Intégralité des exos donnés en 2023-2024-2025

Table des matières

1 Langages formels

Exercice 1 (*) *Parties de mots*

1. On considère le mot m suivant : embarrassante.
   a) Donner un préfixe propre de m qui soit un mot de la langue française.
   b) Donner un facteur de m (ni préfixe, ni suffixe) qui soit un mot de la langue française.
   c) Donner trois mots de la langue française qui sont des sous-mots de m sans en être facteur.
2. Soit u un mot de longueur n dont toutes les lettres sont distinctes.
   a) Combien u a-t-il de préfixes ? De suffixes ? De facteurs ? De sous-mots ?
   b) Que peut-on dire dans le cas où les lettres de u ne sont plus supposées toutes distinctes ?

1. a) Le mot *embarras* convient.
   b) Quelques mots qui conviennent : bar, ras, sa.
   c) Quelques mots qui conviennent : rasant, basse, mante, marrante.
2. a) Si u est de taille n, u un préfixe de taille 0 (à savoir, ε), un de taille 1, ..., un de taille n (lui-même) donc u a en tout $n+1$ préfixes. Le raisonnement est le même pour les suffixes.

   On raisonne similairement pour les facteurs en les regroupant par taille : u a un facteur de taille 0, n facteurs de tailles 1 (chaque lettre de u en est facteur et ces facteurs sont tous différents par hypothèse), $n-1$ facteurs de taille 2, ..., un facteur de taille n. Le nombre de facteurs de u est :

$$1 + \sum_{i=1}^{n} i = 1 + \frac{n(n+1)}{2}$$

   Construire un sous-mot de u revient à parcourir ses lettres dans l'ordre et décider si oui ou non cette lettre fera partie du sous-mot. Pour chacune des n lettres de u on a deux choix (la prendre dans le sous-mot ou pas) donc on obtient 2^n sous-mots. Une autre façon de voir les choses est de regrouper les sous-mots par taille : il y a $\binom{n}{k}$ sous-mots de taille k dans u donc $\sum_{k=1}^{n} \binom{n}{k} = \sum_{k=1}^{n} \binom{n}{k} 1^k 1^{n-k} = (1+1)^n = 2^n$ sous-mots en tout.

   b) Le mot u continue d'avoir $n+1$ préfixes puisque lesdits $n+1$ préfixes sont de tailles deux à deux différentes. Idem pour les suffixes. En revanche, les valeurs obtenues précédemment pour les facteurs et sous-mots deviennent des bornes supérieures qui ne sont plus nécessairement atteintes : par exemple aa n'a que 3 facteurs et non 4 et n'a que 3 sous-mots et non 4.

Exercice 2 (**) *Langages génériques : vrai ou faux ?*

Dans la suite, A, B et C désignent des langages sur un même alphabet Σ. Pour chacune des affirmations suivantes, dire si elle est vraie ou fausse en justifiant dans tous les cas.

1. $(A^\star)^\star = A^\star$.
2. $(AB)^\star = A^\star B^\star$.
3. $A(B+C) = AB + AC$.
4. $A(B \cap C) = AB \cap AC$.
5. $A^+ = A^\star \setminus \{\varepsilon\}$.
6. $A \subset AA$.

1. Vrai, par double inclusion.
2. Faux, contre-exemple : $abab \in (ab)^\star$ mais $abab \notin a^\star b^\star$.

3. Vrai, par double inclusion.
4. Faux, contre-exemple avec $A = \{a, ab\}$, $B = \{bb\}$ et $C = \{b\}$: $A(B \cap C) = \emptyset$ alors que $AB \cap AC = \{abb\}$. L'inclusion $A(B \cap C) \subset AB \cap AC$ est vraie, c'est la réciproque qui est fausse.
5. Faux, contre-exemple : n'importe quel langage qui contient ε.
6. Faux, contre-exemple : n'importe quel langage qui ne contient pas ε.

Exercice 3 (**) *Autour du lemme de Levi*

Soit Σ un alphabet. Dans la suite, un mot désigne (bien sûr) un mot de $\Sigma^\star$.

1. Soient u, v, w, z quatre mots tels que $uv = wz$. Montrer qu'il existe un unique mot t tel que l'une des conditions suivantes est vérifiée :

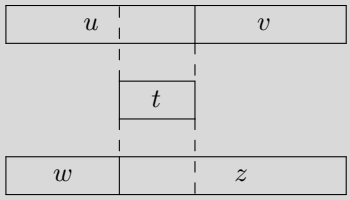
   - $u = wt$ et $z = tv$
   - $w = ut$ et $v = tz$

   Remarque : Ce résultat est le lemme de Levi et se généralise à certaines familles de monoïdes.
2. Soit x et y deux préfixes d'un même mot m. Montrer que x est préfixe de y ou y est préfixe de x. Si on remplace "préfixe" par "suffixe", le résultat reste-t-il vrai ?
3. Déduire du lemme de Levi que deux mots commutent pour la concaténation si et seulement si ils sont puissances d'un même mot. Reformulé, il s'agit de montrer que :

$$\forall\, u, v \in \Sigma^\star,\ uv = vu \text{ si et seulement si } \exists\, w \in \Sigma^\star, \exists\, (\alpha, \beta) \in \mathbb{N}^2, u = w^\alpha \text{ et } v = w^\beta$$

   Indication : Pour le sens le plus difficile, on pourra procéder par récurrence sur $|uv|$.

1. Constatons tout d'abord l'évidence de ce résultat avec un petit dessin (qui n'est pas une preuve) :



   L'unicité est simple : s'il existe t, t' vérifiant par exemple la première condition, on aurait $uv = wtv = wz = wt'v$ et en simplifiant à gauche et à droite (plus pompeusement : par régularité à gauche et à droite) on obtient $t = t'$. De même dans le cas où la deuxième condition est vérifiée.

   Pour l'existence, supposons tout d'abord que $|w| \leq |u|$. L'égalité $uv = wz$ assure alors que w est préfixe de u (comme sur le dessin) : il existe t tel que $u = wt$. Mézalors en remplaçant, on a $wtv = wz$ et en simplifiant par w, on a bien $z = tv$ et on vient de montrer qu'on est dans le premier cas. Si $|w| > |u|$, le raisonnement est similaire et on est dans le second cas.
2. Si x et y sont préfixes de m, par définition il existe u, v des mots tels que $m = xu = yv$. On peut alors appliquer le lemme de Levi pour conclure mais on peut aussi plus simplement ne refaire que la partie du raisonnement de la question 1 nécessaire pour conclure : si $|x| \leq |y|$, comme $xu = yv$ alors x est préfixe de y et sinon y est préfixe de x. Le résultat reste vrai avec "suffixe".
3. La réciproque est immédiate car un mot commute avec lui même donc avec ses puissances.

   Montrons le sens direct par récurrence forte sur $|uv|$. Si $|uv| = 0$ alors $u = v = \varepsilon$ et dans ce cas u et v sont puissance 0-ème de n'importe quel mot. Soit $n \in \mathbb{N}$ tel que le résultat soit acquis pour $|uv| = k$ pour tout $k \in [\![0, n]\!]$ et soit u, v qui commutent et tels que $|uv| = n + 1$. Alors deux possibilités :

   - Si u vaut ε alors choisir $w = v$, $\alpha = 0$ et $\beta = 1$ permet de conclure. Symétrique si $v = \varepsilon$.
   - Sinon, ni u ni v ne vaut ε. On suppose sans perte de généralité que $|u| \leq |v|$. Comme $uv = vu$ alors u est préfixe de v donc il existe un mot t tel que $\mathbf{v = ut}$ (on vient de redémontrer un morceau du lemme de Levi dans un cas particulier). Mézalors $uut = uv = vu = utu$ donc $ut = tu$ et comme $u \neq \varepsilon$, $|ut| = |v| < |uv|$. Cela garantit qu'on peut appliquer l'hypothèse de récurrence (forte) à t

et u et nous assure l'existence d'un mot w et de deux entiers α et β tels que $u = w^\alpha$ et $t = w^\beta$. Mais, en réinjectant dans l'égalité en gras, $v = ut = w^{\alpha+\beta}$ dans ces conditions ce qui conclut.

Exercice 4 (**) *Mots et langage de Dyck*

Dans cet exercice, on travaille sur l'alphabet $\Sigma = \{a, b\}$. On introduit une valuation ν sur Σ telle que $\nu(a) = 1$ et $\nu(b) = -1$ qu'on étend aux mots de $\Sigma^\star$ de la façon suivante : pour tout mot $m = m_0...m_n \in \Sigma^\star$, $\nu(m) = \sum_{i=0}^{n} \nu(m_i)$. Un mot $m \in \Sigma^\star$ est un mot de Dyck (on dit aussi, "mot bien parenthésé" pour des raisons qui devraient être évidentes vu leur définition) si $\nu(m) = 0$ et tout préfixe p de m vérifie $\nu(p) \geq 0$. On note $\mathcal{D}$ l'ensemble des mots de Dyck et on l'appelle le langage de Dyck.

1. Donner quelques exemples de mots de Dyck et de mots de $\Sigma^\star$ qui ne sont pas des mots de Dyck.
2. Montrer que $\varepsilon \in \mathcal{D}$ et que pour tous mots $u, v \in \mathcal{D}$, le mot $aubv$ est aussi dans $\mathcal{D}$.
3. Montrer que tout mot de Dyck $m \neq \varepsilon$ se décompose sous la forme $aubv$ avec $u, v \in \mathcal{D}$.

   Indication : Considérer le plus petit préfixe non vide de m qui a une valuation nulle.
4. Montrer que la décomposition précédente est unique.

Remarque : Nous reparlerons du langage de Dyck, notamment lors du cours sur les grammaires.

1. Par exemple, *aabb* et *aababb* sont des mots de Dyck mais pas *aaa* ou *bab*.
2. La somme sur un ensemble vide valant 0, il est clair que $\varepsilon \in \mathcal{D}$. Soit désormais $u, v \in \mathcal{D}$ et observons $aubv$. On a $\nu(aubv) = \nu(a) + \nu(u) + \nu(b) + \nu(v) = 1 + 0 - 1 + 0 = 0$ car u et v sont des mots de Dyck. D'autre part, si p est un préfixe de $aubv$ alors :
   - soit $p = \varepsilon$, auquel cas $\nu(p) \geq 0$,
   - soit $p = ap'$ avec p' préfixe de u, auquel cas $\nu(p) = 1 + \nu(p') \geq 1$,
   - soit $p = aub$, auquel cas $\nu(p) = 1 + 0 - 1 = 0 \geq 0$,
   - soit $p = aubp'$ avec p' un préfixe de v, auquel cas $\nu(p) = \nu(p') \geq 0$.

   Donc tout préfixe de $aubv$ a une valuation positive. Les deux conditions caractérisant les mots de Dyck sont ainsi satisfaites par $aubv$.
3. Soit $m \in \mathcal{D} \setminus \{\varepsilon\}$. L'ensemble $E = \{|p| \,|\, p \text{ est préfixe de } m, p \neq \varepsilon \text{ et } \nu(p) = 0\}$ est non vide puisque $|m|$ est dans E : il admet donc un élément minimal ce qui assure l'existence d'un plus petit préfixe (au sens de la longueur) p de m non vide et de valuation nulle. En particulier, il existe un mot v tel que $m = pv$. Alors :
   - $v \in \mathcal{D}$. En effet, $\nu(v) = \nu(m) - \nu(p) = 0 - 0 = 0$ et si v' est préfixe de v alors pv' est préfixe de m et donc $0 \leq \nu(pv') = \nu(p) + \nu(v') = 0 + \nu(v')$.
   - $p = aub$. En effet, p est non vide et ne peut pas commencer par b sinon b serait un préfixe de m ce qui est impossible car la valuation de b est strictement négative. De même, p ne peut pas terminer par a, sinon p serait de la forme $ap'a$. Mézalors ap' serait préfixe de m donc on aurait $\nu(ap') \geq 0$ et en même temps $0 = \nu(p) = \nu(ap'a) = \nu(ap') + 1$ c'est-à-dire $\nu(ap') = -1$.
   - Dans la décomposition précédente, $u \in \mathcal{D}$. Déjà, $0 = \nu(p) = \nu(aub) = 1 + \nu(u) - 1$ donc $\nu(u) = 0$ est acquis. Soit un préfixe u' de u. Alors au' est préfixe de m donc $\nu(au') \geq 0$. Mais, $\nu(au')$ ne peut pas être nul car sinon au' serait un préfixe de m non vide, de valuation nulle et strictement plus petit que le plus petit tel préfixe ! Donc on a $\nu(au') \geq 1$ puis $\nu(u') = \nu(au') - 1 \geq 0$.
4. Soit $m \in \mathcal{D} \setminus \{\varepsilon\}$ et supposons qu'il existe des mots de Dyck u, v, u', v' tels que $m = aubv = au'bv'$. On peut simplifier par a, puis, sans perte de généralité, supposer que u' est préfixe de u. Si u' était différent de u, alors u aurait $u'b$ parmi ses préfixes mais $\nu(u'b) = \nu(u') - 1 = 0 - 1 < 0$ ce qui contredit le fait que u est un mot de Dyck. Donc $u = u'$ et par suite $v = v'$. D'où l'unicité attendue.

Exercice 5 (***) *Résiduels d'un langage*

Soit Σ un alphabet fini. Si m est un mot de $\Sigma^\star$ et L est un langage sur $\Sigma^\star$, on appelle *résiduel de L par rapport à m* le langage $m^{-1}L = \{u \in \Sigma^\star \,|\, mu \in L\}$. L'ensemble $\{m^{-1}L \,|\, m \in \Sigma^\star\}$ s'appelle *l'ensemble des résiduels de L*.

1. Dans cette question, L est le langage dénoté par l'expression rationnelle $b^\star ab^\star$ sur l'alphabet $\{a, b\}$.
   a) Calculer le résiduel de L par rapport à a (c'est-à-dire, caractériser le langage $a^{-1}L$).
   b) Reprendre la question précédente en changeant a par ab. Que constate-t-on ?
   c) Montrer que l'ensemble des résiduels de L est fini et le calculer.
2. Dans cette question, $L = \{u^2 \mid u \in \{a, b\}^\star\}$. Montrer que L admet une infinité de résiduels. *Indication : Considérer les résiduels de L par rapport à chacun des mots ba^n où $n \geq 0$.*

Remarque (qui s'éclairera après le chapitre sur les automates) : On peut montrer qu'un langage est rationnel si et seulement si son nombre de résiduels est fini - ce qui donne une façon de prouver le caractère rationnel d'un langage. Il existe d'autre part un lien fort entre le nombre de résiduels d'un langage rationnel L et le nombre minimal d'états d'un automate permettant de reconnaître L.

1. a) Si u est un mot, $u \in a^{-1}L \Leftrightarrow au \in b^\star ab^\star \Leftrightarrow u \in b^\star$ car au contient déjà une fois la lettre a.
   b) Le même raisonnement montre que $(ab)^{-1}L = b^\star$ aussi.
   c) On fait une disjonction de cas selon le nombre de a dans le mot u :

$$u^{-1}L = \begin{cases} L \text{ si } |u|_a = 0 \\ b^\star \text{ si } |u|_a = 1 \\ \emptyset \text{ si } |u|_a \geq 2 \end{cases}$$

   Ceci se prouve par doubles inclusions mais c'est clair intuitivement : $u^{-1}L$ est le langage des mots avec lesquels on peut compléter u pour obtenir un mot de L. Par exemple, si u contient déjà deux a, alors il est impossible de le compléter pour obtenir un mot de L d'où le dernier cas.

2. Montrons que la fonction suivante est injective : $\varphi : \begin{cases} \mathbb{N} \to \{u^{-1}L \mid u \in \Sigma^\star\} \\ n \mapsto (ba^n)^{-1}L \end{cases}$

   Si $\varphi(n) = \varphi(m)$ alors en particulier $ba^n \in (ba^m)^{-1}L$ puisque $ba^n ba^n \in L$, donc $ba^n \in (ba^n)^{-1}L = (ba^m)^{-1}L$. On en déduit qu'il existe $u \in \Sigma^\star$ tel que $ba^n ba^m = u^2$. Cette égalité oblige à ce que $u = bu'$ et en identifiant les b de la seule manière possible dans l'égalité $ba^n ba^m = bu'bu'$ on obtient que $u' = a^n = a^m$ d'où on déduit que $n = m$. On en déduit que $\{(ba^n)^{-1}L \mid n \in \mathbb{N}\}$ est une famille infinie de résiduels de L.

   Ceci montre au passage, d'après la remarque, que L n'est pas rationnel, ce dont on ne doutait pas.

Exercice 6 (*) *Mots de langages rationnels*

On se place sur l'alphabet $\Sigma = \{A, T, C, G\}$. Dans chaque cas, déterminer tous les mots de taille au plus 3 qui appartiennent au langage dénoté l'expression rationnelle considérée et donner 2 mots de taille 3 qui n'y appartiennent pas :

1. $(T|TA)^\star$
2. $AC^\star + C$
3. $(G + \varepsilon)TT^\star$
4. $(A + C)^\star ACC$
5. $A^\star(C|G)T^\star$
6. $(CG|\varepsilon)G$

Énumérer les mots par tailles permet de ne pas en oublier.

1. Appartiennent : ε, T, TA, TT, TTA. N'appartiennent pas : TAA, ATA.
2. Appartiennent : C, A, AC, ACC. N'appartiennent pas : AAA, AAC.
3. Appartiennent : T, GT, TT, GTT, TTT. N'appartiennent pas : GGG, GGT.
4. Appartiennent : ACC. N'appartiennent pas : CAC, ACA.
5. Appartiennent : C, G, AC, AG, CT, GT, AAC, AAG, ACT, AGT, CTT, GTT. N'appartiennent pas : CAT, AAA.
6. Appartiennent : G, CGG. Ce sont en fait les deux seuls mots de ce langage.

Exercice 7 (*) *Du langage rationnel à sa description*

On se place sur l'alphabet à deux lettres $\Sigma = \{a, b\}$.

1. Donner une description simple en français des langages dénotés par les expressions régulières :

   a) $(\varepsilon + \Sigma)(\varepsilon + \Sigma)$

   b) $(\Sigma^2)^\star$

   c) $(b + ab)^\star(a + \varepsilon)$

   d) $(ab^\star a + b)^\star$

   e) $a^\star(a + b)^\star$

   f) $(a^\star b^\star)^\star$

2. Que peut-on dire des expressions rationnelles e et f ?

3. Lesquelles parmi ces expressions rationnelles sont linéaires ?

1. a) Les mots sur Σ ayant au plus deux lettres.
   b) Les mots sur Σ ayant un nombre pair de lettres.
   c) Les mots sur $\{a, b\}$ dans lesquels il n'y a jamais deux a consécutifs.
   d) Les mots sur $\{a, b\}$ ayant un nombre pair de a.
   e) Tous les mots sur $\{a, b\}$.
   f) Tous les mots sur $\{a, b\}$.
2. On remarque que les deux dernières expressions dénotent le même langage.
3. Seule l'expression f est linéaire.

Exercice 8 (*) *De sa description au langage rationnel*

Dans chacun des cas suivants, déterminer une expression rationnelle dénotant le langage :

1. Des mots sur l'alphabet $\{0, 1\}$ qui contiennent au moins un 0.
2. Des mots sur l'alphabet $\{0, 1\}$ qui contiennent au plus un 0.
3. Des mots sur l'alphabet $\{0, 1, 2\}$ qui ont 202 comme facteur.
4. Des mots sur l'alphabet $\{0, 1, 2\}$ qui ont 202 comme sous mot.
5. Des mots sur l'alphabet $\{0, 1\}$ tels que toute série de 0 consécutifs est de longueur paire.
6. Des mots sur l'alphabet $\{0, 1\}$ tels que deux lettres consécutives sont toujours distinctes.
7. Des mots u sur l'alphabet $\{0, 1\}$ tels que $|u|_1 \equiv 0 \bmod 2$.
8. Des mots sur un alphabet quelconque Σ dont la longueur n'est pas divisible par 3.

1. $(0 + 1)^\star 0(0 + 1)^\star$.
2. $1^\star(0 + \varepsilon)1^\star$.
3. $(0 + 1 + 2)^\star 202(0 + 1 + 2)^\star$.
4. En notant $\Sigma = \{0, 1, 2\}$: $\Sigma^\star 2\Sigma^\star 0\Sigma^\star 2\Sigma^\star$.
5. $(00 + 1)^\star$.
6. $(01)^\star + (10)^\star + 1(01)^\star + 0(10)^\star$.
7. $(01^\star 0 + 1)^\star$.
8. $\Sigma(\Sigma^3)^\star + \Sigma^2(\Sigma^3)^\star$.

Exercice 9 (**) *Vrai ou faux ?*

Dans la suite, a et b sont des lettres d'un alphabet. Pour chacune des affirmations suivantes, déterminer si elle est vraie ou fausse en justifiant :

1. Les langages $a^\star b^\star$ et $\{a, b\}^\star$ sont égaux.
2. Les langages $(a^\star + b^\star)^\star$ et $\{a, b\}^\star$ sont égaux.
3. L'assertion précédente reste vraie si on remplace a et b par n'importe quelles expressions rationnelles.
4. Le langage $a^\star bc^\star$ est local.
5. Le langage $a^\star ba^\star$ est local.

6. Un langage rationnel est local.
7. Tout langage rationnel est infini.
8. Il y a une infinité de langages non rationnels.
9. Tout langage inclus dans un langage rationnel est rationnel.
10. Tout langage rationnel est dénoté par une infinité d'expressions rationnelles différentes.

1. Faux, $ba \in \{a,b\}^\star$ mais n'est pas dans $a^\star b^\star$.
2. Vrai. On a évidemment $(a^\star + b^\star)^\star \subset \{a,b\}^\star$ puisque le deuxième langage est le langage de tous les mots sur $\{a,b\}$. De plus $a \subset a^\star + b^\star$ et $b \subset a^\star + b^\star$ donc $\{a,b\} \subset a^\star + b^\star$ puis $\{a,b\}^\star \subset (a^\star + b^\star)^\star$.
3. Vrai, avec la même preuve.
4. Vrai, puisqu'il est dénoté par une expression linéaire.
5. Faux, s'il était local alors il devrait contenir *abab* ce qui n'est pas.
6. Faux, d'après la question précédente.
7. Faux, $\emptyset$ par exemple est rationnel et fini.
8. Vrai, et même une infinité non dénombrable. S'il y avait un nombre fini de langages non rationnels, comme l'ensemble des langages rationnels est dénombrable, alors l'ensemble de tous les langages serait (dénombrable $\cup$ fini) lui aussi dénombrable ce qui n'est pas.
9. Faux. Pour n'importe quel alphabet Σ, $\Sigma^\star$ est rationnel. Or il existe un langage non rationnel L sur Σ (si on ne sait pas l'exhiber : par argument de cardinalité) et $L \subset \Sigma^\star$ donne la contradiction.
10. Vrai : si L est dénoté par e alors L est aussi dénoté par $e + \underbrace{\emptyset + ... + \emptyset}_{k \text{ fois}}$ ou par $e\,\underbrace{\varepsilon ... \varepsilon}_{k \text{ fois}}$ pour tout $k \in \mathbb{N}$.

Exercice 10 (**) *Stabilités de langages locaux*

1. L'intersection de deux langages locaux est-il un langage local ? Justifier.
2. Le complémentaire d'un langage local est-il un langage local ? Justifier.
3. Si L est un langage local, montrer que le langage L' de ses facteurs est aussi un langage local.

1. Oui. Soit L_1 et L_2 deux langages locaux. Notons $P = P(L_1) \cap P(L_2)$, $D = D(L_1) \cap D(L_2)$ et $N = N(L_1) \cup N(L_2)$. Alors (la deuxième égalité vient de la localité de L_1 et L_2) :

$$\begin{aligned}(L_1 \cap L_2) \setminus \{\varepsilon\} &= (L_1 \setminus \{\varepsilon\}) \cap (L_2 \setminus \{\varepsilon\}) \\ &= [P(L_1)\Sigma^\star \cap \Sigma^\star D(L_1) \setminus \Sigma^\star N(L_1)\Sigma^\star] \cap [P(L_2)\Sigma^\star \cap \Sigma^\star D(L_2) \setminus \Sigma^\star N(L_2)\Sigma^\star] \\ &= (P\Sigma^\star \cap D\Sigma^\star) \setminus \Sigma^\star N \Sigma^\star\end{aligned}$$

   Donc par définition $L_1 \cap L_2$ est local.
2. Non. On sait que $L_1 \cup L_2 = (L_1^c \cap L_2^c)^c$ donc si la localité était stable par complémentaire, comme elle est stable par intersection vu la question 1, elle serait stable par union ce qui n'est pas le cas.
3. Notons Σ l'ensemble des lettres qui interviennent dans au moins un mot de L. Remarquons alors que $P(L') = D(L') = \Sigma$. Montrons que $loc(l') := (P(L')\Sigma^\star \cap \Sigma^\star D(L')) \setminus \Sigma^\star N(L')\Sigma^\star \subset L' \setminus \{\varepsilon\}$; comme l'autre inclusion est toujours vraie, ça concluera.

   Soit $u = u_1...u_p$ un mot de $loc(L)$. Comme $u_1 \in P(L') = \Sigma$, u_1 est une lettre d'un mot de L donc il existe $b_1, ..., b_n$ des lettres telles que $b_1...b_n u_1$ est un préfixe d'un mot de L (avec n potentiellement nul). Le même raisonnement montre qu'il existe des lettres $c_1, ..., c_m$ telles que $u_p c_1...c_m$ est préfixe d'un mot de L. On montre alors pédestrement que $b_1...b_n u_1...u_p c_1...c_m$ commence par une lettre dans $P(L)$, termine par une lettre de $D(L)$ et ne contient aucun facteur de $N(L)$ donc appartient à $L \setminus \{\varepsilon\}$ par localité de L. D'où on déduit que u est facteur d'un mot de L ce qui conclut.

Exercice 11 (exo cours) *Mots finissant par* 0

1. Donner une expression rationnelle dénotant le langage L des mots sur $\{0,1\}$ finissant par 0.

2. Déterminer une expression rationnelle e pour L^c et montrer que $L(e) = L^c$.

1. $(1+0)^\star 0$.
2. On propose $e = (1+0)^\star 1 + \varepsilon$. Preuve par double inclusion.

Exercice 12 (exo cours) *Puissances paires de a*

1. Décrire en français le langage sur $\Sigma = \{a, b\}$ dénoté par $(a^2)^\star a$.
2. On considère le langage L sur Σ des puissances de a de longueur paire. Proposer une expression rationnelle e dénotant L^c et prouver que $L^c = L(e)$.

1. C'est le langage des mots formés de a et de longueur impaire.
2. On propose $e = (a^2)^\star a + (a+b)^\star b (a+b)^\star$. Preuve par double inclusion. Le sens direct vient du fait qu'un mot de L^c est soit un mot qui est une puissance de a de longueur impaire, soit un mot contenant un b.

Exercice 13 (exo cours) *Mots binaires*

Soit Σ l'alphabet $\{0, 1\}$. Un mot binaire est un mot de $\Sigma^\star$. Un mot binaire normalisé est un mot binaire qui commence par 1 ou vaut exactement 0. La valeur d'un mot binaire $u = u_{n-1}...u_0$ est définie par $V(u) = \sum_{i=0}^{n-1} 2^i u_i$.

1. Donner une expression rationnelle dénotant chacun des langages suivants.
   a) L_1 est le langage des mots binaires normalisés.
   b) L_2 est le langage des mots binaires dont la valeur est paire.
   c) L_3 est le langage des mots binaires normalisés dont la valeur est paire.
   d) L_4 est le langage des mots binaires normalisés dont la valeur est une puissance de 2.
2. Dans le cas de L_4, prouver que l'expression e proposée vérifie bien $L(e) = L_4$.

1. a) $0 + 1(0+1)^\star$
   b) $(1+0)^\star 0$
   c) $0 + 1(1+0)^\star 0$
   d) $10^\star$
2. Par double inclusion. Si $u \in L(e)$, il existe $n \in \mathbb{N}$ tel que $u = 10^n$ et alors $V(u) = 2^n$ et u commence par 1 donc u est normalisé et une puissance de 2. Réciproquement, si u est normalisé et une puissance de 2, $V(u) = 2^k = \sum_{i=0}^{n-1} 2^i u_i$ avec $u = u_{n-1}...u_0$ et $u_{n-1} = 1$ (car 0 n'est pas puissance de 2) Alors $2^k = 2^{n-1} +$ reste et donc $n - 1 \leq k$ et si on avait $n - 1 < k$, l'égalité serait fausse car le reste est trop petit. Il suit que tous les u_i sauf le premier sont nuls.

Exercice 14 (exo cours) *Local ou pas local ?*

Parmi les expressions suivantes, déterminer lesquelles dénotent un langage local en justifiant :

1. $abbab$
2. $(ab)^\star$
3. $a^\star$
4. $a^\star(ab)^\star$
5. a^+

Si L est un langage, on note $\mathrm{loc}(L) = (P(L)\Sigma^\star \cap \Sigma^\star D(L)) \setminus \Sigma^\star N(L) \Sigma^\star$ avec $P(L)$ les premières lettres, $D(L)$ les dernières et $N(L)$ les non facteurs.

1. Non local car $ab \in \mathrm{loc}(L)$ mais pas à L.
2. Local car linéaire.
3. Local car linéaire.

4. Non local car $aba \in \text{loc}(L)$ mais pas à L.
5. Local en montrant que $\text{loc}(L) \subset L \setminus \{\varepsilon\}$.

Exercice 15 (exo cours) *Existence de langages non rationnels*

On se dote d'un alphabet Σ non vide.

1. Montrer que $\text{Rat}(\Sigma^\star)$ est dénombrable.
2. En admettant qu'un ensemble n'est jamais en bijection avec l'ensemble de ses parties, montrer qu'il existe des langages sur Σ qui ne sont pas rationnels.

1. Il y a moins de langages rationnels que d'expressions rationnelles. Or l'ensemble des expressions rationnelles sur Σ est inclus dans l'ensemble L des mots sur $\Sigma \cup \{(,), *, +\}$ et
$$L = \bigcup_{n \in \mathbb{N}} \{u \in L \,|\, |u| = n\}$$
est une union dénombrable d'ensembles finis donc est dénombrable.
2. La même preuve montre que $\Sigma^\star$ est dénombrable. Comme $\Sigma^\star$ et $\mathcal{P}(\Sigma^\star)$ ne sont pas en bijection, l'ensemble des langages est non dénombrable. La question 1 conclut.

Exercice 16 (exo cours) *Localité des langages linéaires*

1. Montrer que toute expression rationnelle linéaire dénote un langage local.
2. La réciproque de ce résultat est-elle vraie ? Justifier.

1. On procède par induction. On utilise la stabilité des langages locaux par $*$ et la stabilité par $\cdot$ et $\cup$ dans le cas d'alphabets disjoints (ce qui est le cas à cause de la linéarité).
2. Non. $aa^\star$ n'est pas linéaire mais dénote un langage local.

Exercice 17 (exo cours) *Stabilité de la localité par union*

1. Montrer que l'union de deux langages locaux sur des alphabets disjoints reste un langage local.
2. Ce résultat reste-t-il vrai si on supprime l'hypothèse de disjonction des alphabets ? Justifier.

1. Soit L_1 défini sur Σ_1 et L_2 défini sur Σ_2 disjoint de Σ_1 deux langages locaux. On montre que $u = u_1 ... u_n \in \text{loc}(L_1 + L_2)$ est dans $(L_1 + L_2) \setminus \{\varepsilon\}$ l'autre inclusion étant toujours vraie. On a $u_1 \in P(L_1 + L_2) = P(L_1) + P(L_2)$. Si $u_1 \in P(L_1)$, la disjonction des alphabets permet de montrer récursivement que tous les facteurs de taille 2 de u sont dans $F(L_1)$ et que $u_p \in D(L_1)$ donc que $u \in \text{loc}(L_1) = L_1 \setminus \{\varepsilon\}$ par localité de L_1. Même raisonnement si $u_1 \in P(L_2)$.
2. Non. $a^\star$ et $(ab)^\star$ dénotent des langages linéaires donc locaux mais $a^\star + (ab)^\star$ n'est pas local.

Exercice 18 (**) *Mots de Fibonacci*

Les mots de Fibonacci sur l'alphabet $\{a, b\}$ sont définis par les relations suivantes :
$$f_0 = \varepsilon, \quad f_1 = a, \quad f_2 = b, \quad \forall n \geq 1, f_{n+2} = f_{n+1} f_n$$

1. Déterminer f_4.
2. Montrer que pour tout $n \geq 3$, le suffixe de longueur 2 de f_n est ab si n est pair et ba si n est impair.
3. Pour tout $n \geq 3$, on note g_n le préfixe de f_n obtenu en supprimant les deux dernières lettres de ce mot. Montrer que g_n est un palindrome.

1. $f_3 = ba$ et $f_4 = bab$.
2. On montre par récurrence sur $n \geq 3$: "f_3 est suffixe de f_n si n est impair et f_4 est suffixe de f_n lorsque n est pair". Puis on conclut.
3. On procède par récurrence. Pour l'hérédité on distingue selon la parité de $n+1$ pour utiliser la question

2. Par exemple pour $n+1$ pair, $f_{n+1} = f_{n-1}f_{n-2}f_{n-1} = \underbrace{g_{n-1}abg_{n-2}bag_{n-1}}_{=g_{n+1},\text{ palindrome}}ab$.

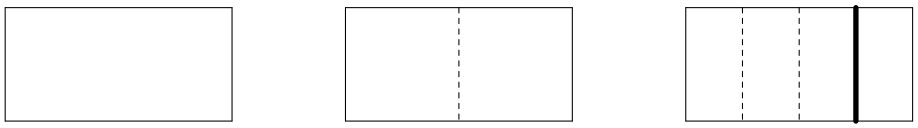
Exercice 19 (***) *Génération de mots par plis*

On considère sur $\{a,b\}^\star$ l'application $m \mapsto \overline{m}$ définie par :

$$\overline{\varepsilon} = \varepsilon \text{ et } \forall m \in \{a,b\}^\star,\ (\overline{am} = \overline{m}b \text{ et } \overline{bm} = \overline{m}a)$$

1. Calculer $\overline{aababa}$.
2. Montrer que pour tous $u, v \in \{a,b\}^\star$, on a $\overline{uv} = \overline{v}\,\overline{u}$.

Prenons une feuille de papier et plions la n fois dans le sens vertical en repliant systématiquement la motié droite sur la moitié gauche. Une fois la feuille redépliée, les plis forment une suite de creux et de bosses. Par exemple, si un trait pointillé représente un creux et un trait épais une bosse, on a :



$n=0$ $n=1$ $n=2$

Cette suite de plis peut être codée par un mot de $\{a,b\}^\star$: si a code un creux et b code une bosse, les premiers mots obtenus sont ε, a, aab. On note u_n le mot associé à la suite de creux et bosses obtenues avec n pliages de la feuille initiale.

3. Déterminer u_3.
4. Exprimer u_{n+1} en fonction de u_n. Calculer la longueur de u_n.

On note $u_{n,k}$ la k-ème lettre de u_n (on numérote à partir de un).

5. Montrer que les lettres d'indice $4p+1$ de u_n sont des a et celles d'indice $4p+3$ des b.
6. En admettant que pour tout k pair inférieur à $|u_n|$, $u_{n,k} = u_{n,k/2}$, déduire de ce qui précède $u_{11,2000}$.

1. $bababb$.
2. Procéder par récurrence sur $|u|$.
3. $u_3 = aabaabb$. On peut s'aider d'une feuille physique.
4. On a $u_{n+1} = u_n a\overline{u_n}$. On montre par récurrence que $|u_n| = 2^n - 1$.
5. Montrons le par récurrence sur n. OK pour $n \leq 2$. Supposons qu'on ait le résultat au rang n. Alors $u_n = aw_2bw_4...aw_mb$ où les w_i sont les mots de deux lettres coincés entre une occurrence de a en position $4p+1$ et une occurrence de b en position $4p+3$. La dernière lettre est un b car son indice est $|u_n| = 2^n - 1 \equiv 3 \bmod 4$.

   Alors par 4 et 2 on $u_{n+1} = aw_2bw_4...aw_mb\ a\,a\overline{w_m}...a\overline{w_2}b$ ce qui conclut.
6. Comme $|u_{11}| = 2^{11} - 1 = 2047 \geq 2048$, la 2000-ème lettre de u_{11} existe. De plus d'après le résultat de l'énoncé (qu'on fera montrer aux forts), $u_{11,2000} = u_{11,125} = a$ car $125 \equiv 1 \bmod 4$.

Exercice 20 (**) *Langage des préfixes*

On note $\text{Pref}(L)$ le langage des préfixes des mots de L.

1. Déterminer $\text{Pref}(L)$ pour $L = \{abb, ba, bab, aaab\}$.
2. Montrer que $L \subset \text{Pref}(L)$. Qu'en est-il de l'inclusion réciproque ?
3. Déterminer $\text{Pref}(L_1L_2)$ pour tous langages L_1, L_2.

1. $\text{Pref}(L) = \{\varepsilon, a, b, aa, ab, ba, aaa, abb, bab, aaab\}$.
2. Un mot est préfixe de lui même. L'inclusion réciproque est fausse par 1.

3. Si L_1 ou L_2 est vide, L_1L_2 et $\text{Pref}(L_1L_2)$ aussi. Sinon on montre par double inclusion que $\text{Pref}(L_1L_2) = \text{Pref}(L_1) + L_1\text{Pref}(L_2)$.

Exercice 21 (***) *Codes préfixes*

On appelle *code* sur un alphabet Σ tout langage L vérifiant la propriété suivante :

$$\text{Si } u_1, u_2, ..., u_p, v_1, v_2, ...v_q \in L \text{ sont tels que } u_1u_2...u_p = v_1v_2...v_q, \text{ alors } p = q \text{ et } \forall i \in [\![1, p]\!], u_i = v_i.$$

Autrement dit, un mot de $L^\star$ se factorise de manière unique.

1. Donner sans preuve un code de quatre mots sur $\{a, b\}$.
2. Montrer que $L_1 = \{a, ab, ba\}$ n'est pas un code sur $\{a, b\}$. Qu'en est-il de $L_2 = \{a, ab\}$?
3. Montrer que si tous les mots de L ont la même longueur alors L est un code.
4. Soit $u, v \in \Sigma^\star$ distincts et non vides. Montrer que $\{u, v\}$ est un code si et seulement si $uv \neq vu$.
5. Montrer que si L est un langage ne contenant pas ε tel qu'aucun mot de L ne soit préfixe strict d'un autre mot de L alors L est un code. *Remarque : de tels codes sont appelés des codes préfixes.*

1. $\{a, ba, bba, bbba\}$ convient (cf question 5).
2. $aba = a \cdot ba = ab \cdot a$ donc L_1 n'est pas un code.

   L_2 est un code. Si ce n'est pas le cas il existe un mot qui se factorise de deux façons $u_1...u_p = v_1...v_q$. On peut prendre un de ceux tels que p est minimal. Mézalors $u_p = v_q$ vu les mots de L_2 et $u_1...u_{p-1} = v_1...v_{q-1}$ contredisant la minimalité de p.
3. Si $u_1...u_p = v_1...v_q$, u_1 est préfixe de v_1 et v_1 est préfixe de u_1. Comme ces mots ont même longueur, ils sont égaux, on peut simplifier et réitérer jusqu'à montrer l'unicité de la factorisation.
4. ($\Rightarrow$) OK. ($\Leftarrow$) On le montre par récurrence forte sur $|uv|$. L'initialisation correspond au cas où u et v sont deux lettres distinctes et alors 3 conclut. Si $u_1...u_p = v_1...v_q$, après simplification d'un éventuel préfixe commun on a $u_1 \neq v_1$ donc par exemple $u_1 = u$ et $v_1 = v$.

   On peut supposer $|u| \leq |v|$ et alors $v = uv'$ avec $|v'| < |v|$ car $u \neq \varepsilon$. Comme $uv \neq vu$, $uv' \neq v'u$ et on a $u_2...u_q = v'v_2...v_q$ $(*)$ deux factorisations sur $\{u, v'\}$ du même mot avec $|uv'| < |uv|$: par hypothèse de récurrence, celle ci est unique. Cette factorisation commence par v' à droite et par u à gauche car u est préfixe de u_2 car $u_2 = u$ ou $u_2 = v = uv'$. Par unicité de la factorisation $(*)$, $u = v'$ et ça contredit $uv' \neq v'u$.
5. (abs) Si $u_1...u_p = v_1...v_q$ avec $u_1 \neq v_1$, on peut supposer $|u_1| < |v_1|$ et u_1 est préfixe strict de v_1.

Exercice 22 (**) *Preuve de non rationnalité par induction*

On se place sur l'alphabet $\{a, b\}$. Pour tout $k \in \mathbb{N}$ et tout $I \subset \mathbb{N}$ infini, on note $L_{k,I} = \{a^n b^{n+k} \mid n \in I\}$. L'objectif de l'exercice est de montrer que $E = \{L_{k,I} \mid k \in \mathbb{N} \text{ et } I \subset \mathbb{N} \text{ infini}\}$ ne contient aucun langage rationnel. Pour ce faire, on procède par l'absurde et on considère une des expressions rationnelles e de taille minimale (c'est-à-dire dont l'arbre est de taille minimale) dénotant un des langages de E.

1. Montrer que e n'est ni $\emptyset$, ni ε, ni a ni b.
2. Montrer que e ne peut pas être de la forme $f + g$ où f, g sont des expressions rationnelles.
3. Montrer que e ne peut pas être de la forme $f^\star$ avec f une expression rationnelle.
4. Si $e = fg$ avec f, g des expressions rationnelles, l'un des langages parmi $L(f)$ et $L(g)$ est infini. Supposons que ce soit $L(f)$ et considérons un mot $w \in L(g)$.
   a) Montrer que w est de la forme a^ib^j ou b^i.
   b) Montrer que s'il existe un mot w dans $L(g)$ de la forme a^ib^j alors w n'est suffixe que d'un seul mot de $L(e)$ et conclure à une contradiction.
   c) Montrer que dans le cas contraire, $L(f)$ est un des langages de E.
5. Conclure.

1. Tous les $L_{k,I}$ avec I infini sont infinis.
2. Par l'absurde, $e = f + g$. Alors l'un parmi $L(f)$ et $L(g)$ est infini car $L(e)$ l'est. Supposons que c'est $L(f)$. Alors il existe $I_f \subset I$ infini tel que $L(f) = L_{k,I_f}$ ce qui contredit la minimalité de e.
3. Par l'absurde, $e = f^\star$. Il existe $m \in L(f)\backslash\{\varepsilon\}$ sinon $L(e)$ n'est pas infini. Comme $m \in L(e)$, $m = a^n b^{n+k}$ et alors $m^2 = a^n b^{n+k} a^n b^{n+k} \in L(e)$ ce qui n'est possible que si $n = k = 0$ donc si $m = \varepsilon$: contradiction.
4. a) w est suffixe d'un mot de $L(e)$.
   b) Notons $a^n b^{n+k}$ un mot (qui existe) dont $w = a^i b^j$ est suffixe. Alors $n = j - k$ donc w n'est suffixe que d'un mot de $L(e)$. Comme $L(f)$ est infini, il existe $m \neq m'$ dans $L(f)$ et $w \in L(g)$ est suffixe de $mw \in L(f)L(g) = L(e)$ et de $m'w \in L(e)$: contradiction.
   c) Dans ces conditions, on a $L(f) = L_{k-i,I} \in E$ ce qui contredit la minimalité de e. En effet :
      - Si $m \in L(f)$, comme $b^i \in L(g)$, $mb^i \in L(e)$ donc $mb^i = a^n b^{n+k}$ donc $m \in L_{k-i,I}$.
      - Si $m = a^n b^{n+k-i}$, montrons que $\{b^i\} = L(g)$. On a $\subset$ et on sait que tous les mots de $L(g)$ sont de la forme b^j car on n'est pas dans le cas b. Donc si $m \in L(g)$, $m = b^j$ et $a^n b^{n-k+i} b^j \in L(e)$ donc $i = j$.
5. Par définition d'une expression rationnelle.

Exercice 23 (*) *Appartenance de ε aux itérés de langages*

1. Montrer que $\varepsilon \in L^+$ si et seulement si $\varepsilon \in L$.
2. Soit L un langage non vide. Montrer que $\varepsilon \in L$ si et seulement si $L \subset LL$.

1. RAS par double implication.
2. Si $\varepsilon \in L$, $L = \{\varepsilon\}L \subset LL$. Si $L \subset LL$, supposons par l'absurde que $\varepsilon \notin L$. Soit m un des plus courts mots de L : on a $|m| > 0$. Alors $m \in LL$ donc $m = m_1 m_2$ avec m_1 et $m_2 \neq \varepsilon$ donc $|m_1|, |m_2| < |m|$ et contradiction avec la minimalité de $|m|$.

Exercice 24 (*) *Opérations sur les langages*

Dans ce qui suit, A, B, C sont des langages sur un même alphabet.

1. Déterminer en justifiant quelles sont les affirmations correctes parmi :
   a) $(A^\star)^\star = A^\star$.
   b) $A(B + C) = AB + AC$.
   c) $A(B \cap C) = AB \cap AC$.
   d) $(A + B)^\star = (A^\star B^\star)^\star$.
2. Comparer les deux langages dans chacun des cas suivants :
   a) $(A + B)^\star$ et $A^\star + B^\star$.
   b) $(AB)^\star$ et $A^\star B^\star$.

1. a) Vrai, par double inclusion.
   b) Vrai, par doucle inclusion.
   c) Faux, $A(B \cap C) \subset AB \cap AC$ mais l'autre inclusion est fausse. Contre exemple avec $A = \{a, ab\}$, $B = \{bb\}$ et $C = \{b\}$.
   d) Vrai, par double inclusion.
2. a) L'inclusion réciproque est vraie mais pas l'inclusion car $ab \in (a + b)^\star$ mais $\notin a^\star + b^\star$.
   b) Aucune des inclusions n'est vraie. $abab \in (ab)^\star$ mais pas $a^\star b^\star$. $aa \in a^\star b^\star$ mais pas $(ab)^\star$.

Exercice 25 (***) *Mots conjugués*

Soit Σ un alphabet. On définit sur $\Sigma^\star$ la relation de conjuguaison C suivante. On note $C(u, v)$ le fait que le mot u soit en relation avec le mot v via C :

$$C(u, v) \text{ si et seulement si } \exists\, x, y \in \Sigma^\star, u = xy \text{ et } v = yx$$

1. Montrer que C est une relation d'équivalence. Ainsi, lorsque $C(u, v)$, on peut dire "u et v sont conjugués".
2. Montrer que $C(u, v)$ si et seulement il existe un mot w tel que $uw = wv$.
3. Soit $n \in \mathbb{N}^\star$. Montrer que $C(u, v)$ si et seulement si $C(u^n, v^n)$.

1. Soit u, v, w des mots. La réflexivité est acquise : avec $x = u$ et $y = \varepsilon$, on a bien $C(u, u)$. La symétrie est immédiate. Enfin, si $C(u, v)$ et $C(v, w)$ alors il existe des mots x, y, z, t tels que $u = xy$ (1), $v = yx$, $v = zt$ et $w = tz$ (2). En particulier, $v = yx = zt$.

   Si $|y| \leq |z|$ alors y est préfixe de z donc $z = yz'$ (3) et alors $yx = yz't$ donc en simplifiant à gauche, $x = z't$ (4). On déduit en réinjectant (4) dans (1) et (3) dans (2) que $u = z'(ty)$ et $w = (ty)z'$ ce qui montre que $C(u, w)$. Le raisonnement est similaire dans le cas où $|y| > |z|$.
2. TODO via lemme de Levi.

Exercice 26 (**) *Racine carrée de langages*

On définit la racine carrée d'un langage L sur un alphabet Σ par $\sqrt{L} = \{u \in \Sigma^\star \mid u^2 \in L\}$.

Déterminer quelles sont les inclusions vraies parmi les suivantes :

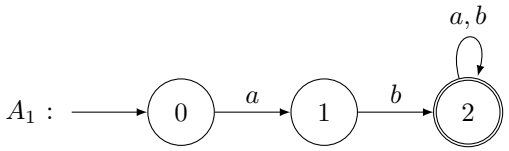
1. $L \subset \sqrt{L^2}$.
2. $\sqrt{L^2} \subset L$.
3. $L \subset (\sqrt{L})^2$.
4. $(\sqrt{L})^2 \subset L$.
5. $(\sqrt{L})^2 \subset \sqrt{L^2}$
6. $\sqrt{L^2} \subset (\sqrt{L})^2$.

1. Vrai, par définition d'une racine.
2. Faux, contre-exemple avec $L = \{\varepsilon, aa\}$.
3. Faux, contre-exemple avec $L = \{a\}$.
4. Faux, contre-exemple avec $L = \{aa, bb\}$.
5. Faux, contre-exemple avec $L = \{aa, bb\}$.
6. Faux, contre-exemple avec $L = \{a\}$.

2 Automates

Exercice 27 (*) *Lectures de mots dans un automate*

On considère deux automates A_1 et A_2. L'automate A_1 est connu via sa représentation graphique :



L'automate A_2 est défini sur l'alphabet $\{a, b\}$, son ensemble d'états est $\{1, 2, 3, 4\}$, son seul état initial est 1, ses états finaux sont 2 et 4 et sa fonction de transition est donnée par la table suivante :

	1	2	3	4
a	2	3	1	1
b		3		2

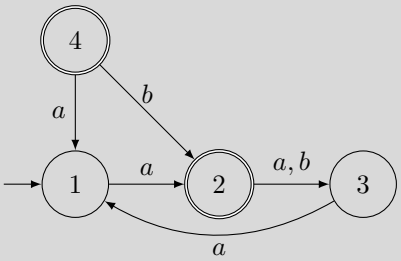
1. L'automate A_1 est-il émondé ? Déterministe ? Complet ?
2. Dans quel état est-t-on dans A_1 après la lecture du mot ab ? Même question avec les mots ε, a, aba^2b, a^2ba^2b, ab^4 et b^3a^2. Parmi ces mots, lesquels sont reconnus par A_1 ? Lesquels sont des blocages ?
3. Décrire les mots reconnus par l'automate A_1.
4. Dessiner l'automate A_2. Est il émondé ? Déterministe ? Complet ?

5. Donner un exemple de mot de longueur 4 reconnu par A_2 et un non reconnu par A_2. Donner une expression rationnelle simple dénotant le langage L_2 reconnu par A_2.
6. Proposer un automate reconnaissant le complémentaire de L_2. Justifier votre construction.

1. Émondé, déterministe mais pas complet.
2. On obtient les résultats suivants :

Mot	ab	ε	a	$abaab$	$aabaab$	$abbbb$	$bbbaa$
État après lecture	2	0	1	2	blocage	2	blocage
Reconnu ?	oui	non	non	oui	non	oui	non

3. Ce sont les mots commençant par ab. En voici une expression rationnelle : $ab(a+b)^\star$.
4. Voici l'automate A_2 :

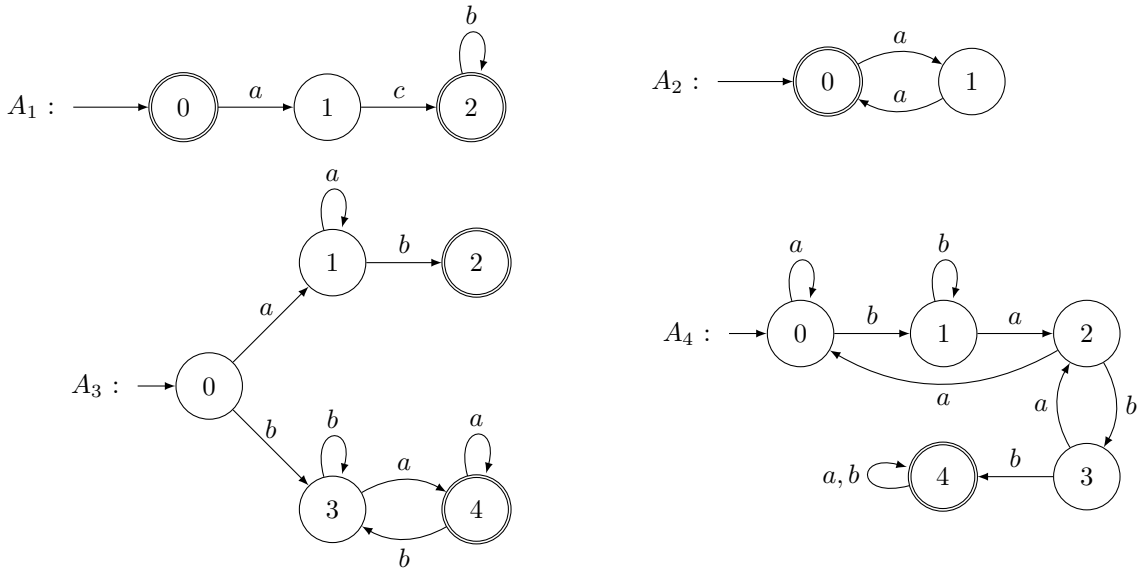


Il est déterministe mais ni complet (on ne peut pas lire b dans 1), ni complet car 4 n'est pas accessible.

5. Par exemple, $abaa$ est reconnu mais pas $abbb$. Le langage reconnu est dénoté par $a + (a(a+b)a)^\star$.
6. On peut déjà émonder A_2 en supprimant l'état 4. On obtient ensuite un automate déterministe qu'on complète en ajoutant un puits. On peut ensuite échanger états finaux et non finaux.

Exercice 28 (*) *Langages reconnus*

Pour chacun des automates ci-dessous, donner en expliquant (mais sans nécessairement donner de preuve formelle) une expression régulière décrivant le langage qu'il reconnaît.

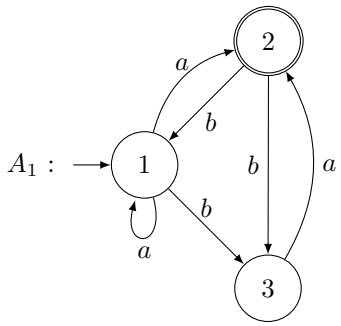


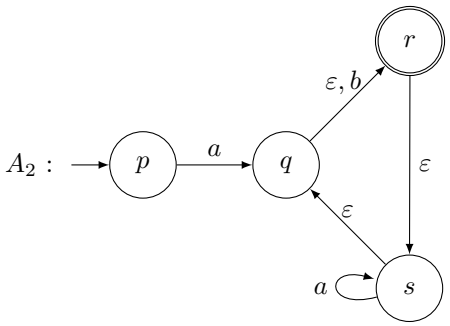
1. Cet automate reconnaît $\varepsilon + acb^\star$.
2. Cet automate reconnaît $(a^2)^\star$.
3. Cet automate reconnaît $\underbrace{a^+b}_{\text{via l'état 2}} + \underbrace{b^+a(a+b^+a)^\star}_{\text{via l'état 4}}$.
4. Cet automate est l'automate des occurrences pour le motif $babb$: il reconnaît $\Sigma^\star babb\Sigma^\star$.

Remarque : il n'est pas attendu que les élèves sachent trouver une expression pour le quatrième automate à ce stade, il sert surtout à ouvrir la discussion sur le fait que sans algorithme (ainsi motivé), il est difficile de trouver une expression correcte à l'oeil, même pour de petits automates.

Exercice 29 (*) *Déterminisations*

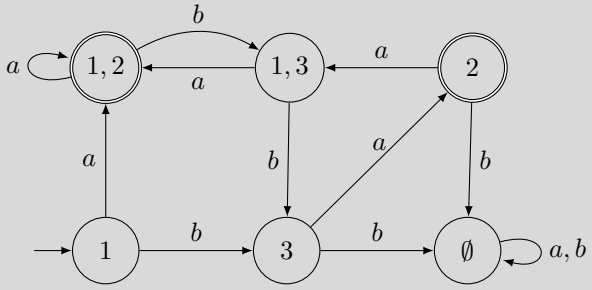
1. Déterminiser les deux automates ci-dessous. On indiquera quelles sont les ε-clôtures des états de A_2. Quel langage reconnait A_2 ?



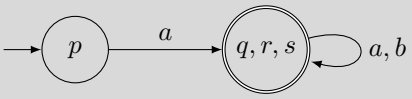


2. Proposer un automate non déterministe permettant de reconnaître le langage sur $\{a, b\}$ des mots dont l'avant dernière lettre est un b. Déterminiser l'automate obtenu.
3. Reprendre la question précédente avec le langage des mots dont $aaba$ est suffixe.
4. L'algorithme de déterminisation exposé en cours construit nécessairement un automate dont tous les états sont accessibles. Les états dudit automate sont-ils nécessairement tous coaccessibles ? Justifier.

1. Tous calculs faits, on devrait trouver l'automate déterministe(et complet) suivant pour A_1 :

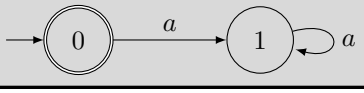


Pour A_2, la clôture de p est $\{p\}$ et celles de q, r, s sont toutes égales à $\{q, r, s\}$. Après déterminisation, on obtient l'automate suivant (en ne dessinant pas l'état puits $\emptyset$) :



On constate alors que le langage reconnu par A_2 est celui des mots sur $\{a, b\}$ commençant par a.

2. RAS.
3. RAS.
4. Pas nécessairement. Par exemple l'automate ci-dessous étant déjà déterministe, il est égal à son déterminisé via l'automate des parties. Dans le déterminisé obtenu, l'état 1 reste non coaccessible :

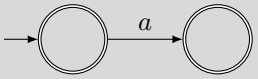


Exercice 30 (**) *Dessine moi un automate*

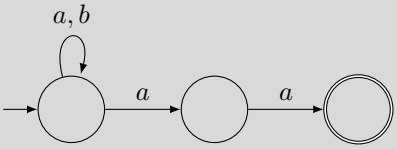
Pour chacun des langages L suivants sur l'alphabet $\{a, b\}$, dessiner un automate **déterministe** qui le reconnaît. La construction proposée doit être suffisamment claire et explicable pour qu'on soit convaincu que l'automate produit reconnaît L. *Indication : on pourra construire des automates intermédiaires y compris non déterministes qu'on modifiera ou raffinera pour répondre aux questions posées.*

1. $L = \{a, \varepsilon\}$
2. $L = \{maa \mid m \in \{a, b\}^\star\}$
3. L est dénoté par $(a^2)^\star$
4. L est dénoté par $(a+b)^\star aba(a+b)^\star$
5. L est le langage des mots qui contiennent ab et bb
6. L est le langage des mots contenant au plus un a
7. L est le langage des mots ayant aba comme sous-mot
8. L est le langage des mots qui ne contiennent pas abb

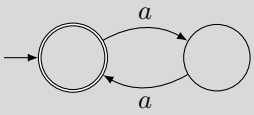
1. L'automate suivant convient :



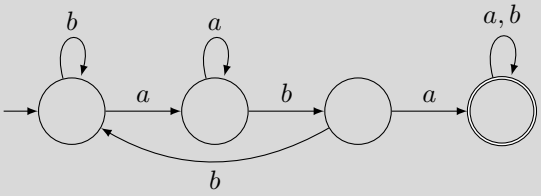
2. On déterminise l'automate suivant :



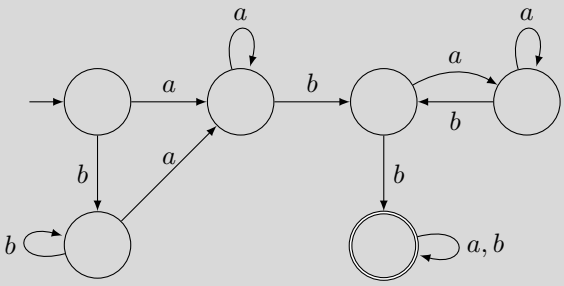
3. L'automate suivant convient :



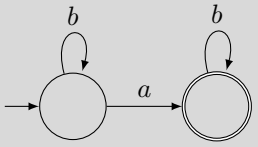
4. On peut déterminiser un automate naïf ou construire l'automate des occurrences pour aba :



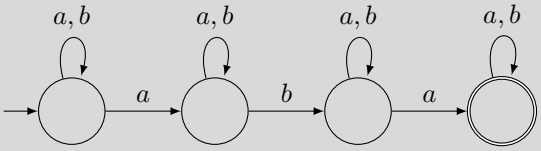
5. On construit un automate pour les mots qui contiennent ab et un pour les mots qui contiennent bb (avec un automate des occurrences par exemple). Puis on en fait le produit :



6. L'automate suivant convient :



7. On déterminise l'automate suivant :



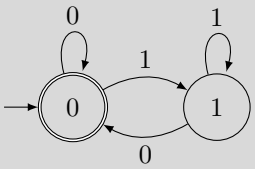
8. On fait un automate pour les mots qui contiennent abb (avec un automate des occurrences ou naïvement). On le complète et on le déterminise. Puis on inverse états finaux et non finaux.

Exercice 31 (**) *Reconnaissances de multiples*

Nous avons étudié en cours la construction d'un automate A_3 reconnaissant les écritures binaires d'entiers divisibles par 3 (lesdites écritures étant potentiellement non purgées de 0 en tête).

1. Proposer un automate déterministe et complet A_2 reconnaissant les mots sur $\{0,1\}$ qui sont écriture binaire (potentiellement non purgée) d'entiers pairs.
2. Calculer l'automate produit de A_2 et A_3. Quel langage reconnaît-il ?
3. Déterminer un automate reconnaissant les écritures binaires d'entiers divisibles par 10.

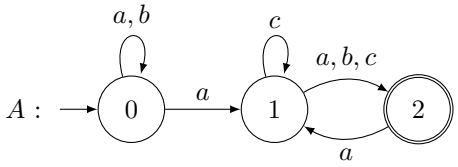
1. L'automate suivant convient :



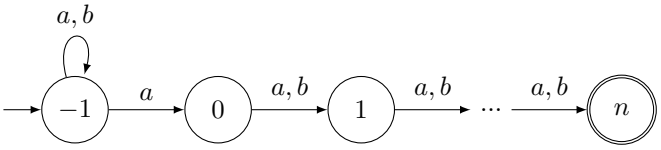
   Les états représentent les restes possibles modulo 2. Lire un 0 consiste à multiplier par 2 l'entier lu et lire un 1 à le multiplier par 2 et lui ajouter 1. Par exemple, si le reste modulo 2 est 1 et qu'on lit un 1 alors le reste modulo 2 devient $2 \times 1 + 1 = 3 \equiv 1$.
2. L'automate obtenu reconnaît les multiples de 6.
3. Il suffit de construire l'automate produit entre A_2, qui reconnaît les multiples de 2, et un automate A_5 reconnaissant les multiples de 5.

Exercice 32 (**) *Une déterminisation coûteuse*

1. Déterminiser l'automate suivant. Quel est le nombre d'états du déterminisé ? Commenter.



L'observation faite ci-dessus est en fait générale : pour tout $n \in \mathbb{N}$ il existe un automate à n états dont la déterminisation via l'automate des parties produit un automate ayant un nombre d'états exponentiel en n. Pire : il existe un automate A à n états tel que le plus petit automate déterministe (en termes de nombre d'états) équivalent à A a un nombre d'états exponentiel en n. Considérons l'automate A_n suivant :



2. Donner une expression rationnelle pour le langage reconnu L_n par A_n.
3. Déterminiser A_1 puis A_2.
4. Combien d'états semble avoir le déterminisé de A_n ?
5. Montrer qu'aucun automate déterministe possédant strictement moins de 2^{n+1} états ne reconnaît L_n. Snif. *Remarque : venez me voir pour un indice au besoin.*

1. On obtient un automate à $2^3 = 8$ états (si on le complète) ce qui est le nombre d'états maximal lors d'une déterminisation accessible via l'automate des parties d'un automate à 3 états.
2. Le langage L_n est dénoté par $(a+b)^\star a(a+b)^n$.
3. RAS. On constate que les automates déterminisés obtenus ont une structure "arborescente".
4. L'automate A_n contient $n+2$ états. Son déterminisé accessible en contient apparemment 2^{n+1} (il y a 2^{n+1} parties contenant -1, or toutes les parties contenant -1 sont accessibles dans le déterminisé).

5. Soit A un automate déterministe dont on note Q les états, d'état initial q_0 reconnaissant L_n. Alors,

$$\varphi : \begin{cases} \{a,b\}^{n+1} & \longrightarrow Q \\ u & \longmapsto \delta^\star(q_0, u) \end{cases}$$

est une fonction bien définie par déterminisme[a] et est injective. Donc, $|Q| \geq |\{a,b\}^{n+1}| = 2^{n+1}$.

L'injectivité de φ se prouve pédestrement. Soit u et v deux mots de $\{a,b\}^{n+1}$ tels que $\varphi(u) = \varphi(v) = q$. Supposons par l'absurde que $u \neq v$. Alors sans perte de généralité il existe une position telle que la lettre à cette position soit a dans u et b dans v : $u = u'au''$ et $v = v'bv''$ avec u' et v' de taille $i \in [\![0, n]\!]$ et donc u'' et v'' de taille $n - i$ (c'est ici qu'on utilise le fait que u et v sont de taille $n+1$, si c'était plus, les mots u'' et v'' pourraient déjà avoir plus de n lettres strictement et donc on ne pourrait pas compléter u et v comme ci-dessous).

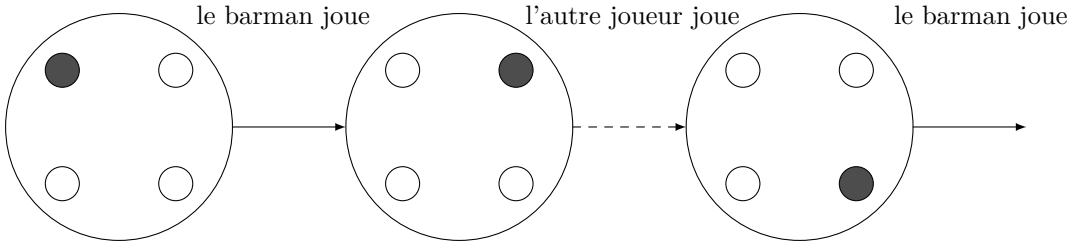
Mézalors $ua^i \in L_n$ alors que $va^i \notin L_n$. Donc en lisant a^i dans l'état q, on devrait aboutir à la fois dans un état final et dans un état qui n'est pas final ce qui est impossible. On en déduit qu'on avait $u = v$.

[a] Il faudrait aussi que A soit complet, mais au pire ça rajoute un état donc on conserve le fait d'avoir besoin d'un nombre exponentiel d'états pour reconnaître L_n de manière déterministe.

Exercice 33 (***) *Le barman aveugle*

Deux joueurs font face à un plateau tournant sur lequel sont placés 4 jetons aux coins d'un carré. Ces jetons ont deux faces de couleurs différentes : l'une est blanche, l'autre est noire. L'un des joueurs (le barman) a les yeux bandés. Son but est de faire en sorte que les 4 jetons soient retournés sur la même couleur : dès que cela arrive, il gagne. Pour ce faire il peut retourner 1, 2 ou 3 jetons à chaque tour. Entre chaque tour, l'autre joueur fait pivoter le plateau d'un quart de tour, d'un demi tour ou de trois-quarts de tour.

On peut ainsi avoir la suite de coups suivants : le barman joue en retournant les deux jetons du haut sans les voir puis l'autre joueur fait pivoter le plateau d'un quart de tour. C'est maintenant au barman de jouer :



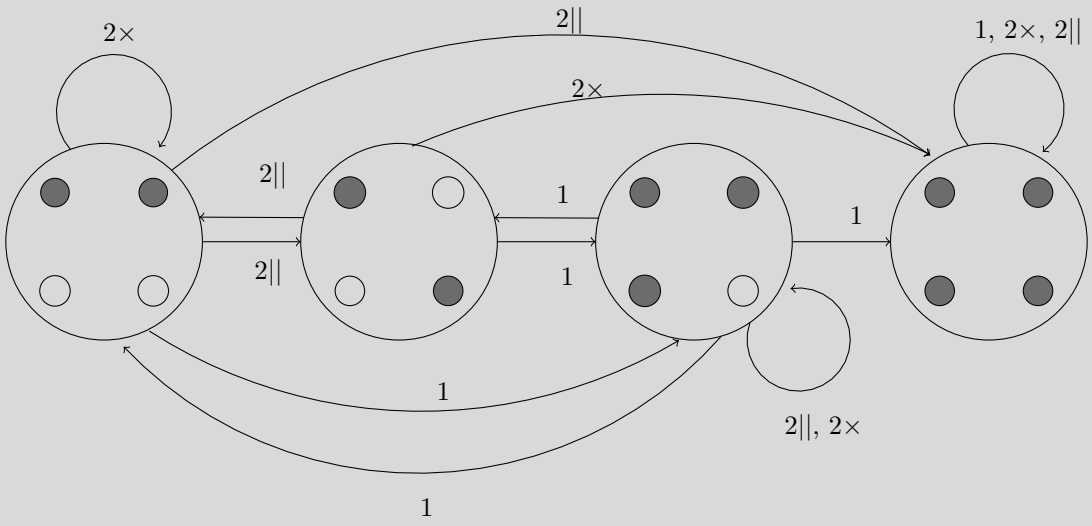
1. Expliquer pourquoi il n'y a que quatre configurations du plateau fondamentalement différentes. En s'aidant de cette observation, modéliser le jeu à l'aide d'un automate.
2. Montrer que le barman a une stratégie gagnante, c'est-à-dire une suite de coups qui le fera gagner quoi que fasse le joueur qui tourne le plateau.

1. Les couleurs des jetons n'ont pas d'importance puisque ce qui compte est d'avoir 4 jetons de la même couleur, peu importe la couleur. Ainsi, retourner un jeton ou trois jetons correspond à la même action. De plus, les rotations du plateau ne modifient pas la configuration. Donc les états du jeu sont :
   - Tous les jetons ont la même couleur (état 4 en partant de la droite ci-dessous).
   - Tous les jetons sauf un sont de la même couleur (état 3).
   - Deux jetons côte à côte sont de la même couleur et les deux autres de l'autre (état 1).
   - Deux jetons en diagonale sont de la même couleur et les deux autres de l'autre (état 2).

   Les actions possibles se résument à :
   - Retourner un des jetons (action 1).
   - Retourner deux jetons côte à côte (action 2||).
   - Retourner deux jetons en diagonale (action (2×)).

   Le jeu se modélise alors via l'automate suivant où les trois premiers états sont initiaux (puisqu'on ne

sait pas dans quelle configuration on commence) et le dernier est final, correspondant à la victoire (d'où le fait qu'on en ait fait un puits : une fois que tous les jetons ont la même couleur, c'est gagné) :



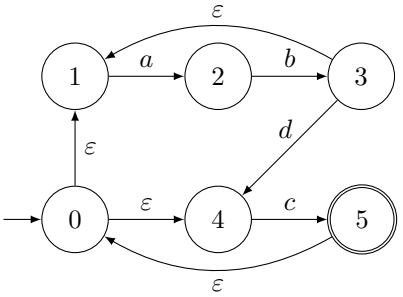
2. On peut ensuite déterminiser mais en ne rendant final que l'état 4 (une suite de coups qui pourrait faire gagner n'est pas suffisante pour gagner, donc contenir un état final n'est pas suffisant pour être un état gagnant). Dans cet automate, l'objectif est ensuite de trouver un mot permettant d'aller de $\{1, 2, 3\}$ à $\{4\}$. On peut aussi observer sur l'automate non déterministe de la question précédente que le mot 2×, 2||, 2×, 1, 2×, 2||, 2× mène à l'état final quel que soit l'état de départ et le chemin non déterministe emprunté : c'est une suite de coups gagnante.

Exercice 34 (**) *Suppressions des ε-transitions*

L'algorithme de déterminisation accessible via l'automate des parties permet de déterminiser les automates à ε-transitions. C'est un moyen de supprimer les transitions instantanées mais il est parfois très coûteux. Dans cet exercice on montre qu'on peut supprimer les ε-transitions sans pour autant déterminiser.

On rappelle que la clôture instantanée d'un ensemble est l'union des clôtures instantanées de ses éléments et que $\delta(E, a)$ où E est un ensemble d'états est l'union des $\delta(e, a)$ pour $e \in E$. Soit $A = (\Sigma, Q, I, F, \delta)$ un automate à ε-transitions. On définit l'automate A' par $A' = (\Sigma, Q, I', F, \eta)$ où I' est la clôture instantanée de I et pour tout $q \in Q$ et tout $a \in \Sigma$, $\eta(q, a)$ est la clôture instantanée de $\delta(q, a)$.

1. Prouver par récurrence sur $|u|$ que pour tout $u \in \Sigma^\star$, $\delta^\star(I, u) = \eta^\star(I', u)$.
2. En déduire que A' est un automate sans ε-transition reconnaissant le même langage que A.
3. A l'aide de la méthode présentée ci-dessus, supprimer les ε-transitions de l'automate suivant.



4. Quelle est la taille de l'automate obtenu par ce procédé en fonction de celle de l'automate initial ?
5. Quel est le coût de la suppression des ε-transitions sur un automate à n états via cette méthode ? Le comparer au coût au pire cas d'une déterminisation et conclure.

1. Si $|u| = 0$ alors $u = \varepsilon$ et alors :
$$\delta^\star(I, u) = \bar{I} = I' = \eta^\star(I', u)$$

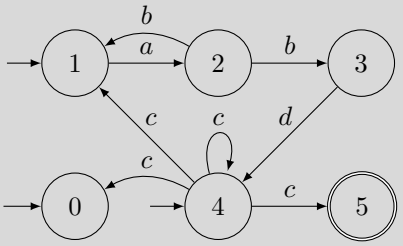
Si $u = va$ avec a une lettre alors :

$$\begin{aligned}\eta^\star(I', u) &= \eta(\eta^\star(I', v), a) \text{ par définition de } \eta^\star \\ &= \eta(\delta^\star(I, v), a) \text{ par hypothèse de récurrence} \\ &= \overline{\delta(\delta^\star(I, v), a)} \text{ par définition de } \eta \\ &= \delta^\star(I, va) \text{ car dans un automate à } \varepsilon\text{-transitions, } \delta^\star(q, wa) = \bigcup_{q' \in \delta^\star(q,w)} \overline{\delta(q', a)}\end{aligned}$$

2. A' est un automate fini puisqu'il a le même nombre d'états que A qui était un automate fini ; sans ε-transition par construction. La question précédente affirme de plus que :

$$m \in L(A) \Leftrightarrow \delta^\star(I, m) \cap f \neq \emptyset \Leftrightarrow \eta^\star(I', m) \cap F \neq \emptyset \Leftrightarrow m \in L(A')$$

3. On obtient l'automate suivant :



4. Le nombre d'états n ne change pas. Au pire, chaque sommet a une clôture instantanée contenant les n états et donc chaque transition "réelle" dans l'automate est dédoublée au plus n fois. On obtient un automate de taille polynomiale en celle de l'automate en entrée. Or, on pourrait obtenir un automate de taille exponentielle en supprimant les ε-transitions par déterminisation.
5. Calculer les clôtures instantanées des n états peut se faire via Floyd-Warshall en $O(n^3)$. Puis, comme mentionné ci-dessus, chacune des t transitions donne lieu au pire à $n-1$ nouvelles transitions pour un coût total majoré par du $O(n^3+nt)$. La complexité en temps de cet algorithme, comme sa complexité en espace, est polynomiale alors que la déterminisation pourrait exiger un temps et un espace exponentiel. Si on veut juste supprimer les ε-transitions, il vaut donc mieux utiliser cette méthode que déterminiser.

Exercice 35 (***) *Algorithme de minimisation de Moore*

Si $A = (\Sigma, Q, \{q_0\}, F, \delta)$ est un automate déterministe complet, on définit pour tout $q \in Q$ le langage $L_q = \{u \in \Sigma^\star \mid \delta^\star(q, u) \in F\}$ puis la relation d'équivalence suivante sur Q :

$$q \sim q' \text{ si et seulement si } L_q = L_{q'}$$

On note $\overline{q}$ la classe d'équivalence de q selon cette relation. On admet dans cet exercice (les intéressé.e.s le montreront en DM) que l'automate $A' = (\Sigma, Q', \{q'_0\}, F', \delta')$ tel que :

- $Q' = \{\overline{q} \mid q \in Q\}$
- $q'_0 = \overline{q_0}$
- $F' = \{\overline{q} \mid q \in F\}$
- pour tout $\overline{q} \in Q'$ et toute lettre $a \in \Sigma$, $\delta'(\overline{q}, a) = \overline{\delta(q, a)}$.

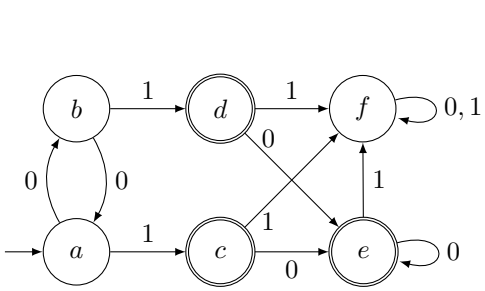
a un nombre d'états minimal parmi tous les automates déterministes et complets reconnaissant $L(A)$ et on l'appelle l'automate minimal reconnaissant $L(A)$. On calcule les classes d'équivalences selon $\sim$ en calculant successivement les classes d'équivalence selon $\sim_k$ telles que

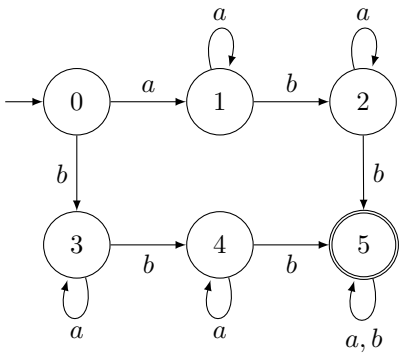
$$q \sim_k q' \text{ si et seulement si } \{u \in L_q \mid |u| \leq k\} = \{u \in L_{q'} \mid |u| \leq k\}$$

1. Déterminer les classes d'équivalences selon $\sim_0$.
2. Montrer que pour tous $q, q' \in Q$ et tout $k \in \mathbb{N}$: $q \sim_{k+1} q' \Leftrightarrow (q \sim_k q' \text{ et } \forall a \in \Sigma,\ \delta(q, a) \sim_k \delta(q', a))$.
3. Montrer qu'à partir d'un certain rang $k \in \mathbb{N}$, $\sim_k$ coïncide avec $\sim$.

L'algorithme sous-jacent aux questions précédentes est l'algorithme de Moore. Mettons-le en application :

4. Dans les deux cas suivants déterminer les classes d'équivalence d'états de l'automate A selon $\sim$ et en déduire un automate minimal reconnaissant le langage $L(A)$:





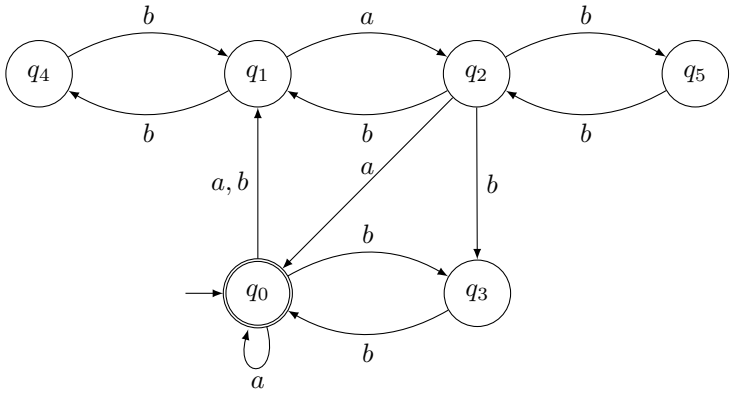
Cf corrigé du DM1 sur la minimisation.

Exercice 36 (***) *Minimisation de Brzozowski*

Si A est un automate fini, le transposé de A, noté $T(A)$ est l'automate dans lequel tous les arcs de A sont inversés, dans lequel les états initiaux deviennent finaux et les finaux initiaux. On note par ailleurs $D(A)$ l'automate déterministe, complet et accessible obtenu à partir de A par déterminisation accessible. Le procédé de minimisation de Brzozowski consiste en l'algorithme suivant :

Entrée : Un automate A reconnaissant L.
Sortie : ?
renvoyer $D(T(D(T(A))))$

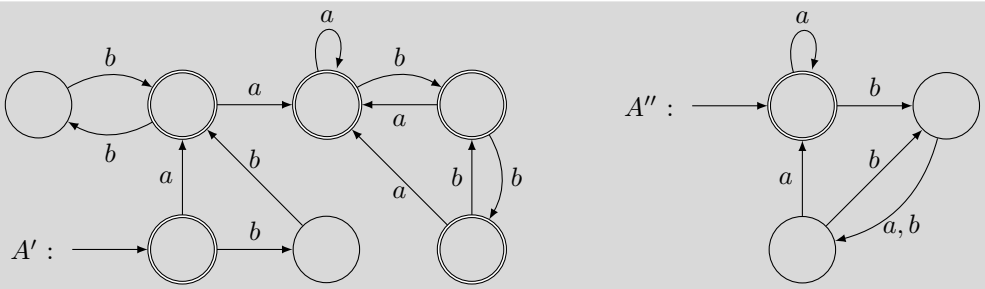
1. Appliquer cet algorithme à l'automate suivant. Quelle propriété inattendue observe-t-on sur l'automate résultat et pourquoi est-ce surprenant ?



Dans la suite de cet exercice, on montre que l'observation faite sur l'exemple précédent ne doit en fait rien au hasard et on identifie ce qu'est la sortie du procédé de Brzozowski.

2. Montrer que l'automate en sortie de l'algorithme de Brzozowki est un automate déterministe et complet qui reconnaît le même langage que l'automate en entrée de l'algorithme.
3. Montrer que, si l'automate en entrée reconnaît un langage L, l'automate en sortie a un nombre d'états minimal parmi tous les automates déterministes et complets qui reconnaissent L. *Cette question est difficile. Il est conseillé de traiter le DM1 pour avoir des pistes de réponse.*
4. Quelle est la complexité a priori de cet algorithme de minimisation d'automate ?

1. En notant $A' = D(T(A))$ et $A'' = D(T(A'))$ l'automate recherché, on obtient :



On constate que A'' a moins d'états que A alors que la déterminisation a tendance à faire augmenter, parfois exponentiellement le nombre d'états.

2. La dernière opération appliquée par l'algorithme de Brzozowski est D donc cet automate est déterministe et complet. De plus, $T(A)$ reconnaît le langage miroir de celui reconnu par A et déterminiser un automate ne change pas le langage qu'il reconnaît. On en déduit que A'' reconnaît le miroir du miroir des mots reconnus par A, c'est-à-dire $L(A)$.

3. Soit A'' un automate reconnaissant L :

   - L'automate $D(T(A''))$ est déterministe donc n'a qu'un seul état initial qu'on note f.
   - On en déduit que $A' = T(D(T(A')))$ a pour seul état final f et donc écrire : $A' = (\Sigma, Q', I', \{f\}, \eta)$.
   - L'automate $A = D(A')$ est le morceau accessible de l'automate des parties associé à A' donc $A = (\Sigma, Q, \{i\}, F, \delta)$ avec $Q \subset P(Q')$ et $i = \{I'\}$.

   Montrons alors que la fonction suivante est injective :

$$\varphi : \begin{cases} Q \longrightarrow \mathcal{R}_L = \{u^{-1}L \,|\, u \in \Sigma^\star\} \\ q \longmapsto u^{-1}L \text{ où } u \text{ est un mot tel que } \delta^\star(I', u) = q \end{cases}$$

   Cela montrera que A (qui est exactement l'automate produit par le procédé de Brzozowski appliqué à A'') a moins d'états que le nombre de résiduels de L. Or cette quantité est égale au nombre d'états d'un automate minimal pour L d'après le DM1. Ceci montrera bien que A est minimal pour L.

   Déjà, comme A est déterministe et accessible, la question 4a du DM1 montre que φ est bien définie. Soit $q, q' \in Q$ tels que $\varphi(q) = \varphi(q')$. Alors il existe $u, v \in \Sigma^\star$ tels que $u^{-1}L = v^{-1}L$, $q = \delta^\star(I', u)$ et $q' = \delta^\star(I', v)$. On cherche à montrer que $\delta^\star(I', u) = \delta^\star(I', v)$ en tant qu'états de A. Pour ce faire, on va montrer que ces états sont égaux en tant qu'ensembles d'éléments de Q'.

   Soit $p \in Q'$ tel que $p \in \delta^\star(I', u)$. Comme $D(T(A''))$ est accessible, en particulier l'état p l'est donc p est coaccessible dans $T(D(T(A'))) = A'$: il existe donc un mot h permettant de passer de l'état p à l'état f dans A'. Mais par définition de p, on peut passer d'un état initial de A' à l'état p en lisant u donc lire uh dans A' mène à un état final. Comme A' reconnaît L, $uh \in L$ donc $h \in u^{-1}L = v^{-1}L$ par hypothèse puis $vh \in L$. Donc lire h dans un des états de $\eta^\star(I', v)$ mène à l'état f dans A'.

   Or, un chemin permettant d'arriver à f en lisant h part nécessairement de l'état p. Sinon, il existerait dans A' un chemin $p' \to^h f$ et un chemin $p \to^h f$ et donc dans $D(T(A))$ il existerait un chemin étiqueté par le transposé de h menant de f à deux états distincts ce qui est impossible dans un automate déterministe. On en déduit que $p \in \eta^\star(I', v)$ donc que $p \in \delta^\star(I', v)$ et $\delta^\star(I', u) \subset \delta^\star(I', v)$.

   L'autre inclusion se fait de la même façon et finalement $q = q'$.

4. Cet algorithme est au moins exponentiel à cause des déterminisations, à comparer avec la déterminisation de Moore qui elle est polynomiale.

Exercice 37 (**) *Construction d'automates*

Cet exercice est une suite de l'exercice *Dessine moi un automate*. Il permet de s'entraîner sur les techniques qui y sont mises en jeu, notamment les algorithmes permettant de faire l'union, l'intersection ou le complémentaire d'automates et de déterminiser. Construire un automate...

1. ... déterministe qui reconnaît $(ba + b)^\star + (bb + a)^\star$.
2. ... qui reconnaît les mots sur $\{0, 1\}$ commençant par 0 et contenant le facteur 01.
3. ... qui reconnaît les mots sur $\{0, 7\}$ ne contenant pas le facteur 007.

4. ... déterministe et complet qui reconnaît les mots sur $\{a, b, c\}$ qui commencent et finissent par a.
5. ... qui reconnaît les mots sur $\{2, 4\}$ n'ayant pas 42 comme sous-mot.

Cet exercice compile les exercices de colle *Déterminisation après construction à la Thompson*, *Construction d'un automate produit*, *No Bond*, *Mots qui commencent et finissent par a* et *Automate* 42.

1. Faire un automate naïf en utilisant l'algorithme de Thompson puis déterminiser
2. Faire le produit d'un automate pour $0(0+1)^\star$ et d'un automate pour $(0+1)^\star 01(0+1)^\star$.
3. Faire un automate pour les mots contenant 007, le compléter et le déterminiser puis le complémentariser.
4. Pas besoin de passer par un automate produit : faire un automate naïf qui reconnaît les mots commençant et finissant par a (attention, vérifier qu'il reconnaît bien le mot a) puis le compléter /déterminiser.
5. Faire un automate qui reconnaît les mots ayant 42 comme sous-mot, le compléter et déterminiser avant de complémentariser.

Exercice 38 (exo cours) *Déterminisation après construction à la Thompson*

1. Construire un automate A acceptant le langage sur $\{a, b\}$ dénoté par $(ba+b)^\star + (bb+a)^\star$. La construction doit être convaincante et on pourra à cette fin utiliser de l'indéterminisme, y compris des ε-transitions.
2. Déterminiser A.

Exercice 39 (exo cours) *Construction d'un automate produit*

1. Proposer un automate A_1 simple qui reconnaît les mots sur $\{0, 1\}$ qui commencent par 0. Construire de même un automate A_2 pour les mots qui contiennent 01.
2. Construire l'automate produit de A_1 et A_2. Que reconnaît-il ?

Exercice 40 (exo cours) *No Bond*

1. Construire un automate non déterministe simple sur $\{0, 7\}$ qui reconnaît les mots contenant 007.
2. En justifiant la démarche, déduire un automate déterministe reconnaissant les mots ne contenant pas 007.

Exercice 41 (exo cours) *Mots qui commencent et finissent par a*

1. Donner un automate non déterministe simple reconnaissant le langage des mots sur l'alphabet $\{a, b, c\}$ qui commencent et qui finissent par la lettre a.
2. Déterminiser et compléter l'automate précédent.

Exercice 42 (exo cours) *Automate* 42

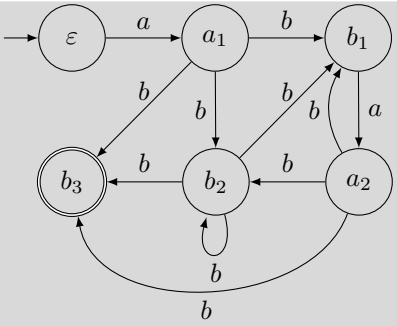
1. Donner un automate non déterministe simple sur l'alphabet $\{2, 4\}$ reconnaissant les mots qui admettent 42 comme sous-mot.
2. En déduire un automate déterministe reconnaissant les mots n'ayant pas 42 comme sous-mot.

Exercice 43 (*) *Algorithme de Berry-Sethi*

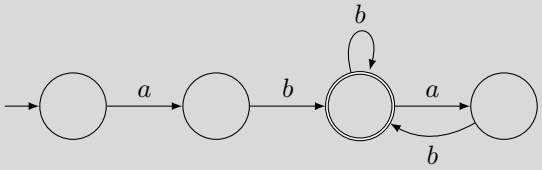
En appliquant l'algorithme de Berry-Sethi, déterminer pour chacune des expressions rationnelles suivantes un automate qui reconnaît le langage qu'elle dénote. Déterminiser les automates obtenus aux questions 1 et 2.

1. $a(ba+b)^\star b$
2. $(a+c)^\star(abb+\varepsilon)$
3. $aab^\star(ab)^\star + ab^\star + a^\star bba$

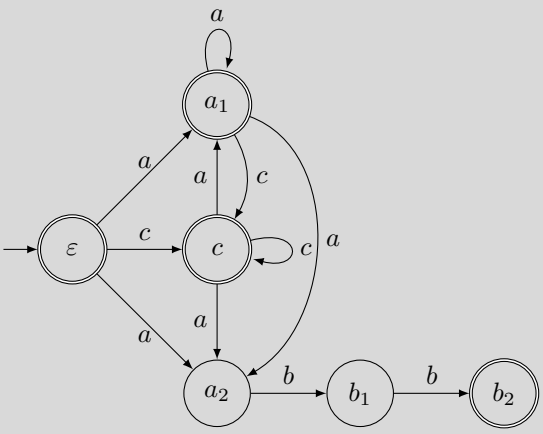
1. Après application de Berry-Sethi on obtient :



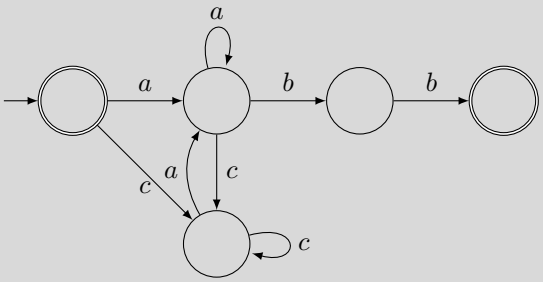
Voici la déterminisation de cet automate :



2. Après application de Berry-Sethi on obtient l'automate suivant. Attention à bien rendre l'état ε final puisque ce mot est dans le langage étudié.



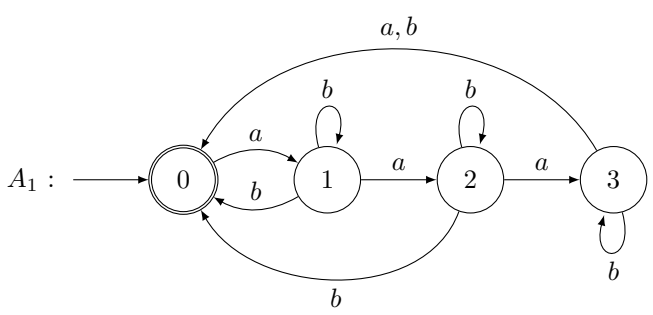
Après déterminisation on obtient :

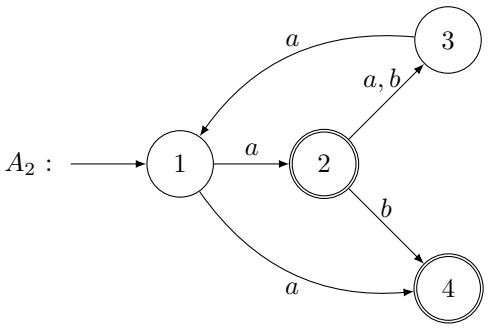


3. Non, je ne vais pas dessiner cet automate à 12 états.

Exercice 44 (*) *Méthode d'élimination des états*

Déterminer le langage reconnu via la méthode d'élimination des états pour chacun des automates suivants. Pour l'automate A_1, on éliminera les états par ordre croissant de numéro, pour A_2, l'ordre est laissé libre.





1. On trouve quelque chose d'aboslument horrible puisque l'odre d'élimination imposé commence par éliminer 0 qui a plein de transitions entrantes.
2. En éliminant dans l'ordre 3,4,1,2, A_2 reconnaît le langage dénoté par $a + a((a + b)a^2)^\star(b + \varepsilon)$.

Exercice 45 (**) *Algorithme de McNaughton et Yamada*

On présente dans cet exercice une variante de l'algorithme de Floyd-Warshall permettant de déterminer une expression rationnelle pour le langage reconnu par un automate.

Cet algorithme prend en entrée un automate A potentiellement non déterministe ou avec ε-transitions, dont on note I les états initiaux et F les finaux. On commence par numéroter ses n états de 0 à $n-1$. On définit pour tous $i, j \in [\![0, n-1]\!]$ et tout $k \in [\![0, n]\!]$ le langage $L_{i,j}^{(k)}$ formé par les étiquettes des chemins allant de l'état i à l'état j et dont les sommets intermédiaires ont tous un numéro strictement inférieur à k. L'objectif est de calculer pour tout $k \in [\![0, n]\!]$ une matrice $R^{(k)}$ contenant des expressions rationnelles de sorte à ce que la case (i, j) de cette matrice contienne une expression rationnelle dénotant $L_{i,j}^{(k)}$.

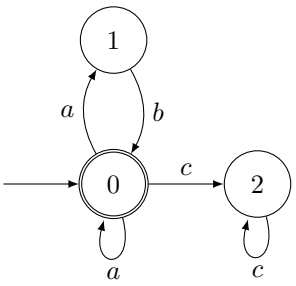
1. Que choisir pour la matrice $R^{(0)}$ pour répondre à cette contrainte ?

Si on a construit la matrice $R^{(k)}$, on en déduit $R^{(k+1)}$ comme suit :

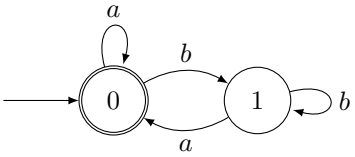
$$\forall i, j \in [\![0, n-1]\!],\ R_{i,j}^{(k+1)} = R_{i,j}^{(k)} + R_{i,k}^{(k)} \left(R_{k,k}^{(k)}\right)^\star R_{k,j}^{(k)}$$

L'expression renvoyée par l'algorithme de McNaughton-Yamada est alors $\sum_{i \in I, j \in F} R_{i,j}^{(n)}$.

2. Expliquer la correction de cet algorithme.
3. Calculer la matrice $R^{(1)}$ pour l'automate suivant sans simplifier les expressions rationnelles obtenues :



4. Simplifier la matrice $R^{(1)}$ obtenue.
5. Sans simplifications, quel est l'ordre de grandeur de la taille de l'expression rationnelle obtenue en fin d'algorithme en fonction du nombre d'états de l'automate en entrée ? Qu'en penser ?
6. Déterminer le langage reconnu par cet automate en appliquant l'algorithme de McNaughton-Yamada :



1. Le langage $L_{i,j}^{(0)}$ est le langage des mots permettant de passer de l'état i à l'état j sans passer par aucun sommet intermédiaire. Il suffit donc de prendre :

$$L_{i,j}^{0} = \begin{cases} \{a \in \Sigma \,|\, i \to^a j\} \text{ si } i \neq j \\ \{a \in \Sigma \,|\, i \to^a j\} \cup \{\varepsilon\} \text{ sinon} \end{cases}$$

2. Il y a deux types de chemins allant de i à j ne passant que par des sommets de numéro strictement inférieur à $k+1$:

   - Ceux qui ne passent que par des sommets de numéro strictement inférieur à k : par hypothèse de récurrence, les étiquettes de ces derniers sont les mots du langage $R_{i,j}^{(k)}$.
   - Ceux qui passent $p > 1$ fois par k. L'étiquette d'un tel chemin est un mot $m = uv_1...v_pw$ tel que :

   $$i \to^u k \to^{v_1} k \to^{v_2} k \to ... \to^{v_p} k \to^w j$$

   où tous les chemins empruntés ne font intervenir que des sommets de numéro strictement inférieur à $k+1$. Par hypothèse de récurrence, on a donc $u \in R_{i,k}^{(k)}$, $v \in R_{k,j}^{(k)}$ et tous les $v_i \in R_{k,k}^{(k)}$ donc $m \in R_{i,k}^{(k)} \left(R_{k,k}^{(k)}\right)^\star R_{k,j}^{(k)}$.

   Cela montre que $R_{i,j}^{(k+1)}$ est inclus dans $R_{i,j}^{(k)} + R_{i,k}^{(k)} \left(R_{k,k}^{(k)}\right)^\star R_{k,j}^{(k)}$ et l'inclusion réciproque se traite similairement. On en déduit que $R_{i,j}^{(n)}$ est l'ensemble des mots permettant de passer de i à j en ne passant que par des sommets de numéro strictement inférieur à n c'est-à-dire sans contrainte puisque les sommets sont numérotés jusque $n-1$. Donc $\sum_{i \in I, j \in F} R_{i,j}^{(n)}$ est l'ensemble des mots permettant de passer d'un état initial à un état final dans A ce qui est la définition du langage reconnu par A.

3. D'après la question la matrice $R^{(0)}$ est :

$$R^{(0)} = \begin{pmatrix} \varepsilon + a & a & c \\ b & \varepsilon & \emptyset \\ \emptyset & \emptyset & \varepsilon + c \end{pmatrix}$$

   Sans simplification on devrait obtenir :

$$R^{(1)} = \begin{pmatrix} \varepsilon + a + (\varepsilon + a)(\varepsilon + a)^\star(\varepsilon + a) & a + (\varepsilon + a)(\varepsilon + a)^\star a & c + (\varepsilon + a)(\varepsilon + a)^\star c \\ b + b(\varepsilon + a)^\star(\varepsilon + a) & \varepsilon + b(\varepsilon + a)^\star a & \emptyset + b(\varepsilon + a)^\star c \\ \emptyset + \emptyset(\varepsilon + a)^\star(\varepsilon + a) & \emptyset + \emptyset(\varepsilon + a)^\star a & \varepsilon + c + \emptyset(\varepsilon + a)^\star c \end{pmatrix}$$

4. On constate que la taille des expressions dans les matrices $R^{(k)}$ va rapidement croître. On peut simplifier la matrice $R^{(1)}$ en la matrice suivante mais même ainsi, le calcule de $R^{(2)}$ serait ardu :

$$R^{(1)} = \begin{pmatrix} a^\star & a^+ & a^\star c \\ ba^\star & \varepsilon + ba^+ & ba^\star c \\ \emptyset & \emptyset & \varepsilon + c \end{pmatrix}$$

5. D'après la question 2, le calcul de $R_{i,j}^{(k+1)}$ combine quatre coefficients de $R^{(k)}$ (de taille au moins un), donc on s'attend à avoir des expressions dans $R^{(n)}$ de taille de l'ordre de 4^n sans simplifications. On fait ensuite la somme d'au plus n^2 de ces expressions (si tous les états sont initiaux et finaux).

6. Tous calculs faits et avec simplifications on devrait obtenir :

$$R^{(0)} = \begin{pmatrix} \varepsilon + a & b \\ a & \varepsilon + b \end{pmatrix} \text{ et } R^{(1)} = \begin{pmatrix} a^\star & a^\star b \\ a^+ & \varepsilon + a^\star b \end{pmatrix}$$

   Comme 0 est le seul état initial et le seul état final, le seule coefficient de $R^{(2)}$ qui nous intéresse est celui en haut à gauche et ce dernier vaut $a^\star + a^\star b(\varepsilon + a^\star b)^\star a^+$.

Exercice 46 (*) *Et les Shadoks pompaient*

En utilisant le lemme de l'étoile, montrer que les langages suivants ne sont pas rationnels :

1. $L_1 = \{u \in \{a,b\}^\star \mid |u|_a = |u|_b\}$.
2. L_2 est l'ensemble des palindromes sur l'alphabet $\{0,1\}$.
3. $L_3 = \{a^{(n^2)} \mid n \in \mathbb{N}\}$.
4. L_4 est le langage de Dyck sur l'alphabet $\{(,)\}$.

1. Par l'absurde, si L_1 était rationnel, il serait reconnu par un automate à N états. Selon le lemme de l'étoile, le mot $a^N b^N$ qui est de taille $2N \geq N$ admettrait alors une factorisation de la forme $a^{n_1}a^{n_2}a^{N-n_1-n_2}b^N$ avec $n_2 \neq 0$ et telle que pour tout $k \in \mathbb{N}$ on ait $a^{n_1}a^{kn_2}a^{N-n_1-n_2}b^N \in L_1$. En particulier avec $k = 0$, on aurait $a^{N-n_2}b^N \in L_1$ mais ce mot contient strictement moins de a que de b puisque n_2 est non nul. On en déduit que L_1 n'est pas rationnel.
2. On applique le lemme de l'étoile avec le mot $1^N 001^N$ avec $k = 0$.
3. On applique le lemme de l'étoile avec a^{N^2} et $k = 0$. Cela fonctionne car si $0 < n_2 \leq N$, alors $N^2 > N^2 - n_2 \geq N^2 - N > (N-1)^2$ (on peut supposer $N > 1$ car un automate à un état ne peut clairement pas reconnaître L_3). Ainsi $N^2 - n_2$ ne peut pas être un carré puisqu'il est strictement compris entre deux carrés consécutifs.
4. On applique le lemme de l'étoile avec le mot $a^N b^N$ et $k = 0$.

Exercice 47 (**) *Rationnel ou pas rationnel ?*

On considère deux langages L et L' sur un même alphabet Σ. Pour chacune des propositions suivantes, la démontrer si elle est vraie et fournir un contre-exemple si elle est fausse.

1. Si L est rationnel et $L \cap L'$ ne l'est pas alors L' n'est pas rationnel.
2. Si L est rationnel et L' ne l'est pas alors $L \cap L'$ n'est pas rationnel.
3. Si $L^\star$ est rationnel alors L est rationnel.
4. Si L et L' sont rationnels alors $L \setminus L'$ aussi.
5. Si LL' est rationnel alors L est rationnel ou L' est rationnel.

1. Vrai. Si L' était rationnel, par stabilité de la rationnalité par intersection, $L \cap L'$ le serait.
2. Faux. $L = \emptyset$ est rationnel, $L' = \{a^n b^n \mid n \in \mathbb{N}\}$ ne l'est pas mais $L \cap L' = \emptyset$ l'est.
3. Faux. $L = \{a^{n^2} \mid n \in \mathbb{N}\}$ n'est pas rationnel et contient a donc $L^\star = a^\star$ qui est rationnel.
4. Vrai. $L \setminus L' = L \cap (L')^c$ et la stabilité des rationnels par intersection et complémentaire conclut.
5. Faux. Prenons L un langage non rationnel qui contient ε, par exemple $\{a^n b^n \mid n \in \mathbb{N}\}$ et $L' = L^c \cup \varepsilon$ qui n'est pas rationnel non plus sinon L le serait. Alors $LL' = \Sigma^\star$ (car $\varepsilon = \varepsilon\varepsilon \in LL'$ et si u est non vide : soit $u \in L$ et alors $u = u\varepsilon \in LL'$, soit $u \in L^c$ et alors $u = \varepsilon u \in LL'$) qui est bien rationnel.

Exercice 48 (***) *La pompe est cassée*

On considère sur $\Sigma = \{a,b\}$ le langage L suivant :

$$L = \{v^t v w \mid v, w \in \Sigma^\star \text{ tels que } |v| \geq 1, |w| \geq 1\}$$

1. Montrer que le langage L vérifie les conclusions du lemme de l'étoile.
2. Montrer que L n'est pas un langage rationnel.
3. Que vient-on de montrer ?

1. Montrons que pour tout $u \in L$ de longueur supérieure ou égale à $N = 4$, u se factorise de manière convenable. Soit donc u un mot de longueur au moins 4. Alors il existe $v, w \in (a+b)^+$ tels que $u = v^t v w$. Comme $v \neq \varepsilon$, $v = \alpha v'$ avec α une lettre et pour la même raison, $w = \beta w'$ avec β une lettre. On en déduit que $u = \alpha v'^t v' \alpha \beta w'$. Deux possibilités :
   - Si $v' \neq \varepsilon$, la factorisation $U = \varepsilon$, $V = \alpha$ et $W = v'^t v' \alpha w$ convient car $|UV| \leq 4$, $V \neq \varepsilon$ et pour

tout $k \in \mathbb{N}$, $\varepsilon\alpha^k W \in L$. En effet :

$$\begin{cases} \text{pour } k = 0, v'^t v'(\alpha w) \in L \text{ puisqu'on est sous l'hypothèse } v' \neq \varepsilon \\ \text{pour } k = 1 \text{ c'est bon puisque } u \in L \\ \text{pour } k \geq 2, \varepsilon\alpha^k W = \alpha\alpha \underbrace{(\alpha^{k-2} W)}_{\neq \varepsilon} \in L \end{cases}$$

- Si $v' = \varepsilon$ alors $u = \alpha\alpha\beta w'$. Mais dans ce cas $w' \neq \varepsilon$ puisque u est de taille au moins 4. Alors la factorisation $U = \alpha^2$, $V = \beta$, $W = w'$ convient car $|\alpha^2\beta| \leq 4$, $\beta \neq \varepsilon$ et pour tout $k \in \mathbb{N}$, $\alpha^2\beta^k w' \in L$: le seul cas délicat est pour $k = 0$ mais $\alpha^2 w'$ est bien dans L puisque $w' \neq \varepsilon$.

2. Supposons par l'absurde que L est reconnu par un automate $A = (\Sigma, Q, q_0, F, \delta)$ déterministe et complet. Ces deux qualités assurent que la fonction suivante est bien définie :

$$\varphi : \begin{cases} \mathbb{N}^\star \to Q \\ n \mapsto \delta^\star(q_0, (ab)^n) \end{cases}$$

Comme A est un automate fini, cette fonction est non injective : il existe donc $n < m$ tel que $q = \delta^\star(q_0, (ab)^n) = \delta^\star(q_0, (ab)^m)$. En lisant $(ba)^n b$ à partir de q, on arrive dans un état final puisque $(ab)^n(ba)^n b \in L$. On en déduit que $u = (ab)^m(ba)^n b$ est également reconnu donc dans L donc de la forme $v^t vw$ avec v et w non vides. En particulier, u contient un carré de deux lettres formé par la concaténation de la dernière lettre de v et la première de son transposé à la jonction entre v et ${}^t v$. Or le seul carré de u est b^2 en position $(2m, 2m+1)$. Donc si u est de la forme $v^t vw$, on a nécessairement $v = (ab)^m$. Mézalors $|u| = 2m + 2n + 1 = 2 \times |v| + |w| = 4m + |w| \geq 4m + 1$ car $|w| \geq 1$. On en déduit que $n \geq m$ ce qui est une contradiction.

3. On vient d'exhiber un langage non rationnel qui vérifie les conclusions du lemme de l'étoile. Cela montre que le lemme de l'étoile est une implication dont la réciproque est fausse.

Exercice 49 (exo cours) *Non rationnalité de* $\{a^n b^n \mid n \in \mathbb{N}\}$

1. Montrer que le langage $\{a^n b^n \mid n \in \mathbb{N}\}$ n'est pas reconnaissable par un automate fini.
2. Si L est inclus dans un langage rationnel, L est-il rationnel ? Justifier.

1. Par l'absurde, L est reconnaissable par un automate $A = (\{a, b\}, Q, q_0, I, \delta)$ qu'on peut loisiblement supposer déterministe est complet. Alors la fonction

$$\varphi : \begin{cases} \mathbb{N} \to Q \\ n \mapsto \delta^\star(q_0, a^n) \end{cases}$$

est correctement définie ($\delta^\star(q_0, a^n)$ existe car a^n n'est pas un blocage car A est complet et est un état et pas un ensemble d'états par déterminisme). Comme Q est fini, elle est nécessairement non injective et il existe donc $n \neq m$ tel que $\varphi(n) = \varphi(m)$. Mézalors :

$$\delta^\star(q_0, a^m b^n) = \delta^\star(\delta^\star(q_0, a^m), b^n) = \delta^\star(\delta^\star(q_0, a^n), b^n) = \delta^\star(q_0, a^n b^n) \in F$$

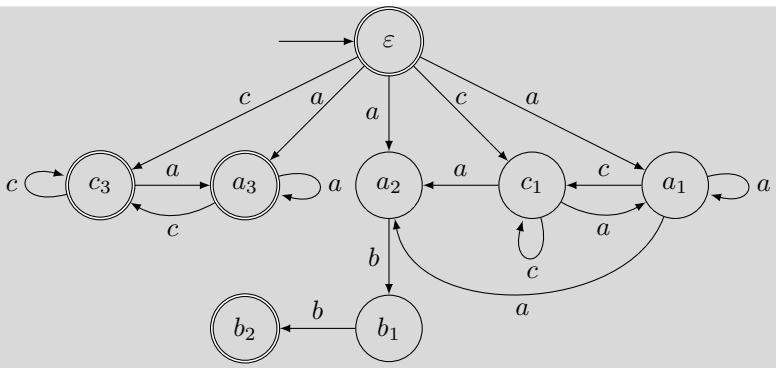
ce qui montre que A reconnaît $a^m b^n$ avec $n \neq m$ et est une contradiction.

2. Non puisque le langage non rationnel de la question 1 est inclus dans le langage rationnel $a^\star b^\star$.

Exercice 50 (exo cours) *Application de l'algorithme de Berry-Sethi*

1. En appliquant l'algorithme de Berry-Sethi, construire un automate reconnaissant le langage dénoté par $(a + c)^\star abb + (a + c)^\star$.
2. Déterminiser l'automate obtenu (on ne le complètera pas).

1. On devrait trouver :

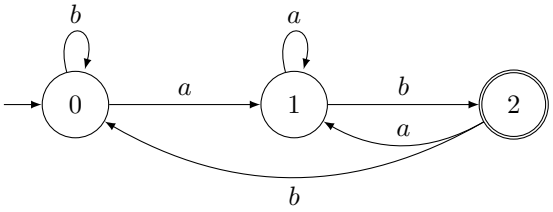


Attention à rendre final l'état initial.

2. RAS, on s'assurera juste que l'algo est connu.

Exercice 51 (exo cours) *Application de la méthode d'élimination des états*

1. En appliquant la méthode d'élimination des états et en supprimant les états par ordre croissant de numéro, donner une expression rationnelle pour le langage reconnu par l'automate suivant :



2. Donner une description simple des mots reconnus par cet automate et en déduire une autre expression rationnelle pour le langage qu'il reconnaît.

1. On devrait trouver $b^\star a^+ b((a + b^+ a)a^\star b)^\star$.
2. Cet automate est l'automate des occurrences pour ab : il reconnaît les mots dont ab est suffixe.

Exercice 52 (exo cours) *Lemme de l'étoile*

Énoncer puis démontrer le lemme de l'étoile.

Énoncé : Si L est un langage rationnel, il existe $N \in \mathbb{N}$ tel que tout mot $m \in L$ de longueur supérieure ou égale à N se factorise sous la forme $m = uvw$ avec :

$$\begin{cases} |uv| \leq N \\ v \neq \varepsilon \\ \forall k \in \mathbb{N}, uv^k w \in L \end{cases}$$

Preuve : Comme L est rationnel, par Kleene il est reconnu par un automate A à N états, d'état initial q_0, qu'on peut supposer déterministe et complet. Si $m = m_1...m_k$ est de taille supérieure à N, la fonction :

$$\varphi : \begin{cases} [\![0, N]\!] \to Q \\ i \mapsto \delta^\star(q_0, m_1...m_i) \end{cases}$$

est correctement définie et non injective : il existe $i < j$ tel que $\varphi(i) = \varphi(j)$. On montre alors que les mots $u = m_1...m_i$, $v = m_{i+1}...m_j$ et $w = m_{j+1}...m_k$ vérifient les propriétés souhaitées.

Exercice 53 (**) *Algorithmique et automates*

Pour chacun des problèmes suivants, donner les grandes lignes d'un algorithme permettant de le résoudre. On essaiera de proposer des solutions raisonnablement efficaces et on évaluera leurs complexités.

1. Déterminer si un mot appartient au langage dénoté par une expression rationnelle donnée.

2. Déterminer si le langage reconnu par un automate (avec ε-transitions ou non) est fini.
3. Déterminer si le langage dénoté par une expression rationnelle est vide.
4. Déterminer si une expression régulière sur l'alphabet Σ dénote $\Sigma^\star$.
5. Déterminer si deux expressions régulières dénotent le même langage.
6. Déterminer si un langage rationnel contient tous les préfixes des mots qu'il contient.

1. On construit l'automate reconnaissant $L(e)$ puis on lit le mot u. Complexité en $O(|e|^3)$ via Berry-Sethi pour la construction puis en $O(|u||Q|^2) = O(|u||e|^2)$ pour la lecture (car non déterminisme).
2. On élimine les ε-transitions éventuelles (en calculant polynomialement les ε-clôtures de chaque état) puis on fait un parcours pour détecter un cycle en $O(|\text{automate}|)$.
3. Se ramène à un problème d'accessibilité, algorithme linéaire en |automate|.
4. On fait un automate reconnaissant $L(e)$, un automate reconnaissant $L(e)^c$ et on teste la vacuité de ce dernier via 3. Complexité potentiellement exponentielle à cause de la déterminisation lors du passage au complémentaire.
5. On fait un automate pour $L(e)\Delta L(e')$ et on teste s'il reconnaît $\emptyset$. Complexité potentiellement exponentielle car le calcul de Δ demande de complémentariser donc de déterminiser.
6. On construit un automate pour L, on l'émonde et on renvoie "oui" si tous les états de cet automate sont finaux. Complexité linéaire en |automate|.

Exercice 54 (**) *Manipulation automatique d'écritures binaires*

Dans cet exercice, $\Sigma = \{0, 1\}$.

1. Construire un automate simple reconnaissant le langage L des mots commençant par 1 sur Σ.

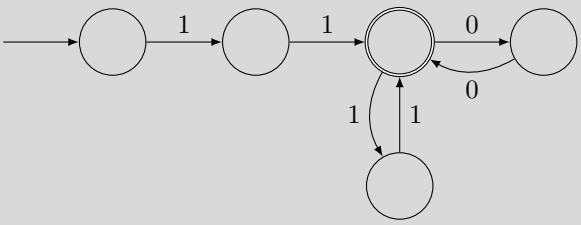
Chaque mot u de L peut être interprété comme l'écriture binaire d'un entier $n > 0$. On note succ(u) le mot de L correspondant à l'écriture binaire de $n + 1$.

2. Dessiner en justifiant un automate déterministe reconnaissant $L' = \{u \in L, |u| = |\text{succ}(u)|\}$.
3. Dessiner en justifiant un automate déterministe reconnaissant $E = \{h(u) \,|\, u \in L\}$ où h est une fonction qui modifie le mot u comme suit : chaque 0 de u est transformé en 00 par h et chaque 1 par 11. Par exemple $h(1001) = 11000011$.

Pour tout mot $u \in L'$, on définit le mot $s(u)$ de la façon suivante : si $u = u_1u_2...u_n$ et succ$(u) = v_1v_2...v_n$ alors $s(u) = u_1v_1u_2v_2...u_nv_n$. On note $S = \{s(u) \,|\, u \in L'\}$.

4. Déterminer deux langages L_1 et L_2 tels que $S = EL_1L_2$.
5. En déduire un automate déterministe reconnaissant S.

1. RAS.
2. Les mots de L' sont ceux commençant par 1 et contenant au moins un 0.
3. Le langage E est dénoté par $11(00 + 11)^\star$. L'automate suivant convient :



4. $L_1 = 01$ et $L_2 = (10)^\star$ conviennent. Le principe est que succ(u) voit tous les 1 en fin de u être transformés en 0 à cause de la propagation de la retenue puis cette dernière transforme un 0 (qui existe) en 1. Ce qui est à gauche de la retenue n'est pas modifié.
5. On combine des automates reconnaissant E, L_1 et L_2 en déterminisant après coup si besoin.

Exercice 55 (**) *Automates et préfixes*

Soit L un langage reconnaissable sur un alphabet Σ. Les langages suivants sont-ils reconnaissables ? Justifier dans chacun des cas.

1. $L_1 = \{w \in L \mid$ aucun préfixe de w n'est dans $L\}$
2. $L_2 = \{w \in \Sigma^\star \mid$ aucun préfixe strict de w n'est dans $L\}$
3. $L_3 = \{w \in L \mid w$ n'est préfixe strict d'aucun mot de $L\}$
4. $L_4 = \{w \in \Sigma^\star \mid w$ est préfixe d'un mot de $L\}$

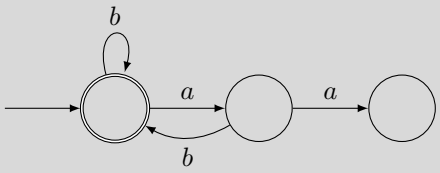
TODO

Exercice 56 (*) *Construction d'automates*

L'alphabet est $\Sigma = \{a, b, c\}$. Pour chacun des langages L suivants, construire si c'est possible un automate qui reconnaît L en expliquant la démarche.

1. L est l'ensemble des mots tels qu'un a est toujours suivi d'un b.
2. L est l'ensemble des palindromes sur Σ.
3. L est l'ensemble des mots qui contiennent un nombre impair de c et ne contiennent pas ab.
4. L est l'ensemble des mots qui commencent par ab ou bc et ne terminent pas par abc.

1. L'automate suivant convient :



2. Le lemme de l'étoile montre que ce langage n'est pas rationnel (avec $a^N b a^N$ par exemple).
3. On fait un automate pour les mots qui contiennent ab qu'on complémentarise en A_1 ; un automate A_2 pour les mots ayant un nombre impair de c puis on fait le produit de A_1 et A_2.
4. Raisonnement similaire à 3 : une union, un complémentaire, un produit.

Exercice 57 (*) *Manipulation formelle d'automates*

On considère un automate déterministe $A = (\Sigma, Q, \{q_0\}, F, \delta)$ tel qu'il existe une lettre $a \in \Sigma$ vérifiant que pour tout état q, $\delta(q, a) = q$.

1. Montrer que pour tout $n \in \mathbb{N}$ et tout $q \in Q$, $\delta^\star(q, a^n) = q$.
2. Prouver qu'on est dans un et un seul des deux cas suivants : $\{a\}^\star \subset L(A)$ ou $\{a\}^\star \cap L(A) = \emptyset$.

On considère à présent un automate $A = (\Sigma, Q, \{q_0\}, \{q_f\}, \delta)$ tel que pour toute lettre $a \in \Sigma$, $\delta(q_0, a) = \delta(q_f, a)$.

3. Montrer que pour tout mot $m \neq \varepsilon$, $\delta^\star(q_0, m) = \delta^\star(q_f, m)$.
4. Montrer que, si $m \neq \varepsilon$ alors on a l'équivalence suivante : m est reconnu par A si et seulement si pour tout $k \in \mathbb{N}^\star$, m^k est reconnu par A.

1. Par récurrence sur n.
2. S'il existe q à la fois final et initial, $\{a\}^\star \subset L(A)$. Sinon, supposons par l'absurde que $\{a\}^\star \cap L(A) \neq \emptyset$. Alors il existe a^n reconnu. Mais le déterminisme de A impose alors que l'unique chemin permettant de reonnaître a^n est composé d'un seul sommet qui est donc à la fois initial et final.
3. Par récurrence sur la longueur du mot m.
4. Réciproque évidente. Sens direct par récurrence ; l'initialisation vient de ce que m est reconnu.

Exercice 58 (**) *Loup, chèvre et choux*

Un berger B doit faire traverser une rivière à un loup L, une chèvre C et une cagette de choux X à l'aide d'une barque. Cette dernière est petite : elle ne peut contenir que deux entités du problème à la fois, et l'une d'elles est nécessairement le berger, seul apte à manœuvrer l'esquif. Sans surveillance du berger, la chèvre mange les choux et le loup mange la chèvre.

1. Modéliser le problème à l'aide d'un automate fini.
2. À l'aide de cet automate, comment trouver une solution au problème ? Il y en a-t-il ?
3. Une solution est dite minimale si le nombre de traversées l'est. Combien y a-t-il de solutions minimales au problème ?
4. Comment déterminer algorithmiquement la taille de cette solution minimale ? Comment déterminer toutes les solutions minimales ? Discuter de la complexité de vos propositions.

1. On étiquette les états pour indiquer qui est sur la rive de départ et qui sur la rive d'arrivée. Par exemple, BC-LX signifie que berger et chèvre sont sur la rive de départ et loup et choux à l'arrivée. Les transitions sont b = berger traverse seul, l = berger traverse avec loup, c = berger traverse avec chèvre et x = berger traverse avec choux. Puis on traduit les contraintes.
2. Savoir s'il y a une solution revient à savoir si le langage reconnu par l'automate est vide.
3. Il y a deux solutions minimales.
4. On calcule les plus courts chemins depuis l'état initial de notre automate (déterministe) avec un parcours en largeur (linéairement) et le min des pcc vers un état final donne la réponse k. Une fois que k est connu, on peut juste tester tous les mots de Σ^k pour savoir lesquels sont reconnus en $O(|\Sigma|^{k+1})$. Peut-on faire mieux ?

Exercice 59 (***) *Reconnaissance de racines*

Si L est un langage, la racine carrée de L est le langage défini par

$$\sqrt{L} = \{u \in \Sigma^\star \mid u^2 \in L\}$$

Si $A = (\Sigma, Q, q_0, F, \delta)$ est un automate déterministe et complet, on définit pour tout $q \in Q$ l'automate A_q de la façon suivante :

- Les états de A_q sont les éléments de Q^2.
- L'état initial de A_q est (q_0, q).
- Les états finaux de A_q sont les états de la forme (q, q_f) avec $q_f \in F$.
- Tous les A_q ont la même fonction de transition δ' telle que $\delta'((q_i, q_j), a) = (\delta(q_i, a), \delta(q_j, a))$.

1. Donner une caractérisation simple du langage reconnu par A_q.
2. La racine carrée d'un langage rationnel est-il un langage rationnel ?

1. $u \in L(A_q)$ si et seulement si $\delta^\star(q_0, u) = q$ et $u \in \sqrt{L}$ car :

$$\begin{aligned} u \in L(A_q) &\Leftrightarrow \exists q_f \in F, \delta'^\star((q_0, q), u) = (q, q_f) \\ &\Leftrightarrow \exists q_f \in F, \delta^\star(q_0, u) = q \text{ et } \delta^\star(q, u) = q_f \\ &\Leftrightarrow \exists q_f \in F, \delta^\star(q_0, u) = q \text{ et } \delta^\star(q_0, uu) = q_f \text{ par déterminisme} \end{aligned}$$

2. Comme A est complet, $\sqrt{L} = \bigcup_{q \in Q} \{u \in \Sigma^\star \mid u \in \sqrt{L} \text{ et } \delta^\star(q_0, u) = q\} = \bigcup_{q \in Q} L(A_q)$.

   Donc $\sqrt{L}$ est rationnel en tant qu'union finie de langages rationnels car reconnaissables.

Remarque : on peut étendre l'exo via l'étude de $L/2 = \{u \in \Sigma^\star \mid \exists v \in \Sigma^\star, uv \in L et |u| = |v|\}$

Exercice 60 (**) *Rationnel ?*

1. Les langages suivants sont-ils rationnels sur l'alphabet $\{a, b\}$? Justifier.

a) $L_1 = \{a^n b^m \mid n < m\}$

b) $L_2 = \{a^n b^m \mid n + m \leq 1024\}$

c) $L_3 = \{a^n b^m \mid n \neq m\}$

d) $L_4 = \{a^n b^m \mid n \equiv m \bmod 2\}$

e) $L_5 = \{a^p \mid p \text{ est premier}\}$

2. On note L l'ensemble des mots sur $\{0,1\}$ correspondant à l'écriture binaire d'un nombre premier. En supposant qu'il existe un nombre infini de nombres premiers de la forme $2^n - 1$ (ce sont les nombres de Mersenne), montrer que L n'est pas rationnel.

1. a) Non rationnel via le lemme de l'étoile avec $a^N b^{N+1}$.

   b) Rationnel car fini.

   c) Non rationnel : s'il l'était, son complémentaire, intersecté avec $a^\star b^\star$, c'est-à-dire $\{a^n b^n \mid n \in \mathbb{N}\}$ le serait, ce qui n'est pas.

   d) Rationnel car égal à $\{a^n b^m \mid n \text{ et } m \text{ pairs}\} \cup \{a^n b^m \mid n \text{ et } m \text{ impairs}\}$ pour lequel une expression est $(a^2)^\star(b^2)^\star + a(a^2)^\star b(b^2)^\star$. On peut aussi exhiber un automate à 4 états.

   e) Non rationnel, sinon il serait reconnu par un automate à N états. Comme $\mathbb{P}$ est infini, il existe $p \in \mathbb{P}$ tel que $|a^p| \geq N$ et se mot se factoriserait en $a^{n_1} a^{n_2} a^{p-n_1-n_2}$ avec $n_2 \neq 0$ avec pour tout $k \in \mathbb{N}$, $a^{p+n_2(k-1)} \in L_5$. Avec $k = p+1$ on a une contradiction car $a^{p(1+n_2)}$ serait alors reconnu et cette puissance n'est pas première car $n_2 + 1 \geq 2$.

2. Si L est reconnu par un automate à N états, l'infinité des nombres de Mersenne nous dit qu'il existe $n \geq N$ tel que $2^n - 1 \in \mathbb{P}$ dont l'écriture binaire 1^n est suffisamment longue pour être factorisée par lemme de l'étoile en $1^{n_1} 1^{n_2} 1^{n-n_1-n_2}$ telle que pour tout $k \in \mathbb{N}$, $1^{n+n_2(k-1)} \in L$.

   Avec $k = n+1$, on aurait $1^{n(n_2+1)} \in L$ donc $2^{n(n_2+1)} - 1 \in \mathbb{P}$. Mais $\sum_{i=0}^{n-1} (2^{n_2+1})^k = \frac{2^{n(n_2+1)}-1}{2^{n_2+1}-1}$, donc $2^{n(n_2+1)} - 1 = (2^{n_2+1} - 1) \times$ (un entier) donc $(2^{n_2} + 1 - 1) \geq 3$ divise un nombre premier.

Exercice 61 (**) *Reconnaissances diverses*

1. On suppose que L est un langage rationnel. Les langages suivants le restent-ils ?

   a) L'ensemble des préfixes des mots de L.

   b) L'ensemble des suffixes des mots de L.

   c) L'ensemble des facteurs des mots de L.

   d) L'ensemble des miroirs des mots de L c'est-à-dire $\{m^t \mid m \in L\}$ avec $(m_1 m_2 ... m_k)^t = m_k ... m_2 m_1$?

2. Si Σ est un alphabet, $m \in \Sigma^\star$ et $a \in \Sigma$, on note $s_a(m)$ le mot m dans lequel on a supprimé toutes les occurrences de la lettre a. Si L est un langage, on note $s_a(L) = \{s_a(m) \mid m \in L\}$.

   a) Si L est rationnel sur Σ, $s_a(L)$ le reste-t-il ?

   b) Si $s_a(L)$ est rationnel, L l'était-il nécessairement ?

1. Oui à tout. Si $A = (\Sigma, Q, q_0, F, \delta)$ est un automate déterministe reconnaissant L :
   - $(\Sigma, Q, q_0, \{\text{états coaccessibles de } A\}, \delta)$ reconnaît Pref(L).
   - $(\Sigma, Q, \{\text{états accessibles de } A\}, F, \delta)$ reconnaît Suff(L).
   - $(\Sigma, Q, \{\text{états accessibles de } A\}, \{\text{états coaccessibles de } A\}, \delta)$ reconnaît Fact(L).
   - $(\Sigma, Q, F, q_0, \delta^t)$ où δ^t consiste à inverser les transitions reconnaît le miroir de L.
2. Oui, il suffit de transformer toutes les transitions étiquetées par a par des ε-transitions.
3. Non : $\alpha(\{a^n b^n \mid n \in \mathbb{N}\}) = b^\star$ est rationnel sans que le langage d'origine le soit.

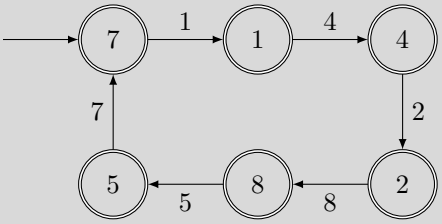
Exercice 62 (***) *Automates décimaux*

Si $\alpha \in \mathbb{R}$, on note $d(\alpha)$ le développement décimal de $\alpha - \lfloor \alpha \rfloor$ où le zéro initial et la virgule ont été supprimés. Par exemple, $d(1/3) = 33333\cdots$. On définit $L(\alpha)$ comme étant le langage que forment les préfixes finis de $d(\alpha)$. Par exemple :

$$L(\pi) = \{\varepsilon, 1, 14, 141, 1415, 14159...\}$$

1. Sachant que $1/7 = 0.142857142857142857...$, $L(1/7)$ est-il rationnel ? Si oui, donner pour ce langage une expression régulière et un automate. Si non, justifier.
2. Si $\alpha \in \mathbb{Q}$, $L(\alpha)$ est-il (aussi) rationnel ?
3. $L(\sqrt{2})$ est-il rationnel ? Si oui, en donner une expression régulière ; si non, justifier.

1. Comme la partie décimale de $1/7$ est périodiquement 142857, $L(1/7) = (142857)^\star(\varepsilon + 1 + 14 + 142 + 1428 + 14285 + 142857)$. L'automate suivant reconnaît $L(1/7)$:



2. Oui car dans ce cas le développement décimal de $\alpha - \lfloor \alpha \rfloor$ est ultimement périodique. Si on note $m(\alpha)$ le mot qui précède l'apparition de la période, $w(\alpha)$ la période et pour tout mot u, $p(u)$ l'ensemble des préfixes de u, alors $L(\alpha) = p(m(\alpha)) + m(\alpha)w(\alpha)^\star(p(w(\alpha)))$.
3. Non. Si c'était le cas, tout mot de $L(\sqrt{2})$ de longueur supérieure au nombre N d'états d'un automate reconnaissant $L(\sqrt{2})$ se factoriserait en uvw avec les propriétés du lemme de l'étoile.

   En particulier, pour tout $k \in \mathbb{N}$, on aurait uv^kw qui serait préfixe de $d(\sqrt{2}) = (m_i)_{i\in\mathbb{N}}$ (car le développement de $\sqrt{2}$ est infini) et cela implique que pour tout $i > |u|$, $m_i = m_{i+|v|}$ donc que le développement de $\sqrt{2}$ est ultimement périodique ce qui contredit son irrationnalité.

Exercice 63 (***) *Automates et palindromes*

Soit Σ un alphabet contenant au moins deux lettres. Pour tout $u = u_0...u_{n-1} \in \Sigma^\star$, on appelle miroir de u le mot $\overline{u} = u_{n-1}...u_0$. On dit que u est un palindrome si $u = \overline{u}$. On note Π le langage des palindromes sur Σ.

1. Montrer que Π n'est pas rationnel.
2. Si A est un automate fini, $L(A) \cap \Pi$ est-il rationnel ?
3. Si A est un automate fini, $\{u \in \Sigma^\star \mid u\overline{u} \in L(A)\}$ est-il rationnel ?

On note dans la suite Π_{pair} l'ensemble des palindromes de longueur paire.

4. Proposer un algorithme qui, étant donné un automate fini A, détermine si $L(A) \cap \Pi_{\text{pair}}$ est vide, fini ou infini. Préciser la complexité en temps et en espace de cet algorithme.
5. Donner un nouvel algorithme pour calculer le cardinal de $L(A) \cap \Pi_{\text{pair}}$ dans le cas où ce dernier est fini en supposant que l'automate A en entrée est déterministe. La complexité est-elle modifiée ?
6. Modifier les algorithmes précédents pour qu'ils s'appliquent à $L(A) \cap \Pi$.

1. On utilise le lemme de l'étoile avec a^Nba^N où a, b sont deux lettres de Σ.
2. Non. $\Sigma^\star$ est reconnaissable par un automate A et on aurait alors $L(A) \cap \Pi = \Pi$ rationnel.
3. Oui. Si $A = (Q, \Sigma, I, F, \delta)$, on construit $\overline{A} = (Q, \Sigma, F, I, \overline{\delta})$ où $\overline{\delta}$ consiste à renverser le sens des transitions de δ : cet automate reconnaît les miroirs des mots de $L(A)$.

   Puis on construit le produit $B = (Q^2, I \times F, \{(q, q) \mid q \in Q\}, \delta')$ où $\delta'((q_1, q_2), a)$ consiste à lire simul-

tanément a depuis q_1 dans A et a depuis q_2 dans $\overline{A}$. B reconnaît le langage souhaité car :

$$\begin{aligned} m \in L(B) &\Leftrightarrow \exists i \in I, \exists f \in F, \exists q \in Q, q \in \delta^\star(i, m) \text{ et } q \in \overline{\delta}^\star(f, m) \\ &\Leftrightarrow \exists i \in I, \exists f \in F, \exists q \in Q, q \in \delta^\star(i, m) \text{ et } f \in \delta^\star(q, \overline{m}) \\ &\Leftrightarrow \exists i \in I, \exists f \in F, \exists q \in Q, f \in \delta^\star(i, m\overline{m}) \\ &\Leftrightarrow m\overline{m} \in L(A) \end{aligned}$$

4. On constate que
$$f : \begin{cases} L(B) \to L(A) \cap \Pi_{\text{pair}} \\ u \mapsto u\overline{u} \end{cases}$$
est bijective. Donc il suffit de déterminer si $L(B)$ avec B défini ci-dessus est vide / fini / infini. La construction de $\overline{A}$ se fait linéairement en $|A|$. La construction de B se fait en $O(|A|^2)$ et produit un automate B de taille $O(|A|^2)$ puis il suffit de parcourir linéairement cet automate pour déterminer les états utiles puis savoir si un état final est accessible / s'il y a un cycle d'où une complexité en $O(|A|^2)$.
5. On peut procéder par programmation dynamique en déterminant pour tout q le nombre de mots acceptés depuis q. Ce dernier vaut 0 ou 1 selon que q est final ou non plus le nombre de mots acceptés depuis chacun des états q' tels que $q \to q'$. Pour l'ordre des calculs il suffit d'utiliser un ordre topologique sur les états de l'automate en commençant par la fin (ce qui est possible car l'automate n'a pas de cycle puisqu'il reconnaît un langage fini).

   La complexité est linéaire en la taille de l'automate B donc en $O(|A|^2)$.
6. Pour les palindromes de longueur impaire, pour chaque lettre a, on construit un automate produit B_a comme en 4 reconnaissant $\{u \in \Sigma^\star \mid ua\overline{u} \in L(A)\}$. On peut ensuite appliquer 4 et 5 et les complexités sont les mêmes, multipliées par $|\Sigma|$.

Exercice 64 (***) *Langages continuables*

Dans cet exercice, on fixe un alphabet Σ de taille au moins deux. Les automates y seront toujours considérés finis, déterministes et complets.

Un mot $u \in \Sigma^\star \setminus \{\varepsilon\}$ est dit primitif s'il n'existe pas de mot $v \in \Sigma^\star$ et d'entier $p > 1$ tel que $u = v^p$. Un langage rationnel L est dit continuable si et seulement si : $\forall u \in \Sigma^\star, \exists v \in \Sigma^\star, uv \in L$.

1. Le mot *abaaabaa* est-il primitif ? Le mot *ababbaabbbababbab* est-il primitif ?
2. Proposer un algorithme naïf qui détermine si un mot est primitif et donner sa complexité temporelle.
3. Donner un exemple de langage rationnel infini qui ne contient aucun mot primitif.
4. Donner un exemple de langage rationnel infini qui ne contient que des mots primitifs.
5. Donner un exemple de langage rationnel infini non continuable.
6. Étant donné un automate A, proposer un algorithme qui détermine si $L(A)$ est continuable. Justifier sa correction et préciser sa complexité temporelle.
7. Soit L un langage rationnel. Montrer que si L est continuable, il a une infinité de mots primitifs.
8. La réciproque à la question précédente est-elle vraie ?

1. Non pour le premier car il vaut $(abaa)^2$. Oui pour le second car sa taille est un nombre premier.
2. Si u est de taille n, il suffit de vérifier pour tout préfixe v de u de taille inférieure à $n/2$ si $u = v^{|u|/|v|}$. Une telle vérification se fait en $O(n)$ d'où une complexité temporelle en $O(n^2)$.
3. Le langage aa^+ convient.
4. Le langage $a^\star b$ convient.
5. Le langage $a^\star$ convient car b n'est pas continuable. On se sert ici du fait que $|\Sigma| \geq 2$.
6. Il suffit d'émonder puis compléter l'automate pour obtenir un automate A' puis de vérifier si tous ses états sont coaccessibles : si oui, le langage est continuable ; si non, non. Cela revient à faire des parcours de graphes, donc se fait linéairement en la taille de l'automate.

Pour la preuve de correction : si il y a un état q non coaccessible dans A', comme il est accessible par émondage, il existe u tel que $q \in \delta^\star(q_0)$ et pour tout v, lire v depuis q n'amène jamais dans un état final donc u n'est pas continuable et $L(A)$ non plus. Réciproque similaire.

7. Soit A reconnaissant L continuable ; dont on peut supposer les n états utiles. Pour tout mot u, il existe un mot v tel que $uv \in L(A) = L$ tel que $|v| < n$. Pour tout $i \geq n-1$, on note $u_i = ab^i$ et on note v_i le plus petit mot permettant de continuer u_i, cad tel que $w_i = u_i v_i \in L$.

   Alors w_i est primitif car si $w_i = x^p$, x commence par un a et le prochain a est au mieux i lettres plus loin et dans v_i donc $|x| \geq i+1 \geq n > |v_i|$ et $|v_i| \geq |x|$ ce qui est impossible. Il y a une infinité de tels w_i car $i+1 \leq |w_i| < i+1+n$ et donc les $(w_{kn})_{k\in\mathbb{N}^\star}$ sont deux à deux distincts.

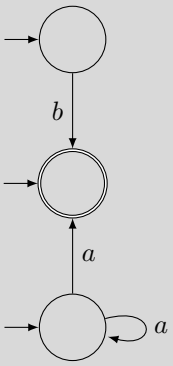
8. Non : tous les mots de $a^\star b$ sont primitifs mais ce langage n'est pas continuable.

Exercice 65 (**) *Automates à contraintes*

On se place sur l'alphabet $\Sigma = \{a, b\}$ dans chacun des cas suivants, dire en justifiant s'il existe ...

1. ... un automate reconnaissant $a^\star + b$ avec un seul état final ?
2. ... un automate reconnaissant les mots de la forme uv avec u inferieur à v lexicographiquement ?
3. ... un automate déterministe reconnaissant les mots de la forme uv avec u préfixe de v ?
4. ... un automate émondé et complet reconnaissant $ba^\star$?

1. L'automate suivant convient :



2. Non. Sinon ce langage serait reconnu par un automate à n états et le mot ba^nba^n, qui fait partie du langage serait reconnu via un chemin qui contiendrait nécessairement un cycle dans la première série de a (car il y en a trop par rapport au nombre d'états). En faisant un tour de plus dans ce cycle, on obtiendrait un mot reconnu alors qu'il ne devrait pas l'être.
3. Oui. Le langage décrit ici est en fait $\Sigma^\star$ puisque tout mot m est égal à εm avec ε qui est bien un préfixe de m. Si on impose en revanche que u soit préfixe non vide de v alors non, avec la même démarche qu'à la question précédente.
4. Non. Dans un automate complet et émondé, { préfixes des mots reconnus } est nécessairement égal à $\Sigma^\star$. Or, bb est un mot qui n'est préfixe d'aucun mot de $ba^\star$.

Exercice 66 (***) *Stabilité des langages rationnels*

Nous avons vu en cours que les langages rationnels sont stable par union, concaténation, étoile de Kleene, intersection, complémentaire et différence. L'exercice *Reconnaissance de racines* étudie la stabilité des langages rationnels par racine carrée. On introduit dans cet exercice de nouvelles opérations sur les langages et on se demande si les langages rationnels sont stables pour ces opérations.

1. Le miroir de L est $\{m_k...m_1 \,|\, m_1...m_k \in L\}$. Les langages rationnels sont-ils stables par miroir ?
2. Le préfixe de L est l'ensemble des préfixes de mots de L. Les langages rationnels sont-ils stables par préfixe ?
3. Les langages rationnels sont-ils stables par suffixe ?
4. Les langages rationnels sont-ils stables par facteur ?

5. Les langages rationnels sont-il stables par moitié sachant que la moitié de L est :

$$L/2 = \{u \in \Sigma^\star \mid \exists v \in \Sigma^\star, |u| = |v| \text{ et } uv \in L\}$$

6. Le produit de mélange de L et L', noté $L\#L'$ est l'ensemble des mots de la forme $u_1v_1...u_kv_k$ avec les $u_i, v_i \in \Sigma^\star$ tels que $u_1u_2...u_k \in L$ et $v_1v_2...v_l \in L'$. Les langages rationnels sont-ils stables par mélange ?
7. Le conjugué de L est $\{vu \mid uv \in L\}$. Les langages rationnels sont-ils stables par conjugué ?
8. Si u est un mot fixé, le résiduel de L par rapport à u est $u^{-1}L = \{v \mid uv \in L\}$. Les langages rationnels sont-ils stables par résiduel ?

Un morphisme μ est une fonction $\Sigma^\star \to \Gamma^\star$ où Σ et Γ sont deux alphabets qui est compatible avec la concaténation, c'est-à-dire que $\mu(\varepsilon) = \varepsilon$ et pour tous mots $u, v \in \Sigma^\star$, $\mu(uv) = \mu(u)\mu(v)$. Si L est un langage, on note $\mu(L) = \{\mu(u) \mid u \in L\}$ et $\mu^{-1}(L) = \{u \in \Sigma^\star \mid \mu(u) \in L\}$.

9. Les langages rationnels sont-ils stable par morphisme ?
10. Les langages rationnels sont-ils stables par morphisme inverse ?
11. On considère une fonction φ qui transforme les mots de la manière suivante : $\varphi(\varepsilon) = \varepsilon$ et pour tout mot non vide, φ permute chaque lettre d'indice pair avec la lettre qui lui succède immédiatement. Par exemple, $\varphi(abbab) = baabb$. On note $\varphi(L) = \{\varphi(u) \mid u \in L\}$. Si L est rationnel, $\varphi(L)$ le reste-t-il ?

TLDR : toutes ces transformations conservent les langages rationnels. Le principe quasiment à chaque fois est de construire un automate pour le langage transformé à partir d'automates pour les langages en entrée et de conclure via le théorème de Kleene. Soit $A = (\Sigma, Q, q_0, F, \delta)$ et $A' = (\Sigma, Q', q_0', F', \delta')$ deux automates détrministes reconnaissant respectivement L et L'.

1. $(\Sigma, Q, F, q_0, \delta^t)$ où δ^t consiste à inverser les transitions reconnaît le miroir de L.
2. $(\Sigma, Q, q_0, \{\text{états coaccessibles de } A\}, \delta)$ reconnaît Pref(L).
3. $(\Sigma, Q, \{\text{états accessibles de } A\}, F, \delta)$ reconnaît Suff(L).
4. $(\Sigma, Q, \{\text{états accessibles de } A\}, \{\text{états coaccessibles de } A\}, \delta)$ reconnaît Fact(L).
5. Même type de preuve que pour $\sqrt{L}$, voir l'exercice *Reconnaissance de racines*.
6. $(\Sigma, Q \times Q', \{(q_0, q_0')\}, F \times F', \Delta)$ avec $\Delta((q, q'), a) = \{(r, q') \mid \delta(q, a) = r\} \cup \{(q, r') \mid \delta'(q', a) = r'\}$ reconnaît $L\#L'$ (c'est un simili-produit dans lequel on peut avancer soit dans A soit dans A').
7. On peut supposer que A est normalisé (un seul état initial avec aucune transition entrante et un seul état final avec aucune transition sortante). Pour chaque état q de A différent de son état initial i et de son état final f, on note $L_q = \{m = m_1m_2 \mid$ il existe un chemin dans A $i \to^{m_2} q \to^{m_1} f\}$.

   On dédouble alors l'automate A en notant avec des barres les copies des états. Alors $(\Sigma, \widetilde{Q}, \{q\}, \{\overline{q}\}, \delta \cup \overline{\delta})$ avec $\widetilde{Q} = Q \cup \overline{Q}$ où f et $\overline{i}$ sont fusionnés reconnaît L_q. Le conjugué de L vaut $L \cup \bigcup_{q \neq i, q \neq f} L_q$ et est donc reconnu en tant qu'union finie de langages reconnus.
8. $(\Sigma, Q, \{\text{état dans lequel on arrive en lisant } u\}, F, \delta)$ reconnaît $u^{-1}L$.
9. Un morphisme est défini par l'image qu'il donne des lettres. Pour chaque lettre a, remplacer les occurrences de a par $\mu(a)$ dans une expression rationnelle pour L donne une expression pour $\mu(L)$.
10. $(\Sigma, Q, q_0, F, \{(p, a, q) \mid$ il existe un chemin $p \to^{\mu(a)} q$ dans $A\})$ reconnaît $\mu^{-1}(L)$.
11. Cf Mines 2009 option informatique.

3 Algorithmique des graphes

Exercice 67 (**) *Parcours en profondeur précis*

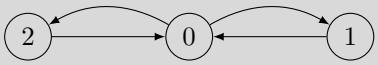
On considère la version du parcours en profondeur donnée en cours permettant la classification des arcs. Les tableaux de début et fin d'exploration d'un sommet sont notés D et F. Dans chacun des cas suivants, dire en justifiant si l'affirmation est correcte :

1. Si y est accessible depuis x alors on initialisera $D[y]$ pendant l'exploration de x.
2. Si y est accessible depuis x et $D[x] < D[y]$ alors $D[y] < F[x]$.

3. S'il existe un arc (x, y) et que $D[x] < D[y]$ alors $D[y] < F[x]$.

1. Faux : 0 est accessible depuis 1 mais dans un parcours à partir de 0, $D[0]$ ne sera pas initialisé lors de l'exploration de 1 puisque cette date de début de traitement sera déjà connue :

2. Faux. Parcourons le graphe suivant à partir de 0 :



On peut obtenir les dates de début et fin de traitement suivantes :

Sommet	0	1	2
Début	1	2	4
Fin	6	3	5

Le sommet 1 est accessible depuis 2, $D[1] = 2 < 4 = D[2]$ et pourtant $D[2] > F[1]$.

3. Vrai. Comme $D[x] < D[y]$, x est le premier sommet à être exploré parmi $\{x, y\}$, et comme il existe un arc (x, y), l'exploration de y est donc lancée lors de l'exploration de x. Donc l'exploration de y termine avant celle de x à cause des propriétés d'une pile : $F[y] < F[x]$ et donc $D[y] < F[x]$.

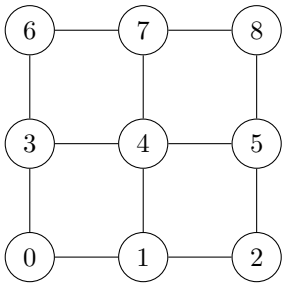
Exercice 68 (**) *Whatever First Search*

On considère deux versions de l'algorithme générique de parcours vu en cours (la version de gauche est exactement celle du cours) auquel on ajoute de l'information pour reconstruire les arbres de parcours :

```
WFS1(G, s_0) =
Mettre (−1, s_0) dans le sac
tant que le sac n'est pas vide
    Sortir un élément (p, s) du sac
    si s n'est pas marqué alors
        Marquer s
        Ajouter (p, s) à l'arbre de parcours
        pour tout voisin t de s
            Mettre (s, t) dans le sac
```

```
WFS2(G,s_0) =
Mettre (−1, s_0) dans le sac
Marquer s_0
tant que le sac n'est pas vide
    Sortir un élément (p, s) du sac
    Ajouter (p, s) à l'arbre de parcours
    pour tout voisin t de s
        si t n'est pas marqué alors
            Mettre (s, t) dans le sac
            Marquer t
```

1. On suppose que le sac est une file. Déterminer si les données suivantes changent entre les deux parcours :
   a) L'ordre de parcours.
   b) La complexité temporelle.
   c) La complexité spatiale.
2. On suppose que le sac est une pile. En supposant que lorsqu'un choix est possible on traite les sommets par ordre croissant de numéros, appliquer WFS1 au graphe suivant et déterminer l'ordre dans lequel les sommets sont traités et l'arbre du parcours. Faire de même pour WFS2 et comparer.



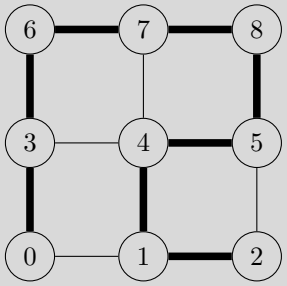
1. La différence entre les deux est la temporalité du marquage : dans WFS1 on parque au sortir du sac alors que dans WFS2 on marque à l'entrée du sac.
   a) Ne change pas car les sommets sont ajoutés pour la première fois à la file dans le même ordre.

b) Avec des opérations en $O(1)$ sur la file, on obtient du $O(|V| + |E|)$ dans les deux cas.

c) La complexité spatiale peut être en $O(|A|)$ pour WFS1 (un sommet peut entrer dans la file autant de fois que son nombre de prédecesseurs) alors qu'elle est en $O(|S|)$ pour WFS2 grâce au marquage.

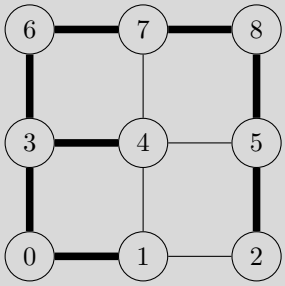
2. Ici, "traiter en premier le petit numéro" veut bien dire "lancer le parcours, c'est-à-dire, mettre dans la pile le petit numéro" et pas "se déplacer d'abord vers le petit numéro" (qui demanderait à ce que ce numéro soit au sommet de la pile donc y ait été mis en dernier). On obtient :

Pour WFS1 :



Ordre du parcours : $0, 3, 7, 8, 5, 4, 1, 2$

Pour WFS2 :

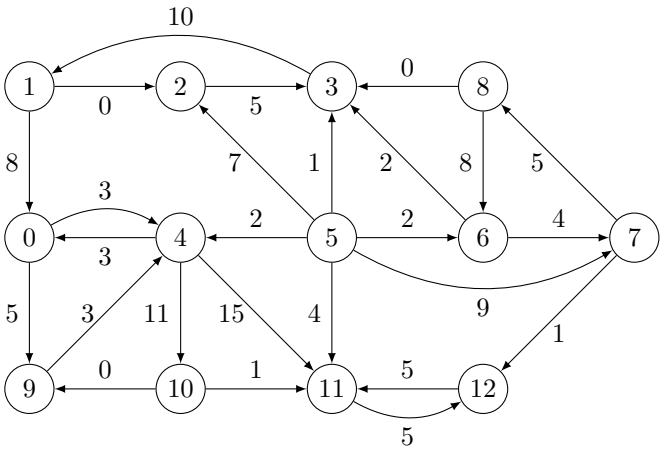


Ordre du parcours : $0, 3, 6, 7, 8, 5, 2, 4, 1$

Les deux parcours sont différents, WFS2 n'est pas un "vrai" parcours en profondeur.

Exercice 69 (*) *Quelques gammes*

On considère le graphe G suivant :



1. Appliquer l'algorithme de Kosaraju à ce graphe (en indiquant les dates de début et de fin de traitement de chaque sommet lors du premier parcours ; dès qu'un choix se présente dans ledit parcours, on considèrera en premier le sommet de plus petit numéro possible).
2. Calculer le graphe quotient associé à G et en donner un tri topologique.

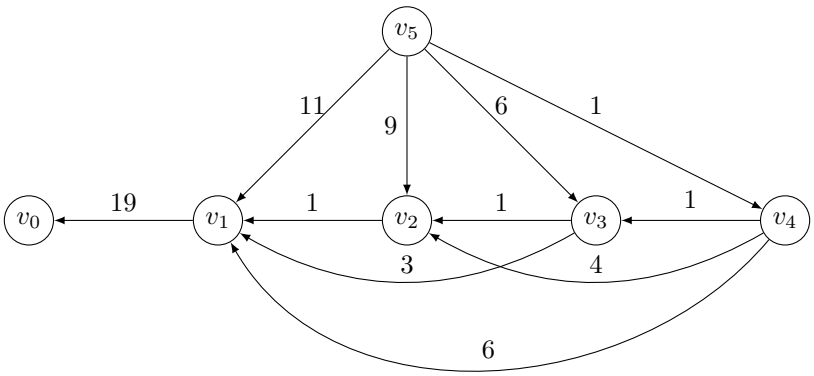
On considère à présent le graphe G', égal à G dans lequel on a supprimé les orientations (on peut, car à chaque fois qu'il existe un couple d'arêtes orientées (u, v) et (v, u), ces deux arêtes ont le même poids).

3. Calculer les plus courts chemins d'origine le sommet 1 et leurs poids grâce à l'algorithme de Dijkstra.
4. Déterminer un arbre couvrant minimal pour G' (et son poids) à l'aide de l'algorithme de Kruskal.

Applications directes du cours.

Exercice 70 (*) *Heuristiques pour A^**

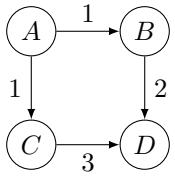
On souhaite déterminer un plus court chemin entre la source v_5 et la destination v_0 à l'aide de l'algorithme A^*. Les valeurs de l'heuristique h pour chaque sommet sont indiquées ci-dessous :



Sommet v	v_0	v_1	v_2	v_3	v_4	v_5
Valeur de $h(v)$	0	0	3	7	13	23

1. L'heuristique h est-elle admissible ? Est-elle monotone ?
2. Appliquer l'algorithme de Dijkstra à cet exemple depuis v_5 puis lui appliquer A^*. Commenter.

On considère à présent le graphe G suivant dans lequel on cherche un plus court chemin de A à D :



3. Proposer une heuristique monotone pour G.
4. Proposer une heuristique admissible mais non monotone pour G.
5. Proposer une heuristique non admissible mais telle que le déroulé de A^* renvoit le résultat escompté lorsqu'on tente de calculer un plus court chemin de A à D.
6. Proposer une heuristique pour laquelle A^* ne renvoit pas un plus court chemin de A à D.

1. h est admissible mais pas monotone car pour l'arc (v_5, v_4) : $h(v_5) = 23 > p(v_5, v_4) + h(v_4) = 14$.
2. Évolution du tableau des distances avec Dijkstra et du contenu de la file de priorité (si on l'implémente avec modification des priorités au fil de l'algorithme) avec comme convention que les couples représentent (sommet, priorité) :

File	$D[0]$	$D[1]$	$D[2]$	$D[3]$	$D[4]$	$D[5]$
$(5,0)$	∞	∞	∞	∞	∞	0
$(1,11),(2,9),(3,6),(4,1)$	∞	11	9	6	1	0
$(1,6),(2,5),(3,2)$	∞	7	5	2		
$(1,5),(2,3)$	∞	5	3			
$(1,4)$	∞	4				
	23					

Avec A*, on tombe dans un pire cas dans lequel les sommets repassent plusieurs fois dans la file de priorité avec des priorités de plus en plus basses. Voici le début de l'exécution ($P[i]$ = priorité du sommet i) :

$P[0]$	$P[1]$	$P[2]$	$P[3]$	$P[4]$	$P[5]$	$D[0]$	$D[1]$	$D[2]$	$D[3]$	$D[4]$	$D[5]$
					0	∞	∞	∞	∞	∞	0
	11	12	13	14		∞	11	9	6	1	0
30		**12**	13	14		30	11	9	6	1	0
30	**10**		13	14		30	10	9	6	1	0
29		**12**	13	14		29	10	9	6	1	0
29			**13**	14		29	10	9	6	1	0
29	**9**	10		14		29	9	7	6	1	0
28		**10**		14		28	8	7	6	1	0
28	**8**			14		27	8	7	6	1	0
28				**14**		27	8	5	2	1	0
28	**7**	8	10			...					

3. 4. 5. 6. Pour les questions suivantes on propose de manière récapitulée :

Sommet	A	B	C	D
Heuristique des sommets pour Q3	3	2	2	0
Heuristique des sommets pour Q4	0	0	0	0
Heuristique des sommets pour Q5	2	3	4	0
Heuristique des sommets pour Q6	2	4	1	0

Exercice 71 (**) *Caractérisation des graphes bipartis*

Un graphe non orienté $G = (V, E)$ est dit biparti s'il existe une partition de V en $V = V_1 \sqcup V_2$ telle que toute arête de E a une extrémité dans V_1 et l'autre dans V_2.

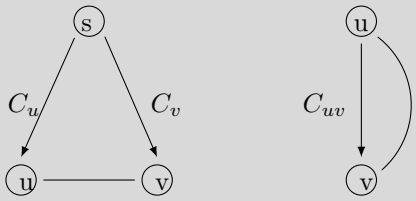
1. Montrer qu'un arbre est un graphe biparti.
2. Montrer qu'un graphe est biparti si et seulement si toutes ses composantes connexes le sont.

Ce dernier résultat montre que pour détecter si un graphe est biparti, il suffit de travailler composante connexe par composante connexe. Sans perte de généralité, on peut donc supposer que G est connexe.

3. Montrer qu'un graphe connexe est biparti si et seulement s'il ne contient aucun cycle de longueur impaire. *Indice : pour la réciproque, raisonner par l'absurde et considérer un arbre couvrant du graphe.*
4. Concevoir un algorithme permettant de détecter si un graphe est biparti. *Indice : parcours !*

1. On décompose les sommets en $V_1 = \{$sommets à distance paire de la racine$\}$ et $V_2 = \{$sommets à distance impaire de la racine$\}$. Cela donne une partition convenable (car pas de cycle dans un arbre).
2. La partition du graphe entier induit une partition correcte sur chacune des composantes. Réciproquement, les sommets de chacune des k composantes se partitionnent en $V_{i,1} \sqcup V_{i,2}$ et alors $V_1 = \bigcup_{i=1}^{k} V_{i,1}$ et $V_2 = \bigcup_{i=1}^{k} V_{i,2}$ partitionnent correctement les sommets du graphe entier.
3. Si G est biparti, supposons par l'absurde qu'il contient un cycle de longueur impaire. Alors son premier sommet est dans V_1 (sans perte de généralité), le second dans V_2... et ainsi de suite jusqu'au premier sommet qui doit être aussi dans V_2. Comme V_1 et V_2 sont disjoints, c'est impossible.

   Si G ne contient pas de cycle de longueur impaire, considérons un arbre A de parcours de G (qui est biparti par Q1) et montrons que cette bipartition $V_1 \sqcup V_2$ des sommets induit une bipartition correcte pour G. Si ce n'était pas le cas, il existerait une arête (u, v) dans G dont les deux extrémités sont dans V_1 (par exemple). Notons s le plus petit ancêtre commun à u et v dans A. Alors, soit $s \notin \{u, v\}$ (dessin de gauche), soit $s = u$ (ou v mais c'est symétrique, dessin de droite) :



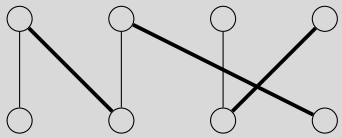
Dans le premier cas, comme u et v sont tous les deux dans V_1, la parité des longueurs des chemins C_u et C_v dans A est la même. On en déduit que le chemin $s \to^{\text{par } C_u} u \to v \to^{\text{par } C_v} s$ est un cycle de longueur impaire (que C_u et C_v soient tous les deux de longueur impaire ou tous les deux de longueur paire). Dans le second cas, comme u et v sont dans V_1, la longueur du chemin C_{uv} dans A est paire donc là encore on a découvert un cycle de longueur impaire dans G : c'est contradictoire.

4. Le piège est de partir sur de la détection de cycle (puisqu'il faudrait calculer la longueur de tous les cycles pour utiliser Q3). À la place, il vaut mieux utiliser un algorithme de parcours en largeur qui "colorie" les sommets en fonction de la parité de la distance à la racine du parcours : le graphe est biparti si et seulement s'il n'y a pas de conflit de coloriage.

Exercice 72 (*) *Couplages dans un graphe quelconque*

1. Donner une condition nécessaire simple pour qu'un graphe connexe admette un couplage parfait. Est-elle suffisante ?
2. Montrer que pour $n \geq 2$, il existe un graphe connexe à n sommets pour lequel la cardinalité maximale d'un couplage vaut un.
3. Si C et C' sont deux couplages dans un même graphe et que C est maximal, montrer que $|C'| \leq 2|C|$.

1. Il faut que le nombre de sommets du graphe soit pair. Ce n'est pas suffisant : il suffit de considérer un graphe à deux sommets sans arête. On peut même trouver un contre-exemple avec un graphe connexe :



En effet, le couplage en gras est maximum (car pas de chemin augmentant) mais pas parfait.

2. Il suffit de considérer un graphe étoilé avec un sommet central relié à tous les autres.
3. Notons (x_i, y_i) les k arêtes du couplage C'. Pour tout $i \in [\![1, k]\!]$, soit x_i soit y_i est couvert par C, sans quoi on pourrait ajouter une arête à C alors qu'il est maximal. Donc le nombre de sommets couverts par C est au moins égal à k, c'est à dire $2|C| \geq |C'|$. *Remarque : en particulier cela signifie qu'un couplage maximal est de taille au moins la moitié d'un couplage maximum.*

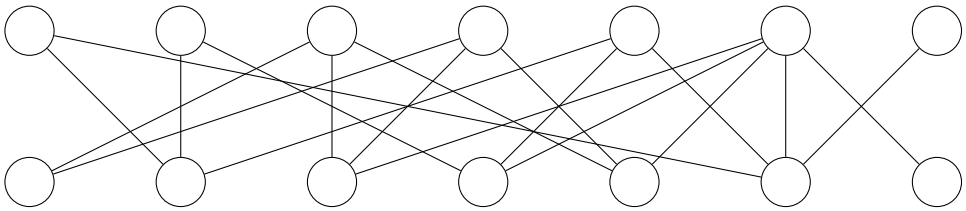
Exercice 73 (***) *Théorème de König*

Si G est un graphe non orienté, non pondéré, on appelle *transversal* de G un sous ensemble T des sommets de G telle que toute arête de G est incidente à au moins un sommet de T (T "couvre" les arêtes). On dit qu'un transversal de G est minimum si aucun transversal de G n'a un cardinal strictement plus petit.

1. Si M est un couplage de G et T un transversal de G, montrer que $|M| \leq |T|$.
2. Le résultat précédent montre que la taille d'un couplage maximum dans un graphe quelconque est inférieure à la taille d'un transversal minimum. A-t-on égalité entre ces deux quantités en général ?
3. On suppose à présent que G est un graphe biparti dont on note $U \sqcup V$ la bipartition des sommets et M un couplage maximum. On note E l'ensemble formé des sommets de U non saturés par M auxquels on ajoute les sommets atteignables à partir d'un de ces sommets par un chemin M-alternant.
   a) Montrer que pour toute arête (u, v) de M on a soit $u, v \in E$ soit $u, v \notin E$.
   b) Montrer que $T = (U \setminus E) \cup (V \cap E)$ est de cardinal égal à $|M|$.
   c) Montrer que T est un transversal de G.

d) Déduire de ce qui précède le théorème de König : dans un graphe biparti, la taille d'un couplage maximum est égale à la taille d'un transversal minimum.

4. Déterminer un transversal minimum dans le graphe suivant :



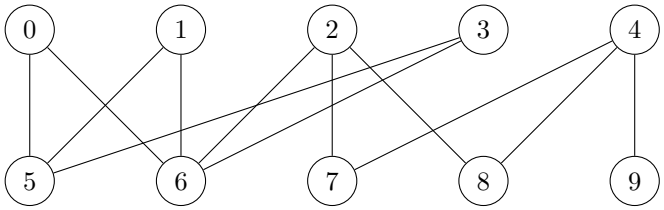
Remarque : Le problème du calcul d'un transversal minimum est un problème connu sous le nom de `VERTEX COVER` *et est NP-complet dans un graphe quelconque. Le théorème de König, tout comme le théorème de Hall, est un cas particulier du théorème flot-max / coupe-min, qui est un résultat important en optimisation.*

1. Tout sommet de T est couvert par au plus une arête de M sinon M ne serait pas un couplage.
2. Non, considérer un 3-cycle : la taille d'un couplage max est 1 mais la taille du transversal min est 2.
3. TODO.
4. On peut utiliser l'algorithme sous-jacent à la question 3 ou utiliser l'algorithme de Berge pour constater qu'un couplage maximum dans ce graphe est de taille 6 puis trouver un transversal de taille 6 à l'oeil.

Exercice 74 (***) *Théorème de Hall*

Soit G un graphe biparti dont on note (U, V) une bipartition des sommets. On cherche à montrer le théorème de Hall : il existe un couplage de taille $|U|$ dans G si et seulement si, pour tout sous-ensemble $X \subset U$, l'ensemble $V(X)$ des voisins de X vérifie $|V(X)| \geq |X|$.

1. Montrer le sens direct de ce théorème.
2. En s'aidant au besoin d'un récurrence sur la taille de $|U|$, montrer le sens réciproque.
3. En déduire que G admet un couplage parfait si et seulement si $|U| = |V|$ et pour tout sous ensemble $X \subset U$, $|V(X)| \geq |X|$.
4. Le graphe suivant admet-il un couplage parfait ? En trouver un couplage maximum en justifiant.



5. Montrer que si G est un graphe biparti k-régulier avec $k \geq 1$ (c'est-à-dire, tous les sommets de G ont degré égal à k), alors G admet un couplage parfait.

Remarque : Le théorème de Hall s'énonce aussi avec le vocabulaire de la théorie des ensembles. On l'appelle aussi "lemme des mariages" : U représente un ensemble de femmes, V un ensemble d'hommes, une arête entre u et v indique que u et v sont d'accord pour se marier et le théorème de Hall donne une condition nécessaire et suffisante pour que toutes les femmes puissent se marier. Ce théorème fait partie d'un ensemble de théorèmes en analyse combinatoire, chacun ayant une adpatation en théorie des graphes.

1. Supposons que G admette un couplage de taille $|U|$. Par l'absurde, s'il existait une partie $X \subset U$ telle que $|V(X)| < |X|$, il y aurait au moins un sommet de X qui ne serait apparié à aucun autre dans le couplage, ce qui l'empêcherait d'être de taille $|U|$.
2. Montrons la réciproque par récurrence forte. C'est OK si $|U| = 1$ puisqu'il suffit d'apparier l'unique sommet de U à un de ses voisins dans V, qui existe. Si $|U| \geq 2$:
   - Si pour tout $\emptyset \subsetneq X \subsetneq U$ on a en fait une inégalité plus forte selon laquelle $|V(X) \geq |X| + 1$, on peut choisir une arête (a, b) de G et dans le graphe G' dans lequel on retire les sommets a et b on aura que tout $X \subset U \setminus \{a\}$ vérifie $|V_{G'}(X)| = |V_G(X)| - 1 \geq |X|$ (où on note $V_H(X)$ le voisinage

de X dans le graphe H). On peut alors appliquer l'hypothèse de récurrence à G' et compléter le couplage de taille $|U|-1$ qu'elle produit par l'arête (a,b) pour conclure.

- Sinon, il existe un ensemble $\emptyset \subsetneq U' \subsetneq U$ tel que $|V(U')| = |U'|$. Considérons le graphe G' induit par les sommets de $U' \cup V(U')$: on peut lui appliquer l'hypothèse de récurrence ce qui fournit un couplage M' qui sature U'. On considère à présent G'' induit par les sommets autres que ceux de G'. Pour tout $X \subset U'' = U \setminus U'$ on a :

$$|V_{G''}(X)| \geq |V_G(X \cup U')| - |V_G(U')| \geq |X \cup U'| - |U'| = |X|$$

Donc on peut aussi appliquer l'hypothèse de récurrence à G'' pour obtenir un couplage M'' saturant U''. Alors $M' \cup M''$ est un couplage qui sature U.

3. Si G admet un couplage parfait, il sature tous les sommets donc $|U| = |V|$ puisque le graphe est biparti et la seconde condition est vraie d'après Q1. Réciproquement, la seconde condition indique qu'il existe un couplage de taille $|U|$ dans G et la première assure qu'il est alors parfait.
4. Non, car $V(\{0,1,3\}) = \{5,6\}$ est de trop petit cardinal. Le couplage $\{(0,5),(1,6),(2,7),(4,9)\}$ est maximum car il est de taille 4 et qu'on sait qu'il n'existe pas de couplage de taille 5.
5. Si U, V est la bipartition de G et que G est k-régulier, on sait déjà que $|U| = |V|$. De plus, si $X \subset U$, il est incident à exactement $k|X|$ arêtes qui font partie des $k|V(X)|$ arêtes incidentes à $V(X)$. Donc $k|X| \leq k|V(X)|$ puis $|V(X)| \geq |X|$ et la question 3 assure que G admet un couplage parfait.

Exercice 75 (**) *Arbres couvrants minimaux et arêtes de poids minimal*

On considère un graphe $G = (S, A)$ non orienté, pondéré via une fonction de pondération p injective (tous les poids des arêtes sont donc distincts) et tel que $|A| \geq 3$. Soit T un arbre couvrant minimal de G.

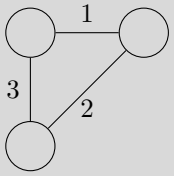
1. Si T' est un arbre couvrant minimal de G, montrer que $T = T'$.
2. Montrer que T contient les deux arêtes de poids minimal de A.
3. Montrer que T ne contient pas nécessairement les trois arêtes de poids minimal de A.
4. Montrer que pour chaque cycle de G, T ne contient pas l'arête de poids maximal du cycle.
5. T contient-il l'arête de poids minimal de chaque cycle ?

1. Supposons par l'absurde que $T \neq T'$ et notons $T = (S, B)$ et $T' = (S, B')$. Considérons l'arête de poids minimal dans $B \Delta B'$. On peut supposer que cette arête $a \in B \setminus B'$. Ainsi, rajouter a à T' créé un cycle, qui contient une arête $a' \in B' \setminus B$ sans quoi il y aurait un cycle dans l'arbre T. On considère alors T'' qui correspond à l'arbre T' dans lequel on ajoute a et on enlève a' : c'est un arbre couvrant de poids :

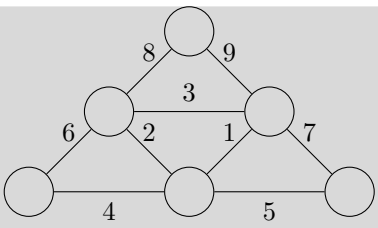
$$p(T'') = p(T') + p(a') - p(a) < p(T) \text{ puisque } p(a) < p(a') \text{ vu la construction de } a$$

On a trouvé un arbre couvrant de poids strictement plus petit que celui d'un arbre couvrant minimal.

2. La question précédente montre l'unicité de l'arbre couvrant minimal. Donc, l'arbre couvrant minimal calculé par l'algorithme de Kruskal est le seul arbre couvrant minimal du graphe. Or, dans l'arbre calculé par l'algorithme de Kruskal, les deux arêtes de poids minimal sont présentes car ce sont les deux premières considérées et elles ne peuvent pas à elles deux créer de cycle.
3. Le graphe suivant produit un contre-exemple :



4. Par l'absurde, si l'arête a de poids maximal d'un cycle est dans l'unique arbre couvrant minimal T, en supprimant a de T on obtient deux composantes connexes et rajouter n'importe quelle arête du cycle autre que a produit un arbre de poids strictement plus petit que T par injectivité de p.
5. Eh non :



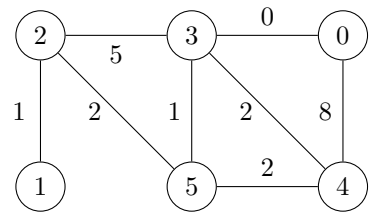
Les trois arêtes centrales sont chacune l'arête minimale d'un cycle mais aucun arbre couvrant minimal ne peut les contenir toutes (ce qui est d'ailleurs plus fort que ce qu'on voulait démontrer).

Exercice 76 (**) *Algorithme de Prim*

On considère un graphe $G = (S, A)$ non orienté, connexe, pondéré par une fonction de pondération p. On souhaite calculer un arbre couvrant minimal pour G via l'algorithme de Prim-Jarník décrit ci-après :

```
Prim(G) =
E ← ∅
V ← {s_0} où s_0 est un élément de S pris au hasard
F ← S \ {s_0}
tant que F ≠ ∅
    Choisir une arête (u, v) ∈ A telle que u ∈ V, v ∉ V et p(u, v) est minimal
    F ← F \ {v}
    E ← E ∪ {(u, v)}
    V ← V ∪ {v}
renvoyer (V, E)
```

1. Appliquer cet algorithme au graphe suivant avec $s_0 = 0$.



2. Montrer qu'à tout instant, V et F forment une partition de l'ensemble S des sommets.
3. Montrer la terminaison de cet algorithme.
4. Montrer que "le graphe (V, E) est inclus dans un arbre couvrant minimal de G" est un invariant pour la boucle tant que de cet algorithme. En déduire la correction de l'algorithme.

On cherche à présent à implémenter cet algorithme.

5. Traduire l'algorithme de Prim comme étant un Whatever First Search (terme non contractuel) dont la structure de sac est une file de priorité min F bien choisie. Donner l'évolution des structures dans ce parcours lors de l'exécution de l'algorithme de Prim sur l'exemple de la question 1.
6. En supposant que F est implémenté par un tas min, quelle est la complexité de l'algorithme de Prim ?

1. RAS, on devrait obtenir un arbre de poids 6. Il n'y a pas unicité.
2. Cette propriété est un invariant de la boucle tant que.
3. $|F|$ est un variant pour la boucle tant que : c'est un entier naturel valant initialement $|S| - 1$ et qui décroît strictement à chaque itération dans l'ensemble bien fondé $(\mathbb{N}, <)$.
4. Cette propriété est initialement vraie car $(\{s_0\}, \emptyset)$ est inclus dans tout arbre couvrant minimal de G et il en existe au moins un car G est connexe.

   Si (V, E) est inclus dans un arbre couvrant minimal avant une itération, montrons que cela reste vrai avant l'itération suivante. On suppose alors qu'il existe un arbre couvrant minimal (S, T) tel que (V, E) y est inclus et on souhaite prouver l'existence d'un arbre couvrant minimal de G (pas forcément (S, T)) qui contient le graphe $(V \cup \{v\}, E \cup \{(u, v)\})$ où (u, v) est l'arête sélectionnée par l'algorithme de Prim.

- Si $(u, v) \in T$ alors il n'y a rien à faire puisque $(V \cup \{v\}, E \cup \{(u, v)\})$ est bien inclus dans (S, T).
- Sinon, comme (S, T) est un arbre couvrant, il existe un chemin dans cet arbre menant de u à v. Mais au moment de rajouter l'arête (u, v) à notre arbre de Prim en construction, $u \in V$ et $v \notin V$ donc ce chemin emprunte nécessairement une arête (x, y) tel que $x \in V$ et $y \notin V$. Considérons dès lors l'arbre (S, T') où T' est égal à T dans lequel on a retiré l'arête (x, y) et ajouté l'arête (u, v). Cet arbre reste couvrant. Par définition de l'arête choisie par l'algorithme de Prim, $p(u, v) \leq p(x, y)$.

  Comme T est minimal on a ainsi :

  $$p(T) \leq p(T') = p(T) - p(x, y) + p(u, v) \leq p(T)$$

  Donc $p(T) = p(T')$ et cela signifie que (S, T') est en fait un arbre couvrant minimal. Comme il contient $(V \cup \{v\}, E \cup \{(u, v)\})$, on a gagné !

Pour conclure la preuve, on observe qu'en fin d'algorithme (V, E) est connexe par construction, couvrant puisque $V = S$ et $E \subset A$ et vérifie $|E| = |V| - 1$ puisqu'on ajoute une arête pour chaque sommet de G sauf s_0. Ainsi, (V, E) est un arbre couvrant de G et comme l'invariant assure qu'il est inclus dans un arbre couvrant minimal de G, c'est bien un arbre couvrant minimal.

5. On traduit l'algorithme de Prim comme étant un Whatever First Search où le sac est une file de priorité min F. Ses éléments sont des triplets (p, s, d) où s est un sommet, p son parent dans le parcours et la priorité d est le poids de l'arête (p, s). On dispose bien sûr d'un tableau de marquage qui marque les sommets déjà visités, qui sont exactement ceux de l'ensemble V. Autrment dit :

```
Créer une structure de marquage
Créer une file de priorité min F
Insérer (−1, s_0, 0) dans F
tant que F ≠ ∅
    (p, s, d) ← extraire de F
    si s n'est pas marqué alors
        Marquer s
        Ajouter (p, s) à l'arbre
        pour tous les voisins v de s
            Ajouter (s, v, p(s, v)) à F
```

Si on implémente le Whatever First Search naïvement (en autorisant l'insertion répétée d'un même sommet avec différentes priorités plutôt que de se doter d'une opération de diminution de priorité), on obtient une trace suivante à partir de $F = \{(-1, s_0 = 0, 0)\}$:

Évolution de F	Évolution du marquage	Évolution de E
$\underline{(-1, 0, 0)}$	Marquage de 0	
$\underline{(0, 3, 0)}, (0, 4, 8)$	Marquage de 3	Ajout de $(0, 3)$
$(0, 4, 8), \underline{(3, 0, 0)}, (3, 2, 5), (3, 4, 2), (3, 5, 1)$		
$(0, 4, 8), (3, 2, 5), (3, 4, 2), \underline{(3, 5, 1)}$		
$(0, 4, 8), (3, 2, 5), (3, 4, 2), \underline{(5, 3, 1)}, (5, 2, 2), (5, 4, 2)$	Marquage de 5	Ajout de $(3, 5)$
$(0, 4, 8), (3, 2, 5), \underline{(3, 4, 2)}, (5, 2, 2), (5, 4, 2)$		
$(0, 4, 8), (3, 2, 5), (5, 2, 2), \underline{(5, 4, 2)}, (4, 0, 8), (4, 3, 2), (4, 5, 2)$	Marquage de 4	Ajout de $(3, 4)$
$(0, 4, 8), (3, 2, 5), (5, 2, 2), (4, 0, 8), (4, 3, 2), \underline{(4, 5, 2)}$		
$(0, 4, 8), (3, 2, 5), (5, 2, 2), (4, 0, 8), \underline{(4, 3, 2)}$		
$(0, 4, 8), (3, 2, 5), \underline{(5, 2, 2)}, (4, 0, 8)$		
$(0, 4, 8), (3, 2, 5), (4, 0, 8), \underline{(2, 1, 1)}, (2, 3, 5), (2, 5, 2)$	Marquage de 2	Ajout de $(5, 2)$
$(0, 4, 8), (3, 2, 5), (4, 0, 8), (2, 3, 5), (2, 5, 2), (1, 2, 1)$	Marquage de 1	Ajout de $(2, 1)$
Ensuite tout est défilé		

Les éléments soulignés sont ceux extraits de la file. Les ajouts des arêtes à E se font dès qu'on extrait un triplet (p, s, d) où s n'est pas visité (en effet, on a alors $p \in V$ et $s \notin V$).

6. Le coût des opérations dans une file implémentée par un tas min est logarithmiques en la taille de la file. Or la taille de F est majorée par $|A|$. Avec l'implémentation ci-dessus, on a un Whatever First Search avec des opérations en $O(\log|A|) = O(\log|S|)$ donc si le graphe est représenté par listes d'adjacence, on obtient une complexité en $O(|S| + |A|\log|S|)$.

Exercice 77 (**) *Détection de graphes bipartis*

On considère l'algorithme suivant, prenant en entrée un graphe non orienté connexe G et un sommet s_0 de G :

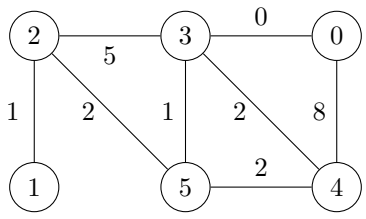
```
1  mystere(G, s_0) =
2  numéro(s_0) ← 1
3  Mettre (s_0, 1) dans un sac
4  tant que le sac n'est pas vide
5  |  Extraire un couple (s, c) du sac
6  |  pour tout voisin t de s
7  |  |  si numéro(t) = c alors
8  |  |  |  renvoyer faux
9  |  |  sinon si t n'a pas encore de numéro alors
10 |  |  |  numéro(t) ← 3 − c
11 |  |  |  Mettre (t, 3 − c) dans le sac
12 renvoyer vrai
```

1. Donner une spécification précise pour cet algorithme et justifier sa correction vis-à-vis de cette spécification.
2. En précisant la façon dont seraient implémentées les structures de données, déterminer sa complexité.
3. Comment ferait-on pour adapter cet algorithme aux graphes non connexes ?

1. Cet algorithme renvoie vrai si et suelement si le graphe en entrée est biparti. Cela vient de ce qu'il renvoie faux si et seulement si un cycle de longueur impaire existe dans le graphe (cf exercice *Caractérisation des graphes bipartis*).
2. Représentons le graphe $G = (V, E)$ par listes d'adjacence. On peut implémenter le sac via une file avec des opérations en $O(1)$ et gérer les numéros via un tableau T de taille $|V|$ pour pouvoir accéder au numéro d'un sommet en temps constant. Dans ces conditions, l'initialisation de T se fait en $O(|V|)$. De plus, chaque sommet passe au plus une fois dans la file grâce au marquage obtenu via les couleurs et pour chacun, on fait une extraction puis pour chacun de ses voisins, des comparaisons en temps constant et potentiellement une insertion ; d'où une complexité en $O(|V| + \sum_{s \in V} \deg(s)) = O(|S| + |E|)$.
3. Il suffit de lancer ce parcours modifié depuis chacun des sommets en l'interrompant si la couleur du sommet est déjà fixée.

Exercice 78 (*) *Évolution d'une structure union-find*

On considère le graphe G suivant :



Appliquer l'algorithme de Kruskal à G en indiquant à chaque étape de calcul l'évolution du tableau représentant la partition des sommets de G en supposant que ce dernier rend compte d'une structure union-find implémentée de manière arborescente naïve.

Une exécution possible peut être :

- Partition initiale représentée par $[-1, -1, -1, -1, -1, -1]$.
- Choix (obligatoire) de $(3, 0)$. Partition : $[3, -1, -1, -1, -1, -1]$.
- Choix de l'arête $(1, 2)$. Partition : $[3, 2, -1, -1, -1, -1]$.
- Choix (obligatoire) de $(3, 5)$. Partition : $[3, 2, -1, 5, -1, -1]$.
- Choix de $(2, 5)$. Partition : $[3, 2, 5, 5, -1, -1]$.
- Choix de $(5, 4)$. Partition : $[3, 2, 5, 5, -1, 4]$

Les unions ne sont pas forcées d'être des unions par hauteur dans cet exercice.

Exercice 79 (exo cours) *Modélisation et couplages*

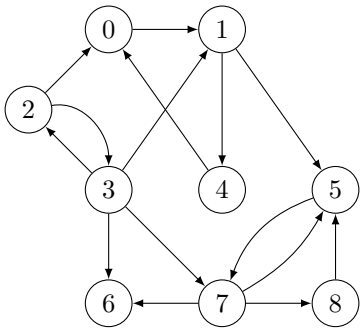
Une compagnie aérienne dispose de 5 avions a_1, a_2, a_3, a_4, a_5 qu'elle fait voler sur 5 lignes aériennes l_1, l_2, l_3, l_4, l_5. Tous les avions ne peuvent pas voler sur toutes les lignes, plus précisément : a_1 ne peut assurer que la liaison l_1, a_2 les liaisons l_1 et l_2, a_3 les liaisons l_1, l_3, l_4, l_5, a_4 les liaisons l_3, l_4, l_5 et a_5 les liaisons l_1, l_2.

1. Dessiner un graphe modélisant la situation.
2. Déterminer un couplage maximum dans ce graphe en expliquant l'algorithme utilisé et en déduire combien de liaisons la compagnie peut assurer simultanément au maximum. Le couplage trouvé est-il parfait ?

1. On modélise la situation à l'aide d'un graphe biparti en plaçant d'un côté 5 sommets correspondant aux avions et de l'autre 5 sommets correspondant aux liaisons. Une arête (a_i, l_j) symbolise le fait que a_i peut assurer l_j.
2. On utilise l'algorithme de Berge et on obtient un couplage maximum de taille 4 qui n'est pas parfait.

Exercice 80 (exo cours) *Parkour !*

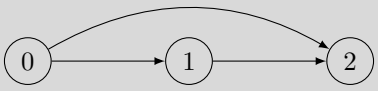
On considère le graphe G suivant :



1. Effectuez un parcours en profondeur de G en commençant par parcourir 0 et en calculant les dates de début et de fin de traitement de chaque sommet ainsi que les arbres de parcours.
2. Effectuez un parcours en largeur de G en commençant par parcourir 0 et en calculant les arbres de parcours.
3. Les arbres dans un parcours en largeur et dans un parcours en profondeur sont-ils toujours les mêmes ?

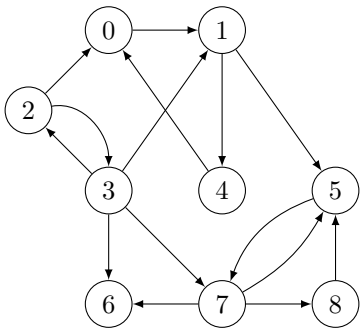
On devrait obtenir les mêmes arbres de parcours avec le parcours en largeur et le parcours en profondeur. Les ordres de visite des sommets peuvent varier selon les choix effectués lorsqu'il y a plusieurs voisins.

De manière générale, les arbres de parcours en largeur et en profondeur ne sont pas les mêmes. Par exemple, avec un parcours depuis le sommet 0 sur le graphe suivant, ils sont différents si on parcourt 1 en premier :



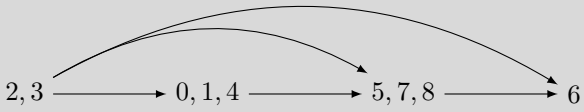
Exercice 81 (exo cours) *Algorithme de Kosaraju*

On considère le graphe G suivant :



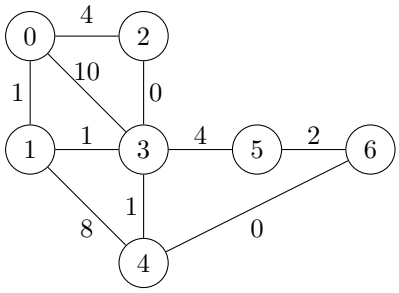
1. En calculer les composantes fortement connexes via l'algorithme de Kosaraju.
2. En déduire le graphe des composantes fortement connexes associé en classant les sommets dudit graphe dans un ordre topologique.

On devrait obtenir le graphe des composantes fortement connexes suivant :



Exercice 82 (exo cours) *Algorithme de Dijkstra*

On considère le graphe G suivant :



En utilisant l'algorithme de Dijkstra et en détaillant les étapes, déterminer les plus courts chemins d'origine 0 dans le graphe G ainsi que leurs poids respectifs.

Évolution de la file de priorité (si on l'implémente avec une opération de modification des priorités) et du tableau des distances (une distance n'évolue plus dès que le sommet correspondant est ôté de la file) :

Contenu de la file	$D[0]$	$D[1]$	$D[2]$	$D[3]$	$D[4]$	$D[5]$	$D[6]$
(0, _)	0	∞	∞	∞	∞	∞	
$(2,4),(3,10),(\mathbf{1},\mathbf{1})$	0	1	4	10	∞	∞	∞
$(2,4),(\mathbf{3},\mathbf{2}),(4,9)$		1	4	2	9	∞	∞
$(\mathbf{2},\mathbf{2}),(4,3),(5,6)$			2	2	3	6	∞
$(\mathbf{4},\mathbf{3}),(5,6)$			2		3	6	∞
$(5,6),(\mathbf{6},\mathbf{3})$					3	6	3
$(\mathbf{5},\mathbf{5})$						5	3
$\emptyset$						5	

Le dernier nombre dans la colonne $D[i]$ est la distance de 0 au sommet i.

Exercice 83 (exo cours) *Monotones et admissibles*

On considère un graphe $G = (V, E)$ pondéré par une fonction de pondération positive p.

1. Dans le cadre du calcul d'un plus court chemin de s à b dans G, rappeler la définition d'une heuristique admissible. Faire de même pour une heuristique monotone.

2. Montrer que toute heuristique h monotone et telle que $h(b) = 0$ est admissible.

1.
   - h est monotone si pour tout arc $(u, v) \in E$, $h(u) \leq p(u, v) + h(v)$.
   - h est admissible si pour tout $u \in V$, $h(u) \leq \delta(u, b)$.
2. On montre par récurrence sur n que pour tout $v \in V$ pour lequel le nombre d'arêtes dans un plus court chemin de v à b est n, $h(v) \leq \delta(v, b)$. OK pour $n = 0$ car $h(b) = 0$.

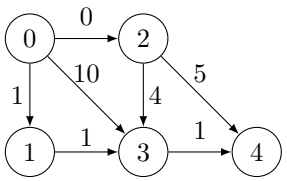
   Sinon, soit $v \to u_1 \to ... \to u_n \to b$ un plus court chemin de v à b de longueur $n + 1$. Alors :

   $$\begin{aligned} h(v) &\leq p(v, u_1) + h(u_1) \text{ car } h \text{ est monotone} \\ &= \delta(v, u_1) + h(u_1) \text{ car } v \to u_1 \text{ est un pcc de } v \text{ à } u_1 \\ &\leq \delta(v, u_1) + \delta(u_1, b) \text{ par hypothèse appliquée au pcc } u_1 \to ... \to b \\ &= \delta(v, b) \end{aligned}$$

   Donc on a $h(v) \leq \delta(v, b)$ pour tout v tel qu'on peut accéder à b depuis v et pour les autres l'inégalité tient puisque le membre droit est infini.

Exercice 84 (exo cours) *Algorithme A**

On considère le graphe G suivant :



1. On cherche à calculer un plus court chemin de 0 à 4 dans G.
   a) Donner pour ce faire une heuristique non nulle et monotone.
   b) Donner pour ce faire une heuristique non nulle, admissible mais pas monotone.
2. Sous quelles conditions sur l'heuristique h est-on sûr que A^* utilisant h renvoie un résultat correct ?
3. Appliquer A^* avec une heuristique adaptée pour déterminer le poids d'un plus court chemin de 0 à 4 dans G.

1.

Sommet	0	1	2	3	4
Heuristique pour a)	0	0	0	0	1
Heuristique pour b)	2	0	0	3	0

   La deuxième n'est pas monotone car $h(0) > h(1) + p(0, 1)$.
2. Si h est admissible, A^* est correct.
3. RAS, on utilise par exemple l'heuristique admissible de 1.

Exercice 85 (exo cours) *Lemme de Berge*

Soit $G = (V, E)$ un graphe non orienté et M un couplage dans G.

1. Montrer que, si M n'est pas maximum dans G, alors il existe un chemin M-augmentant dans G.
2. La réciproque de ce résultat est-elle vraie ?

1. Il existe un couplage M' de cardinal strictement supérieur à $|M|$. On considère le graphe $G' = (V, M \Delta M')$. Le degré d'un sommet dans ce graphe est inférieur à deux car M et M' sont des couplages. Les composantes connexes de ce graphe sont donc des sommets isolés, des cycles ou des chemins ; ces deux derniers étant forcément alternants.

   De plus $M \Delta M'$ contient une arête de plus dans M' que dans M et comme les cycles de G' consomment autant d'arêtes de M que de M' cette arête subsidiaire est sur un chemin alternant contenant une arête

de plus dans M' que dans M. Ce chemin convient.

2. Oui, dans ce cas $M\Delta$(le chemin) reste un couplage de cardinal un de plus que $|M|$.

Exercice 86 (exo cours) *Complexité d'union-find avec union par hauteur*

1. Rappeler le principe des opérations `trouver` et `unir` sur une structure union-find implémentée via une arborescence dans laquelle les unions sont faites par hauteur.
2. Montrer que dans ce contexte, tout arbre dans la partition de taille t et de hauteur h vérifie $t \geq 2^h$.
3. En déduire que la complexité pire cas des opérations `trouver` et `unir` avec une telle implémentation est en $O(\log n)$ où n est le cardinal de l'ensemble partitionné.

1. `trouver` revient à remonter l'unique chemin menant de l'argument x de `trouver` vers la racine de son arbre. `unir` revient à déterminer les racines des arbres de ses arguments x et y (via `trouver`), comparer leurs hauteurs et faire de l'arbre le moins haut le fils de l'arbre le plus haut.
2. On montre par induction que la hauteur h d'un arbre dans la partition et sa taille t vérifient $t \geq 2^h$. Pour $h = 0$ c'est vrai puisqu'alors l'arbre est une racine.

   De plus, lors d'une union d'arbres de hauteurs h_1 et h_2 respectivement :

   - Soit $h_1 \neq h_2$ et on peut loisiblement supposer que $h_1 > h_2$. L'arbre résultant de l'union a alors une hauteur $h = h_1$ et taille $t = t_1 + t_2$. Par hypothèse sur l'arbre 1 on a bien :

   $$t = t_1 + t_2 \geq t_1 \geq 2^{h_1} = 2^h$$

   - Soit $h_1 = h_2$ et alors l'arbre résultant de l'union a hauteur $h = h_1 + 1$ et taille alors :

   $$t = t_1 + t_2 \geq 2^{h_1} + 2^{h_2} = 2^{h_1} \times 2 = 2^{h_1+1} = 2^h$$

   Comme `trouver` et `unir` sont de complexité linéaire en la hauteur des arbres considérés, que ces dernières sont majorées par le logarithme de leurs tailles par ce qui précède et que cette taille est bornée par n, on conclut.

Exercice 87 (*) *Couverture par dominos*

On considère un échiquier de taille $n \times n$ contenant des pièces sur certaines cases. On souhaite savoir s'il est possible de recouvrir toutes les cases inoccupées de l'échiquier à l'aide de dominos de longueur 2 et largeur 1.

En modélisant le problème par un graphe biparti qu'on explicitera, décrire un algorithme permettant de répondre à cette question.

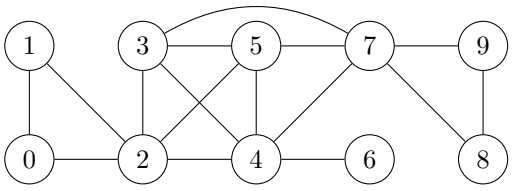
On considère le graphe G dont les sommets sont les cases libres et $\{u, v\}$ est une arête ssi u et v sont des cases voisines de l'échiquier. Ce graphe est biparti (bipartition en cases noires / cases blanches).

On calcule un couplage maximum dans G via l'algorithme de Berge. Il est parfait si et seulement si le problème admet une solution (et quand c'est le cas, le couplage indique comment placer les dominos).

Exercice 88 (***) *Sommets coupants*

Soit $G = (V, E)$ un graphe non orienté connexe. On dit qu'un sommet s est coupant si le graphe induit par la suppression du sommet s dans G n'est plus connexe.

1. Déterminer les sommets coupants du graphe suivant :



2. Décrire un algorithme qui à partir de G et s détermine si s est un sommet coupant de G. En déduire un algorithme naïf qui détermine tous les sommets coupants d'un graphe G et estimer sa complexité.

On note A l'arbre de parcours en profondeur de G depuis un sommet quelconque s.

3. Montrer que si un noeud de A a pour fils u et v alors il n'existe aucune arête dans G entre un descendant de u et un descendant de v.
4. Montrer que le sommet s est coupant si et seulement si s a au moins deux fils dans A.
5. Montrer que $t \neq s$ est un sommet coupant de G si et seulement si t possède un fils u tel qu'aucun descendant de u dans A n'est voisin dans G d'un ancêtre strict de t dans A.
6. Proposer un algorithme permettant de déterminer tous les sommets coupants de $|G|$ en s'appuyant sur les questions précédentes. Déterminer sa complexité.

1. $2, 4, 7$.
2. Retirer s et faire un parcours en $O(|V| + |E|)$ pour déterminer le nombre de composantes connexes après suppression permet de savoir si s est coupant. Déterminer les sommets coupants se fait naïvement en $O(|V|(|V| + |E|)) = O(|V||E|)$ puisque G est connexe.
3. Découle du principe d'un parcours en profondeur. Si une telle arête (x, y) entre un descendant x de u et un descendant y de v, sans perte de généralité u est exploré avant v lors du parcours de s et y aurait du être un descendant de u.
4. Si s a au moins deux fils u et v, il n'y a pas d'arête entre un descendant de u et un de v (par 3) donc tout chemin entre un descendant de u et un de v passe par s : s est coupant.

   Si s est coupant, il ne peut pas avoir 0 ou 1 fils sinon tout sommet de $T \setminus \{s\}$ continue d'être relié à tous les autres via les arêtes de T et donc retirer s laisse le graphe connexe.
5. ($\Rightarrow$) Par l'absurde, supposons que tout fils de t a un descendant voisin d'un ancêtre strict de t. Alors pour tous $u, v \in V \setminus \{t\}$, il y a un chemin de u à v qui ne passe pas par t ce qui contredit son caractère coupant. On procède par disjonction de cas en faisant des dessins :
   - ni u ni v n'est descendant de t.
   - u est descendant de t mais pas v (et inversement).
   - u et v sont descendants d'un même fils de t.
   - u et v sont descendants de t mais pas du même fils de t.

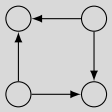
   ($\Leftarrow$) Dans ces conditions, soit x un descendant de u et montrons qu'il n'y a pas de chemin de x à la racine s qui ne passe pas par t. Déjà (x, s) n'existe pas sinon on aurait un descendant de u voisin d'un ancêtre strict de t. De plus un chemin de x à s doit passer par un sommet non descendant de u mais la seule arête permettant de passer d'un descendant de u à un non descendant de u est (u, t). Supprimer t déconnecte donc x et s.
6. On fait un parcours et pour chaque sommet différent de la racine on garde en mémoire l'ancêtre le plus haut auquel il est relié au fil du parcours. Ceux pour qui aucun fils n'a d'arc arrière vers un acêtre sont les sommets coupants différents de la racine. On traite la racine à part en vérifiant si elle a deux fils ou pas. Le coût de calcul de tous les sommets coupants devient celui d'un parcours, en $O(|V| + |E|)$.

Exercice 89 (**) *Semi-connexité*

Soit $G = (S, A)$ un graphe orienté. On dit que G est semi-connexe si pour tous $(s, t) \in S$ il existe un chemin de s à t ou il existe un chemin de t à s.

1. Donner un exemple de graphe orienté à 4 sommets, sans cycle, non semi-connexe et tel que le graphe obtenu en en supprimant les orientations est connexe.
2. Si G est acyclique, on peut considérer ses sommets dans un ordre topologique $(s_0, ..., s_{n-1})$. Montrer que G est semi-connexe si et seulement si pour tout $i \in [\![0, n-2]\!]$, $(s_i, s_{i+1}) \in A$. En déduire la complexité d'un algorithme qui teste la semi-connexité d'un graphe acyclique.
3. Proposer un algorithme qui décide si un graphe orienté est semi-connexe.

1. Ce graphe convient :



2. ($\Rightarrow$) par contraposée. S'il existe i tel que $(s_i, s_{i+1}) \notin A$ alors il n'y a pas de chemin de s_i à s_{i+1} car un tel chemin devrait passer par s_j avec $j > i+1$ ce qui est aburde car s_{i+1} est avant s_j dans un ordre topologique. Comme s_{i+1} est après s_i dans un ordre topologique, il n'y a pas de chemin de s_{i+1} à s_i non plus.

   ($\Leftarrow$) Dans ce cas, $s_0 \to s_1 \to ... \to s_{n-1}$ est un chemin garantissant la semi-connexité.

   Algorithme : calculer un ordre topologique en $O(|S| + |A|) + \forall i$ tester si $(s_i, s_{i+1}) \in A$ en $O(|A|)$.

3. G est semi-connexe si et seulement si le graphe G_{CFC} des composantes fortements connexes de G l'est donc on calcule G_{CFC} via Kosaraju en $O(|S| + |A|)$ puis on se ramène à 2.

Exercice 90 (***) *Un seul arc*

Soit $G = (S, A)$ un graphe orienté. On suppose que pour tout $(u, v) \in S^2$, il y a exactement un arc parmi (s, t) et (t, s) dans A.

1. Montrer qu'il existe un chemin dans G qui passe une et une seule fois par chaque sommet.
2. Déterminer un algorithme pour trouver un tel chemin et sa complexité.
3. Montrer que ce chemin est unique si et seulement si G ne possède pas de cycle de taille 3.

1. On le montre par récurrence sur le nombre n de sommets du graphe. OK si $n = 1$ ou 2. Soit G un graphe à $n+1$ sommets. Alors par hypothèse, il existe un chemin $(s_0, ..., s_{n-1})$ passant une et une seule fois par chaque sommet de $\{s_0, ..., s_{n-1}\}$. Notons i_0 le plus petit i tel que $(s_n, s_i) \in A$:
   - S'il n'y en a pas, alors en particulier $(s_{n-1}, s_n) \in A$ et permet de prolonger le chemin.
   - Sinon, pour tout $i < i_0$, c'est (s_i, s_n) qui appartient à A parmi les deux arcs possibles et alors $(s_0, ..., s_{i_0-1}, s_n, s_{i_0}, ..., s_{n-1})$ est un chemin convenable.
2. On suit le procédé de la question 1 : $C \leftarrow [0]$ et pour tout $i \in [\![1, n-1]\!]$, on insère le sommet i dans C avant la première valeur t telle que $(i, t) \in A$. Se fait en $O(|S|^2)$ si le test d'existence d'une arête est constant, ce qui est possible si G est représenté par matrice d'adjacence.
3. ($\Rightarrow$) Par l'absurde, (u, v, w) est un cycle de taille 3. Alors en insérant les autres sommets dans (u, v, w) puis dans (v, w, u) via l'algo de Q2, on obtient deux chemins différents.

   ($\Leftarrow$) Par contraposée, supposons qu'il y a deux chemins $u = (u_1, ..., u_{n-1})$ et $v = (v_1, ..., v_{n-1})$ passant exactement une fois par tous les sommets et construisons un 3-cycle dans G. Comme u et v sont distincts, on peut considérer $i \in [\![0, n-1]\!]$ le plus petit indice où u_i et v_i diffèrent. En fait, $i < n-2$ car :
   - Si $i = n-1$, u et v coincident sauf sur le dernier sommet ce qui est impossible.
   - Si $i = n-2$, $u_{n-2} = v_{n-1}$ et $u_{n-1} = v_{n-2}$ donc (u_{n-2}, u_{n-1}) et (u_{n-1}, u_{n-2}) sont des arêtes.

   Alors, u_{i+2} existe et si $(u_{i+2}, u_i) \in A$ alors (u_i, u_{i+1}, u_{i+2}) est un 3-cycle. Sinon, c'est (u_i, u_{i+2}) qui existe et si $(u_{i+3}, u_i) \in A$, (u_i, u_{i+2}, u_{i+3}) est un 3-cycle. On poursuit ce raisonnement jusque soit avoir trouvé un 3-cycle, soit avoir montré que $(u_i, u_j) \in A$ pour tout $j > i$. Le même raisonnement s'applique au chemin v.

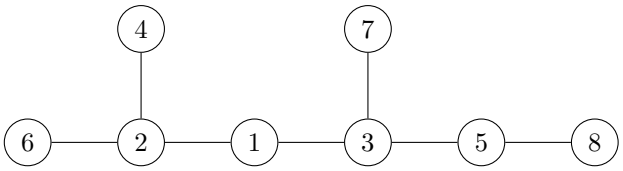
   Or, il n'est pas possible que pour tout $j > i$ $(u_i, u_j) \in A$ et $(v_i, v_j) \in A$. Comme $u_i \neq v_i$ on sait qu'au delà de i, le sommet u_i doit apparaître dans v et le sommet v_i doit apparaître dans u. Donc il existe $j > i$ et $k > i$ tel que $u_j = v_i$ et $v_k = u_i$. Le fait que $(u_i, u_j) \in A$ se réécrit $(v_k, v_i) \in A$ avec $k > i$ et donc on a à la fois l'arête (v_i, v_k) et l'arête (v_k, v_i) dans G : contradiction.

Exercice 91 (**) *Codage de Prüfer*

On considère un arbre $G = (S, A)$ dont les sommets sont numérotés de 1 à $n \geq 2$. Le codage de Prüfer de cet arbre est une suite finie de $n-2$ entiers de l'intervalle $[\![1, n]\!]$ et on le calcule via l'algorithme suivant :

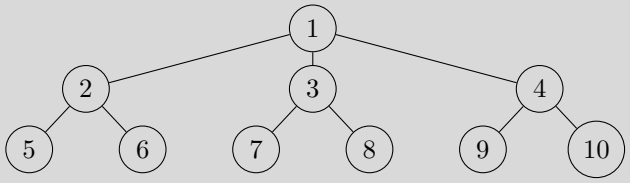
```
1 codage(G = (S, A)) =
2 L ← []
3 tant que |S| > 2
4 |   i ← min{s ∈ S | deg(s) = 1}
5 |   j ← unique sommet de S auquel est relié i dans G
6 |   Concaténer j en fin de L
7 |   S ← S \ {i}
8 |   A ← A \ {(i, j)}
9 renvoyer L
```

1. Expliquer pourquoi la ligne 4 de cet algorithme ne pose pas problème. Montrer sa terminaison. Le codage de Prüfer d'un arbre est-il unique ?
2. Déterminer un codage de Prüfer pour l'arbre suivant :



3. On admet pour cette question qu'il est possible de reconstruire l'arbre correspondant à un codage de Prüfer uniquement à partir de ce dernier. Déterminer le nombre d'arbres à n noeuds numérotés de 1 à n différents qu'il est possible de construire.
4. À quel arbre correspond le code $(2, 2, 1, 3, 3, 1, 4, 4)$?
5. Concevoir un algorithme permettant de reconstruire l'arbre correspondant à un codage de Prüfer donné.

1. L'algo termine car dans un arbre il existe toujours un somet de degré 1 ce qui assure qu'on pourra trouver un i convenable et le retirer à chaque itération faisant strictement décroître l'entier $|S|$. Le codage est unique.
2. $(2, 2, 1, 3, 3, 5)$.
3. Il y a bijection entre les arbres à n noeuds étiquetés par $[\![1, n]\!]$ et les suites à $n-2$ éléments de $[\![1, n]\!]$ via le codage de Prüfer d'où n^{n-2} arbres convenables.
4. On obtient :



5. L'ensemble des noeuds est $S = [\![1, |L| + 2]\!]$ où L est la liste donnant le codage. Puis on en calcule les arêtes via :

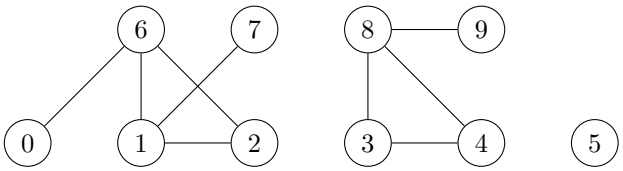
```
A ← ∅
tant que |S| > 2
|   i ← min{j ∈ S | j ∉ L}
|   A ← A ∪ {(i, tête(L))}
|   L ← queue(L)
|   S ← S \ {i}
A ← A ∪ {(a, b)} où S = {a, b}
```

Le principe est que quand on lit un élément v de la liste, il faut rajouter une arête de la forme (u, v) où u a été supprimé et est le plus petit qui était supprimable. Si un sommet apparaît plus loin dans L, c'est qu'il n'a pas été supprimé au moment de l'ajout de (u, v), donc u est nécessairement le plus petit sommet non encore utilisé qui n'apparaît pas dans la suite de L.

Exercice 92 (**) *Nombre cyclomatique*

Si $G = (S, A)$ est un graphe non orienté à k composantes connexes, le nombre cyclomatique de G, est l'entier naturel $c(G) = |A| - |S| + k$.

1. Déterminer le nombre cyclomatique du graphe ci-dessous :



2. Déterminer le nombre cyclomatique d'un arbre.
3. Que peut-on dire du nombre cyclomatique d'un graphe acyclique ? Prouvez le.
4. Montrer que si $c(G) = 0$ alors il existe un sommet de G de degré inférieur ou égal à un.
5. La réciproque de la propriété montrée à la question 3 est-elle vraie ?

1. $c(G) = 2$.
2. Pour un arbre, $|A| = |S| - 1$ et $k = 1$ donc $c(G) = 0$.
3. Si G est acyclique, ses k composantes connexes sont des arbres $G_i = (S_i, A_i)$ et donc :

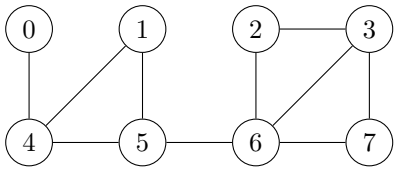
$$c(G) = |A| - |S| + k = \sum_{i=1}^{k} |A_i| - \sum_{i=1}^{k} |S_i| + \sum_{i=1}^{k} 1 = \sum_{i=1}^{k} \underbrace{(|A_i| - |S_i| + 1)}_{=0 \text{ par q2}} = 0$$

4. Sinon pour tout $s \in S$, $\deg(s) \geq 2$ et alors $2|A| = \sum_{s \in S} \deg(s) \geq 2|S|$ puis $c(G) \geq k > 0$.
5. La réciproque est vraie. On montre par récurrence sur $n \in \mathbb{N}^\star$ que tout graphe à n sommets vérifiant $c(G) = 0$ est acyclique. C'est vrai si $n = 1$. Si G est d'ordre $n + 1$ et $c(G) = 0$, retirons à G un de ses sommets s de degré un (qui existe par 4) pour obtenir G' à n sommets :
   - Si $\deg(s) = 0$, k diminue de un et $|S|$ aussi donc $c(G') = 0$. Par hypothèse de récurrence G' est donc acyclique et G aussi.
   - Sinon, k reste constant et $|S|$ et $|A|$ décroissent de un donc $c(G') = 0$ et rebelote.

Exercice 93 (*) *Points d'articulation*

Soit $G = (S, A)$ un graphe non orienté. Un sommet $s \in S$ s'appelle un point d'articulation de G si le graphe induit par la suppression de s dans G (c'est à dire le graphe dans lequel on supprime s et toute les arêtes incidentes à s) a plus de composantes connexes que G.

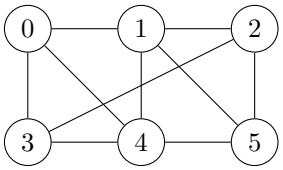
1. Quels sont les points d'articulation dans le graphe suivant ?



2. Montrer qu'un graphe complet (à plus de deux sommets) n'a pas de point d'articulation.

On suppose à présent que $G = (S, A)$ est connexe. Un ensemble d'articulation est un sous ensemble S' de S si le graphe induit par la suppression de tous les sommets de S' n'est plus connexe.

3. Déterminer un ensemble d'articulation le plus petit possible pour le graphe suivant.



4. Montrer que tout graphe connexe non complet admet un ensemble d'articulation.

On note $\kappa(G)$ le nombre minimal de sommets que l'on doit enlever à G pour obtenir soit un graphe non connexe, soit un graphe à un seul sommet.

5. Montrer que $\kappa(G) = n - 1$ si et seulement si G est le graphe complet à n sommets.

1. $4, 5, 6$.
2. Par récurrence sur le nombre de sommets.
3. L'ensemble $\{1, 2, 4\}$ convient.
4. Comme G n'est pas complet, il existe v, w tels que (v, w) n'est pas une arête du graphe. En supprimant tous les sommets sauf v et w on obtient un graphe non connexe.
5. Sens direct par récurrence sur le nombre de sommets. Pour le sens réciproque, si G n'est pas complet, $\kappa(G) \leq n - 2$ d'après la question 4 ce qui conclut.

Exercice 94 (***) *Théorème de Mantel*

1. Montrer que si x et y sont deux réels alors $\left(\frac{x+y}{2}\right)^2 \geq xy$ avec égalité si et seulement si $x = y$.

On appelle *ensemble de sommets indépendants* dans un graphe G un sous ensemble I des sommets de G tel que pour tout $(u, v) \in I^2$, les sommets u et v ne sont pas reliés par une arête dans G. On considère dans la suite un graphe $G = (S, A)$ simple, non orienté et ne contenant aucun cycle de longueur 3 et I un ensemble de sommets indépendants dans G de taille maximale. On note x cette taille maximale.

2. Montrer que pour tout sommet $s \in S$, $\deg(s) \leq x$.
3. Montrer que $|A| \leq \sum_{s \in S \setminus I} \deg(s)$.
4. Soit G un graphe simple non orienté à n sommets ne contenant pas de triangle. Déduire des questions précédentes que G a moins de $n^2/4$ arêtes. Montrer que l'égalité est atteinte si et seulement si n est pair et G est le graphe biparti complet avec $n/2$ sommets dans chaque ensemble de la bipartition.

Remarque : Le théorème de Turán donne une borne supérieure sur le nombre d'arêtes d'un graphe ne contenant pas de sous graphe complet à plus de k sommets. Le cas particulier où $k = 2$, montré en question 4, est le théorème de Mantel.

1. Revient à montrer que $\left(\frac{x-y}{2}\right)^2 \geq 0$ avec égalité si et seulement si $x = y$.
2. Soit $s \in S$. Deux voisins de s ne peuvent pas être adjacents sinon il y aurait un cycle de longueur 3 donc $\{$voisins de $s\}$ est un ensemble indépendant dans G donc son cardinal $\deg(s)$ est inférieur au cardinal maximal d'un ensemble indépendant c'est-à-dire x.
3. Chaque arête qui a une extrémité dans I a son autre extrémité dans $S \setminus I$ sinon I n'est pas indépendant. On en déduit que $\sum_{s \in I} \deg(s) \leq |A|$. Alors,

$$2|A| = \sum_{s \in S} \deg s = \sum_{s \in I} \deg s + \sum_{s \in S \setminus I} \deg s \leq |A| + \sum_{s \in S \setminus I} \deg s$$

   et on conclut en soustrayant $|A|$ des deux côtés.
4. La première inégalité ci-dessous vient de 3 ; la seconde de 2 ; la dernière de 1 :

$$|A| \leq \sum_{s \in S \setminus I} \deg(s) \leq x|S \setminus I| = x(|S| - x) \leq \left(\frac{x + (|S| - x)}{2}\right)^2 = \frac{|S|^2}{4}$$

   Si il y a égalité, toutes les inégalités ci-dessus sont des égalités, en particulier $x = |S| - x$ (donc $|S|$ est pair) et tout sommet est de degré $|S|/2$. Alors il existe un ensemble indépendant I dans G de taille $x = |S|/2$ et $G = K_{|S|/2, |S|/2}$ (la bipartition des sommets est $I \sqcup (S \setminus I)$). Réciproquement, si G est le graphe biparti complet à $|S|/2$ sommets, l'égalité $|A| = |S|^2/4$ est vraie.

Exercice 95 (***) *Deuxième arbre couvrant minimal*

Soit $G = (S, A)$ un graphe non orienté pondéré par p et connexe et $T = (S, B)$ un arbre couvrant de G.

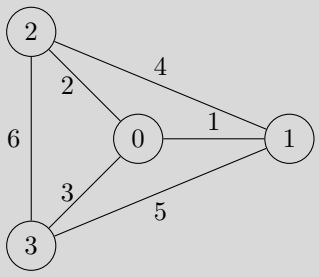
1. Montrer que T est un arbre couvrant minimal si et seulement si pour toute arête $a \in A \setminus B$, a est de poids maximal dans le cycle créé en ajoutant a aux arêtes de T.

On considère dans la suite que la fonction de pondération p sur G est injective.

2. Montrer que dans ce cas, G admet un unique arbre couvrant minimal. On le note T_1.
3. Notons T_2 le deuxième arbre couvrant de G ayant le plus petit poids après T_1.
   a) Montrer que T_2 et T_1 ont toutes leurs arêtes en commun sauf 2.
   b) En déduire un algorithme pour calculer T_2 et sa complexité.
   c) Y a-t-il unicité de l'arbre T_2 ?

1. ($\Rightarrow$) Si on avait une arête $a \in A \setminus B$ qui n'est pas de poids maximal dans son cycle alors en la rajoutant et en supprimant une arête de poids plus grand, on contedit la minimalité de T.

   ($\Leftarrow$) On trie les arêtes par ordre de poids croissant en rejetant les arêtes de $A \setminus B$ à la fin du cycle qu'elles crééent. Alors appliquer Kruskal en prenant les arêtes dans cet ordre produit T, qui est donc minimal par correction de l'algorithme de Kruskal.
2. Si on en a deux différents T et T', on prend a la plus petite arête de $T \Delta T'$. On peut supposer que $a \in T \setminus T'$. La rajouter à T' fait un cycle qui contient une arête $b \in T' \setminus T$, sinon il y avait un cycle dans l'arbre T. Par construction de a et injectivité de p, $p(a) < p(b)$. Mézalors, ajouter a dans T' et lui retirer b produit un arbre couvrant de poids strictment plus petit que celui de T' qui était minimal : contradiction.
3. a) Si $|T_1 \Delta T_2| > 2$, on montre de manière similaire à q2 qu'on peut construire un arbre de poids strictement entre celui de T_1 et celui de T_2. Cette quantité ne pas être nulle par unicité de T_1, ni égale à 1 ce qui conclut.
   b) Le graphe est connexe donc $|S| \leq |A|$. On calcule T_1 via Kruskal. Ensuite deux options :
      - Pour tout $a \in A$, la supprimer et refaire Kruskal dans G. Le plus petit arbre ainsi construit est T_2. Complexité en $O(|A|^2 \log |S|)$ (union-find avec union par hauteur).
      - Pour tout $a \in A$, la rajouter à T, en faire un parcours pour trouver l'unique cycle ainsi créé et en supprimer l'arête de plus gros poids (hormis a). Le plus petit arbre ainsi construit est T_2. Complexité en $O(|A|(|A| + |S|)) = O(|A|^2)$.
   c) Non, contre-exemple :



   L'arbre couvrant minimal est de poids 6, il n'y a aucun arbre couvrant de poids 7 et deux arbres couvrants de poids 8 : $((2,0),(0,1),(1,3))$ et $((0,3),(0,1),(1,2))$.

Exercice 96 (**) *Complexités d'unions*

On représente une structure union-find de manière arborescente naïve via la structure suivante :

```
type partition = int array
```

Si `p` est de type `partition`, `p` est un tableau de taille n (où n est le cardinal de l'ensemble partitionné) dans lequel la case i contient le père de l'élément i dans son arbre. Le représentant d'un élément est la racine de son arbre et les fusions de morceaux de partitions se font de manière arbitraire.

1. Écrire une fonction `creer_partition` renvoyant une nouvelle partition formée de singletons.
2. Écrire une fonction `trouver` prenant en entrée une partition et un entier (supposé appartenir à l'ensemble) et renvoyant son représentant.

3. Écrire une fonction `fusionner` qui fusionne les ensembles de deux éléments donnés.
4. Quelle est la complexité pire cas d'une fusion avec cette implémentation ?

On souhaite à présent étudier la complexité *amortie* d'une fusion. Il s'agit de C/n où C est le pire coût total de n fusions consécutives et n est le nombre d'éléments dans l'ensemble.

5. Montrer que la complexité amortie d'une fusion au pire cas est en fait la même que la complexité pire cas. *Indication : trouver une suite de n fusions coûteuse.*

Pour diminuer la complexité de l'opération de fusion, on souhaite unir les arbres par hauteur plutôt que de manière arbitraire. Pour ce faire, on modifie le type `partition` en :

```
type partition = {parents : int array; hauteurs : int array}
```

6. Réécrire dans ce contexte les fonctions `creer_partition` et `fusionner`.
7. Quelle est la complexité de `fusionner` avec ce choix d'implémentation ?

1. Il suffit de créer un tableau qui contient i en case i :

   ```
   let creer_partition (n:int) :partition =
       Array.init n (fun i -> i)
   ```

   On peut aussi choisir de signaler que i est racine par la présence de -1 dans la case i mais dans ce cas il faudra modifier un peu le code pour `trouver`.

2. On remonte à la racine :

   ```
   let rec trouver (p:partition) (i:int) :int =
       if p.(i) = i then i
       else trouver p p.(i)
   ```

3. On détermine les deux racines et on fait de l'une la fille de l'autre :

   ```
   let fusionner (p:partition) (i:int) (j:int) :unit =
       let ri = trouver p i in
       let rj = trouver p j in
       if ri <> rj then p.(rj) <- ri
   ```

   La condition `ri <> rj` est en fait inutile car si les deux sont égaux, on remplacerait une valeur par elle même avec notre convention pour repérer une racine.

4. Le coût de la fusion est le même que celui d'un `trouver` qui est linéaire en la hauteur de l'arbre remonté. Au pire cet arbre est filiforme et contient tous les éléments (sauf 1) donc est de hauteur $\Theta(n)$.

5. On considère la séquences de fusions suivantes : `fusionner p 0 1`, `fusionner 1 2`, `fusionner 2 3`, ..., `fusionner (n-2) (n-1)`. Après i fusions, la forêt est constituée de $n-i$ arbres singletons et d'un arbre filiforme $0 \leftarrow 1 \leftarrow 2 \leftarrow \ldots \leftarrow i$ donc la i-ème fusion se fait en temps proportionnel à i. Le coût cumulé des n fusions est donc de l'ordre de $\sum_{i=1}^{n} i = O(n^2)$ et donc le coût amorti d'une fusion dans cette séquence est en $O(n)$ au pire cas. Le coût amorti pire cas ne peut pas être plus grand que le coût pire cas et on vient d'exhiber une séquence de fusions pour laquelle il y a atteinte.

6. Pour la création, on initialise toutes les hauteurs à zéro (hauteur d'une racine) :

   ```
   let creer_partition (n:int) :partition =
       {parents = Array.init n (fun i -> i);
        hauteurs = Array.make n 0}
   ```

   On fusionne de sorte à faire de l'arbre le moins haut un fils de l'arbre le plus haut. Quand les hauteurs sont égales, on fusionne arbitrairement en modifiant la hauteur de la nouvelle racine.

```
let fusionner (p:partition) (i:int) (j:int) :unit =
    let ri = trouver p i in
    let rj = trouver p j in
    let hi = p.hauteurs.(ri) in
    let hj = p.hauteurs.(rj) in
    if hi < hj then p.parents.(ri) <- rj
    else if hi > hj then parents.(rj) <- ri
    else if ri <> rj then (p.parents.(ri) <- rj;
                           p.hauteurs.(rj) <- p.hauteurs.(rj) +1)
```

7. On obtient une complexité pour `trouver` donc pour `fusionner` en $O(\log n)$.

Exercice 97 (**) *Poids des arêtes d'un arbre couvrant*

Soit $G = (S, A, f)$ un graphe connexe non orienté et pondéré par f. On note $n = |S|$. Soit $T^* = (S, B^*)$ un arbre couvrant minimal de G. On note $a_1, ..., a_{n-1}$ les arêtes de B^* triées par ordre croissant de poids. Soit enfin $T = (S, B)$ un arbre couvrant de G (pas forcément minimal). On note $b_1, ..., b_{n-1}$ les arêtes de B triées par ordre croissant de poids. On veut montrer que pour tout $i \in [\![1, n-1]\!]$, $f(a_i) \leq f(b_i)$.

1. Dans cette question, on se place dans le cas particulier où T est en fait un arbre couvrant minimal. On suppose par l'absurde qu'il existe $i \in [\![1, n-1]\!]$ tel que $f(a_i) > f(b_i)$.
   a) Justifier que $(S, B \setminus \{b_i\})$ est un graphe a deux composantes connexes C_1 et C_2.
   b) Montrer que, si $a \in B^*$ est une arête entre un sommet de C_1 et un de C_2, alors $f(a) \geq f(b_i)$.
   c) En déduire une contradiction.
2. Supposons à présent que T n'est plus nécessairement minimal. Montrer que pour tout $i \in [\![1, n-1]\!]$, il existe une arête $b \in \{b_1, b_2, ..., b_i\}$ telle que $(S, \{a_1, a_2, ..., a_{i-1}, b\})$ est sans cycle.
3. Conclure quant à l'objectif annoncé en début d'énoncé.

1. a) T est un arbre donc connexe minimal : lui retirer une arête produit donc deux composantes.
   b) Par l'absurde, s'il existe $a \in B^*$ telle que $f(a) < f(b_i)$ alors $T' = (S, B \cup \{a\} \setminus \{b_i\})$ est un arbre couvrant de poids strictement plus petit que T qui est minimal : contradiction.
   c) Prenons le plus petit i tel que $f(a_i) > f(b_i)$. On continue de noter C_1 et C_2 les composantes créées par le retrait de b_i dans T. Il existe nécessairement une arête $a_k \in A^*$ qui traverse entre C_1 et C_2 et d'après Q1b, $f(a_k) \geq f(b_i)$. Mais cette inégalité n'est pas stricte, sinon en retirant a_k de T^* et en y ajoutant b_i on obtiendrait un arbre de poids strictement plus petit que T^*, qui est minimal. Donc $f(a_k) = f(b_i)$. Comment peut se situer k par rapport à i ?
      - Si $k < i$, comme les arêtes de T^* sont triées par ordre croissant, on a $f(a_k) \leq f(a_i)$ donc $f(b_i) = f(a_k) \leq f(a_i)$ ce qui contredit l'hypothèse.
      - Si $k > i$, on aurait $f(a_k) = f(b_i) < f(a_i)$ ce qui contredit le fait que les arêtes de T^* sont triées dans l'ordre croissant.

      Ainsi, on a nécessairement $k = i$ donc $f(a_i) = f(b_i)$ et ça contredit $f(a_i) > f(b_i)$.
2. Notons $E = \{a_1, ..., a_{i-1}\}$ et $F = \{b_i, ..., b_i\}$. Supposons par l'absurde que $(S, E \cup \{b\})$ contient un cycle pour tout $b \in F$. Notons $C_1, ..., C_j$ les composantes connexes de (S, E). Notons pour chaque composante $F_j = \{b \in F \mid b \text{ est une arête entre deux sommets de } C_j\}$ et $E_j = \{a \in E \mid a \text{ intervient dans } C_j\}$.

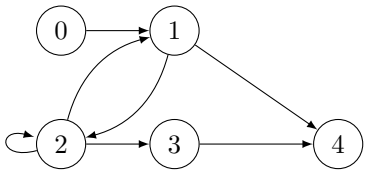
   Puisque $(S, E \cup \{b\})$ contient un cycle pour tout $b \in F$, cela veut dire que les arêtes b relient deux sommets d'une même composante C_j. Ainsi, les a_k sont des arêtes reliant deux sommets d'une même composante (il y en a $i-1$) et les b_k sont aussi des arêtes reliant deux sommets d'une même composante (il y en a i) donc par principe des tiroirs, il existe une composante j telle que $|F_j| > |E_j|$. Or, (C_j, E_j) est un arbre (on restreint un arbre couvrant) donc $|C_j| - 1 = |E_j| < |F_j|$. Ainsi, (C_j, F_j) contient un cycle mézalors T aussi ce qui contredit que T est un arbre.
3. Soit T_K un arbre couvrant minimal de G calculé par l'algorithme de Kruskal et $a'_i, ..., a'_{n-1}$ ses arêtes triées par ordre croissant de poids. D'après le fonctionnement de cet algorithme, au moment de compléter la forêt formée par $\{a_1, ..., a'_{i-1}\}$, on ajoute l'arête de plus petit poids qui ne créé pas de cycle

donc d'après Q2, $f(a'_i) \leq f(b) \leq f(b_i)$ (puisque les b_k sont triés par ordre croissant). Or, T^* et T_K sont minimaux, donc Q1 assure aussi que $f(a_i) \leq f(a'_i)$. Donc $f(a_i) \leq f(b_i)$.

Exercice 98 (**) *Calcul dynamique de clôture transitive*

La fermeture transitive d'un graphe $G = (S, A)$ orienté est le graphe $G' = (S, A')$ dans lequel $(u, v) \in A'$ si et seulement si il existe un chemin de u vers v dans G. Dans toute la suite, on considère un graphe G dont les sommets sont $[\![1, n]\!]$ et représenté par matrice d'adjacence.

1. Dessiner la fermeture transitive du graphe ci-dessous.



2. Décrire dans les grandes lignes un algorithme simple qui calcule la fermeture transitive de G.
3. Quelle est la complexité de l'algorithme proposé ?

On se propose à présent de calculer la fermeture transitive de G à n sommets par programmation dynamique en utilisant l'algorithme $\mathcal{A}$ suivant :

```
C_0 ← matrice d'adjacence de G
pour k allant de 1 à n
    Initialiser une matrice C_k de taille n × n
    pour i allant de 1 à n
        pour j allant de 1 à n
            C_k(i,j) ← C_{k-1}(i,j) ∨ (C_{k-1}(i,k) ∧ C_{k-1}(k,j))
renvoyer C_n
```

4. À quel algorithme du cours, l'algorithme $\mathcal{A}$ ressemble-t-il ?
5. Montrer que pour tout $k \in [\![0, n]\!]$ et tous $i, j \in [\![1, n]\!]$, le coefficient (i, j) de C_k vaut vrai s'il existe un chemin de i à j dans G dont les sommets intermédiaires sont tous inférieurs ou égaux à k et faux sinon. En déduire la correction de cet algorithme.
6. Quelle est la complexité temporelle de $\mathcal{A}$? Et sa complexité spatiale ?
7. Expliquer comment on pourrait diminuer la complexité spatiale précédemment trouvée.

1. RAS.
2. Pour chaque sommet, on calcule les accessibles avec un parcours et on ajoute les arcs correspondants.
3. Un parcours coûte $O(n^2)$ (représentation par matrice d'adjacence) donc $O(n^3)$.
4. C'est le même principe que l'algorithme de Floyd-Warshall pour le calcul de plus courts chemins.
5. Par récurrence sur k. C'est vrai initialement par définition d'une matrice d'adjacence. L'hérédité repose sur le fait que s'il existe un chemin de i à j ne passant que par des sommets inférieurs à $k+1$:
   - Soit il ne fait pas intervenir $k+1$, auquel cas c'est un chemin de i à j dont tous les sommets intermédiaires sont inférieurs ou égaux à k.
   - Soit il fait intervenir $k+1$ et alors il existe un chemin de i à j faisant intervenir exactement une fois $k+1$ (en supprimant les boucles) et ce chemin se décompose en un chemin de i à $k+1$ avec sommets intermédiaires $\leq k$ puis un chemin de $k+1$ à j avec sommets intermédiaires $\leq k$.

   En fin d'algo, C_n contient vrai en case (i, j) si et seulement si il existe un chemin de i vers j pouvant passer par n'importe quels sommets intermédiaires, d'où la correction.
6. Les deux sont en $O(n^3)$ (création de n matrices à n^2 coefficients).
7. On peut en fait modifier C_0 en place car au moment de calculer $C_k(i, j)$, $C_{k-1}(i, j)$ n'a pas encore été changé et les deux autres coefficients ont pu être déjà changés mais sans incidence sur le résultat car $C_{k-1}(i, k) = C_k(i, k)$ et $C_{k-1}(k, j) = C_k(k, j)$. D'où une complexité spatiale en $O(n^2)$.

4 Décidabilité et classes de complexité

Exercice 99 (*) *Formalisation de problèmes*

Dans les cas où cela a du sens, formaliser le problème de décision ou d'optimisation sous-jacent à la description informelle donnée.

1. Trouver le minimum d'un tableau d'entiers.
2. Calculer la composante fortement connexe d'un sommet dans un graphe orienté.
3. Trier un tableau en place.
4. Le problème `SAT`.
5. Étant donnés des objets ayant chacun un poids et une valeur, remplir un sac à dos ayant le plus de valeur possible tout en ne dépassant pas un poids donné.
6. Le problème d'accessibilité dans un graphe.

1. Ce problème est un problème d'optimisation :

$$\begin{cases} \textbf{Entrée :} \text{ Un tableau d'entiers } [t_0, ..., t_{n-1}]. \\ \textbf{Solution :} \text{ Un indice } i \in [\![0, n-1]\!]. \\ \textbf{Optimisation :} \text{ Minimiser } t_i. \end{cases}$$

2. Ce problème est un problème d'optimisation :

$$\begin{cases} \textbf{Entrée :} \text{ Un graphe orienté } G = (S, A) \text{ et } s \in S. \\ \textbf{Solution :} \text{ Un sous ensemble } S' \subset S \text{ tel que } \forall u, v \in S', \ u \text{ et } v \text{ sont mutuellement accessibles.} \\ \textbf{Optimisation :} \text{ Maximiser } |S'|. \end{cases}$$

3. Ce problème n'est ni un problème de décision, ni un problème d'optimisation. On pourrait argumenter que trier est un problème d'optimisation (minimiser le nombre d'inversions par exemple), mais je ne vois pas comment intégrer le caractère en place dans la description d'un problème.
4. Ce problème est un problème de décision :

$$\begin{cases} \textbf{Entrée :} \text{ Une formule } \varphi \text{ du calcul propositionnel.} \\ \textbf{Question :} \ \varphi \text{ est-elle satisfiable ?} \end{cases}$$

5. C'est le problème du sac à dos, qui est un problème d'optimisation :

$$\begin{cases} \textbf{Entrée :} \text{ Des poids } p_1, ..., p_n, \text{ des valeurs } v_1, ..., v_n \text{ et un poids maximal } P, \text{ tous entiers.} \\ \textbf{Solution :} \text{ Un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \sum_{i=1}^{n} p_i \leq P. \\ \textbf{Optimisation :} \text{ Maximiser } \sum_{i=1}^{n} v_i. \end{cases}$$

   Ici, le choix a été fait de considérer les poids et les valeurs comme étant des entiers ; ce pourraient être autre chose mais il faut tout de même pouvoir les représenter en machine.

6. Ce problème est un problème de décision :

$$\begin{cases} \textbf{Entrée :} \text{ Un graphe } G = (S, A) \text{ et } s, t \in S. \\ \textbf{Question :} \text{ Existe-t-il un chemin dans } G \text{ de } s \text{ vers } t \text{ ?} \end{cases}$$

   Pas besoin de se restreindre aux graphes orientés / non orientés.

Exercice 100 (**) *Variantes autour du problème de l'arrêt*

Pour chacun des problèmes suivants, déterminer en justifiant s'il est décidable ou non. Dans les cas où le problème concerné est indécidable, on s'aidera d'une réduction depuis le problème de l'arrêt.

1. $\begin{cases}\textbf{Entrée} : \text{Une fonction } f \text{ et un argument } x.\\ \textbf{Question} : \text{Le calcul de } f(x) \text{ est-il infini ?}\end{cases}$
2. $\begin{cases}\textbf{Entrée} : \text{Une fonction } f, \text{ un argument } x \text{ et un entier } n.\\ \textbf{Question} : \text{Est-ce que l'exécution de } f \text{ sur l'entrée } x \text{ termine en moins de } n \text{ opérations élémentaires ?}\end{cases}$
3. $\begin{cases}\textbf{Entrée} : \text{Une fonction } f \text{ et un argument } x.\\ \textbf{Question} : \text{Est-ce que l'exécution de } f \text{ sur } x \text{ renvoie } 0 \text{ ?}\end{cases}$
4. Dans cette question, f est une fonction fixée en amont.

   $\begin{cases}\textbf{Entrée} : \text{Un argument } x.\\ \textbf{Sortie} : \text{Est-ce que } f(x) \text{ termine en temps fini ?}\end{cases}$
5. $\begin{cases}\textbf{Entrée} : \text{Une fonction } f.\\ \textbf{Question} : \text{Existe-t-il une entrée } x \text{ telle que l'exécution de } f(x) \text{ termine en temps fini ?}\end{cases}$
6. $\begin{cases}\textbf{Entrée} : \text{Une fonction } f.\\ \textbf{Question} : \text{Le nombre d'arguments } x \text{ pour lesquels } f(x) \text{ termine est-il supérieur à } 10 \text{ ?}\end{cases}$

1. Indécidable. Sinon, il existerait une fonction `coarret` qui décide ce problème. Alors la fonction `arret` suivante permet de résoudre le problème de l'arrêt ce qui est impossible.

   ```
   let arret f x = not (coarret f x)
   ```

2. Décidable : on exécute f sur une machine universelle (qui existe) pendant n étapes et on regarde après ces n étapes si l'exécution a fini ou pas.
3. Indécidable. Sinon, il existerait une fonction `zero` qui déciderait ce problème et alors :

   ```
   let arret f x =
       let g x = let f x in 0
       in zero g x
   ```

   permettrait de décider le problème de l'arrêt ce qui est impossible.
4. Cela dépend de la fonction f choisie. On sait qu'il y en a certaines pour lesquelles le problème est indécidable (sinon, l'arrêt serait décidable) ; mais il y en a aussi pour lesquelles il est décidable : toutes les fonctions pour lesquelles on a fait une preuve de terminaison !
5. Indécidable. Sinon, il existe une fonction `arret_exists` qui le déciderait et alors :

   ```
   let arret f x =
       let g () = f x
       in arret_exists g
   ```

   déciderait le problème de l'arrêt ce qui est impossible.
6. Indécidable. Sinon il existe une fonction `arret_sup10` qui le décide. Alors considérons :

   ```
   let arret f x =
       let g y = f x in
       arret_sup10 g
   ```

   Si le calcul de $f(x)$ termine alors il y a une infinité d'entrées pour lesquelles la fonction `g` termine et par conséquent `arret_sup10 g` renvoie `true`. Si le calcul de $f(x)$ ne termine pas alors `g y` ne termine pour aucune entrée `y` et donc `arret_sup10 g` renvoie `false`. On en déduit que `arret` résout correctement le problème de l'arrêt ce qui est une impossible puisque ce problème est indécidable.

Exercice 101 (***) *Problème de correspondance de Post*

Le problème de correspondance de Post, `PCP`, est le problème de décision suivant :

$$\begin{cases}\textbf{Entrée} : \text{Une suite } (u_1, v_1), ..., (u_n, v_n) \text{ de couples dont les éléments sont des mots d'un alphabet } \Sigma. \\ \textbf{Question} : \text{Existe-t-il } k \in \mathbb{N} \text{ et une suite d'indices } i_1, ..., i_k \in [\![1, n]\!] \text{ tels que } u_{i_1}...u_{i_k} = v_{i_1}...v_{i_k} \, ?\end{cases}$$

Le problème de correspondance de Post marqué, `MPCP`, lui est très similaire :

$$\begin{cases}\textbf{Entrée} : \text{Une suite } (u_1, v_1), ..., (u_n, v_n) \text{ de couples dont les éléments sont des mots sur un alphabet } \Sigma. \\ \textbf{Question} : \text{Existe-t-il } k \in \mathbb{N} \text{ et une suite d'indices } i_2, ..., i_k \in [\![1, n]\!] \text{ tels que } u_1u_{i_2}...u_{i_k} = v_1v_{i_2}...v_{i_k} \, ?\end{cases}$$

Ces problèmes peuvent être visualisés comme suit. Leur entrée est un jeu de dominos verticaux $\boxed{\begin{array}{|c|}\hline u_i \\ \hline v_i \\ \hline \end{array}}$.

L'objectif est de savoir s'il existe un enchaînement de dominos (sachant qu'on peut utiliser plusieurs fois un même domino) tels que le mot formé par la concaténation des parties hautes des dominos soit égal à celui formé par les parties basses. Pour `MPCP`, on impose en plus que la suite commence par un domino spécifique.

1. La suite $(a, baa), (ab, aa), (bba, bb)$ est-elle une instance positive de `PCP` ?
2. Montrer que `MPCP` $\leq$ `PCP`.
3. En déduire que `PCP` est décidable si et seulement si `MPCP` l'est.

On admet que `MPCP` est indécidable. Ce résultat va nous servir à montrer l'indécidabilité de problèmes de décision portant sur les langages algébriques. Si $(u_1, v_1), ..., (u_n, v_n)$ est une instance de `PCP` dans laquelle les lettres des mots appartiennent à un alphabet Σ, on peut toujours introduire un alphabet $A = \{a_1, ..., a_n\}$ formé de lettres n'appartenant pas à Σ et on note alors L_u le langage :

$$L_u = \{u_{i_1}...u_{i_k}a_{i_k}...a_{i_1} \,|\, k \geq 0 \text{ et } 1 \leq i_r \leq n\}$$

4. Montrer que L_u est un langage algébrique.
5. Montrer l'indécidabilité du problème `INTERSECTION_VIDE` :

$$\begin{cases}\textbf{Entrée} : \text{Deux grammaires algébriques } G \text{ et } G'. \\ \textbf{Question} : \text{L'intersection des langages engendrés par } G \text{ et } G' \text{ est-elle vide ?}\end{cases}$$

6. En déduire l'indécidabilité du problème `AMBIGUITE` :

$$\begin{cases}\textbf{Entrée} : \text{Une grammaire algébrique } G. \\ \textbf{Question} : G \text{ est-elle ambiguë ?}\end{cases}$$

1. Oui : aligner $(bba, bb), (ab, aa), (bba, bb), (a, baa)$ produit le mot *bbaabbbaa* en haut et en bas.
2. Soit $(u_1, v_1), ..., (u_n, v_n)$ une instance de `MPCP`. Les mots u_i et v_i sont définis sur un certain alphabet (fini), donc on peut considérer une lettre $*$ qui ne fait pas partie de cet alphabet. Si $m = m_1...m_k$ est un mot, on définit les mots $*m$, $m*$ et $*m*$ par :
   - $*m = *m_1 * m_2... * m_k$.
   - $m* = m_1 * m_2 * ...m_k*$.
   - $*m* = *m_1 * m_2 * ... * m_k*$.

   On considère alors l'instance de PCP formée des dominos $(*u_1, *v_1*)$, $(**, *)$ et pour tout $i \geq 2$, $(*u_i, v_i*)$. L'alternance des $*$ impose qu'il y a une solution pour `MPCP` si et seulement s'il y en a une pour `PCP`.
3. La question 2 montre que si `PCP` est décidable alors `MPCP` aussi. Réciproquement, on peut réduire `PCP` à `MPCP` via une réduction Turing comme suit : pour tout domino de l'instance de `PCP`, on lance l'algo qui résout `MPCP` avec ce domino comme début et on décide ainsi `PCP`.
4. Si $k = 0$, L_u est réduit à ε donc est algébrique. Sinon, il est engendré par :

$$S \to u_1Sa_1 \,|\, u_2Sa_2 \,|\, ... \,|\, u_nSa_n \,|\, \varepsilon$$

5. Supposons par l'absurde que ce problème est décidable et montrons que `PCP` $\leq$ `INTERSECTION_VIDE`. Soit donc une instance de PCP : on peut construire algorithmiquement des grammaires pour L_u et L_v et les fournir à l'algorithme décidant `INTERSECTION_VIDE`. Or, $L_u \cap L_v \neq \emptyset$ si et seulement si l'instance choisie pour `PCP` est positive donc on vient de décider `PCP` donc `MPCP` d'après la question 3.

6. Le langage $L = (L_u + L_v) \setminus \{\varepsilon\}$ est un langage algébrique engendré par

$$S \to S_1 \mid S_2, S_1 \to \sum_{i=1}^{n} u_i S_1 a_i \mid \sum_{i=1}^{n} u_i a_i, S_2 \to \sum_{i=1}^{n} u_i S_2 a_i \mid \sum_{i=1}^{n} u_i a_i$$

Or cette grammaire est ambiguë si et seulement si $L_u \cap L_v \neq \emptyset$ donc si on savait décider l'ambiguité, on saurait décider `INTERSECTION_VIDE` et on a montré que ce n'était pas le cas.

Exercice 102 (*) *Classe de complexité a priori*

Dans chacun des cas, indiquer la classe de complexité déterministe du problème (a priori) en justifiant :

1. **Entrée** : Trois entiers naturels a, b, k.
   Question : Le k-ème bit dans l'écriture binaire de $a \times b$ vaut-il 1 ?
2. **Entrée** : Un graphe $G = (S, A)$, deux sommets $s, t \in S$ et un entier k.
   Question : Y a-t-il un chemin de s à t de longueur au plus k ?
3. **Entrée** : Des poids $p_1, ..., p_n \in \mathbb{N}$, un nombre de boîtes $m \in \mathbb{N}$ et un poids maximal $P \in \mathbb{N}$.
   Question : Peut-on ranger n objets de poids $p_1, ..., p_n$ dans m boîtes sans que le poids d'aucune ne dépasse P ?
4. **Entrée** : Une fonction f et un tableau t de taille n.
   Question : Le calcul de $f(t)$ termine-t-il en moins de n étapes de calcul ?
5. **Entrée** : Un graphe non orienté $G = (S, A)$ et un entier k.
   Question : Existe-t-il dans G un ensemble $B \subset A$ tel que $|B| \leq k$ et tel que pour tout sommet s de G au moins une arête de B est incidente à s ?
6. Trouver un élément minimal dans un arbre binaire étiqueté par des entiers.
7. Déterminer si un graphe non orienté est un arbre.
8. Calculer un arbre couvrant minimal dans un graphe pondéré connexe et non orienté.

1. P, car l'algorithme de multiplication naïf est en $O(\log a \times \log b)$.
2. P, car se résout via un parcours qui peut être linéaire en la taille du graphe.
3. EXP, car a priori il faut tester tous les rangements possibles et il y en a autant que de fonctions $[\![1, n]\!] \to [\![1, m]\!]$, c'est-à-dire m^n.
4. P, car il suffit d'exécuter n étapes de calcul ce qui se fait en $O(n)$. Or la taille de l'instance est au moins celle du tableau qui est de taille n donc cette complexité est linéaire en la taille de l'instance.

   Remarque : si on avait directement n en entrée, on ne pourrait plus faire ce raisonnement car un $O(n)$ deviendrait exponentiel en la taille de l'entrée vu que la taille de l'entier n est de l'ordre de $\log n$.
5. P : on calcule un couplage maximum dans le graphe en temps polynomial. S'il couvre tous les sommets on peut répondre, sinon, on le complète de manière gloutonne pour atteindre les sommets manquants et on peut répondre.

   Remarque : de la même manière que la recherche de chemin eulérien est dans P alors que celle de chemin hamiltonien est NP-complet, la couverture par arêtes est dans P alors que la couverture par sommets est NP-complet.
6. P, il suffit de parcourir les étiquettes de l'arbre, en temps linéaire.
7. P, un parcours permet d'obtenir la connexité et un second l'acyclicité (ou, une fois la connexité acquise, on peut se contenter de vérifier que le nombre d'arêtes vaut le nombre de sommets moins un).
8. P, se fait en $O(|A| \log |S|)$ avec Kruskal par exemple.

Exercice 103 (*) *Problèmes NP*

Montrer que les problèmes suivants sont dans NP.

1. `SUBSET SUM` : $\begin{cases}\textbf{Entrée} : \text{Un ensemble fini d'entiers naturels } E \text{ et une cible } C \in \mathbb{N}.\\ \textbf{Question} : \text{Existe-t-il un sous ensemble de } E \text{ dont la somme des éléments vaut } C\,?\end{cases}$

2. `RANGEMENT` : $\begin{cases}\textbf{Entrée} : \text{Des poids } p_1, ..., p_n \in \mathbb{N}, \text{ un nombre de boîtes } m \in \mathbb{N} \text{ et un poids maximal } P \in \mathbb{N}.\\ \textbf{Question} : \text{Peut-on ranger } n \text{ objets de poids } p_1, ..., p_n \text{ dans } m \text{ boîtes sans que le poids d'aucune ne dépasse } P\,?\end{cases}$

3. `SAC A DOS` : $\begin{cases}\textbf{Entrée} : \text{Une liste de poids } p_1, ..., p_n \in \mathbb{N}, \text{ une liste de valeurs } v_1, ..., v_n \in \mathbb{N},\\ \qquad \text{un poids maximal } P \in \mathbb{N} \text{ et une valeur minimale } V \in \mathbb{N}.\\ \textbf{Question} : \text{Existe-t-il un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \sum_{i\in I} p_i \leq P \text{ et } \sum_{i\in I} v_i \geq V\,?\end{cases}$

4. `CHEMIN EULERIEN` : $\begin{cases}\textbf{Entrée} : \text{Un graphe } G = (S, A).\\ \textbf{Question} : \text{Existe-t-il un chemin qui passe une et une seule fois par chaque arête de } A\,?\end{cases}$

5. `COUPLAGE PARFAIT` : $\begin{cases}\textbf{Entrée} : \text{Un graphe } G = (S, A) \text{ non orienté.}\\ \textbf{Question} : \text{Existe-t-il un couplage parfait dans } G\,?\end{cases}$

6. `COUPE` : $\begin{cases}\textbf{Entrée} : \text{Un graphe } G = (S, A) \text{ et un entier } k.\\ \textbf{Question} : \text{Existe-t-il une partition } X \sqcup Y \text{ de } S \text{ telle qu'au moins } k \text{ arêtes relient } X \text{ à } Y\,?\end{cases}$

1. Étant donné un sous-ensemble $X \subset E$ (qui est de taille polynomiale en $|E|$), vérifier que sa somme est inférieure à C se fait en $O(|X|) = O(|E|)$ donc polynomialement en la taille de l'instance.
2. Étant donné un tableau de taille n dont la case i indique le numéro de la boîte dans laquelle ranger l'objet i (qui est de taille $n \log m$ donc polynomiale en la taille de l'instance), vérifier que ce rangement est correct demande de calculer le poids de chacune des boîtes et de vérifier qu'il est inférieur à P ce qui se fait linéairement en la taille de l'instance en parcourant le tableau.
3. Étant donné $I \subset [\![1, n]\!]$ (qui est de taille polynomiale en la taille de l'instance), un vérificateur consiste à calculer deux sommes à moins de n éléments et à faire deux comparaisons se qui se fait en $O(n)$.
4. Étant donné une suite de au plus $|A| + 1$ sommets de G (donc de taille polynomiale en la taille de l'instance), vérifier que c'est un chemin qui passe une et une seule fois par chaque arête de A se fait polynomialement en la taille de l'instance en le parcourant.
5. Étant donné un ensemble X de moins de $|S|$ arêtes (qui est donc de taille polynomiale en la taille de l'entrée), vérifier que c'est un couplage parfait se fait polynomialement en la taille de l'instance en vérifiant que chaque sommet est saturé exactement une fois ce qui se fait en parcourant les arêtes de X.
6. Étant donné un ensemble $X \subset S$ (donc de taille polynomiale en celle de l'instance), vérifier qu'il y a au moins k arêtes entre X et $S \setminus X$ se fait polynomialement en parcourant les arêtes.

Exercice 104 (***) *Voyageur de commerce*

On rappelle qu'un cycle est hamiltonien dans un graphe s'il passe une et une seule fois par chacun des sommets de ce graphe. On considère les deux problèmes de décision suivants :

`CYCLE HAMILTONIEN` : $\begin{cases}\textbf{Entrée} : \text{Un graphe non orienté } G = (S, A).\\ \textbf{Question} : \text{Existe-t-il dans } G \text{ un cycle hamiltonien}\,?\end{cases}$

`VOYAGEUR DE COMMERCE` : $\begin{cases}\textbf{Entrée} : \text{Un graphe non orienté } G = (S, A) \text{ pondéré par } p \text{ et un entier naturel } k.\\ \textbf{Question} : \text{Existe-t-il dans } G \text{ un cycle hamiltonien de poids inférieur ou égal à } k\,?\end{cases}$

1. En utilisant le fait que `CHO` (cycle hamiltonien orienté) est NP-complet, montrer via une réduction polynomiale bien choisie que `CYCLE HAMILTONIEN` est aussi NP-complet.
2. Montrer que `VOYAGEUR DE COMMERCE` est NP-complet.

On note `CH` = `CYCLE HAMILTONIEN` et `PVC` = `VOYAGEUR DE COMMERCE`.

1. Vérifier qu'une suite C de moins de $|S|$ sommets est un cycle hamiltonien dans le graphe $G = (S, A)$ peut se faire en temps polynomial en $|G|$ en vérifiant qu'il n'y a pas de doublons dans C, que $|C| = |S|$

et qu'il y a bien une arête de G entre deux sommets consécutifs de C (y compris entre le dernier et le premier). Donc CH $\in$ NP.

Montrons que CHO $\leq$ CH. Soit $G = (S, A)$ un graphe orienté. On construit le graphe $G' = (S', A')$ avec :

- $S' = \bigcup_{s \in S} \{s_1, s_2, s_3\}$: autrement dit, on détriple chaque sommet de G.
- $A' = \bigcup_{s \in S} \{(s_1, s_2), (s_2, s_3)\} \cup \{(s_3, t_1) \,|\, (s, t) \in A\}$, ces arêtes étant non orientées.

Le graphe G' est non orienté donc est une instance de CH. Il est possible de construire G' en temps polynomial en $|G|$ car il y a $3|S| = O(|G|)$ sommets et $|A| + 2|S| = O(|G|)$ arêtes dans G'. Enfin, G est une instance positive de CHO si et seulement si G' est une instance positive de CH. En effet :

($\Rightarrow$) Si G admet un cycle hamiltonien $s^{(1)}, s^{(2)}, ..., s^{(n)}$, alors $s_1^{(1)}, s_2^{(1)}, s_3^{(1)}, s_1^{(2)}, s_2^{(2)}, s_3^{(2)}, ..., s_3^{(n)}$ est un cycle hamiltonien dans G' (en fait, chaque sommet se transforme en trois sommets numérotés par $1, 2, 3$ et il suffit de parcourir ces 3 sommets dans le sens $1 \to 2 \to 3$).

($\Leftarrow$) Réciproquement, si G' admet un cycle hamiltonien C alors comme C n'est pas orienté, on peut supposer que ce cycle parcourt les sommets s_1, s_2, s_3 dans le sens $1 \to 2 \to 3$ pour tout sommet $s \in S$. Ceci donne naissance à un cycle dans G par "fusion" pour tout $s \in S$ des trois sommets (nécessairement consécutifs dans C) s_i en le sommet s.

On vient donc d'établir que CHO se réduit polynomialement en CH et comme CHO est NP-difficile, CH aussi. Ce problème est NP-difficile et NP donc est NP-complet.

2. La preuve du caractère NP de PVC se fait de manière similaire à celle de CH.

   Soit $G = (S, A)$ une instance de CH. Alors le graphe $G' = (S, A)$ pondéré par la fonction p qui donne comme poids 1 à toutes les arêtes est un graphe qu'on peut construire polynomialement en $|G|$ donc $(G', |S|)$ est une instance de PVC constructible en temps polynomial en $|G|$.

   Or, si G admet un cycle hamiltonien, alors ce cycle est un cycle hamiltonien dans G' de poids égal à $|S|$ et réciproquement, si G' admet un cycle hamiltonien de poids supérieur ou égal à $|S|$ alors ce cycle est aussi un cycle hamiltonien dans G. Donc G est une instance positive pour CH si et seulement si G' est une instance positive de PVC.

   Ceci montre que CH $\leq$ PVC et comme CH est NP-difficile d'après la question 1, PVC aussi ce qui était le point qui manquait pour montrer sa NP-complétude.

Exercice 105 (**) *Réductions autour du problème du sac à dos*

Dans cet exercice, on considère les problèmes SUBSET SUM, SAC A DOS et RANGEMENT, définis à l'exercice 4 ainsi que le problème PARTITION suivant :

$$\begin{cases} \textbf{Entrée} : \text{Une liste d'entiers } a_1, ..., a_n. \\ \textbf{Question} : \text{Existe-t-il un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \sum_{i \in I} a_i = \sum_{i \notin I} a_i \,? \end{cases}$$

1. Montrer que SUBSET SUM $\leq$ SAC A DOS.
2. Montrer que SUBSET SUM $\leq$ PARTITION.
3. Montrer que PARTITION $\leq$ RANGEMENT.
4. En admettant que le problème SUBSET SUM est NP-complet (ce résultat sera montré à l'exercice 8), que peut-on en déduire sur la classe de complexité des problèmes de cet exercice ?

1. Soit $t = [t_1, ..., t_n], C$ une instance de SUBSET SUM. Alors en prenant $p_i = v_i = t_i$ et $P = V = C$, on construit une instance de SAC A DOS en temps polynomial en $|t| + |C|$. De plus :

$$\begin{aligned} (t, C) \text{ est une instance positive de SUBSET SUM} &\Leftrightarrow \text{ il existe } I \subset [\![1, n]\!] \text{ tel que } \sum_{i \in I} t_i = C \\ &\Leftrightarrow \text{ il existe } I \subset [\![1, n]\!] \text{ tel que } \sum_{i \in I} p_i = P \text{ et } \sum_{i \in I} v_i = V \\ &\Leftrightarrow (p_i, v_i, P, V) \text{ est une instance positive de SAC A DOS} \end{aligned}$$

2. Soit $t = [t_1, ..., t_n], C$ une instance de `SUBSET SUM`. Notons $S = \sum_{i=1}^{n} t_i$ et considérons $a_1 = t_1, ..., a_n = t_n, a_{n+1} = S - 2C$. Alors $(a_i)_{i \in [\![1, n+1]\!]}$ est une instance de `PARTITION` constructible en temps polynomial en $|t| + |C|$ qui est positive si et seulement si (t, C) est positive pour `SUBSET SUM` :

   S'il existe $I \subset [\![1, n]\!]$ tel que $\sum_{i \in I} t_i = C$ alors $\sum_{i \notin I} t_i = S - C$ et alors en notant $I' = I \cup \{n+1\}$ on a :

$$\sum_{i \in I'} a_i = \left(\sum_{i \in I} t_i \right) + a_{n+1} = C + S - 2C = S - C = \sum_{i \notin I'} t_i = \sum_{i \in I'} a_i$$

   Réciproquement, s'il existe $I \subset [\![1, n+1]\!]$ tel que $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$ alors comme la somme de tous les a_i fait $2S - 2C$, chacune des deux sommes vaut $S - C$. Sans perte de généralité, on peut supposer que $(n+1) \in I$ et alors les t_i indicés par un élément de $I \setminus \{n+1\} \subset [\![1, n]\!]$ ont une somme égale à C car :

$$\sum_{i \in I \setminus \{n+1\}} t_i = \sum_{i \in I \setminus \{n+1\}} a_i = \left(\sum_{i \in I} a_i \right) - a_{n+1} = S - C - (S - 2C) = C$$

3. Soit $a_1, ..., a_n$ une instance de `PARTITION`. Notons $p_i = 2 \times a_i$, $m = 2$ et $P = \sum_{i=1}^{n} a_i$. Alors (p_i, m, P) est une instance de `RANGEMENT` constructible en temps polynomial en la taille de $a_1, ..., a_n$ qui est positive si et seulement si $a_1, ..., a_n$ est positive pour `PARTITION`.

   Remarque : on préfère cette transformation à $p_i = a_i$, $m = 2$ et $P = \frac{1}{2} \sum_{i=1}^{n} a_i$ car avec cette transformation, P n'est pas forcément un entier.

4. Tous ces problèmes sont NP-complets.

Exercice 106 (***) *Subset sum*

L'objectif de cet exercice est de montrer que le problème :

$$\texttt{SUBSET SUM} : \begin{cases} \textbf{Entrée} : \text{Un tableau } T = [t_1, ..., t_n] \text{ d'entiers naturels et } C \in \mathbb{N}. \\ \textbf{Question} : \text{Existe-t-il un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \sum_{i \in I} t_i = C \, ? \end{cases}$$

est NP-complet. Pour montrer qu'il est NP-difficile, on procède via une réduction depuis `3SAT`. Soit φ une formule sous 3-CNF dont les variables sont $x_1, ..., x_n$ et les clauses $C_0, ..., C_{m-1}$. Pour tout $i \in [\![1, n]\!]$, on définit a_i et b_i par :

$$a_i = 6^{i+m-1} + \sum_{\substack{j=0 \\ x_i \in C_j}}^{m-1} 6^j \text{ et } b_i = 6^{i+m-1} + \sum_{\substack{j=0 \\ \overline{x_i} \in C_j}}^{m-1} 6^j$$

De plus, pour tout $j \in [\![0, m-1]\!]$, on pose $c_j = d_j = 6^j$ et on note $C = 3 \sum_{i=0}^{m-1} 6^i + \sum_{j=m}^{m+n-1} 6^j$.

1. Considérons tout d'abord la formule exemple $\varphi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3})$. Dans un tableau à 15 lignes et 7 colonnes donner la représentation en base 6 des a_i, b_i, c_j, d_j et de C. La colonne de droite sera celle des "bits" de poids faible.
2. Toujours dans ce cas particulier, si on ne prend pas en compte la ligne représentant la décomposition de C, pour chacune des quatre premières colonnes, combien de lignes au maximum dans le tableau précédent contiennent un 1 dans celle-ci ? Même question pour les trois dernières colonnes.

On pose $T = [a_1, ..., a_n, b_1, ...b_n, c_0, ...c_{m-1}, d_0, ...d_{m-1}]$.

3. On suppose que φ est satisfiable. Montrer que (t, C) est une instance positive de `SUBSET SUM`.
4. Réciproquement, montrer que si (t, C) est solution de `SUBSET SUM` alors φ est satisfiable.
5. En déduire que `SUBSET SUM` est NP-complet.

Si $([t_1, ..., t_n], C)$ est une instance de `SUBSET SUM`, pour tout $i \in [\![0, n]\!]$ et $c \in [\![0, C]\!]$, on note $SP(i, c)$ le booléen valant vrai s'il existe un sous ensemble $I \subset [\![1, i]\!]$ tel que $\sum_{j \in I} t_j = C$ et faux sinon.

6. Déterminer les valeurs de $SP(i,c)$ lorsque $c=0$, $c<0$ ou $i=0$.
7. Déterminer une formule de récurrence liant $SP(i,c)$ à $SP(i-1,c)$ et $SP(i-1,c-t_i)$.
8. En déduire un algorithme permettant de résoudre `SUBSET SUM` avec une complexité en $O(nC)$.
9. Vient-on de montrer que P = NP ?

TODO

Exercice 107 (exo cours) *Problème de l'arrêt*

Montrer qu'il est impossible d'écrire une fonction `arret : ('a -> 'b) -> 'a -> bool` en OCaml qui aurait le comportement suivant : pour toute fonction `f` et toute entrée `x`, l'exécution de `arret f x` termine toujours en temps fini et renvoie `true` si et seulement si l'exécution de `f` sur `x` termine.

Si c'était le cas on pourrait écrire la fonction suivante :

```
let rec paradoxe () = if arret paradoxe () then paradoxe ()
```

qui termine sur son entrée si et seulement si elle ne termine pas. Or c'est impossible.

Exercice 108 (exo cours) *Réduction au plus court chemin*

1. Formaliser le problème d'optimisation `PCC` consistant à déterminer un plus court chemin (en termes de nombre d'arêtes) entre deux sommets dans un graphe non orienté.
2. Déterminer le problème de décision `PCC-deci` associé au problème d'optimisation `PCC`.
3. On considère le problème suivant :

$$\texttt{ACCESSIBILITE} : \begin{cases} \textbf{Entrée} : \text{Un graphe } G=(S,A) \text{ non orienté et deux sommets } s,t \in S. \\ \textbf{Question} : \text{Existe-t-il un chemin entre } s \text{ et } t \,? \end{cases}$$

   Montrer que `ACCESSIBILITE` se réduit polynomialement à `PCC-deci`.

1. $\begin{cases} \textbf{Entrée :} G=(S,A) \text{ un graphe non orienté et } s,t \in S. \\ \textbf{Solution :} \text{ Un chemin } (s_0,...,s_k) \text{ dans } G \text{ tel que } s_0=s \text{ et } s_k=t \\ \textbf{Optimisation :} \text{ Minimiser } k. \end{cases}$
2. $\begin{cases} \textbf{Entrée :} G=(S,A) \text{ un graphe non orienté, } s,t \in S \text{ et } k \in \mathbb{N}. \\ \textbf{Question :} \text{ Existe-t-il un chemin de } s \text{ à } t \text{ dans } G \text{ de longueur inférieure à } k \,? \end{cases}$
3. Si $i_1=(G=(S,A),s,t)$ est une instance de `ACCESSIBILITE`, $i_2=(G,s,t,|A|)$ est une instance de `PCC-deci` constructible en temps polynomial en $|i_1|$. De plus i_1 est positive si et seulement si i_2 l'est car un chemin de s à t est nécessairement de longueur inférieure à $|A|$ (et même $|S|-1$).

Exercice 109 (exo cours) *Coloriabilité*

1. Formaliser le problème de décision `k-COLORIABLE` consistant à déterminer si on peut colorier les sommets d'un graphe non orienté en n'utilisant que $k \in \mathbb{N}^\star$ couleurs différentes et de telle sorte à ce que deux sommets voisins n'aient jamais la même couleur.
2. On admet que `3SAT` $\leq$ `3-COLORIABLE`. Montrer que `3-COLORIABLE` est NP-complet.
3. Quelle est la classe de complexité de `2-COLORIABLE` ? Justifier brièvement.

1. $\begin{cases} \textbf{Entrée :} \text{ Un graphe non orienté } G=(S,A) \text{ et } k \in \mathbb{N}^\star. \\ \textbf{Question :} \text{ Existe-t-il une fonction } f : S \to [\![1,k]\!] \text{ telle que pour tout } (s,t) \in A,\ f(s) \neq f(t) \,? \end{cases}$
2. La réduction montre la NP-difficulté. On peut vérifier en temps polynomial si une coloration (de taille $|S|$ donc polynomiale) est correcte en parcourant les arêtes ; d'où le caractère NP.
3. Ce problème est dans P. Il suffit de faire un parcours en largeur du graphe en coloriant avec 1 les

sommets à distance impaire de l'origine et 2 ceux à distance paire : le graphe est 2-coloriable si et seulement si il n'y a aucun conflit.

Exercice 110 (exo cours) *Réduction ou pas réduction ?*

1. Rappeler la définition des problèmes de décision `CHO` (cycle hamiltonien orienté), `SAT` et `kSAT` pour $k \geq 2$.
2. Est-il vrai que `3SAT` se réduit polynomialement à `CHO` ? Justifier.
3. Est-il vrai que `SAT` se réduit polynomialement à `2SAT` ? Justifier.

1. RAS.
2. Oui car `3SAT` est dans NP (toute formule φ satisfiable l'est par une valuation v de taille linéaire en $|\varphi|$ et l'évaluation de $v(\varphi)$ se fait en temps polynomial) et `CHO` est NP-complet.
3. Probablement non, car `SAT` est NP-complet et `2SAT` $\in$ P. Si c'était vrai, on aurait P = NP.

Exercice 111 (exo cours) *Transitivité des réductions*

Si A et B sont deux problèmes de décision, on note $A \leq B$ si A se réduit polynomialement à B.

1. Rappeler les définitions de "$A \leq B$" et de "A est NP-difficile".
2. Montrer rigoureusement que $\leq$ est réflexive et transitive.
3. Montrer que si $A \leq B$ et A est NP-difficile alors B l'est aussi.

1. $A \leq B$ s'il existe $f : I_A \to I_B$ qui s'exécute en temps polynomial et telle que pour tout $x \in I_A$, x est positive pour A si et seulemnt si $f(x)$ est positive pour B.
2. $A \leq A$ car l'identité est polynomiale. La transitivité vient de ce qu'une composée de polynômes est un polynôme. On demande une preuve formelle.
3. Soit $C \in$ NP. Comme A est NP-difficile, $C \leq A$. Par transitivité $C \leq B$ ce qui conclut.

Exercice 112 (exo cours) *Cliques*

1. Formaliser le problème d'optimisation `CLIQUE MAX` suivant : dans un graphe non orienté G, trouver un sous graphe de G qui soit complet avec le plus de sommets possible.
2. Déterminer le problème de décision `CLIQUE-deci` associé à `CLIQUE MAX`.
3. Montrer que `CLIQUE-deci` est un problème NP.

1. $\begin{cases} \textbf{Entrée :} \text{ Un graphe } G = (S, A) \text{ non orienté.} \\ \textbf{Solution :} \text{ Un sous graphe complet } G' = (S', A') \text{ de } G. \\ \textbf{Optimisation :} \text{ Maximiser } |S'|. \end{cases}$
2. On ajoute un seuil aux entrées et on demande s'il existe une clique de taille supérieure à ce seuil.
3. Si une instance est positive, il existe $X \subset S$ (donc de taille polynomiale en l'entrée) qui induit un graphe complet. Vérifier qu'il est complet et de la bonne taille se fait polynomialement.

Exercice 113 (exo cours) *Décidabilité de l'équivalence*

Le problème `EQUIV` suivant est-il décidable ?

$\begin{cases} \textbf{Entrée} : \text{Deux fonctions } f \text{ et } g. \\ \textbf{Question} : f \text{ et } g \text{ ont-elles le même comportement sur toute entrée (c'est-à-dire renvoient la même} \\ \qquad\qquad \text{valeur si elles terminent ou ne terminent ni l'une ni l'autre) ?} \end{cases}$

Non. Sinon il serait décidé par une fonction `equiv` et alors cette fonction décide le problème de l'arrêt :

```
let arret f x =
    let f1 () = let _ = f x in ()
    and let rec f2 () = f2 () in
    not (equiv f1 f2)
```

Exercice 114 (**) *STABLE est NP-complet*

Un stable dans un graphe G est un sous-ensemble X des sommets de ce graphe tel que deux sommets de X ne sont jamais adjacents dans G. On considère le problème de décision `STABLE` suivant :

$$\begin{cases} \textbf{Entrée} : \text{Un graphe } G = (S, A) \text{ non orienté et } k \in \mathbb{N}. \\ \textbf{Question} : \text{Existe-t-il } S' \subset S \text{ un stable de } G \text{ tel que } |S'| \geq k \,? \end{cases}$$

1. Montrer que `STABLE` $\in$ NP.

On cherche à présent à montrer que `3SAT` $\leq$ `STABLE`. Soit donc $\varphi = \bigwedge_{i=1}^{m} C_i$ une formule du calcul propositionnel dont les clauses C_i contiennent au plus 3 littéraux et impliquant un ensemble $V = \{v_1, ..., v_n\}$ de variables propositionnelles. On définit le graphe non orienté $G_\varphi = (S, A)$ comme suit :

- Pour chaque **occurrence** de littéral dans φ, on construit un sommet de S.
- Si $s, t \in S$, on ajoute l'arête $\{s, t\}$ à A si et seulement si (s et t sont deux littéraux d'une même clause) ou (s et t sont la négation l'un de l'autre).

2. Tracer le graphe G_φ associé à $\varphi = (\overline{x} \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y})$.
3. Montrer que si G_φ possède un stable de taille m, alors φ est satisfiable par une valuation qu'on explicitera. La réciproque est-elle vraie ?
4. Montrer que `STABLE` est NP-complet.
5. En déduire que `CLIQUE` est NP-complet avec `CLIQUE` le problème défini par :

$$\begin{cases} \textbf{Entrée} : \text{Un graphe } G = (S, A) \text{ non orienté et } k \in \mathbb{N}. \\ \textbf{Question} : \text{Existe-t-il un sous-graphe complet de } G \text{ ayant plus de } k \text{ sommets} \,? \end{cases}$$

1. RAS.
2. Il y a un sommet par **occurrence** de littéral donc le graphe à 12 sommets.
3. Si G_φ admet un stable X de taille m alors :
   - Pour chaque clause C, comme les sommets correspondants à ses littéraux sont liés et qu'il y a m clauses, il y a exactement un littéral de C dans X.
   - Il n'y a aucun sommet étiqueté par x dans X tel que $\neg x$ soit aussi dans X.

   Alors la valuation attribuant 1 à tous les littéraux du stable et 0 aux autres est bien définie et satisfait toutes les clauses de φ. La réciproque est vraie : si φ est satisfiable via v, pour chaque clause il existe un littéral l_i tel que $v(l_i) = 1$ et $\{l_i \,|\, i \in [\![1, m]\!]\}$ est un stable de taille m dans G_φ.
4. Transformer φ en G_φ se fait en temps polynomial en $|\varphi|$ ce qui combiné à 3) montre que `3SAT` $\leq$ `STABLE`. Comme `3SAT` est NP-difficile, `STABLE` l'est aussi et comme il est NP on conclut.
5. Ce problème est NP et NP-difficile car `STABLE` $\leq$ `CLIQUE` via la réduction qui à $(G = (S, A), k)$ instance de `STABLE` associe l'instance de `CLIQUE` $((S, S^2 \setminus A), k)$.

Exercice 115 (**) *Couverture par sommets*

On dit qu'un sous-ensemble X des sommets d'un graphe G est une couverture par sommets de G si toute arête de G a au moins l'une de ses extrémités dans X. Une clique dans un graphe G est un sous-graphe complet de G. On considère les problèmes `VERTEX COVER` et `CLIQUE` définis par :

`VERTEX COVER` : $\begin{cases} \textbf{Entrée} : \text{Un graphe non orienté } G = (S, A) \text{ et } k \in \mathbb{N}. \\ \textbf{Question} : \text{Existe-t-il une couverture par sommets } X \text{ de } G \text{ telle que } |X| \leq k \,? \end{cases}$

`CLIQUE` : $\begin{cases}\textbf{Entrée} : \text{Un graphe } G=(S,A) \text{ non orienté et } k \in \mathbb{N}.\\ \textbf{Question} : \text{Existe-t-il une clique } X \text{ de } G \text{ telle que } X \geq k\,?\end{cases}$

On admet que `CLIQUE` est un problème NP-complet.

1. Montrer que `VERTEX COVER` $\in$ NP.
2. Montrer que `CLIQUE` $\leq$ `VERTEX COVER`. *Indication : Si (G,k) est une instance de `CLIQUE`, on pourra considérer le graphe complémentaire de G.*
3. En déduire que `VERTEX COVER` est NP-complet.
4. Le problème d'optimisation associé à `VERTEX COVER` vise à couvrir les arêtes par les sommets avec le moins de sommets possible. Formaliser le problème d'optimisation consistant au contraire à couvrir les sommets par les arêtes avec le moins d'arêtes possible. Que peut-on dire de la classe de complexité de ce problème ?

1. RAS.
2. Soit $i_1 = (G=(S,A),k)$ une instance de `CLIQUE`. Alors $i_2 = (\overline{G} = (S, S^2 \setminus A), |S|-k)$ est une instance de `VERTEX COVER` calculable en temps polynomial en $|i_1|$.

   De plus, si i_1 est positive pour `CLIQUE` alors il existe $X \subset S$ une clique de taille au moins k. Alors $X' = S \setminus X$ est de taille inférieure à $|S|-k$ et c'est une couverture de $\overline{G}$. En effet, si $(u,v) \notin A$, soit u soit v n'appartient pas à la clique X et donc soit u soit v appartient à $S \setminus X = X'$.

   Réciproquement, si i_2 est positive pour `VERTEX COVER`, il existe une couverture X de taille inférieure à $|S|-k$. Alors $X' = S \setminus X$ est de taille au moins k et est une clique de G. En effet, si $u,v \in X'$, $(u,v) \in A$ sinon cette arête est couverte par X donc $u \in X$ ou $v \in X$ ce qui est impossible.
3. La question 1 montre le caractère NP et la question 3 la NP-difficulté.
4. On peut résoudre ce problème en temps polynomial. Il suffit de calculer un couplage maximum dans le graphe (ce qui se fait en temps polynomial même dans un graphe non biparti) puis de couvrir les sommets non saturés par une arête quelconque. La couverture ainsi créée est de cardinal minimal (preuve par l'absurde).

Exercice 116 (*) *Isomorphisme de graphes*

Si $G_1 = (S_1, A_1)$ et $G_2 = (S_2, A_2)$ sont deux graphes non orientés, on dit que G_1 et G_2 sont isomorphes s'il existe une bijection $f : S_1 \to S_2$ telle que pour tout $s,t \in S_1$, il y a une arête entre s et t dans G_1 si et seulement si il y a une arête entre $f(s)$ et $f(t)$ dans G_2. On considère les problèmes :

`ISO GRAPHES` : $\begin{cases}\textbf{Entrée} : \text{Deux graphes non orientés } G \text{ et } H.\\ \textbf{Question} : G \text{ et } H \text{ sont-ils isomorphes ?}\end{cases}$

`ISO SOUS GRAPHES` : $\begin{cases}\textbf{Entrée} : \text{Deux graphes non orientés } G \text{ et } H.\\ \textbf{Question} : G \text{ est-il isomorphe à un sous-graphe de } H\,?\end{cases}$

`CLIQUE` : $\begin{cases}\textbf{Entrée} : \text{Un graphe } G=(S,A) \text{ non orienté et } k \in \mathbb{N}.\\ \textbf{Question} : \text{Existe-t-il un sous-graphe complet de } G \text{ ayant au moins } k \text{ sommets ?}\end{cases}$

On admet que le problème `CLIQUE` est NP-complet.

1. Montrer que `ISO GRAPHES` $\in$ NP.
2. Montrer que `ISO SOUS GRAPHES` l'est aussi.
3. Montrer par réduction polynomiale que `ISO SOUS GRAPHES` est NP-complet.

Remarque : A ce jour, on ne sait pas si `ISO GRAPHES` est NP-complet ou pas.

1. Un isomorphisme de $G = (S_G, A_G)$ vers $H = (S_H, A_H)$ peut être représenté par un tableau de taille $|S_G|$ contenant des valeurs de $[\![0, |S_H|-1]\!]$ et vérifier qu'un tel tableau représente un isomorphisme se fait bien en temps polynomial.
2. Similaire à la question 1.
3. `CLIQUE` $\leq$ `ISO SOUS GRAPHES` via la transformation (polynomiale) qui à (G,k) instance de `CLIQUE`

associe l'instance de `ISO SOUS GRAPHES` (K_k, G) où K_k est le graphe complet à k sommets.

Exercice 117 (**) *Ensembles dominants*

On dit qu'un sous ensemble D des sommets d'un graphe G est dominant si tout sommet de G est soit dans D soit adjacent à un sommet de D. On dit qu'un sous ensemble C des sommets de G est une couverture par sommets de G si chacune des arêtes de G a au moins une extrémité dans C. On note :

$$\texttt{DOM} : \begin{cases} \textbf{Entrée} : \text{Un graphe non orienté } G = (S, A) \text{ et } k \in \mathbb{N}. \\ \textbf{Question} : \text{Y a-t-il un ensemble dominant dans } G \text{ de taille au plus } k \,? \end{cases}$$

On introduit aussi le problème `VERTEX COVER` dont on admet le caractère NP-complet :

$$\texttt{VERTEX COVER} : \begin{cases} \textbf{Entrée} : \text{Un graphe non orienté } G = (S, A) \text{ et } k \in \mathbb{N}. \\ \textbf{Question} : \text{Existe-t-il dans } G \text{ une couverture par sommets de taille au plus } k \,? \end{cases}$$

1. Montrer que `DOM` $\in$ NP.

On cherche à présent à réduire polynomialement le problème `VERTEX COVER` à `DOM`. Soit donc $G = (S, A)$ une instance de `VERTEX COVER`. On construit le graphe $G' = (S', A')$ à partir de G comme suit :

- $S' = (S \setminus I) \cup E$ où I est l'ensemble des sommets isolés de G et $E = \{s_a \,|\, a \in A\}$ est un ensemble de nouveaux sommets.
- $A' = A \cup \{(x, s_a) \,|\, x \text{ est une extrémité de l'arête } a\}$.

2. Montrer que G admet une couverture par sommets de taille au plus k si et seulement si G' admet un ensemble dominant de taille au plus k.
3. En déduire que `DOM` est NP-complet.

1. RAS.
2. Si G admet une couverture C de taille moins que k, C est dominant dans G'. En effet, si $s \in S'$:
   - Soit s est un sommet non isolé de G et alors il existe $(u, s) \in A$. Cette arête est couverte par C donc u ou s est dans C et donc s est dans C ou voisin de C.
   - Soit $s = s_a$ pour $a \in A$. Alors par construction de G' s est relié à u et à v tels que $a = (u, v)$. Comme C couvre (u, v), $u \in C$ ou $v \in C$ donc $s = s_a$ est voisin de C.

   Réciproquement, supposons C dominant dans G' de taille au plus k. Soit a une arête de G. Si aucune des extrémités de A n'est dans C alors $s_a \in C$ sinon ce sommet ne serait ni dans C ni voisin de C alors que C est dominant. Alors $C' = (C \setminus \{s_a\}) \cup \{\text{une des extrémités de } a\}$ couvre a et reste un ensemble dominant de G' de taille au plus k (la taille peut strictement décroître si l'extrémité de a qu'on a ajoutée était déjà dans C). On répète l'opération pour toute arête de G et cela donne une couverture de G de taille au plus k.
3. La transformation de G en G' se fait en temps polynomial donc la question 2 montre que `VERTEX COVER` $\leq$ `DOM` ce qui montre la NP-difficulté. Le caractère NP a été montré en question 1.

Exercice 118 (**) *Problème de rangement*

On considère les deux problèmes suivants :

$$\texttt{STABLE} : \begin{cases} \textbf{Entrée} : \text{Un graphe } G = (S, A) \text{ non orienté et } k \in \mathbb{N}. \\ \textbf{Question} : \text{Existe-t-il } S' \subset S, \text{ ensemble de sommets deux à deux non adjacents tel que } |S'| = k \,? \end{cases}$$

$$\texttt{RANGEMENT} : \begin{cases} \textbf{Entrée} : \text{Un ensemble fini } E, \text{ une collection } \{E_1, ..., E_n\} \text{ de sous ensembles de } E, \, k \in \mathbb{N}. \\ \textbf{Question} : \text{Y a-t-il une sous collection } C \text{ de cardinal } k \text{ d'ensembles deux à deux disjoints} \,? \end{cases}$$

1. Montrer que ces deux problèmes sont dans NP.
2. En admettant que `STABLE` est NP-difficile, montrer que `RANGEMENT` est NP-complet.

1. RAS.
2. Soit $i_1 = (G = (S, A), k)$ une instance de `STABLE`. Alors $i_2 = (A, \{E(s) \,|\, s \in S\}, k)$ est une instance de

RANGEMENT qu'on peut construire en temps polynomial en $|i_1|$.

Si i_1 est positive pour STABLE, il existe un stable X de taille k dans G et alors $\{E(s) \mid S \in X\}$ est une collection d'ensembles deux à deux disjoints de taille k car si $s \neq t$, $E(s) \cap E(t)$ est au mieux égal $\{(s,t)\}$ mais cette arête n'existe pas sinon X ne serait pas un stable.

Réciproquement, si $\{E(s) \mid S \in X\}$ est une collection d'ensembles deux à deux disjoints, X est un stable dans G donc si i_2 est positive pour RANGEMENT, i_1 est positive pour STABLE.

Exercice 119 (***) *Réductions depuis ZOE*

Dans cet exercice, on admet que le problème ZOE (zero-one equations) est NP-complet :

$$\begin{cases} \textbf{Entrée} : \text{Une matrice } A \in M_{m,n}(\{0,1\}) \\ \textbf{Question} : \text{Y a-t-il un vecteur } X \in M_{n,1}(\{0,1\}) \text{ tel que } AX = \mathbf{1} \,? \end{cases}$$

On considère par ailleurs les deux problèmes suivants :

$$\texttt{SOMME PARTIELLE} : \begin{cases} \textbf{Entrée} : \text{Un tableau } T = [t_1, ..., t_n] \text{ d'entiers naturels et } C \in \mathbb{N}. \\ \textbf{Question} : \text{Y a-t-il un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \sum_{i \in I} t_i = C \,? \end{cases}$$

$$\texttt{PLNE} : \begin{cases} \textbf{Entrée} : \text{Une matrice } A \in M_{m,n}(\mathbb{R}) \text{ et un vecteur } B \in M_{m,1}(\mathbb{R}). \\ \textbf{Question} : \text{Y a-t-il un vecteur } X \in M_{n,1}(\mathbb{Z}) \text{ tel que } AX \leq B \text{ (inégalités ligne à ligne)} \,? \end{cases}$$

1. Montrer que les problèmes SOMME PARTIELLE et PLNE sont dans NP.

On cherche à présent à montrer que ZOE $\leq$ PLNE. Soit $A \in M_{m,n}(\{0,1\})$ une instance de ZOE. On pose $A' = (a'_{ij})_{i,j \in [\![1,2m+2n]\!] \times [\![1,n]\!]}$ et $B = (b_i)_{i \in [\![1,2m+2n]\!]}$ avec :

(1) Pour $i \leq m$, $a'_{ij} = a_{ij}$ et $b_i = 1$.

(2) Pour $m < i \leq 2m$, $a'_{i,j} = -a_{ij}$ et $b_i = -1$.

(3) Pour $2m < i \leq 2m + n$, $a'_{ij} = \delta_{i,2m+j}$ et $b_i = 1$.

(4) Pour $2m + n < i$, $a'_{ij} = -\delta_{i,2m+n+j}$ et $b_i = 0$.

2. A l'aide de ce qui précède montrer que PLNE est NP-complet.
3. Montrer que ZOE $\leq$ SOMME PARTIELLE et en déduire que SOMME PARTIELLE est NP-complet. *Indication : Si A est une matrice, considérer les $t_j = \sum_{i=1}^{m} a_{ij} \times 2^{i-1}$ et la quantité $\sum_{i=1}^{m} 2^{i-1}$.*

1. RAS.
2. Déjà, la transformation étudiée est polynomiale en $|A|$. Si X est tel que $A'X \leq B$:
   (1) Impose que $AX \leq \mathbf{1}$.
   (2) Impose que $-AX \leq -\mathbf{1}$ donc que $AX \geq \mathbf{1}$.
   (3) Impose que tous les x_j sont inférieurs à 1.
   (4) Impose que tous les $-x_j$ sont inférieurs à 0 donc que les x_j sont positifs.

   Si (A', B) est positive pour PLNE, il existe un vecteur X d'entiers tels que $A'X \leq B$. (3) et (4) montrent que les coordonnées de X sont des entiers entre 0 et 1 donc que $X \in M_{n,1}(\{0,1\})$. (1) et (2) montrent que $AX = \mathbf{1}$. Donc X est un témoin de la positivité de l'instance A de ZOE.

   Si A est positive pour ZOE, (A', B) est évidemment positive pour PLNE.
3. Si $A \in M_{m,n}(\{0,1\})$ est une instance de ZOE, on la transforme en temps polynomial en (T, C) avec les t_j de l'énoncé et $C = \sum_{i=1}^{m} 2^{i-1}$. A est positive pour ZOE si et seulement si (T, C) est positive pour SOMME PARTIELLE. En effet, le vecteur X fourni par ZOE indique exactement quels j choisir pour obtenir une somme égale à C : ce sont pour lesquels $x_j = 1$.

Exercice 120 (***) *Semi-décidabilité*

On dit qu'un problème de décision est semi-décidable s'il existe un algorithme qui :

- Renvoie "oui" en temps fini pour toute instance positive du problème.
- Renvoie "non" ou ne termine pas ou échoue sur les instances négatives.

Pour tout problème de décision `A`, le problème complémentaire de `A`, noté `coA`, est le problème de décision ayant les mêmes instances que `A` et dont les instances positives (resp. négatives) sont les instances négatives (resp. positives) de `A`.

1. Montrer qu'un problème `A` est décidable si et seulement si `A` et `coA` sont semi-décidables.
2. Le problème `coARRET` est-il semi-décidable ?

On définit le problème de l'arrêt universel par :

$$\texttt{ARRET}_\forall : \begin{cases} \textbf{Entrée :} \text{ Une fonction } f. \\ \textbf{Question :} \text{ L'exécution de } f \text{ sur } x \text{ termine-t-elle pour tout } x \,? \end{cases}$$

3. Ce problème est-il décidable ?
4. Le problème $\texttt{coARRET}_\forall$ est-il semi-décidable ?
5. Le problème $\texttt{ARRET}_\forall$ est-il semi-décidable ?

1. Le sens direct vient de ce qu'un problème décidable est semi-décidable. Réciproquement, si `A` est semi-décidable via f et `coA` est semi-décidable via g alors cet algorithme décide `A` :

```
1 Exécuter f(x) pendant une unité de temps
2 Si on a obtenu "oui", renvoyer oui
3 Sinon exécuter g(x) pendant une unité de temps
4 Si on a obtenu "oui", renvoyer "non"
5 Sinon, recommencer à partir de la ligne 1
```

2. Non. `ARRET` est semi-décidable donc si `coARRET` l'était, par la question 1, `ARRET` le serait.
3. Non. S'il l'était via une fonction OCaml `arret_pour_tout` alors cette fonction déciderait l'arrêt :

```
let arret f x = arret_pour_tout (fun y -> f x)
```

4. Non. S'il était semi-décidable via `coarret_pour_tout` alors la fonction suivante semi-déciderait le problème `coARRET` qui n'est pas semi-décidable par la question 2 :

```
let coarret f x = coarret_pour_tout (fun y -> f x)
```

5. Non plus. Si c'était le cas via une fonction `arret_pour_tout` on pourrait semi-décider `coARRET` via l'algorithme suivant (prenant en entrée (f, x) une instance de `coARRET`) :

```
Définir une fonction g prenant en entrée un entier n telle que :
  g exécute les n premières étapes du calcul de f(x)
  Si le calcul de f(x) a terminé à l'issue de ces n étapes, boucler
  Sinon, terminer
renvoyer arret_pour_tout g
```

Exercice 121 (**) *Arbres couvrants à arité bornée*

On rappelle que le problème `CHO` (cycle hamiltonien orienté) est NP-complet. On considère les problèmes `ARBRE COUVRANT`, `ARBRE COUVRANT BORNE` et `CYCLE HAM` (cycle hamiltonien) définis respectivement par :

$$\begin{cases} \textbf{Entrée :} \text{ Un graphe pondéré } G = (S, A, p) \text{ et un seuil } P \in \mathbb{N}. \\ \textbf{Question :} \text{ Existe-t-il un arbre couvrant } T \text{ de } G \text{ tel que } p(T) \leq P \,? \end{cases}$$

$$\begin{cases} \textbf{Entrée :} \text{ Un graphe pondéré } G = (S, A, p), \text{ un seuil } P \in \mathbb{N} \text{ et un entier } k. \\ \textbf{Question :} \text{ Existe-t-il un arbre couvrant } T \text{ de } G \text{ tel que } p(T) \leq P \text{ et dont les noeuds sont de degré au plus } k \,? \end{cases}$$

$$\begin{cases} \textbf{Entrée :} \text{ Un graphe } G \text{ non orienté.} \\ \textbf{Question :} \text{ Existe-t-il un cycle hamiltonien dans } G \,? \end{cases}$$

1. Dans quelle classe de complexité déterministe se trouve le problème `ARBRE COUVRANT` ? Justifier.

Si $G = (S, A)$ est une instance de `CHO`, on considère le graphe non orienté $G' = (S', A')$ défini par :

- $S' = \{s^{(1)} \mid s \in S\} \cup \{s^{(2)} \mid s \in S\} \cup \{s^{(3)} \mid s \in S\}$; autrement dit, on détriple les sommets de G.
- $A' = \{(s^{(1)}, s^{(2)}) \mid s \in S\} \cup \{(s^{(2)}, s^{(3)}) \mid s \in S\} \cup \{(s^{(3)}, t^{(1)}) \mid (s, t) \in A\}$.

2. En utilisant cette construction, montrer que `CYCLE HAM` est NP-difficile.
3. On admet que ceci implique que `CHEMIN HAM`, vérifiant l'existence d'un *chemin* hamiltonien dans un graphe non orienté, est un problème NP-complet. En déduire que `ARBRE COUVRANT BORNE` est NP-complet aussi.

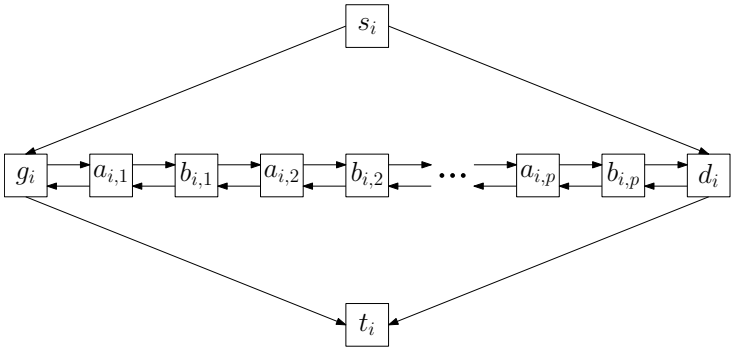
1. Il est dans P via l'algorithme (polynomial) de Kruskal.
2. La construction de G' se fait en temps polynomial en $|G|$. Si $s_1, ..., s_n$ est un cycle hamiltonien dans G alors $s_1^{(1)}, s_1^{(2)}, s_1^{(3)}, s_2^{(1)}, ..., s_n^{(3)}$ est un cycle hamiltonien dans G' (parcours des sommets dans G' dans l'ordre $1, 2, 3$). Réciproquement, si C est un cycle dans G', comme il est non orienté on peut supposer que le parcours de s_1, s_2, s_3 se fait dans l'ordre $1, 2, 3$ ce qui donne un cycle dans G en fusionnant ces trois sommets.
3. Si $G = (S, A)$ est une instance de `CHEMIN HAM`, (G pondéré par 1 partout, $|S| - 1, 2$) est une instance de `ARBRE COUVRANT BORNE`. Cette construction montre que `CHEMIN HAM` $\leq$ `ARBRE COUVRANT BORNE` ce qui montre la NP-difficulté et le caractère NP est évident.

Exercice 122 (**) *Cycle hamiltonien orienté*

1. Rappeler la définition du problème de décision `CHO` et montrer qu'il est dans NP.

On veut démontrer que ce problème est NP-difficile par réduction depuis `3SAT`. Soit φ une formule instance de `3SAT` comportant n variables et p clauses. Pour chaque variable x_i de φ, on définit le graphe $G_i = (V_i', E_i')$ par :

- Pour chaque clause C_j de φ, G_i contient un sommet étiqueté $a_{i,j}$ et un sommet étiqueté $b_{i,j}$. On rajoute à ces sommets quatre sommets d_i, g_i, s_i et t_i.
- Les arcs de G_i sont représentés ci-dessous :



On définit ensuite le graphe $G = (V, E)$ tel que $V = V_1' \cup ... \cup V_n' \cup \{s, t\}$ et

$$E = E_1' \cup ... \cup E_n' \cup \{(t_i, s_{i+1}) \mid 1 \leq i < n\} \cup \{(s, s_1), (t_n, t), (t, s)\}$$

2. Dessiner schématiquement le graphe G.
3. Combien y a-t-il de cycles hamiltoniens dans G ?
4. Définir un graphe H à partir du graphe G en y ajoutant pour chaque clause de φ un sommet c_j et deux arcs choisis de telle sorte à ce que H admette un cycle hamiltonien si et seulement si φ est satisfiable.
5. Conclure quant à la NP-complétude de `CHO`.

1. RAS.
2. RAS.
3. La chaîne centrale de chaque G_i peut être parcourue de gauche à droite ou de droite à gauche et il y autant de G_i que le nombre n de variables donc 2^n.
4. Moralement chaque cycle hamiltonien de G correspond à une valuation (parcourir G_i de gauche à

droite ↔ fixer x_i à vrai / parcourir G_i de droite à gauche ↔ fixer x_i à faux). Pour chaque littéral l_i de C_j, on ajoute les arcs (a_{ij}, c_j) et (c_j, b_{ij}) si l_i est positif et (b_{ij}, c_j) et (c_j, a_{ij}) s'il est négatif (si un littéral apparaît positivement et négativement dans une clause on la supprime).

5. Construire G' se fait en temps polynomial en n et k donc en $|\varphi|$. Si v satisfait φ, on parcourt G_i de gauche à droite si $v(x_i) = 1$ et de droite à gauche si $v(x_i) = 0$ en faisant un détour par chaque c_j non encore visité (et on peut car pour chaque clause au moins un de ses littéraux est rendu vrai par v). Réciproquement, un chemin hamiltonien de H donne une valuation satisfaisant φ déduite des sens de parcours des G_i. Cela montre la NP-difficulté et la question 1 conclut.

Exercice 123 (***) *Couvertures exactes*

Le problème `EXACT COVER` (couverture exacte) est le problème de décision suivant :

$$\begin{cases} \textbf{Entrée :} \text{ Une matrice } M \in M_{n,p}(\{0,1\}). \\ \textbf{Question :} \text{ Existe-t-il un ensemble de lignes de } M \text{ tel que la matrice formée par ces lignes ait} \\ \qquad\qquad \text{exactement un 1 dans chaque colonne ?} \end{cases}$$

1. Montrer que `EXACT COVER` peut être défini de manière équivalente par :

$$\begin{cases} \textbf{Entrée :} \text{ Un ensemble fini } U \text{ et une collection } S \text{ de sous-ensembles de } U. \\ \textbf{Question :} \text{ Existe-t-il } S' \subset S \text{ tel que } U = \bigsqcup_{X \in S'} X \text{ ?} \end{cases}$$

   Le problème ci-dessus est donc lui aussi appelé `EXACT COVER`.

2. On considère l'instance suivante de `EXACT COVER` : $U = [\![1, 7]\!]$ et $S = \{A, B, C, D, E, F\}$ avec :

   - $A = \{3, 5, 6\}$
   - $B = \{1, 4, 7\}$
   - $C = \{2, 3, 6\}$
   - $D = \{1, 4\}$
   - $E = \{2, 7\}$
   - $F = \{4, 5, 7\}$

   Donner la version matricielle de cette instance et indiquer si elle est positive.

3. Considérons la formule $\varphi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$ avec :

$$C_1 = x_1 \vee \neg x_2 \qquad C_2 = \neg x_1 \vee x_2 \vee x_3 \qquad C_3 = x_2 \qquad C_4 = \neg x_2 \vee \neg x_3$$

   On y associe l'instance de `EXACT COVER` (U, F) suivante :

$$U = \{x_1, x_2, x_3, C_1, p_{11}, p_{12}, C_2, p_{21}, p_{22}, p_{23}, C_3, p_{31}, C_4, p_{41}, p_{42}\}$$

   et la collection S contient tous les sous-ensembles suivants :

$$\{p_{11}\}, \{p_{12}\}, \{p_{21}\}, \{p_{22}\}, \{p_{23}\}, \{p_{31}\}, \{p_{41}\}, \{p_{42}\}$$
$$\{C_1, p_{11}\}, \{C_1, p_{12}\}, \{C_2, p_{21}\}, \{C_2, p_{22}\}, \{C_2, p_{23}\}, \{C_3, p_{31}\}, \{C_4, p_{41}\}, \{C_4, p_{42}\}$$
$$E_{1,F} = \{x_1, p_{11}\}, E_{1,V} = \{x_1, p_{21}\}$$
$$E_{2,F} = \{x_2, p_{22}, p_{31}\}, E_{2,V} = \{x_2, p_{12}, p_{41}\}$$
$$E_{3,F} = \{x_3, p_{23}\}, E_{3,V} = \{x_3, p_{42}\}$$

   Donner l'ensemble des valuations qui satisfait φ et l'ensemble des solutions associées à (U, S).

4. En s'inspirant du procédé ci-dessus, montrer que `EXACT COVER` est NP-complet.

1. Comme U est fini, on peut en numéroter les éléments en $u_1, ..., u_{|U|}$ et alors un sous ensemble X de U est représentable par un tableau de taille $|U|$ contenant un 1 en case i si $u_i \in X$ et 0 sinon. Se donner U et S revient donc à se donner une matrice $M \in M_{|S|,|U|}(\{0,1\})$ et par construction, (U, S) est une instance positive de `EXACT COVER` version ensembliste si et seulement si M est positive pour `EXACT COVER` version matricielle.

2. La version matricielle de cette instance (en écrivant les lignes dans l'ordre alphabétique) est :

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

   Cette instance est positive car $\{A, D, E\}$ est une couverture exacte de U.

3. Si v est une valuation satisfaisant φ, elle doit satisfaire C_3 donc $v(x_2) = V$ et comme C_1 et C_4 doivent aussi être satisfaites, on a nécessairement $v(x_1) = V$ et $v(x_3) = F$. Réciproquement, cette valuation satisfait bien φ. On constate par ailleurs que la seule couverture de U est :

$$E_{1,V}, E_{2,V}, E_{3,F}, \{C_1, p_{11}\}, \{C_2, p_{22}\}, \{C_3, p_{31}\}, \{C_4, p_{42}\}$$

4. Soit $\varphi = C_1 \wedge C_2 \wedge ... \wedge C_r$ une instance de `CNF-SAT`. Chaque clause C_j est disjonctive et vaut donc $l_{j1} \vee ... \vee l_{jr_j}$ où les l_{ji} sont les r_j littéraux de C_j. Alors en parcourant cette formule (donc en temps polynomial en sa taille), on peut construire l'instance de `EXACT COVER` suivante :

   - $U = \{\text{variables de } \varphi\} \cup \{\text{clauses de } \varphi\} \cup \{p_{jk} \,|\, 1 \leq j \leq r, 1 \leq k \leq m_j\}$. Le dernier ensemble consiste à ajouter un élément à U pour chaque occurrence de variable dans φ en l'indiçant par la clause dont il provient et la position du littéral qu'il représente dans cette clause.
   - On place dans S les sous-ensembles de U suivants :
     - Tous les singletons $\{p_{jk}\}$.
     - Toutes les paires de la forme $\{C_j, p_{jk}\}$.
     - Pour chaque variable x_i, l'ensemble $E_{i,V} = \{x_i\} \cup \{p_{jk} \,|\, l_{jk} = \neg x_i\}$ contenant x_i et toutes les occurrences négatives de x_i et l'ensemble $E_{i,F} = \{x_i\} \cup \{p_{jk} \,|\, l_{jk} = x_i\}$ contenant x_i et toutes les occurrences positives de x_i.

   Il reste à montrer que φ est satisfiable si et seulement si (U, S) est positive pour `EXACT COVER`. Si φ est satisfaite par v, on construit une X couverture de U comme suit :

   - On couvre x_i avec $E_{i,V}$ si $v(x_i) = V$ et avec $E_{i,F}$ sinon. Cela couvre certaines des positions p_{jk} mais jamais deux fois la même car à une position donnée il n'y a qu'une seule variable.
   - On couvre chaque C_j avec un des $\{C_j, p_{jk}\}$ tel que $v(l_{jk}) = V$. Il existe un tel ensemble puisque chaque clause est satisfaite par v donc contient au moins un littéral satisfait. De plus on ne peut pas couvrir deux fois le même p_{jk} avec cette étape puisque les clauses ne partagent pas de position et p_{jk} n'était pas couvert par l'étape précédente, sinon il est dans un des $E_{i,V}$ et alors c'est la position d'une occurrence négative de x_i avec $v(x_i) = V$ donc n'est pas la position d'un littéral satisfait de C_j. Idem si p_{jk} est dans l'un des $E_{i,F}$.
   - Il ne reste plus qu'à couvrir les positions qui ne le sont pas encore avec des singletons $\{p_{jk}\}$.

   Réciproquement, si on dispose d'une couverture exacte X de U, alors les variables sont couvertes, donc pour chaque variable x_i, on a soit $E_{i,V} \in X$, soit $E_{i,F} \in X$ mais pas les deux. On peut donc construire la valuation v telle que $v(x_i) = V$ si c'est $E_{i,V}$ qui est dans X et $v(x_i) = F$ sinon.

   Cette valuation satisfait φ. Soit en effet une clause C_j. Elle est couverte par X donc vu la forme des sous-ensembles de S, il y a un unique ensemble de la forme $\{C_j, p_{jk}\}$ dans X. Alors le littéral en position p_{jk} dans C_j est satisfait par v. Sinon, il serait égal à x_i avec $v(x_i) = F$ ou à $\neg(x_i)$ avec $v(x_i) = V$. Dans le premier cas, cela voudrait dire que $E_{i,F}$ est dans X mézalors p_{jk} est dans cet ensemble puisque c'est la position d'une occurrence positive de x_i donc p_{jk} serait couvert deux fois. C'est impossible, et le second cas se traite de la même façon.

5 Grammaires algébriques

Exercice 124 (*) *Du langage à la grammaire*

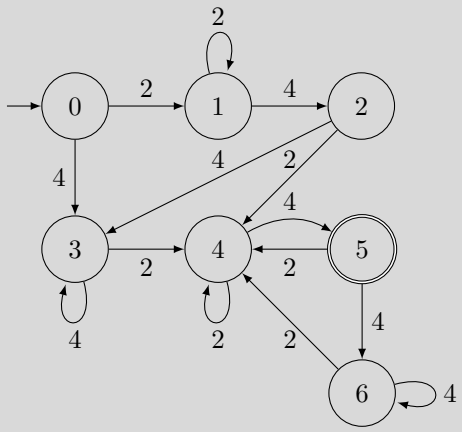
Dans chacun des cas suivants, décrire une grammaire engendrant le langage donné. On ne demande pas de preuve formelle du fait que la grammaire proposée engendre effectivement le langage considéré.

1. $L_1 = a^\star bc^\star$.
2. L_2 est le langage sur $\{0,1\}$ des mots qui commencent et finissent par la même lettre.
3. L_3 est le langage sur $\{2,4\}$ des mots qui terminent par 24 et contiennent 42.
4. L_4 est le langage sur $\{a,b\}$ des mots ayant autant de a que de b.
5. L_5 est le langage sur $\{a,=,+\}$ des mots représentant une addition correcte de deux suites de caractères a. Par exemple, $aa + aaa = aaaaa \in L_5$ mais $a + a = a$ ou $a + a + a = aaa$ n'en font pas partie.
6. L_6 est le langage dont les mots sont les listes de chiffres bien formées en OCaml. Par exemple, `[]`, `[0;1]` et `[5;4;1;1;2]` sont des mots de L_6 mais `[10;1]`, `[1;]` ou `[-1,5]` n'en font pas partie.
7. L_7 est le langage dont les mots sont les objets que l'on peut construire à l'aide du type récursif :

```
type arbre_binaire = Vide | Feuille | Noeud of arbre_binaire * arbre_binaire
```

   Par exemple, `Vide` et `Noeud(Noeud(Feuille, Vide), Feuille)` sont des mots de L_7 alors que les mots `Noeud(Noeud,Noeud)` ou `(Noeud(Feuille, Vide))` n'en font pas partie.
8. L_8 est le langage des formules du calcul propositionnel sur l'ensemble de variables $\{p,q,r\}$.

1. $S \to AbC$, $A \to aA \,|\, \varepsilon$ et $C \to cC \,|\, \varepsilon$ convient.
2. $S \to 0X0 \,|\, 1X1 \,|\, 0 \,|\, 1$ et $X \to 0X \,|\, 1X \,|\, \varepsilon$ convient.
3. On peut s'appuyer sur un automate qui reconnaît L_3 en construisant l'intersection d'un automate (des occurrences) pour $(2+4)^\star 24$ et d'un pour $(2+4)^\star 42 (2+4)^\star$. On obtient :



   D'où la grammaire d'axiome X_0 suivante : $X_0 \to 2X_1 \,|\, 4X_3$, $X_1 \to 2X_1 \,|\, 4X_2$, $X_2 \to 2X_4 \,|\, 4X_3$, $X_3 \to 4X_3 \,|\, 2X_4$, $X_4 \to 2X_4 \,|\, 4X_5$, $X_5 \to 2X_4 \,|\, 4X_6 \,|\, \varepsilon$ et $X_6 \to 2X_4 \,|\, 4X_6$.
4. $S \to aB \,|\, bA \,|\, \varepsilon$, $A \to a \,|\, aS \,|\, bAA$ et $B \to b \,|\, bS \,|\, aBB$ convient. On montre en effet que A génère les mots qui ont un a de plus que de b et B génère les mots qui ont un b de plus que de a.
5. Ce langage est $\{a^n + a^m = a^{m+n} \,|\, n,m \in \mathbb{N}\}$. On propose $S \to a + Ma \,|\, aSa$, $M \to aMa \,|\, a = a$:
   - La première règle génère le dernier a des a^n à droite et à gauche.
   - La seconde génère les a^{n-1} lettres a à gauche comme a droite.
   - La troisième génère a^{m-1} des a de chaque côté.
   - La dernière génère le dernier a des a^m à gauche et le premier à droite.
6. On propose la grammaire suivante sur $\Sigma = [\![0,9]\!] \cup \{;,[,]\}$: $S \to [\,] \,|\, [L]$, $L \to C \,|\, C;L$, $C \to 0 \,|\, 1 \,|\, \ldots \,|\, 9$.
7. On propose la grammaire suivante sur $\Sigma = \{(,),,,F,e,u,i,l,e,V,d,N,o\}$:

$$S \to Noeud(A) \,|\, Vide \,|\, Feuille \qquad A \to S,S$$

8. On propose $S \to B \,|\, \neg S \,|\, (S \vee S) \,|\, (S \wedge S) \,|\, (S \to S) \,|\, (S \leftrightarrow S)$ et $B \to \top \,|\, \bot \,|\, p \,|\, q \,|\, r$.

De manière générale, les objets syntaxiques construits inductivement sont facile à générer par grammaire.

Exercice 125 (*) *De la grammaire au langage*

Dans chacun des cas suivants, déterminer si ce n'est déjà fait le langage L engendré par la grammaire G proposée et montrer rigoureusement que $L = L(G)$.

1. G_1 est la grammaire d'axiome S sur $\{a, b\}$ dont les règles sont $S \to SaS \,|\, b$.
2. G_2 est la grammaire d'axiome S et dont les règles sont $S \to aSa \,|\, bSb \,|\, \varepsilon$. Montrer que $L(G_2)$ est exactement l'ensemble des palindromes de longueur paire sur $\{a, b\}$.
3. G_3 est la grammaire décrite par $S \to aSS \,|\, b$. Montrer que $L(G_3)$ est l'ensemble des mots u tels que $|u|_a = |u|_b - 1$ et pour tout préfixe strict v de u (c'est-à-dire, $v \neq u$), $|v|_b \leq |v|_a$.

Culture générale : Le langage de la dernière question est le langage de Łukasiewicz. C'est le langage des expressions préfixes à un opérateur bianire (ici noté a). Il entretient des rapports étroits avec le langage de Dyck.

1. Notons $L_1 = (ba)^\star b$ et montrons que $L(G_1) = L_1$ par double inclusion.

   Montrons par récurrence forte sur $n \in \mathbb{N}^\star$ que tout mot dérivé de S en n étapes est dans L_1. La seule dérivation de longueur 1 dans G_1 est $S \Rightarrow b$ et $b \in L_1$. De plus, si $S \Rightarrow^{n+1} w$ alors on a nécessairement $S \Rightarrow SaS \Rightarrow^n w$. On en déduit qu'il existe $n_1, n_2 \leq n$ et $w_1, w_2 \in \{a, b\}^\star$ tels que $n_1 + n_2 = n$, $w = w_1 a w_2$, $S \Rightarrow^{n_1} w_1$ et $S \Rightarrow^{n_2} w_2$. L'hypothèse de récurrence s'applique à ces deux dérivations donc $w_1, w_2 \in L_1$ donc $w_1 = (ba)^{k_1} b$ et $w_2 = (ba)^{k_2} b$ puis $w = w_1 a w_2 = (ba)^{k_1+k_2+1} b \in L_1$.

   Réciproquement, montrons par récurrence forte sur $n \in \mathbb{N}^\star$ que tout mot de L_1 de taille n est dans $L(G_1)$. Le seul mot de longueur 1 dans L_1 est b qui est bien engendré par G_1. Si $|w| = n + 1$, comme $|w| > 1$, il existe $k \in \mathbb{N}^\star$ tel que $w = (ba)^k b$ et alors $w' = (ba)^{k-1} b \in L_1$ donc par hypotèse de récurrence, $S \Rightarrow^\star w'$ puis $S \Rightarrow SaS \Rightarrow baS \Rightarrow^\star baw' = w$.

2. Même schéma de preuve qu'à la question précédente, il faut juste distinguer selon la première lettre du mot étudié (soit a, soit b) par endroits.

3. Même principe mais l'une des inclusions est un peu plus délicate. L'inclusion facile est $L(G_3) \subset Ł$, ce qu'on montre par récurrence sur la longueur des dérivations. Si $S \Rightarrow^{n+1} w$ alors $S \Rightarrow aSS \Rightarrow^n w$ et donc $w = aw_1w_2$ avec $w_1, w_2 \in Ł$ par hypothèse de récurrence. Alors :

   - $|w|_a = 1 + |w_1|_a + |w_2|_a = 1 + |w_1|_b - 1 + |w_2|_b - 1 = |w|_b - 1$.
   - Si v est préfixe strict de w alors :
     - Soit $v = \varepsilon$ et $|v|_a \geq |v|_b$.
     - Soit $v = av_1$ avec v_1 préfixe strict de w_1 et alors $|v|_a = 1 + |v_1|_a \geq 1 + |v_1|_b \geq |v|_b$.
     - Soit $v = aw_1v_2$ avec v_2 préfixe strict de w_2 et $|v|_a = 1+|w_1|_a+|v_2|_a \geq 1+|w_1|_b-1+|v_2|_b = |v|_b$.

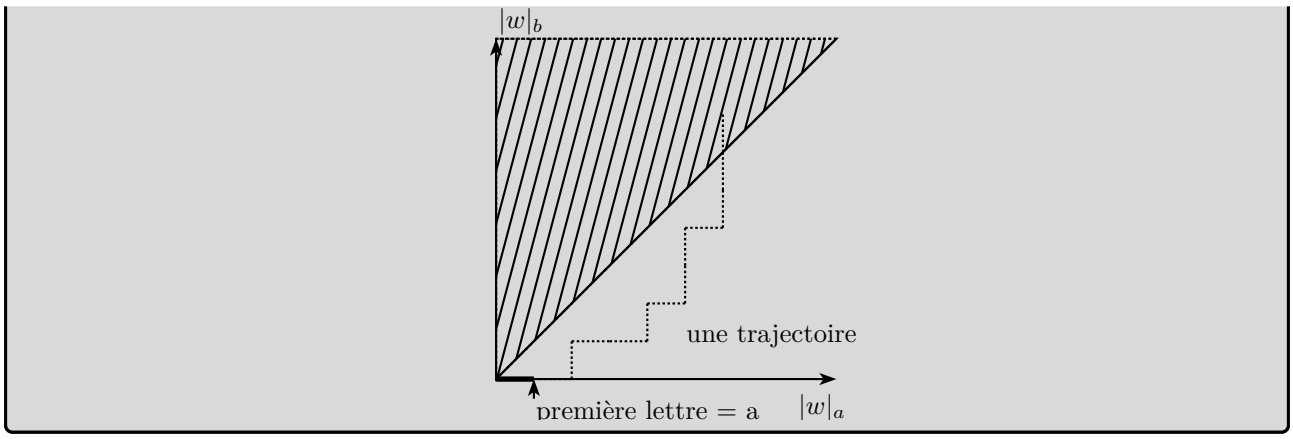
   Donc w est bien un mot de Łukasiewicz.

   Réciproquement, on procède par récurrence sur la taille des mots de Ł. Si $w \in Ł$ est de taille $n+1$, sa taille est au moins deux donc sa première lettre en est préfixe strict donc doit avoir plus de a que de b. Ainsi, w commence par a. Notons w_1 le plus petit mot tel que aw_1 est préfixe de w et ayant autant de a que de b. Ce mot existe car w doit avoir moins de a que de b strictement et commence par a et :

   - $|aw_1|_a = |aw_1|_b$ donc $|w_1|_a = |w_1|_b - 1$.
   - Pour tout préfixe strict v de w_1, $|av|_a > |av|_b$ par minimalité de w_1 donc $|v|_a \geq |v|_b$.

   On en déduit que w_1 est un mot de Łukasiewicz. Mézalors écrivons $w = aw_1w_2$. Comme $w \in Ł$, $|w|_a = |w|_b - 1$ puis $1 + |w_1|_a + |w_2|_a = |w_1|_b + |w_2|_b - 1$. Mais comme $w_1 \in Ł$, $|w_2|_a = |w_2|_b - 1$. De plus, si v est préfixe strict de w_2 alors aw_1v est préfixe strict de w donc $|aw_1v|_a \geq |aw_1v|_b$ donc en utilisant le fait que $w_1 \in Ł$, $|v|_a \geq |v|_b$ et finalement $w_2 \in Ł$. Par hypothèse de récurrence on peut donc engendrer w_1 et w_2 à partir de S, donc la dérivation $S \Rightarrow aSS \Rightarrow^\star aw_1w_2 = w$ montre que $w \in L(G_3)$.

   Remarque : l'idée de considérer w_1 vient de l'observation de l'évolution du nombre de a et de b quand on construit un mot de Łukasiewicz : sa première lettre est a et à la fin le mot contiendra plus de b que de a, donc qu'on sera dans la partie hachurée, il est donc nécessaire qu'on traverse la diagonale :



Exercice 126 (**) *Grammaires et langages naturels*

On considère deux grammaires dont l'axiome est P dans les deux cas : G_1 permet de produire quelques phrases en anglais et G_2 fait de même en français. Les règles de ces grammaires sont données ci-dessous ; les majuscules correspondent aux variables et les mots en minuscules sont les symboles non terminaux :

Règles de G_1

$P \to NV$
$N \to NP$
$N \to$ the dog | the stick | the fire
$V \to$ burned | bit | beat

Règles de G_2

$P \to SVC$
$S \to N \mid NC$
$C \to$ avec $N \mid \varepsilon$
$V \to WN$
$N \to$ elle | une femme | un téléscope
$W \to$ voit

1. On considère le mot suivant engendré par la grammaire G_1 : the dog the stick the fire burned beat bit
   a) Donner une dérivation droite pour ce mot.
   b) Donner une dérivation gauche pour ce mot.
   c) Donner un arbre syntaxique pour ce mot. Comment traduirait-on cette phrase en français ?
2. On considère le mot suivant engendré par la grammaire G_2 : elle voit une femme avec un téléscope
   a) Donner deux sens possibles à cette phrase.
   b) Montrer que G_2 est une grammaire ambiguë.

Culture générale : De manière générale, les grammaires algébriques ne sont pas tout à fait adaptées à la description d'une langue naturelle. Chomsky lui-même convenait de cette limite et proposa pour la lever la notion de grammaire transformationnelle.

Exercice à refondre

Exercice 127 (**) *Désambiguation*

1. On cherche à construire une grammaire destinée à écrire des expressions arithmétiques construites à partir de l'opérateur binaire de soustraction et dont les opérandes sont des entiers qu'on note génériquement e (autrement dit, e peut être dérivé en n'importe quel entier). On considère pour ce faire la grammaire G_1 décrite par : $S \to S - S \mid e$.
   a) À l'aide du mot $m = 10 - 2 - 3$, montrer que G_1 est ambiguë. Pourquoi est-ce gênant ?
   b) On considère donc plutôt la grammaire G_2 décrite par les règles $S \to S - T \mid T$ et $T \to e$. Expliquer pourquoi G_1 et G_2 engendrent le même langage.
   c) Dessiner l'arbre syntaxique du mot m selon G_2. Cette grammaire est-elle ambiguë ?
2. Dans cette question, on considère une grammaire décrivant les instructions d'un langage de programmation dont l'axiome est S, les non terminaux sont S et I et dont les règles sont :

   $S \to I$
   $I \to$ if e then I | if e then I else I | a

a) Montrer que cette grammaire est ambiguë.

b) Proposer une solution pour rendre cette grammaire non ambiguë. Est-il possible de rendre cette grammaire non ambiguë sans modifier le langage engendré ?

1. a) On trouve deux arbres selon qu'on dérive le S à gauche ou le S à droite en $S-S$. L'un correspond à la sémantique $(10-2)-3=5$ et l'autre à la sémantique $10-(2-3)=11$, qui sont différentes.

   b) On peut toujours faire autant de soustractions qu'on veut, sauf qu'on construit d'abord l'emplacement des opérandes avec $S \to S-T$ puis on remplace tous les T par e.

   c) On trouve un seul arbre de sémantique $(10-2)-3$: on a forcé l'associativité à gauche de $-$.

2. a) Le mot : if e then if e then a else a admet deux arbres de dérivations (problème du dangling-else).

   b) On peut introduire du parenthésage (mais ça change le langage) où s'inspirer de la question précédente en distinguant le if-then-else du if-then et en interdisant le then d'un un if-then-else à être un if-then : $I \to$ if e then $I \mid J$ et $J \to a \mid$ if e then J else I.

Exercice 128 (**) *Ambiguité et langages rationnels*

1. On considère la grammaire G d'axiome S et dont les règles sont

$$S \to abA \mid AbA \text{ et } A \to \varepsilon \mid aA$$

   a) Quel est le langage engendré par G ?

   b) Montrer que G est ambiguë.

   c) Exhiber une grammaire non ambiguë qui engendre le langage $L(G)$.

2. Montrer de manière générale qu'un langage rationnel n'est jamais inhéremment ambigu, autrement dit, montrer que pour tout langage rationnel, il existe une grammaire non ambiguë qui l'engendre.

1. a) Il s'agit de $a^\star ba^\star$, ce qu'on prouve par double inclusion en montrant tout d'abord que $L(G,A)=a^\star$.

   b) Le mot aba admet deux arbres de dérivation, selon qu'on commence par $S \to abA$ ou $S \to AbA$.

   c) $S \to AbA$ et $A \to \varepsilon \mid aA$ engendre ce langage de manière non ambiguë.

2. Il suffit de dériver ladite grammaire d'un automate déterministe reconnaissant le langage.

Exercice 129 (**) *Théorème de lecture unique*

Le but de cet exercice est de montrer :

Résultat : Théorème de lecture unique

Soit F une formule du calcul propositionnel sur un ensemble de variables V. Alors, on est dans un et un seul des cas suivants :

- F est égale à un et un seul des éléments de $V \cup \{\top, \bot\}$.
- Il existe une unique formule G telle que $F = \neg G$.
- Il existe un unique couple (G,H) et un unique symbole $\alpha \in \{\vee, \wedge \to, \leftrightarrow\}$ tel que $F = (G \alpha H)$.

1. Montrer que pour toute formule F, on a $|F|_(= |F|_)$.
2. Montrer que pour tout préfixe u d'une formule F, $|u|_(\geq |u|_)$ et que l'inégalité est stricte si le premier symbole de F est (et que u est propre (non égal ni à F ni à ε).
3. Montrer qu'un préfixe propre d'une formule n'est pas une formule.
4. Montrer le théorème de lecture unique.
5. Le langage des formules du calcul propositionnel sur V est-il (intrinsèquement) ambigu ?

Remarque : On comprend mieux pourquoi on peut se permettre de parler de l'arbre syntaxique d'une formule du calcul propositionnel plutôt que d'UN arbre syntaxique : on vient de montrer qu'il n'y en a qu'un. Remarquez aussi que le vocabulaire est bien fait : l'arbre syntaxique d'une formule est un arbre syntaxique.

1. Se montre par induction sur l'ensemble des formules $\mathcal{F}$.
2. La première partie se montre par induction sur $\mathcal{F}$. Pour la deuxième partie, on est dans le cas $F = (G\alpha H)$ et si u est préfixe strict de F alors on distingue selon que $u = (v$ avec v préfixe de G ou $u = (G\alpha v$ avec v préfixe strict de H.
3. Encore une induction sur $\mathcal{F}$ en utilisant Q2 et Q1 pour justifier qu'un préfixe propre de $(G\alpha H)$ a trop de parenthèses ouvrantes par rapport aux fermantes pour pouvoir être une formule.
4. Observer la première lettre de $F \in \mathcal{F}$ montre que les trois cas sont mutuellement exclusifs. De plus :
   - Si $F \in V \cup \{\top, \bot\}$, comme $|F| = 1$ il y a unicité du symbole égale à F.
   - Si F est de la forme $\neg G$ et que $\neg G = \neg H$, par régularité $G = H$.
   - Si $F = (G\alpha H) = (G'\alpha' H')$, sans perte de généralité, G' est préfixe de G. Il ne peut pas être égal à ε par construction de la syntaxe et ne peut pas être strict par Q3 donc est égal à G puis on en déduit que $\alpha = \alpha'$ et $H = H'$ par régularité.
5. Non puisque la grammaire naturelle qui l'engendre (appuyée sur la définition inductive de $\mathcal{F}$) n'est pas ambiguë d'après le théorème de lecture unique.

Exercice 130 (***) *Lemme d'Ogden et applications*

On admet dans cet exercice le lemme d'Ogden :

Résultat : Lemme d'Ogden

Soit $G = (\Sigma, V, S, \mathcal{R})$ une grammaire algébrique et $X \in V$. Alors il existe un entier N tel que tout mot $m \in L(G, X)$ ayant au moins N lettres marquées se factorise en $m = xuyvz$ avec :

1. $X \Longrightarrow^* xAz$, $A \Longrightarrow^* uAv$ et $A \Longrightarrow^* y$ avec $A \in V$
2. (x, u et y contiennent des lettres marquées) ou (y, v et z contiennent des lettres marquées).
3. uyv contient moins de N lettres marquées.

1. À l'aide du lemme d'Ogden, prouver le lemme d'itération vu en cours facultatif.
2. Considérons le langage $L = \{a^n b^n c^m \,|\, n, m \in \mathbb{N}\} \cup \{a^n b^m c^m \,|\, n, m \in \mathbb{N}\}$.
   a) Montrer que L est algébrique.
   b) A l'aide du lemme d'Ogden, montrer que L est intrinsèquement ambigu. *Indication : Considérer le mot $a^N b^N c^{N+N!}$ avec N donné par le lemme d'Ogden et où tous les b sont marqués et montrer que $a^{N+N!} b^{N+N!} c^{N+N!}$ admet deux arbres de dérivation différents.*

Culture générale : Parfois, le terme "lemme d'itération" réfère au lemme d'Ogden. La démonstration du lemme d'Ogden (difficile) se trouve en page 100 de Langages formels, calculabilité et complexité par Olivier Carton.

1. Le lemme d'itération est un cas particulier du lemme d'Ogden où on marque toutes les lettres. En effet, dans ces conditions, ce lemme dit qu'il existe $N \in \mathbb{N}$ tel que tout mot $m \in L(G)$ de longueur au moins N se factorise en $m = xuvyw$ avec :
$$\begin{cases} S \Rightarrow^\star xAz, A \Rightarrow^\star uAv \text{ et } A \Rightarrow^\star y \\ \text{soit } u \text{ soit } v \text{ contient une lettre donc } |uv| > 0 \\ |uyv| \leq N \end{cases}$$
La première condition implique que pour tout $n \in \mathbb{N}$, $S \Rightarrow^\star xu^n yv^n z$ donc que ce mot est dans $L(G)$.
2. a) L est l'union de deux langages algébriques donc est algébrique. Le premier morceau par exemple est engendré par $S \to EC$, $E \to \varepsilon \,|\, aEb$, $C \to \varepsilon \,|\, cC$.
   b) Soit G une grammaire engendrant L. Alors elle engendre $a^N b^N c^{N+N!}$ et si on marque tous les b, on en déduit que ce mot se factorise en $xuyvz$ tel que pour tout $n \geq 0$, $xu^n yv^n z \in L$ ce qui contraint le fait que $u = a^i$ et $v = b^i$ avec $1 \leq i \leq N$. On a donc $x = a^j$, $u = a^i$, $y = a^{N-i-j} b^k$, $v = b^i$, $z = b^{N-i-k} c^{N+N!}$. En itérant $1 + N!/i \in \mathbb{N}$, on construit le mot $a^{N+N!} b^{N+N!} c^{N+N!}$ via un arbre qui contient un sous arbre dont la frontière est $a^{N!+N-j} b^{N!+i+k}$.

En appliquant le même raisonnement à $a^{N+N!}b^Nc^N$, on obtient un arbre pour $m = a^{N+N!}b^{N+N!}c^{N+N!}$ qui contient un sous arbre dont la frontière est de la forme $b^\beta c^\gamma$ avec $\beta \geq N! + 1$.

On déduit (TODO : check) de la place que prennent ces sous arbres qu'ils ne peuvent pas cohabiter dans un même arbre de dérivation pour m donc qu'on vient d'exhiber deux arbres pour m contenant deux sous-arbres différents donc deux arbres différents ce qui montre l'ambiguité de G.

Exercice 131 (exo cours) *Grammaire pour un langage bien connu*

1. Déterminer le langage engendré par la grammaire G d'axiome S dont les règles sont $S \to aSb \,|\, \varepsilon$.
2. Prouver rigoureusement que le langage proposé est bien égal à $L(G)$.

1. Cette grammaire engendre $L = \{a^n b^n \,|\, n \in \mathbb{N}\}$.
2. Montrons par récurrence sur $n \in \mathbb{N}^\star$ que tout mot u tel que $S \Rightarrow^n u$ est bien dans L. L'initialisation est acquise car $\varepsilon \in L$. Si $S \Rightarrow^{n+1} u$ alors nécessairement $S \Rightarrow aSb \Rightarrow^n u$ sinon on aurait une dérivation trop courte. Donc il existe un mot v tel que $S \Rightarrow^n v$ et $u = avb$. On applique l'hypothèse de récurrence à v, qui assure que $v = a^k b^k$ et donc $u = a^{k+1}b^{k+1} \in L$.

   Réciproquement, montrons par récurrence forte sur $n \in \mathbb{N}$ que tout mot de L de taille n est engendré par G. Le seul mot de L de taille 0 l'est bien. De plus, si $u \in L$ est de taille $n+1$ alors il existe $k \in \mathbb{N}^\star$ tel que $u = a^k b^k$. Comme $k \neq 0$, $a^{k-1}b^{k-1} \in L$ et est de taille $n-1$ donc par hypothèse de récurrence il est engendré par S. On en déduit une dérivation $S \Rightarrow aSb \Rightarrow^\star aa^{k-1}b^{k-1}b = u$ pour le mot u.

Exercice 132 (exo cours) *Grammaire pour un complémentaire*

On considère le langage L défini comme étant le complémentaire du langage dénoté par $a(b+a)^\star a$ sur $\{a, b\}$. Déterminer en justifiant une grammaire algébrique permettant d'engendrer le langage L.

On construit un automate simple pour $a(b+a)^\star a$ qu'on déterminise et complète pour le complémentariser puis on déduit de l'automate obtenu une grammaire pour L (via la lecture des transitions).

Exercice 133 (exo cours) *Suppression d'ambiguité*

1. Montrer que la grammaire G décrite par les règles $S \to SaS \,|\, b$ est ambiguë.
2. Déterminer le langage L engendré par G et prouver rigoureusement que $L = L(G)$.
3. Proposer une grammaire non ambiguë qui engendre G.

1. *babab* admet deux arbres syntaxiques différents.
2. $L = (ba)^\star b$. Preuve par double inclusion, l'une par récurrence sur la longueur des dérivations, l'autre par récurrence sur la longueur des mots de L.
3. $S \to Tb$ et $T \to Tba \,|\, \varepsilon$ convient. Elle est non ambiguë car une variable se dérive en un mot ne contenant qu'une variable : il y a donc unicité de la dérivation permettant de produire un mot.

Exercice 134 (exo cours) *Langage engendré*

1. Déterminer le langage engendré par la grammaire G d'axiome S et dont les règles sont

$$S \to 0S0 \,|\, 1S1 \,|\, 0 \,|\, 1 \,|\, \varepsilon$$

2. Prouver rigoureusement que le langage proposé est bien égal à $L(G)$.

1. C'est le langage L des palindromes sur $\{0, 1\}$.
2. Preuve par double inclusion, chaque inclusion se faisant par récurrence.

Exercice 135 (exo cours) *Grammaire et préfixes*

On considère la grammaire G décrite par les règles suivantes : $S \to aS \,|\, aSbS \,|\, \varepsilon$. Montrer que

$$L(G) = \{m \in \{a,b\}^\star \,|\, \text{ pour tout préfixe } u \text{ de } m,\ |u|_a \geq |u|_b\}$$

Montrons par récurrence forte sur la longueur des dérivations que $L(G) \subset L$. Si $S \Rightarrow^{k+1} u$ alors $S \Rightarrow aS \Rightarrow^k u$ ou $S \Rightarrow aSbS \Rightarrow^k u$.

- Dans le premier cas, $u = av$ avec $v \in L$ par hypothèse de récurrence. Un préfixe de u est donc ε ou av' avec v' préfixe de v donc contient plus de a que de b puisque $v \in L$.
- Dans le second, $u = avbw$ avec $v, w \in L$ par hypothèse de récurrence. Un préfixe de u est donc ε, av' avec v' préfixe de v ou $avbw'$ avec w' préfixe de w ; dans tous les cas il a plus de a que de b.

Montrons par récurrence forte sur la taille des mots de L que $L \subset L(G)$. Si $|u| = n+1$ et $u \in L$ alors :

- Soit u est une puissance de a. Si cette puissance est nulle, u est généré via $S \Rightarrow \varepsilon$ et sinon via $S \Rightarrow aS \Rightarrow ... \Rightarrow a^kS \Rightarrow a^k$. Donc $u \in L(G)$.
- Soit u contient un b. TODO.

Exercice 136 (exo cours) *Grammaire pour un langage rationnel*

On considère le langage $L = \{m \in \{a,b\}^\star \,|\, |m|_a = |m|_b \bmod 2\}$.

1. Montrer que L est rationnel.
2. Donner en justifiant une grammaire permettant d'engendrer L.

1. $L = \{m \in \{a,b\}^\star \,|\, |m|_a$ et $|m|_b$ sont pairs $\} \cup \{m \in \{a,b\}^\star \,|\, |m|_a$ et $|m|_b$ sont impairs $\}$. On en déduit un automate à 4 états (a pair, b pair), (a pair, b impair), (a impair, b pair) et (a impair, b impair) reconnaissant L.
2. L'automate précédent en fournit les règles.

Exercice 137 (exo cours) *Grammaire pour un langage naturel*

On considère la grammaire G d'axiome S, de non terminaux $\{S, GN, GV, Det, N, GP, Prep\}$ et décrite par les règles suivantes. Est-elle ambiguë ? Justifier.

$S \to GN\,GV$
$GN \to Det\,N \,|\, GN\,GP \,|\, N$
$Det \to$ un | une | des | le | la | les
$N \to$ chat | Toto | Mickey | téléscope
$GV \to V\,GN \,|\, GV\,GP$
$V \to$ regarde | mange
$GP \to Prep\,GN$
$Prep \to$ avec | chez

Oui. La phrase "Toto regarde Mickey avec un téléscope" admet deux arbres de dérivation :

- Si on dérive le groupe verbal en V + GN : Toto regarde (Mickey avec un téléscope).
- Si on dérive le groupe verbal en GV + GP : Toto regarde Mickey (avec un téléscope).

Exercice 138 (exo cours) *Arbres syntaxiques*

Donner un arbre de dérivation pour le mot u dans la grammaire G_i et deux si c'est possible :

1. Si $u = aabbba$ et G_1 est définie par $S \to aaX$, $X \to B \,|\, \varepsilon$ et $B \to bB \,|\, a$.
2. Si $u = abaabb$ et G_2 est définie par $S \to aSbS \,|\, bSaS \,|\, \varepsilon$.
3. Si $u = abb$ et G_3 est définie par $S \to aS \,|\, aSS \,|\, b$.

Parmi G_1, G_2 et G_3, lesquelles sont des grammaires ambiguës ?

1. Une seule dérivation possible donc un seul arbre possible ; grammaire non ambiguë.
2. (Au moins) deux arbres syntaxiques différents : $S \Rightarrow aSbS$ peut se poursuivre en dérivant le premier S en ε ou en $bSab$. La grammaire est donc ambiguë.
3. Une seule dérivation possible pour u mais la grammaire est quand même ambiguë via $aabb$.

Exercice 139 (***) *Génération par décomposition*

Dans cet exercice, on suppose que l'alphabet est $\Sigma = \{a, b\}$. On considère tout d'abord le langage

$$L_1 = \{ww' \,|\, w \neq w' \text{ et } |w| = |w'|\}$$

1. Montrer que $L_1 = L_a L_b \cup L_b L_a$ avec $L_a = \{uav \,|\, |u| = |v|\}$ et $L_b = \{ubv \,|\, |u| = |v|\}$.
2. Montrer que L_1 est algébrique.

On considère le langage $L_2 = \{w\#w' \,|\, |w| \neq |w'|\}$ où $\#$ est une nouvelle lettre.

3. Montrer que L_2 est algébrique.

1. Par double inclusion en décomposant les mots en lettres.
2. L_a est engendré par $A \to ZAZ \,|\, a$ et $Z \to a \,|\, b$, on procède de même pour L_b et donc la question 1 permet d'assurer que L_1 est engendré par $S \to AB \,|\, BA$, $A \to ZAZ \,|\, a$, $B \to ZbZ \,|\, b$ et $Z \to a \,|\, b$.
3. On décompose $L_2 = L_< \cup L_>$ avec $L_< = \{w\#w' \,|\, |w| < |w'|\}$ engendré par $S \to aSa \,|\, aSb \,|\, bSa \,|\, bSb \,|\, Ta \,|\, Tb$ et $T \to Ta \,|\, Tb \,|\, \#$. $L_>$ s'engendre de manière similaire ce qui conclut.

Exercice 140 (**) *Mots non carrés*

Dans cet exercice, l'alphabet est $\Sigma = \{a, b\}$. On note L le complémentaire du langage

$$\{m \in \Sigma^\star \,|\, \exists u \in \Sigma^\star, m = uu\}$$

1. Donner deux exemples de mots dans L, l'un de longueur 4, l'autre de longueur 5.
2. Montrer que tout mot de longueur impaire est dans L et que tout mot $m_1 ... m_{2n}$ de longueur paire appartient à L si et seulement si il existe un indice $i \in [\![1, n]\!]$ tel que $m_i \neq m_{n+i}$.

On considère la grammaire algébrique G d'axiome S dont les règles sont :

$$\begin{array}{l} S \to A \,|\, B \,|\, AB \,|\, BA \\ A \to a \,|\, aAa \,|\, aAb \,|\, bAa \,|\, bAb \\ B \to b \,|\, aBa \,|\, aBb \,|\, bBa \,|\, bBb \end{array}$$

3. Décrire le langage $L(G, A)$ des mots qu'il est possible de dériver à partir de A.
4. Montrer que tout mot de longueur impaire est dans $L(G)$.
5. Montrer que tout mot $m \in L$ de longueur paire est dans $L(G)$.
6. Montrer que le langage L est algébrique.

1. $aabb$ et $ababb$ conviennent : ce ne sont pas des carrés.
2. On le prouve par contraposition. Pour la première partie : si $m \in L^c$, $|m|$ est paire. Pour la seconde, $m \in L^c$ ssi $m = u^2$ avec u de taille n ssi $\forall\, i \in [\![1, n]\!]$, $m_i = m_{i+n}$ avec $|m| = 2n$.
3. $L(G, A) = \{uav \,|\, |u| = |v|\}$. C'est similaire pour $L(G, B)$.
4. Si m est de longueur impaire, il a une lettre centrale qui vaut soit a soit b donc est soit dans $L(G, A)$ soit dans $L(G, B)$ et comme $S \to A \,|\, B$, $m \in L(G)$.
5. Si $m \in L$ est de longueur $2n$ par la question 2 il existe $i \in [\![1, n]\!]$ tel que $m_i \neq m_{i+n}$. Disons que $m_i = a$ et $m_{i+n} = b$. Alors :
$$m = \underbrace{m_1 ... m_i ... m_{2i-1}}_{=u} \underbrace{m_{2i} ... m_{i+n} ... m_{2n}}_{=v}$$
avec u de longueur impaire avec a comme lettre centrale donc généré par A et v de longueur impaire avec b comme lettre centrale donc généré par B. Comme $S \to AB$, on conclut. Le cas où $m_i = b$ et

$m_{i+n} = a$ se traite de la même façon sauf qu'on utilise $S \to BA$.

6. 4 et 5 montrent que $L \subset L(G)$. On montre l'inclusion réciproque à l'aide de 2 et 3.

Exercice 141 (**) *Dangling else*

On considère un fragment simplifié de la grammaire d'un langage de programmation :

$$I \to \texttt{foo} \mid \texttt{bar} \mid \texttt{aux} \mid C$$
$$C \to \texttt{if } E \texttt{ then } I \texttt{ else } I \mid \texttt{if } E \texttt{ then } I$$
$$E \to \texttt{true} \mid \texttt{false} \mid \texttt{happy}$$

Les non terminaux sont écrits en majuscules et l'axiome de cette grammaire est I.

1. Donner un arbre syntaxique pour :

   `if happy then if true then foo else bar else aux`

   Y en a-t-il d'autres ?
2. Donner deux arbres d'analyse distincts pour : `if happy then if false then foo else bar`. Que peut-on dire de la grammaire présentée ?
3. En supposant que dans ce langage, `begin` I `end` (où I est une instruction) a le même effet que I seule, comment un programmeur peut-il réécrire l'instruction de la question 2 pour obtenir chacune des deux interprétations induites par les deux arbres précédents ?
4. Modifier la grammaire proposée de sorte à obtenir une grammaire faiblement équivalente dans laquelle un seul des arbres d'analyses obtenus en 2 est possible.

1. Il y a deux `else` qui doivent chacun se rapporter à un `if` donc il est obligatoire d'utiliser deux fois la règle $C \to$ `if` E `then` I `else` I. D'où un seul arbre possible.
2. Ce mot admet deux arbres de dérivation donc la grammaire est ambiguë.
3. L'un des arbres correspond à `if happy then begin if false then foo else bar end` et l'autre à `if happy then begin if false then foo end else bar`.
4. On peut forcer la première interprétation (cad rattacher le `else` au `if` le plus proche) via :

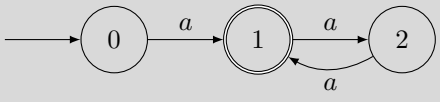
   $I \to$ comme avant / $E \to$ comme avant
   $C \to$ `if` E `then` I_e `else` I | `if` E `then` I
   $I_e \to$ `foo` | `bar` | `aux` | C_e
   $C_e \to$ `if` E `then` I_e `else` I_e

Exercice 142 (**) *Grammaires linéaires*

Une grammaire algébrique est dite linéaire à droite si toutes ses règles sont de la forme $X \to u$ ou $X \to uY$ avec X, Y non terminaux et $u \in \Sigma^\star$ avec Σ l'alphabet des terminaux.

1. La grammaire $G = (\{a\}, \{S\}, \{S\}, S \to aaS \mid \varepsilon)$ est linéaire à droite. Identifier le langage L qu'elle engendre et montrer que $L = L(G)$.
2. Montrer que $L(G)^c$ peut être engendré par une grammaire linéaire à droite.
3. Montrer que si un langage L est rationnel alors il est engendré par une grammaire linéaire à droite.
4. La réciproque de la question précédente est-elle vraie ?
5. Une grammaire algébrique est dite linéaire à gauche si toutes ses règles sont de la forme $X \to u$ ou $X \to Yu$ avec X, Y non terminaux et u un mot sur l'alphabet des terminaux. Reprendre les questions 3 et 4 dans ce contexte.

1. $L = (a^2)^\star$, preuve par double inclusion.
2. On construit un automate déterministe pour L^c :



On en déduit que L^c est engendré par $X_0 \to aX_1$, $X_1 \to aX_2 \,|\, \varepsilon$, $X_2 \to aX_1$, linéaire à droite.

3. Les transitions d'un automate pour L fournissent une telle grammaire.
4. Oui. Si $G = (\Sigma, V, S, \mathcal{R})$ est linéaire à droite on peut construire un automate A pour $L(G)$: son état initial est S, ses états sont V complétés par :
   - Un état final q_f.
   - Pour toute $X \to a_1...a_kY \in \mathcal{R}$ on ajoute $k-1$ états P_i et les transitions : $X \xrightarrow{a_1} P_1...P_{k-1} \xrightarrow{a_k} Y$.
   - Pour toute $X \to a_1...a_k \in \mathcal{R}$ on ajoute $k-1$ états P_i et les transitions : $X \xrightarrow{a_1} P_1...P_{k-1} \xrightarrow{a_k} q_f$.

   Il y a bien un nombre fini d'états car il y a un nombre fini de règles et pour chacune un nombre fini de lettres dans le membre droit.
5. Les langages rationnels sont aussi ceux engendrés par grammaires linéaires gauches. Par exemple, L^c est engendré par $X_0 \to \varepsilon$, $X_1 \to X_2a \,|\, X_0a$, $X_2 \to X_1a$ d'axiome X_1 (sémantique : X_i engendre les mots qui permettent d'arriver dans i. Par ex, $X_1 \to X_2a$ car les mots qui permettent d'arriver en 1 sont ceux permettant d'arriver dans 2 suivi d'un a).

Exercice 143 (**) *Grammaires quasi-propres*

L'objectif de cet exercice est de supprimer les ε-productions d'une grammaire algébrique, c'est-à-dire les règles de la forme $X \to \varepsilon$ sans (trop) modifier le langage engendré. Plus précisément, on cherche à montrer que pour toute grammaire G, il existe une grammaire G' sans ε-production telle que $L(G') = L(G) \setminus \{\varepsilon\}$.

1. En considérant la grammaire G dont les règles sont $S \to SAS \,|\, b$ et $A \to a \,|\, \varepsilon$, d'axiome S, montrer que simplement supprimer les ε-productions ne produit pas une grammaire G' convenable.

On propose la construction suivante pour obtenir G' à partir de $G = (\Sigma, V, S, \mathcal{R})$:

- On note $E(G)$ l'ensemble des non terminaux X de G tels que $X \Longrightarrow^\star \varepsilon$.
- On remplace chacune des règles $X \to m$ de G, par $X \to s(m)$ où s substitue chaque lettre de m ainsi :

$$s : \begin{cases} \Sigma \cup V \cup \{\varepsilon\} & \to \mathcal{P}((\Sigma \cup V)^\star) \\ \varepsilon & \mapsto \varepsilon \\ a \in \Sigma & \mapsto a \\ Y \notin E(G) & \mapsto Y \\ Y \in E(G) & \mapsto (Y + \varepsilon) \end{cases}$$

- On développe les pseudo-règles obtenues. Par exemple, la pseudo-règle $X \to a(Y+\varepsilon)(b+\varepsilon)$ est remplacée par $X \to aYb \,|\, aY \,|\, ab \,|\, a$.
- On supprime toutes les ε-productions des règles obtenues. On obtient alors une grammaire G'.

2. Appliquer cet algorithme à la grammaire dont les règles sont :

$$S \to aAB \,|\, BA \,|\, b$$
$$A \to BBB \,|\, a$$
$$B \to AB \,|\, b \,|\, \varepsilon$$

3. Expliquer comment calculer $E(G)$.
4. Montrer que $L(G') \subset L(G) \setminus \{\varepsilon\}$. Expliquer pourquoi l'inclusion réciproque est vraie et conclure.

1. Si on supprime $A \to \varepsilon$ on ne peut plus produire bb.
2. On a $E(G) = \{S, A, B\}$. Les nouvelles règles sont $S \to aAB \,|\, aA \,|\, aB \,|\, a \,|\, BA \,|\, A \,|\, B \,|\, b$, $B \to BBB \,|\, BB \,|\, B \,|\, a$ et $A \to AB \,|\, A \,|\, B \,|\, b$.
3. On calcule $(E_n)_{n\in\mathbb{N}}$ avec $E_0 = \{X \in V \,|\, V \to \varepsilon\}$ et $E_{n+1} = \{X \in V \,|\, X \to X_1...X_n \text{ et les } X_i \in E_n\}$. Cette suite croît pour l'inclusion et stationne en $E(G)$.
4. On montre pour tout $X \in V$ que $L(G', X) \subset L(G, X) \setminus \{\varepsilon\}$ en montrant par récurrence sur $k \in \mathbb{N}^\star$ que pour tout $X \Rightarrow^k u$ dans G', $X \Rightarrow^\star u$ dans G.

Exercice 144 (***) *Équations aux langages*

Soit G une grammaire dont les symboles terminaux sont donnés par A, les symboles non terminaux par $V = \{X_1, ..., X_n\}$, dont l'axiome est X_1 et les règles sont données par l'ensemble P. Si $L = (L_1, ..., L_n)$ est un n-uplet de langages sur A, on définit pour tout mot $m \in (A+V)^\star$ le langage $m(L)$ inductivement par :

$$\varepsilon(L) = \{\varepsilon\},\ a(L) = \{a\} \text{ pour tout } a \in A,\ X_i(L) = L_i \text{ pour tout } X_i \in V$$

et $xy(L) = x(L)y(L)$ pour tout $x, y \in (A+V)^\star$. Le *système associé à* G, noté $S(G)$ est le système d'équations aux langages dont les variables sont les L_i et les équations sont :

$$L_i = \sum_{X_i \to m \in P} m(L) \quad \forall i \in [\![1, n]\!]$$

Par ailleurs, pour tout $m \in (A+V)^\star$, on note $L(G, m)$ le langage des mots qu'on peut engendrer à partir du mot m dans G. On note aussi L_G le n-uplet $(L(G, X_1), ..., L(G, X_n))$.

1. Déterminer $S(G)$ lorsque G a pour règles $X_1 \to X_1X_2 \,|\, a$ et $X_2 \to aX_2 \,|\, b$. Reprendre cette question avec la grammaire dont les règles sont $X_1 \to X_1X_1 \,|\, \varepsilon$.
2. Montrer que pour tout mot $m \in (A+V)^\star$, $L(G, m) = m(L_G)$.
3. Soit L une solution de $S(G)$. Si $m, m' \in (A+V)^\star$ vérifient que $m \Longrightarrow^\star m'$ montrer que $m'(L) \subset m(L)$.
4. Montrer que L_G est solution de $S(G)$ et que c'est la solution minimale de ce système pour l'inclusion composante par composante.
5. La question précédente montre que le système $S(G)$ admet toujours une solution. Est-elle toujours unique ?

1. Pour le premier : $\begin{cases} L_1 = L_1L_2 + a \\ L_2 = aL_2 + b \end{cases}$. Pour le second : $L_1 = L_1^2 + \varepsilon$.
2. On le montre par induction. $L(G, \varepsilon) = \{\varepsilon\} = \varepsilon(L_G)$. $L(G, a) = \{a\} = a(L_G)$. $L(G, X_i) = i$-ème composante de $L_G = X_i(L_G)$. Enfin, on a :
$$\begin{aligned} L(G, m_1...m_n) =& L(G, m_1)...L(G, m_n) \\ & m_1(L_G)...m_n(L_G) \text{ par les cas de base} \\ & m_1...m_n(L_G) \text{ par définition de } m(L_G) \end{aligned}$$
3. C'est vrai pour une dérivation immédiate (puis on procède par récurrence sur le nombre de dérivations). Si $m \Rightarrow m'$, par définition $m = uX_iv$ et $m' = uwv$ avec $X_i \to w$ une règle. Comme L est solution de $S(G)$, $L_i = \sum_{X_i \to \gamma \in P} \gamma(L)$, $w(L) \subset L_i$ donc $uw(L)v \subset uL_iv$ donc $m'(L) \subset m(L)$.
4. L_G est solution car $L(G, X_i) = \sum_{X_i \to \alpha \in P} L(G, \alpha) = \sum_{X_i \to \alpha \in P} \alpha(L_G)$ par 2. Si $L = (L_1, ..., L_n)$ est solution de $S(G)$, montrons que $L(G, X_i) \subset L_i$. Si $m \in L(G, X_i)$, il existe une dérivation $X_i \Rightarrow^\star m$ donc par 3, $m(L) \subset X_i(L)$. Comme $m \in \Sigma^\star$, $m(L) = m$ donc on a bien $m \in L_i$.
5. Non. La grammaire $X_1 \to X_1X_1 \,|\, \varepsilon$ génère un système pour lequel $K^\star$ est solution pour tout K.

Remarque : parallèle avec le lemme d'Arden. Les langages rationnels sont solutions minimales de systèmes linéaires / les langages algébriques sont solutions minimales de systèmes polynomiaux.

6 Programmation

Exercice 145 (exo cours) *Unlimited power*

On considère le code OCaml suivant :

```
let f x n r =
  let rec aux n =
    if n = 0 then 1
    else match (n mod 2 = 0) with
         | true -> (aux (n/2) * aux (n/2)) mod r
         | false -> (x * aux (n/2) * aux (n/2)) mod r
```

```
  in aux n
```

1. Inférer la spécification de `f`. Le code est-il syntaxiquement correct ?
2. Analyser la complexité de cette implémentation. Commenter et modifier la fonction au besoin.

1. `f x n r` calcule x^n modulo r. Il est syntaxiquement correct.
2. En notant $C(n)$ la complexité quand l'exposant est de taille n on a la relation de récurrence $C(n) = 2C(n/2) + O(1)$ donc une complexité en $O(n)$ alors que clairement on voulait faire une exponentiation rapide en $O(\log n)$. Pour corriger le défaut on modifie le premier cas de filtrage en `let k = aux (n/2) in k*k mod r` et de même pour le second.

Exercice 146 (exo cours) *Appariements*

On considère le code OCaml suivant :

```
let apparie (f:int list) (s:int list) :int list list = match f, s with
  |[], [] -> []
  |a::b, c::d -> [a,c]::(apparie b d)
  |_ -> failwith "listes pas de la même longueur"
```

1. Inférer la spécification de la fonction `apparie`.
2. Ce code est-il syntaxiquement correct ? Si non, le corriger.

1. Cette fonction renvoie la liste des listes formées des deux entiers à la même position dans les deux listes en entrée. Par exemple `apparie [1;2;3] [4;5;6]` renvoie `[[1;4];[2;5];[3;6]]`.
2. Deux problèmes de syntaxe. Déjà `apparie` doit être récursive. Ensuite, `[a,c]` n'a pas le bon type : ce devrait être une `int list` et pas une `(int*int) list` : on corrige en `[a;c]`.

Exercice 147 (exo cours) *Composition*

On considère le code OCaml suivant :

```
let fois x = 2. *. x

let rec somme l = match l with
  |[] -> 0
  |t::q -> t + (somme q)

let comp f g = fun x -> f (g x)

let a = (comp fois somme) [1;5;4;7]
```

1. Inférer la spécification et la signature de chacune des fonctions. Que devrait valoir `a` ?
2. Ce code est-il syntaxiquement correct ? Si non, corriger.

1. `fois : float -> float` multiplie son entrée flottante par 2. `somme : int list -> int` somme les éléments d'une liste d'entiers. `comp : ('a->'b) -> ('c->'a) -> 'c -> 'b` applique à son entrée la composition des fonctions en entrée. A priori, on affiche $(1+5+4+7) \times 2 = 34$.
2. Non, on a un problème de type à corriger (plusieurs façons de faire) car `somme [1;5;4;7]` est un `int` qu'on donne à `fois` qui attend un `float`.

Exercice 148 (exo cours) *Copie de matrice*

On considère le code OCaml suivant :

```
let copy_matrix (m:'a array array) = Array.copy m
```

1. Inférer le comportement attendu pour `copy_matrix`. Le code est-il syntaxiquement correct ?

2. Cette fonction a-t-elle le comportement attendu ? Justifier à l'aide d'un exemple précis et corriger.

1. A priori on veut faire une copie de la matrice en entrée. C'est syntaxiquement correct.
2. Non car `Array.copy` ne fait qu'une copie superficielle. Par exemple après exécution de :

```
let m = Array.make_matrix 2 2 1
let copy_m = copy_matrix m
let _ = copy_m.(0).(0) <- 2
```

la case $(0, 0)$ de la matrice `m` contiendra 2 ce qui n'est a priori pas voulu. Pour corriger :

```
let copy_matrix m = Array.init (Array.length m) (fun i -> Array.copy m.(i))
```

ou plus simplement, créer une matrice de la bonne taille avec `Array.make_matrix` et en remplir les cases avec le contenu de celles de l'entrée à l'aide deux deux boucle pour.

Exercice 149 (exo cours) *Modification de chaînes*

On considère le code C suivant :

```
void f(char* s, char b, char a)
{
    for (int i = 0; i < strlen(s); i++)
    {
        if (s[i] = b) s[i] = a;
    }
    return;
}
```

1. Inférer la spécification de la fonction `f`. Ce code est-il syntaxiquement correct ?
2. Étudier la complexité temporelle de cette fonction et commenter.

1. Ce code remplace les occurrences du caractère `b` par le caractère `a` dans `s`. Il y a une erreur de syntaxe : la condition devrait être `(s[i] == b)`.
2. Complexité en $O(n^2)$ où $n = |s|$ à cause de la complexité linéaire de `strlen` à chaque itération. Il serait judicieux de faire le calcul de longueur une bonne fois pour toutes en dehors de la boucle.

Exercice 150 (exo cours) *Segfault*

On considère le code C suivant :

```
1  typedef struct toto {
2      int* states;
3  }  toto;
4
5  void toto_create(toto* a, int n) {
6      a = malloc(sizeof(toto));
7      a->states = malloc(sizeof(int)*n);
8  }
9
10 void toto_destroy(toto* a) {
11     free(a);
12 }
13
14 int main() {
15     toto* a = NULL;
16     toto_create(a, 3);
```

```
17      toto_destroy(a);
18  }
```

1. Le sanitizer indique une fuite mémoire à l'exécution. Proposer une correction.
2. Une fois cette modification faite, le sanitizer indique :

```
==666050==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x55aa36d2727e bp 0x7ffd4d262780 sp 0x
==666050==The signal is caused by a READ memory access.
==666050==Hint: address points to the zero page.
    #0 0x55aa36d2727d in toto_destroy [redacted]
    #1 0x55aa36d272c8 in main [redacted]
    #2 0x7f1e3df7d082 in __libc_start_main ../csu/libc-start.c:308
    #3 0x55aa36d2712d in _start [redacted]

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV [redacted] in toto_destroy
```

Expliquer et corriger le problème. Pourquoi n'apparaissait-il pas avant ?

1. Ajouter `free(a->states)` avant `free(a)` dans `toto_destroy`.
2. La fonction `toto_create` manipule et modifie une copie locale de `a` : ces modifications sont donc invisibles en dehors de cette fonction. Par conséquent, en ligne 17, `a` est toujours `NULL` et tenter d'accéder au champ `states` de `a` provoque donc une erreur de lecture. Avant de modifier `toto_destroy`, la ligne 17, faisait juste s'exécuter `free(NULL)` ce qui est licite.

   On se rend compte rétrospectivement que la fuite mémoire précédente n'était donc pas due à `toto_destroy` mais au fait que l'appel à `toto_create` en ligne 16 allouait de la mémoire à laquelle on perdait accès immédiatement et donc n'était pas libérée, quoi que fasse `toto_destroy`.

   Pour corriger, soit il faut passer un pointeur d'un pointeur sur un `toto` dans `toto_create` soit modifier son prototype en `toto* toto_create(int n)`.

Exercice 151 (exo cours) *Somme d'entiers*

On souhaite calculer la somme $\sum_{i=1}^{42} i$. Pour ce faire, un programmeur propose le code suivant en C :

```c
int* suite (int n) {
    int* t = malloc(sizeof(int)*n);
    for (int i = 0; i <= n; i++) {
        t[i] = i;
    }
    return t;
}

int somme(int* t, int n) {
    int s;
    for (int i = 0; i < n; i++) {
        s += t[i];
    }
    return s;
}

int main(void) {
    int* tab = suite(42);
    printf("%d\n", somme(tab,42));
    free(tab);
}
```

1. À l'exécution, le sanitizer signale un `heap-buffer-overflow`. Expliquer et corriger le problème.
2. Après ces corrections, l'exécution produit l'affichage attendu. Mais, rendu méfiant, le programmeur décide de mieux tester ses fonctions. Il teste `somme` sur le tableau défini par `int t[] = {1,2,3,4};` et obtient le résultat 1542381. Expliquer le problème et pourquoi il n'était pas visible auparavant.

1. La fonction `suite` est a priori destinée à contenir les entiers de 0 à n, donc $n+1$ entiers, or on n'a alloué de la place que pour n. On sort donc de la zone allouée sur le tas à la dernière itération de la ligne 4. Plusieurs façons de corriger, l'une consiste à mettre une inégalité stricte en ligne 3 et remplacer la ligne 4 par `t[i] = i+1;`.
2. Le problème est la non initialisation de `s` en ligne 10, qui peut donc contenir autre chose que 0. Si l'affichage était correct auparavant, c'était simplement par chance !

7 Algorithmes d'approximation

Exercice 152 (**) *Sac à dos illimité*

On considère le problème d'optimisation `SAC A DOS ILLIMITE` :

$$\begin{cases} \textbf{Entrée} : \text{Une liste de poids } p_1, ..., p_n \in \mathbb{N}^\star, \text{ une liste de valeurs } v_1, ..., v_n \in \mathbb{N}^\star \text{ et } P \in \mathbb{N}^\star. \\ \textbf{Solution} : \text{Une liste d'entiers naturels } a_1, ..., a_n \text{ telle que } \sum_{i=1}^{n} a_i p_i \leq P. \\ \textbf{Optimisation} : \text{Maximiser } \sum_{i=1}^{n} a_i v_i. \end{cases}$$

Ce problème est une variante du problème du sac à dos dans lequel chaque objet est disponible en quantité illimitée. Comme son homologue, il est NP-difficile. On considère l'algorithme suivant :

$S \leftarrow 0$
Trier les densités v_i/p_i par ordre décroissant
Renuméroter les objets de sorte à ce que le i-ème ait la i-ème plus grande densité
pour tout $i \in [\![1, n]\!]$
  Déterminer $a_i \in \mathbb{N}$ maximal tel que $S + a_i p_i \leq P$
  $S \leftarrow S + a_i p_i$
renvoyer $a_1, ..., a_n$

1. Déterminer la complexité de cet algorithme.
2. Montrer que cet algorithme est une 1/2-approximation de `SAC A DOS ILLIMITE`. *Indication : quelle est la proportion du sac remplie par les premiers objets mis dans le sac ?*

1. Le tri peut se faire en $O(n \log n)$. Une itération de la boucle se fait en $O(1)$ en calculant le quotient dans la division euclidienne du poids restant $(P - S)$ par p_i. D'où une complexité en $O(n \log n)$.
2. On note V la valeur du sac renvoyé par l'algo et $V^\star$ sa valeur optimale. Si tous les objets ont un poids strictement supérieur à P alors $V = V^\star = 0$. Sinon, considérons le premier objet de poids inférieur à P dans l'ordre indiqué par l'algorithme : il porte l'indice k. On aura ainsi $a_k \geq 1$. Or :

   $$V^\star = (\text{poids du sac}) \times (\text{meilleure valeure au kilo qui rentre dans le sac})$$

   Cette meilleur valeur est celle de l'objet k à cause du tri effectué. Ainsi :

   $$V^\star = P \times \frac{v_k}{p_k} = a_k v_k + \left(\frac{P}{p_k} - a_k\right) v_k$$

   Le premier terme correspond à la valeur des objets k mis dans le sac et le second à la valeur des autres objets, qui est inférieure à $a_k v_k$ car les a_k premiers objets occupent strictement plus de la moitié du sac, autrement, on pourrait en mettre un de plus dans le sac et on contredirait le fonctionnement de l'algorithme. Ainsi, $V^\star \leq 2a_k v_k \leq 2V$, c'est-à-dire $V \geq (1/2)V^\star$.

Exercice 153 (**) *Approximation pour une couverture par sommets*

Dans un graphe $G = (S, A)$, une couverture par sommets de G est un sous ensemble $R \subset S$ des sommets de G tel que pour toute arête $(u, v) \in A$, $u \in R$ ou $v \in R$. Autrement dit, c'est un ensemble de sommets tel que chaque arête est incidente à l'un d'entre eux. Le problème `COUVERTURE PAR SOMMET` est le problème d'optimisation :

$$\begin{cases} \textbf{Entrée} : \text{Un graphe non orienté } G = (S, A). \\ \textbf{Solution} : \text{Une couverture par sommets } R \text{ de } G. \\ \textbf{Optimisation} : \text{Minimiser } |R|. \end{cases}$$

Le problème de décision associé à ce problème d'optimisation est NP-complet. On considère l'algorithme A_1 prenant en entrée le graphe $G = (S, A)$:

```
R ← ∅
tant que il y a des arêtes non couvertes
    Choisir une arête (s, t) non couverte
    R ← R ∪ {s}
renvoyer R
```

1. Montrer que A_1 n'est pas un algorithme d'approximation de `COUVERTURE PAR SOMMETS` à facteur constant, c'est-à-dire qu'il n'existe pas de $\alpha > 0$ tel que A_1 soit une α-approximation de ce problème.

On considère à présent l'algorithme A_2 suivant prenant en entrée le graphe $G = (S, A)$:

```
C ← ∅
tant que A ≠ ∅
    Choisir a = (s, t) dans A
    C ← C ∪ {a}
    Supprimer de A toutes les arêtes incidentes à s ou à t
renvoyer C
```

2. Montrer que l'ensemble renvoyé par A_2 est un couplage maximal (pour l'inclusion) de G.
3. Déduire de A_2 un algorithme A_3 qui renvoie une couverture par sommets de G.
4. Montrer que A_3 est une 2-approximation de `COUVERTURE PAR SOMMETS`.
5. Montrer qu'on peut implémenter A_3 avec une complexité linéaire en la taille du graphe et conclure.

1. On considère pour tout $n \in \mathbb{N}$ le graphe en étoile à n sommets. A_1 pourrait renvoyer la couverture constituée de tous les sommets sauf le central, de taille $C_n = n - 1$ alors que la couverture optimale est de taille $C_n^\star = 1$ (prendre le sommet central). Comme $C_n / C_n^\star \to +\infty$, ça conclut.
2. "C est un couplage de G" est un invariant. Il est maximal car pour tout $(u, v) \in A$, si $(u, v) \notin C$, cette arête a été supprimée par l'algo puisque A est vide à la fin et cela implique qu'il y a une arête de C incidente soit à u soit à v donc on ne peut pas rajouter (u, v) au couplage C.
3. A_3 = calculer un couplage maximal via A_2 et renvoyer les extrémités des arêtes de ce couplage.
4. On note N le nombre de sommets renvoyé par A_3 et $N^\star$ la taille optimale d'une couverture sur une instance donnée. Une couverture optimale couvre en particulier toutes les arêtes du couplage maximal C calculé par A_3 donc contient une des extrémités de chaque arête de C. Donc $N = 2|C| \leq 2N^\star$.
5. On représente G par liste d'adjacence et on se dote d'une liste L destinée à stocker les sommets de la couverture et d'un tableau T à $|S|$ cases contenant vrai en case i ssi i est couvert par une des arêtes de C. Pour chaque sommet s, s'il n'est pas couvert, on parcourt sa liste d'adjacence. Si on en trouve un non couvert t, on ajoute s et t à L et on les marque comme couverts. T permet en fait de savoir s'il faut considérer une arête sans avoir à la supprimer. Complexité : $O(|S| + |A|)$.

   On a troqué une résolution exacte exponentielle pour une résolution approchée polynomiale.

Exercice 154 (***) *Calcul de k-centre*

Si $G = (S, A)$ est un graphe non orienté et $S' \subset S$, on dit que :

- S' est un ensemble indépendent de G si deux sommets de S' ne sont jamais liés par une arête de G.
- S' est un ensemble dominant de G si tout sommet de S est soit dans S', soit voisin d'un sommet de S'. On note $dom(G)$ le cardinal minimal d'un ensemble dominant dans G.

On note de plus pour tout sous-ensemble $S' \subset S$ et tout sommet $u \in S \setminus S'$: $\delta(u, S')$ le poids minimal d'une arête reliant u à un sommet de S' et $c(S') = \max_{u \in S \setminus S'} \delta(u, S')$. On considère les problèmes suivants :

$$\texttt{DOM} : \begin{cases} \textbf{Entrée} & : G = (S, A) \text{ non orienté et } k \in \mathbb{N}. \\ \textbf{Question} & : \text{Y a-t-il un ensemble dominant dans } G \text{ de taille inférieure à } k \,? \end{cases}$$

$$\texttt{CENTRE} : \begin{cases} \textbf{Entrée} & : G = (S, A) \text{ non orienté, complet, pondéré par une fonction } w \text{ vérifiant l'inégalité} \\ & \quad \text{triangulaire, } m \in \mathbb{N} \text{ et } k \in \mathbb{N}^\star. \\ \textbf{Question} & : \text{Existe-t-il } S' \subset S \text{ de cardinal } k \text{ tel que } c(S') \leq m \,? \end{cases}$$

On admet que DOM est NP-complet. Le problème d'optimisation associé à CENTRE est connu sous le nom de problème du k-centre. Il consiste à trouver un sous ensemble S' des sommets d'un graphe, de taille k, et tel que la distance maximale à parcourir depuis un sommet quelconque du graphe pour atteindre S' soit la plus petite possible.

1. Donner une application dans laquelle déterminer un k-centre peut être utile.
2. Par réduction depuis DOM, montrer que calculer un k-centre est un problème NP-difficile.

On cherche à construire une 2-approximation pour le calcul d'un k-centre. Pour ce faire, on note $a_1, ..., a_m$ (avec $m = |A|$) les arêtes de G supposées triées par ordre de poids croissant. On note $G_i = (S, A_i)$ avec $A_i = \{a_1, ...a_i\}$ l'ensemble des i arêtes de plus petit poids. On définit par ailleurs le carré d'un graphe $H = (V, E)$, noté H^2, comme étant le graphe (V, E') avec $(u, v) \in E'$ si et seulement si $((u, v) \in E)$ ou (il existe $s \in V$ tel que $(u, s) \in E$ et $(s, v) \in E$). Autrement dit, il y a une arête entre u et v dans H^2 si et seulement si il y a un chemin de longueur au plus 2 entre u et v dans H.

3. Montrer que résoudre le problème du k-centre revient à trouver le plus petit i tel que G_i admet un ensemble dominant de cardinal au plus k.
4. Si H est un graphe et I un ensemble indépendant dans H^2, montrer que $|I| \leq dom(H)$.

Voici l'algorithme d'approximation qu'on se propose d'étudier (prenant en entrée le graphe G) :

```
1 Calculer G_1^2, G_2^2, ..., G_m^2
2 Trouver de manière gloutonne un ensemble indépendant inextensible (auquel
3    on ne peut pas rajouter de sommet), noté M_i, dans chaque G_i^2
4 Déterminer le plus petit indice i tel que |M_i| ≤ k, notons le j
5 renvoyer M_j
```

5. Montrer que $w(a_j) \leq C^\star$ où $C^\star$ est le coût d'une solution optimale au problème du k-centre.
6. Montrer qu'il s'agit bien d'une 2-approximation pour le calcul du k-centre.
7. Déterminer la complexité de cet algorithme.
8. Montrer que le facteur d'approximation de cet algorithme n'est pas strictement plus petit que 2 en exhibant un graphe pour lequel il est atteint.
9. Soit $0 < \varepsilon < 2$. Montrer que si P $\neq$ NP alors il n'existe pas de $(2 - \varepsilon)$ approximation polynomiale au problème du k-centre.

1. Un exemple : on peut construire k arrêts de bus sur une ligne et on veut les répartir de sorte à ce que tous les usagers aient un arrêt de bus relativement proche de chez eux.
2. Déjà, CENTRE est dans NP car étant donné $S' \subset S$, vérifier qu'il est de la bonne taille et que $c(S') \leq m$ se fait en calculant tous les $\delta(u, S')$ pour $u \in S \setminus S'$ ce qui peut se faire en $O(|S|^2)$ en représentant S' à l'aide d'un tableau de taille $|S|$ et d'une représentation du graphe par matrice d'adjacence.

   Réduisons DOM à CENTRE. Soit $(G = (S, A), k)$ une instance de DOM. On le transforme (polynomialement en $|G|$) en $(G' = (S, S^2), 1)$ dans lequel les arêtes de A ont poids 1 et celles en dehors ont poids 2. Remarquons que pour tout $S' \subset S$ dans G', on a $c(S') \in \{1, 2\}$ par construction.

   Si (G, k) est positive pour DOM, il existe un ensemble dominant $D \subset S$ de taille k et choisir cet ensemble dans le graphe complet G' résout le problème CENTRE pour $m = 1$. Réciproquement, si on dispose d'un ensemble $C \subset S$ dans G' pour lequel $c(C) = 1$, alors C est dominant dans G : si $s \in S$ est dans $S \setminus C$, il est a distance un d'un sommet de C donc est bien voisin d'un sommet de C.

Évudemment dans la suite de l'exercice, le problème étudié est le problème d'optimisation associé à CENTRE, qu'on continue d'appeler CENTRE en surchargeant la notation.

3. On constate que :

$$\begin{aligned} D \text{ est dominant dans } G_i &\Leftrightarrow \forall v \in S \setminus D, \exists s \in D, (v,s) \in A_i \\ &\Leftrightarrow \forall v \in S, \exists s \in D, w(v,s) \leq w(a_i) \text{ puisque } a_i \text{ a le plus grand poids dans } G_i \\ &\Leftrightarrow \forall v \in S \setminus D, \delta(v,D) \leq w(a_i) \text{ par définition de } \delta \\ &\Leftrightarrow c(D) \leq w(a_i) \end{aligned}$$

   Ainsi, minimiser $c(S')$ dans un graphe G avec S' de taille k revient à trouver un S' dominant dans le G_i tel que $w(a_i)$ est minimal. Comme les a_i sont dans l'ordre croissant, cela revient à trouver un ensemble dominant de taille k dans G_i avec i minimal.

4. Soit I un ensemble indépendant dans H^2 et D un ensemble dominant dans H.

   - Tout sommet de I est (relié ou égal à) au moins un sommet de D dans H puisque D est dominant.
   - Tout sommet de D est (relié ou égal à) au plus un sommet de I dans H. Sinon, on aurait $x, y \in I$ dominés par un même sommet de H et alors x et y seraient à distance un ou deux dans H donc seraient reliés dans H^2 ce qui contredit le fait que I est indépendant dans H^2.

   On en déduit que tout sommet de I est apparié à un sommet de D (en revanche il peut y avoir des sommets de D en plus) et donc $|I| \leq |D|$. Ceci est valable pour tout D donc $|I| \leq dom(H)$.

5. D'après la question 3, $C^\star = \min\{w(a_i) \,|\, dom(G_i) \leq k\}$. Or, d'après la ligne 4 de l'algorithme, pour tout $i < j$, $|M_i| > k$ et donc d'après la question 4, $dom(G_i) > k$. On en déduit que :

$$C^\star = \min_{i \geq j}\{w(a_i) \,|\, dom(G_i) \leq k\} \geq w(a_j) \text{ puisque les } a_i \text{ sont triées par ordre croissant de poids}$$

6. Il s'agit de montrer que $c(M_j) \leq 2C^\star$. Soit $v \in S \setminus M_j$. Alors il existe $s \in M_j$ tel que (v,s) est une arête de G_j^2, sinon, M_j ne serait pas indépendant maximal pour l'inclusion dans G_j^2 puisqu'on pourrait y ajouter v. Observons le poids de cette arête :

   - Si cette arête (v,s) est dans G_j alors par construction $w(v,s) \leq w(a_j)$.
   - Sinon, il existe un sommet z tel que (v,z) et (z,s) sont des arêtes dans G_j. On peut alors utiliser l'inégalité triangulaire pour conclure que $w(v,s) \leq w(v,z) + w(z,s) \leq 2w(a_j)$.

   Dans tous les cas, $w(v,s) \leq 2w(a_j) \leq 2C^\star$ d'après la question précédente. On a ainsi montré que pour tout $v \in S \setminus M_j$, $\delta(v, M_j) \leq 2C^\star$ et donc en passant au maximum, $c(M_j) \leq 2C^\star$ comme prévu.

7. On note n le nombre de sommets de G et m son nombre d'arêtes. Le tri des arêtes se fait en $O(m \log m)$ Si on représente G par matrice d'adjacence, le calcul d'un graphe G_i^2 se fait en $O(n^3)$ en élevant au carré la matrice d'adjacence de G_i (qui a moins de n sommets). Donc le calcul de tous les G_i se fait en $O(mn^3)$. Le calcul d'un M_i peut se faire de manière gloutonne (on pioche un sommet, on l'ajoute à M_i, on le supprime lui et tous ses voisins) en $O(n^2)$. La ligne 4 se fait en $O(m)$. Total : $O(mn^2)$.

8. TODO

9. Si une telle approximation existait, on pourrait décider `DOM` en temps polynomial ce qui contredirait sa NP-difficulté. En effet, si $(G = (S, A), k)$ est une instance de `DOM`, construire le graphe complet induit par G' dans lequel les arêtes de A ont poids un et celles en dehors ont poids 2 produit une instance de `CENTRE` (version optimisation) en temps polynomial. Alors :

   - Si $dom(G) \leq k$, G' admet un k-centre de coût un.
   - Si $dom(G) > k$ alors tout k-centre de G' a coût deux.

   On applique alors notre $(2 - \varepsilon)$ approximation polynomiale à G' : si elle renvoie une solution (nécessairement de coût un puisqu'elle ne peux pas utiliser d'arête de poids deux), G est positive, sinon, elle est négative.

Exercice 155 (**) *Approximation next-fit pour `BIN PACKING`*

Le problème étudié dans cet exercice est `BIN PACKING`, défini par :

$$\begin{cases} \textbf{Entrée} : n \in \mathbb{N}^\star \text{ objets de volumes } v_1, ..., v_n \text{ et un volume maximal } V \in \mathbb{N}^\star. \\ \textbf{Solution} : \text{Un entier } m \text{ et une fonction } f : [\![1, n]\!] \to [\![1, m]\!] \text{ tels que pour tout } i \in [\![1, m]\!], \sum_{f(v_j)=i} v_j \leq V. \\ \textbf{Optimisation} : \text{Minimiser } m. \end{cases}$$

Autrement dit, on cherche à ranger les n objets dans m boîtes de volume maximal V de telle sorte à ce que la somme des volumes des objets dans une boîte n'excède jamais V et en utilisant le nombre de boîtes minimal. On rappelle que le problème `PARTITION` suivant est NP-complet :

$$\begin{cases} \textbf{Entrée} : \text{Des entiers naturels } c_1, ..., c_n. \\ \textbf{Question} : \text{Y a-t-il un sous-ensemble } S \subset [\![1, n]\!] \text{ tel que } \sum_{i \in S} c_i = \sum_{i \notin S} c_i \, ? \end{cases}$$

1. Décrire le problème de décision `BPD` associé au problème d'optimisation `BIN PACKING`.
2. Montrer que `BPD` $\in$ NP.
3. Étant donnée une instance $c_1, ..., c_n$ de `PARTITION`, on construit l'instance suivante de `BPD` : les volumes sont $2c_1, ...2c_n$, le volume total est $V = \sum_{i=1}^{n} c_i$ et le seuil (donc le nombre maximal de boîtes) est égal à 2. Montrer que cette transformation prouve que `PARTITION` se réduit polynomialement à `BPD` et conclure.

On cherche à approcher les solutions pour `BIN PACKING` via l'algorithme *next-fit*. Son principe est le suivant : il maintient à jour une boîte courante. Pour chaque objet, *next-fit* détermine s'il peut rentrer dans la boîte courante : si oui, il l'y place, si non, il ferme définitivement la boîte courante, ouvre une nouvelle boîte qui devient la nouvelle boîte courante et y place l'objet. On note m le nombre de boîtes déterminé via la stratégie *next-fit* sur une instance donnée de `BIN PACKING` et $m^\star$ le nombre optimal de boîtes pour cette même instance.

4. Montrer que la somme des volumes occupés par des objets de deux boîtes consécutives selon la stratégie *next-fit* est strictement supérieure à V.
5. Montrer que $m^\star V \geq \sum_{i=1}^{n} v_i$.
6. En déduire que *next-fit* est une 2-approximation de `BIN PACKING`.
7. Déterminer la complexité de *next-fit* et commenter.
8. Existe-t-il un $\alpha < 2$ tel que *next-fit* soit une α-approximation de `BIN PACKING` ?

Soit à présent $\varepsilon > 0$ et supposons qu'il existe un algorithme polynomial qui soit une $(3/2 - \varepsilon)$-approximation de `BIN PACKING`.

9. En s'inspirant de la réduction de la question 3, montrer que dans ces conditions il existe un algorithme polynomial permettant de résoudre `PARTITION`. Que peut-on en conclure ?

1. Le problème `BPD` consiste à se demander s'il est possible de ranger les n objets dans moins de m boîtes sans en dépasser la capacité :

$$\begin{cases} \textbf{Entrée} : v_1, ..., v_n \in \mathbb{N}^\star, V \in \mathbb{N}^\star \text{ et } m \in \mathbb{N}. \\ \textbf{Question} : \text{Existe-t-il un entier } k \leq m \text{ et une fonction } f : [\![1, n]\!] \to [\![1, k]\!] \text{ tels que :} \\ \qquad\qquad \text{pour tout } i \in [\![1, k]\!], \sum_{f(v_j)=i} v_j \leq V \, ? \end{cases}$$

2. *Version succinte* : Vérifier si un tableau décrivant un rangement (avec par exemple en case i le numéro de la boîte dans lequel i est rangé) est correct peut se faire en temps polynomial en la taille de l'instance en comptant le nombre de boîtes et en vérifiant que la capacité de chacune est respectée. La taille d'un tel tableau est de l'ordre de n donc polynomiale en la taille de l'entrée.

   Version détaillée dans laquelle on prend en compte la taille des entiers. Si $v_1, ..., v_n, V, m$ est une instance de `BPD`, sa taille est de l'ordre de $n \times \max(\log v_i) + \log m + \log V$ si on ne néglige pas la taille des entiers impliqués.

   Étant donnés une instance de `BPD` et un tableau de taille n contenant en case i un entier inférieur à n (ce qui est loisible car au pire on place chacun des n objets dans une boîte ; la taille de ce tableau est ainsi en $O(n \log n)$ donc polynomiale en la taille de l'instance) correspondant au numéro de la boîte dans laquelle ranger i, on peut vérifier que ce tableau encode un rangement correct en temps polynomial en la taille de l'instance comme suit :

- On calcule le poids de chacune des au plus n boîtes en parcourant le tableau, ce qui peut se faire en $O(n \max(\log v_i))$ puisqu'on va faire au plus n sommes d'entiers de taille inférieure à $\max(\log v_i)$ et on stocke ces poids.
- Pour chacun de ces au plus n poids, on vérifie qu'ils sont inférieurs à V en $O(\log V)$.
- On vérifie qu'il y a moins de m boîtes impliquées ce qui implique une comparaison entre un entier de taille au plus $\log n$ et un entier de taille $\log m$. Là encore, cette opération se fait en temps polynomial en la taille de l'instance.

3. Cette transformation peut être effectuée en temps polynomial en la taille de l'instance de `PARTITION` puisqu'elle exige simplement de faire n sommes et n multiplications par une constante sur les c_i.

   Si $c_1, ..., c_n$ est une instance positive de `PARTITION`, il existe $I \subset [\![1, n]\!]$ tel que $\sum_{i \in I} c_i = \sum_{i \notin I} c_i$. Ainsi :

   $$2V = \sum_{i=1}^{n} 2c_i = \sum_{i \in I} 2c_i + \sum_{i \notin I} 2c_i = 2\left(\sum_{i \in I} 2c_i\right) \text{ donc } \sum_{i \in I} 2c_i = \sum_{i \notin I} 2c_i = V$$

   avec V le volume des boîtes dans l'instance de `BPD` construite à partir de celle de `PARTITION`. Il est donc possible de placer les objets de poids $2c_1, ..., 2c_n$ dans exactement deux boîtes (complètement remplies) de volume V et en prime, le sous-ensemble I nous indique comment les répartir.

   Réciproquement, supposons que $2c_1, ...2c_n, V = \sum_{i=1}^{n} c_i, 2$ est une instance positive de `BPD`. Alors il existe $I \subset [\![1, n]\!]$ tel que les objets dans la première boîte sont ceux indicés par I et ceux dans la deuxième ceux indicés par les éléments n'appartenant pas à I. Comme le volume total de tous les objets vaut $2V$ et qu'il rentrent dans 2 boîtes de volume V alors le volume occupé par les objets de chacune des deux boîtes est exactement égal à V. En particulier ces deux volumes sont égaux donc :

   $$\sum_{i \in I} 2c_i = \sum_{i \notin I} 2c_i$$

   ce qui montre que $c_1, ..., c_n$ est une instance positive de `PARTITION` après simplification par deux.

   Comme `PARTITION` est NP-difficile (puisque NP-complet) et que `PARTITION` se réduit polynomialement à `BPD` d'après ce qui précède, on en déduit que `BPD` est NP-difficile. Comme il est aussi NP d'après la question 2, il est NP-complet.

4. Si on avait deux boîtes consécutives B_1 et B_2 (B_2 ayant été créée après B_1 donc après avoir rencontré un objet qui ne pouvait pas rentrer dans B_1) dont les volumes occupés v_1 et v_2 vérifiaient $v_1 + v_2 \leq V$ alors tous les objets de ces deux boîtes auraient pu rentrer dans B_1 et donc B_2 n'aurait pas été créée d'après le fonctionnement de l'heuristique next-fit. C'est une contradiction.

5. La quantité $m^\star V$ est le volume total des m boîtes et la quantité $\sum_{i=1}^{n} v_i$ est le volume total des objets. Comme les objets rentrent dans les $m^\star$ boîtes, le premier volume est plus grand que le second.

6. Si on note $B_1, ..., B_m$ les m boîtes déterminées par next-fit, en sommant sur les paires de boîtes consécutives et en utilisant la question 4 on a :

   $$\begin{aligned}(m-1)V &< \sum_{i=1}^{m-1} (\text{volume occupé dans } B_i + \text{ volume occupé dans } B_{i+1}) \\ &\leq 2 \times \sum_{i=1}^{m} (\text{volume occupé dans } B_i) \\ &= 2 \times \sum_{i=1}^{n} v_i \quad \text{car le volume total occupé est égal au volume total des objets} \\ &\leq 2m^\star V \quad \text{d'après la question 5}\end{aligned}$$

   On en déduit que $m < 2m^\star + 1$ et toutes ces quantités étant entières, $m \leq 2m^\star$ ce qui conclut.

7. L'ajout d'un nouvel objet dans la liste de boîtes en construction selon la stratégie next-fit peut se faire en temps constant. On traite en tout n objets d'où une complexité en $O(n)$. On vient donc de

troquer un algorithme a priori exponentiel permettant de résoudre `BIN PACKING` de manière exacte pour un algorithme polynomial et même linéaire qui calcule une solution, certes approchée, mais avec la garantie que cette dernière est au pire 2 fois plus grande que l'optimale.

8. Non. Pour tout $n \in \mathbb{N}$, on considère l'instance I_n de `BIN PACKING` suivante : le volume V vaut n et on souhaite ranger $2n$ objets dont les volumes sont donnés par la suite $n, 1, n, 1, ..., n, 1$. Dans ces conditions, next-fit utilise $C_n = 2n$ boîtes en plaçant un objet dans chacune alors que la solution optimale consiste à remplir n boîtes avec les objets de volume n et une boîte avec les n objets de volume 1 pour un total de $C_n^\star = n + 1$ boîtes.

   Comme $C_n/C_n^\star \to 2$ lorsque $n \to +\infty$, cela garantit qu'on ne peut pas trouver de meilleur facteur d'approximation que 2 pour next-fit.

9. On note $\mathcal{P}$ la $(3/2 - \varepsilon)$-approximation dont l'existence est supposée par l'énoncé.

   Soit $c_1, ..., c_n$ une instance de `PARTITION`. On considère l'instance I de `BIN PACKING` $2c_1, ...2c_n, V = \sum_{i=1}^{n} c_i, 2$. Si $c_1, ..., c_n$ est une instance positive de `PARTITION` alors le nombre optimal de boîtes pour I est 2 et le nombre de boîtes m renvoyé par $\mathcal{P}$ sur l'instance I vérifiera donc $m \leq 2 \times (3/2 - \varepsilon) = 3 - 2\varepsilon$. Puisque m est un entier, on aura ainsi $m = 2$. Si $c_1, ..., c_n$ n'est pas une instance positive de `PARTITION`, alors l'algorithme $\mathcal{P}$ appliqué à I renverra au moins 3.

   Ainsi, pour résoudre `PARTITION` polynomialement, il suffit de construire l'instance de `BIN PACKING` correspondante, ce qui se fait polynomialement comme vu en question 3 puis de lui appliquer $\mathcal{P}$, qui est lui aussi polynomial : si le résultat de cet algorithme est 2 alors l'instance considérée de `PARTITION` est positive, sinon, elle est négative.

   On en déduit que, si P $\neq$ NP, alors il n'existe pas d'algorithme d'approximation à facteur constant strictement inférieur à 3/2 pour `BIN PACKING`.

Exercice 156 (exo cours) *Non approximation*

On fixe un problème P de maximisation.

1. Décrire une méthode générique pour montrer qu'un algorithme A n'est pas un algorithme d'approximation à facteur constant pour le problème P. Montrer que la méthode proposée est correcte.
2. Cette méthode fonctionne-t-elle si P est un problème de minimisation ? Si non, comment l'adapter ?

1. Il suffit d'exhiber une suite $(I_n)_{n\in\mathbb{N}}$ d'instances de P pour laquelle le coût C_n de la solution renvoyée par A sur I_n et le coût $C_n^\star$ d'une solution optimale pour I_n vérifient $C_n/C_n^\star \to 0$. Par l'absurde, si A est une $(\alpha > 0)$-approximation, on aurait pour tout n : $\alpha C_n^\star \leq C_n$ ce que cette limite contredit.
2. Pour un problème de minimisation, on construit plutôt I_n de sorte que $C_n/C_n^\star \to +\infty$.

Exercice 157 (***) *Approximation pour `SET COVER`*

On considère le problème d'optimisation `SET COVER` (dont on admet la NP-complétude) :

$$\begin{cases} \textbf{Entrée} : \text{Un ensemble } E \text{ fini et } E_1, ..., E_n \text{ des sous ensembles de } E. \\ \textbf{Solution} : \text{Un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \bigcup_{i\in I} E_i = E. \\ \textbf{Optimisation} : \text{Minimiser } |I|. \end{cases}$$

Autrement dit, on veut couvrir E avec le moins de E_i possible. Observons l'algorithme glouton suivant :

$I \leftarrow \emptyset$ et $U \leftarrow E$
tant que $U \neq \emptyset$
  Déterminer i tel que $E_i \cap U$ est maximal
  Ajouter i à I
  $U \leftarrow U \setminus E_i$
renvoyer I

1. Appliquer cet algorithme glouton à l'instance suivante de `SET COVER` : $E = [\![1, 8]\!]$, $E_1 = \{5, 6, 7, 8\}$, $E_2 = \{3, 4\}$, $E_3 = \{1\}$, $E_4 = \{2\}$, $E_5 = \{1, 3, 5, 7\}$, $E_6 = \{2, 4, 6, 8\}$. Montrer que la couverture optimale est de taille 2 mais que l'algorithme glouton peut en renvoyer une de taille 4.

2. Montrer que cet algorithme est polynomial en expliquant les choix de structures effectués.

3. Montrer que cet algorithme n'est pas un algorithme d'approximation à facteur constant pour `SET COVER`.

Notons $m = |E|$, $C^\star = (E_i)_{i\in I}$ une couverture de plus petite taille pour l'instance $(E, E_1, ...E_n)$ et C la couverture calculée par l'algorithme glouton sur cette même instance. On note enfin m_i le nombre d'éléments qui ne sont pas encore couverts au i-ème tour de boucle : pour $i = 0$ on a donc $m_0 = m$.

4. Montrer que pour tout $i \geq 0$, $m_i \leq m\left(1 - \frac{1}{|C^\star|}\right)^i$.

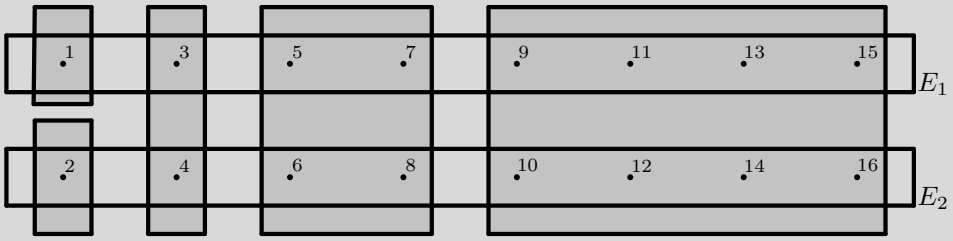
5. En déduire que $|C| \leq |C^\star| \times \lceil \log m \rceil$. On pourra utiliser l'inégalité de convexité : pour tout $x > 0$,

$$\left(1 - \frac{1}{x}\right)^x < \frac{1}{e}$$

1. Le principe de l'algo est de choisir le sous ensemble contenant le plus d'éléments non couverts. $\{E_5, E_6\}$ forme une couverture optimale mais l'algo glouton peut choisir $\{E_1, E_2, E_3, E_4\}$.

2. Si on représente les ensembles par des tableaux de taille $m = |E|$, une itération coûte $O(mn)$ et comme on fait au plus n itérations, la complexité est en $O(n^2m)$.

3. Le principe est de reproduire le comportement de la question 1 (ci dessous, version à 16 éléments) :



Formellement, on note I_n l'instance définie par : $E = [\![1, 2^n]\!]$, E_1 = nombres pairs de E, E_2 = nombres impairs de E, et $n+1$ autres ensembles disjoints de tailles respectivement $1, 1, 2, 4, ..., 2^{n-1}$ (ce qui est possible car $1 + (1 + 2 + 4 + ... + 2^{n-1}) = 1 + 2^n - 1 = 2^n$). La solution optimale compte $C^\star_n = 2$ sous-ensembles mais l'algo glouton pourrait en sélectionner $C_n = n + 1$ et $C_n/C^\star_n \to +\infty$.

4. On procède par récurrence sur i. OK si $i = 0$. Au i-ème tour de boucle, il reste m_i éléments à couvrir. Mais $C^\star$ couvre ces éléments avec $|C^\star|$ ensembles donc un des sous-ensembles de $C^\star$ couvre au moins $m_i/|C^\star|$ des m_i éléments restants. Donc le sous-ensemble choisi à l'étape i couvre au moins ce nombre d'éléments et après l'avoir choisi il reste au plus $m_i - m_i/|C^\star|$ éléments à couvrir. Donc :

$$m_{i+1} \leq m_i - \frac{m_i}{|C^\star|} = m_i\left(1 - \frac{1}{|C^\star|}\right) \leq m\left(1 - \frac{1}{|C^\star|}\right)^{i+1} \text{ par hypothèse récurrente}$$

5. L'algo s'arrête quand $m_i < 1$ (plus d'éléments à couvrir) et donc $|C|$ vaut le nombre de tours de boucle effectué par cet algo. Pour $i = |C^\star|\lceil \log m \rceil$, la question 4 et l'inégalité rappelée donnent :

$$m_i \leq \left(1 - \frac{1}{|C^\star|}\right)^{|C^\star|\lceil \log m \rceil} < m \times \frac{1}{e^{\lceil \log m \rceil}} < \frac{m}{m} = 1$$

Donc l'algo fait au plus $|C^\star|\lceil \log m \rceil$ itérations, donc $|C| \leq |C^\star|\lceil \log m \rceil$.

Exercice 158 (**) *Recherche de bons clusters*

On considère le problème `CLUSTERING` suivant :

$$\begin{cases} \textbf{Entrée} : \text{Un ensemble } X = \{x_1, ..., x_n\} \text{ de points du plan muni de la distance euclidienne } d, \text{ un entier } k. \\ \textbf{Solution} : \text{Une partition de } X \text{ en } k \text{ clusters } C_1, ..., C_k. \\ \textbf{Optimisation} : \text{Minimiser le diamètre } \max\limits_{1\leq i\leq k} \max\limits_{x,y\, \in\, C_i} d(x, y). \end{cases}$$

Autrement dit, on cherche à rassembler les points de X en k paquets de sorte que les deux points les plus éloignés dans un même paquet restent relativement proches.

1. Donner un exemple de contexte dans lequel on pourrait vouloir résoudre ce problème.
2. Définir le problème de décision associé à ce problème d'optimisation.
3. Montrer que ce problème est NP. On admet qu'il est en fait NP-complet.

On considère l'algorithme suivant. Son entrée est une instance (X, k) de `CLUSTERING`. Les y_i introduits par l'algorithme s'appellent des centres :

Choisir un point $y_1 \in X$
pour i allant de 2 à k
| Déterminer un point $y_i \in X$ qui maximise $\min_{j<i} d(., y_j)$
pour i allant de 1 à k
| $C_i = \{x \in X \mid \text{ pour tout } j \neq i, d(x, y_i) < d(x, y_j)\}$
renvoyer les C_i ainsi calculés

4. Décrire en français le fonctionnement de cet algorithme. En l'état, certains points ne sont intégrés dans aucun cluster. Expliquer comment modifier l'algorithme pour que chaque point soit intégré à l'un des C_i.
5. Montrer que cet algorithme est une 2-approximation de `CLUSTERING`. On pourra pour ce faire considérer le point x le plus éloigné des $y_1, ..., y_k$ et sa distance $d = \min_{i=1}^{k} d(x, y_i)$ au plus proche de ces points.
6. Montrer que 2 est le meilleur facteur d'approximation pour cet algorithme.

1. Dans le cadre d'un algorithme d'apprentissage non supervisé où on sait qu'il y a k classes.
2. On rajoute un seuil r auquel le diamètre doit être inférieur.
3. Étant donnés moins de n sous-ensembles de X (cette donnée est de taille polynomiale en la taille d'une instance de `CLUSTERING DECI`), on vérifie en temps polynomial que :
   - Ils forment une partition de X.
   - Il y a bien k sous ensembles.
   - Le diamètre de chaque sous-ensemble (calculable polynomialement) est inférieur à r.
4. Première étape : calculer les centres. Le $(i+1)$-ème est le point le plus éloigné de tous les centres déjà existants. Deuxième étape : assigner chaque point au cluster dont le centre est le plus proche. En l'état, si un point est à égale distance de deux centres différents, il n'est placé dans aucun cluster. En cas d'égalité, on peut l'intègrer arbitrairement dans le cluster dont le centre a le plus petit indice.
5. Notons D le diamètre trouvé par l'algo et $D^\star$ le diamètre optimal pour une instance donnée. x est le prochain centre qu'on aurait choisi s'il y avait $k+1$ clusters donc le point le plus éloigné des $y_1, ..., y_k$. Donc, tout point de X est à distance inférieure à d de son centre le plus proche. Donc :

$$\forall i \in [\![1, k]\!], \forall y, z \in C_i, d(y, z) \leq d(y, y_i) + d(y_i, z) \leq 2d$$

   Ainsi, $D \leq 2d$. De plus, deux points de $\{y_1, ..., y_k, x\}$ sont à distance au moins d par construction. Dans la répartition optimale, deux de ces points sont dans le même cluster qui a donc un diamètre plus grand que d. Donc $D^\star \geq d$. Donc $D \leq 2D^\star$ comme prévu.
6. Pour tout $\varepsilon > 0$, on considère les 4 points $x_1 = (0, 0)$, $x_2 = (1, 0)$, $x_3 = (2 - \varepsilon, 0)$ et $x_4 = (3 - \varepsilon, 0)$ à répartir en $k = 2$ clusters. La répartition optimale est $\{x_1, x_2\}$, $\{x_3, x_4\}$ de coût $C^\star_\varepsilon = 1$. Mais si l'algo glouton choisit x_2 comme premier centre alors le second sera x_4 et la répartition sera $\{x_1, x_2, x_3\}$, $\{x_4\}$ de diamètre $C_\varepsilon = 2 - \varepsilon$. Comme $C_\varepsilon / C^\star_\varepsilon \to 2$ quand $\varepsilon \to 0$, 2 est le meilleur facteur d'approximation.

Exercice 159 (**) *Approximation pour `SUBSET SUM`*

On considère le problème d'optimisation `SUBSET SUM` suivant :

$$\begin{cases} \textbf{Entrée} : \text{Un tableau } T = [t_1, ..., t_n] \text{ d'entiers naturels et } C \in \mathbb{N}. \\ \textbf{Solution} : \text{Un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \sum_{i \in I} t_i \leq C. \\ \textbf{Optimisation} : \text{Maximiser } \sum_{i \in I} t_i. \end{cases}$$

La version décisionnelle de ce problème est NP-complète. On propose d'approcher une solution à `SUBSET SUM` à l'aide de l'algorithme glouton suivant :

```
S ← 0 et I ← ∅
pour i allant de 1 à n
    si S + t_i ≤ C alors
        S ← S + t_i
        I ← I ∪ {i}
renvoyer I
```

1. Montrer que ce n'est pas un algorithme d'approximation à facteur constant pour `SUBSET SUM`.

On modifie l'algorithme précédent de sorte à considérer les t_i par ordre décroissant de valeur (plutôt que par ordre croissant d'indice) : cela donne naissance à un algorithme qu'on note A.

2. Déterminer la complexité temporelle de A.
3. Montrer que A est une 1/2-approximation de `SUBSET SUM`.
4. On vient de montrer que le facteur d'approximation de A est au moins égal à 1/2. Est-il meilleur ?

1. Pour tout $n \geq 1$, on considère l'instance de `SUBSET SUM` caractérisée par $C = n$ et $T = [1, n]$, La somme optimale pour I_n est $S_n^\star = n$, l'algo glouton propose une somme $S_n = 1$ sur cette entrée et $S_n/S_n^\star \to 0$.
2. Le tri peut se faire en $O(n \log n)$ puis la boucle se fait en $O(n)$ pour un total en $O(n \log n)$.
3. Pour une instance donnée, on note S la somme obtenue avec l'algo et $S^\star$ la somme optimale :
   - Si la somme de tous les t_i est inférieure à C alors $S = S^\star$.
   - Si aucun t_i n'est inférieur à C alors $S = S^\star = 0$.
   - Sinon, au moins un t_i est dans S et ils n'y sont pas tous. Au moment où on rencontre le premier t_i qui ne rentre plus, on a $S_{\text{courant}} \geq C/2$. Sinon $S_{\text{courant}} < C/2$ mais les t_j dans S_{courant} (et il y en a) sont tous supérieurs à t_i donc $t_i \leq C/2$ et donc on pourrait l'ajouter à la somme. On en déduit que $S \geq S_{\text{courant}} \geq C/2 \geq S^\star/2$ car au mieux $S^\star = C$.

   Dans tous les cas, on a bien $S^\star/2 \leq S$.
4. Non. On considère pour tout $n \geq 1$ l'instance $C = 2n, T = [n+1, n, n]$. La somme trouvée par l'algo glouton sur cette instance est $S_n = n + 1$ alors que l'optimale est $S_n^\star = 2n$ et $S_n/S_n^\star \to 1/2$.

8 Algorithmes probabilistes

Exercice 160 (**) *Algorithme de Karger*

Soit $G = (S, A)$ un multigraphe (il peut y avoir plusieurs arêtes entre deux sommets) connexe non orienté sans boucle (il n'y a pas d'arête entre un sommet et lui même). On note $n = |S|$ le nombre de sommets de G. Une *coupe* de G est une partition de S en deux sous ensembles X et Y tous les deux non vides. L'ensemble des arêtes d'une coupe (X, Y), qu'on appelle souvent une coupe aussi, par abus, est $C_{X,Y} = \{(x, y) \in A \mid x \in X \text{ et } y \in Y\}$. La taille d'une coupe (X, Y) est le cardinal de $C_{X,Y}$.

Dans cet exercice, on étudie le problème d'optimisation `COUPE MIN` (`MIN-CUT` en anglais) :

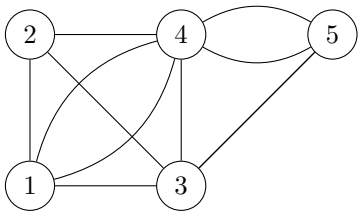
$$\begin{cases} \textbf{Entrée} : \text{Un multigraphe } G \text{ sans boucle connexe non orienté.} \\ \textbf{Solution} : \text{Une coupe } C \text{ de } G. \\ \textbf{Optimisation} : \text{Minimiser la taille de } C. \end{cases}$$

Il existe des algorithmes déterministes polynomiaux pour le calcul d'une coupe minimale mais en pratique les algorithmes probabilistes sont plus efficaces pour résoudre ce problème. On étudie ici l'un d'entre eux : l'algorithme de Karger. C'est un algorithme probabiliste de type Monte Carlo dont l'opération principale est la *contraction* d'arêtes de G. Si G est un multigraphe et (s, t) est une arête dans G, contracter l'arête (s, t) consiste à construire le multigraphe noté $G/(s, t)$ correspondant au multigraphe G dans lequel s et t ont été fusionnés en r, où les arêtes entre s et t ont disparu et dans lequel toute arête entre s et u devient une arête entre r et u (de même pour les arêtes entre t et u). Ainsi :

tant que G a strictement plus de 2 sommets
  Séléctionner uniformément aléatoirement une arête a de G
  $G \leftarrow G/a$
renvoyer la partition induite par les sommets de G

Algorithme 1 - Algorithme de Karger

1. Appliquer l'algorithme de Karger sur ce multigraphe et indiquer la coupe obtenue et sa taille :



2. Montrer que l'algorithme de Karger peut être implémenté de sorte à avoir une complexité en $O(n^2)$.
3. Montrer que l'algorithme de Karger renvoie bien une coupe de G. Est-elle forcément minimale ?
4. Soit C une coupe minimale de G de cardinal k. On note E l'évènement "l'algorithme de Karger appliqué à G renvoie la coupe C" et E_i l'évènement "à la i-ème itération de l'algorithme de Karger sur G l'arête choisie pour être contractée ne fait pas partie de C".
   a) Montrer que G a au moins $kn/2$ arêtes.
   b) Expliquer pourquoi C est la coupe renvoyée par l'algorithme de Karger si et seulement si aucune des arêtes de C n'est sélectionnée pour être contractée.
   c) Montrer que $\mathbb{P}(E_1) \geq (1 - 2/n)$.
   d) Montrer que $\mathbb{P}(E) \geq 2/n^2$.
5. Si on suppose qu'il y a m coupes minimales et qu'on tire aléatoirement et uniformément une coupe de G, quelle est la probabilité qu'elle soit minimale (en fonction de n et m) ? Comparer avec la probabilité que la coupe renvoyée par l'algorithme de Karger soit minimale et commenter.

La probabilité d'avoir calculé une coupe minimale avec l'algorithme de Karger est bien plus élevée que la probabilité d'avoir trouvé une coupe minimale en créant la partition des sommets au hasard mais reste très faible. On peut l'amplifier en répétant l'algorithme de Karger et en conservant la plus petite coupe obtenue.

6. Montrer que pour tout $0 < \varepsilon < 1$, la probabilité que $\frac{n^2}{2} \ln \frac{1}{\varepsilon}$ répétitions indépendantes de l'algorithme de Karger produisent une coupe minimale est supérieure ou égale à $1 - \varepsilon$. *Indication : Montrer auparavant que pour tout $x > 0$, on a :*
$$\left(1 - \frac{1}{x}\right)^x < \frac{1}{e}$$
7. Quel est l'ordre de grandeur de la complexité d'un algorithme utilisant l'algorithme de Karger et produisant une coupe minimale avec une "bonne" probabilité ?

1. En contractant $(3, 4)$ puis une des arêtes entre 2 et $\{3, 4\}$ puis une des arêtes entre $\{2, 3, 4\}$ et 5, on obtient la coupe $\{1\} \sqcup \{2, 3, 4, 5\}$ qui est de taille 4.
2. On fait $n - 2$ contractions et chaque contraction peut se faire en $O(n)$ (union des partitions + modifications des extrémités des arêtes pour en rendre compte).
3. Les meta-sommets de G forment une partition de S est un invariant de la boucle tant que. À la fin, il ne reste que deux meta-sommets : on renvoie donc bien une coupe. Elle n'est pas forcément minimale : la coupe trouvée en Q1 est de taille 4 alors que la coupe $\{5\} \sqcup \{1, 2, 3, 4\}$ est de taille 3.
4. a) Chaque sommet de G est de degré au moins k sinon il existerait u de degré strictement plus petit que k et $\{u\} \sqcup (S \setminus \{u\})$ serait une coupe strictement plus petite qu'une coupe minimale. Comme il y a n sommets de degré au moins k, il y a au moins kn extrémités d'arêtes et comme une arête a deux extrémités, il y a au moins $kn/2$ arêtes dans G.
   b) Si aucune arête de C n'est contractée, ce sont exactement celles du graphe final donc la coupe renvoyée est C. Si la coupe renvoyée est C, aucune arête de C n'a pu être contractée sinon la

partition obtenue in fine ne pourrait pas être celle induite par C.

c) d) D'après Q3b, E est l'évènement "l'algorithme ne choisit jamais une arête de C", c'est-à-dire l'événement $\bigcap_{i=1}^{n-2} E_i$ puisqu'il y a $n-2$ contractions. Les E_i ne sont pas indépendants donc :

$$\mathbb{P}(E) = \mathbb{P}(E_1)\mathbb{P}(E_2|E_1)\mathbb{P}(E_3|E_1 \cap E_2)...\mathbb{P}\left(E_{n-2}|\bigcap_{i=1}^{n-3} E_i\right)$$

Or, la probabilité de choisir une arête de C à la première étape :

$$\frac{\text{nombre d'arêtes dans } C}{\text{nombre d'arêtes total}} \leq k \times \frac{2}{kn} \text{ d'après Q3a}$$

Donc $\mathbb{P}(E_1) \geq 1 - 2/n$. De plus, si E_1 se produit, le graphe contracté contient maintenant $n-1$ sommets et une coupe minimale dans ce graphe est toujours de taille k donc Q3a assure que ce graphe contracté a au moins $k(n-1)/2$ arêtes. Donc $\mathbb{P}(E_2|E_1) \geq 1 - 2/(n-1)$. En réappliquant ce raisonnement, on obtient en fait que $\mathbb{P}(E_i|\bigcap_{j=1}^{i-1} E_j) \geq 1 - 2/(n-i+1)$. Par conséquent :

$$\mathbb{P}(E) \geq \prod_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right) = \prod_{i=1}^{n-2}\frac{n-i-1}{n-i+1} = \frac{(n-2)\times(n-3)\times...\times 1}{n\times(n-1)\times...\times 3} = \frac{2}{n(n-1)} \geq \frac{2}{n^2}$$

5. Pour créer une coupe, on choisit un sommet qu'on met dans X (par exemple) et on répartit les $n-1$ autres sommets dans X et Y en interdisant de tous les mettres dans X : il y a donc $2^{n-1}-1$ possibilités de répartitions. La probabilité d'obtenir une coupe est donc de $m/(2^{n-1}-1)$. On peut montrer (mais pas ici) que $m = n(n-1)/2$, cette quantité est donc bien plus petite que $2/n^2$.
6. La probabilité que Karger ne renvoie pas une coupe minimale est inférieure à $1-2/n^2$ donc la probabilité que k répétitions de Karger indépendantes ne renvoie pas une coupe minimale est majorée par $(1-2/n^2)^k$. On utilise l'inégalité rappelée avec $x = n^2/2$ puis on passe l'inégalité à la puissance $\ln(1/\varepsilon)$ pour obtenir que :
$$\left(1 - \frac{2}{n^2}\right)^{\frac{n^2}{2}\ln\left(\frac{1}{\varepsilon}\right)} < \left(\frac{1}{e}\right)^{\ln\left(\frac{1}{\varepsilon}\right)} = \varepsilon$$
Ainsi, avec $k = (n^2/2)\ln(1/\varepsilon)$ répétitions, la probabilité de ne pas avoir renvoyé une coupe minimale est inférieure à ε, donc on a trouvé une coupe minimale avec probabilité au moins $1-\varepsilon$.
7. Prenons $\varepsilon = 10^{-60}$ (pourquoi pas). D'après Q5, il suffit alors de faire environ $70n^2$ répétitions recherches de coupes minimales, chacune en $O(n^2)$ d'après Q2, pour obtenir une coupe satisfaisante avec probabilité au moins $1-10^{-60}$. Le coût de cette recherche est donc en $O(n^4)$.

Exercice 161 (**) *Quickselect*

On cherche à résoudre le problème suivant : étant donné un tableau $T = [t_0, ..., t_{n-1}]$ et un entier k, déterminer le k-ème plus petit élément de T.

1. Décrire en français un algorithme de complexité temporelle pire cas en $O(n\log n)$ et de complexité spatiale constante permettant de résoudre le problème (on s'autorise à modifier T au cours de cet algorithme).
2. En s'inspirant de l'algorithme du tri rapide randomisé, décrire un algorithme utilisant la stratégie diviser pour régner pour résoudre ce problème et justifier sa correction.
3. Déterminer sa complexité pire cas.

Pour tout $n \geq 1$, on note X_n la variable aléatoire comptant le nombre de comparaisons effectuées par cet algorithme sur un tableau de taille n. On souhaite montrer que $\mathbb{E}[X_n] \leq 4n$. Pour ce faire, on procède par récurrence forte.

4. Justifier l'initialisation de cette récurrence. On suppose à présent le résultat souhaité prouvé pour tout $m < n$. On introduit la variable aléatoire Y comptant le nombre d'éléments à gauche du pivot lorsqu'il

est choisi dans le tableau de taille n. Montrer que

$$\mathbb{E}[X_n] = \frac{1}{n}\sum_{m=0}^{n-1} \mathbb{E}[X_n \,|\, Y = m]$$

5. Montrer que $\sum_{m=0}^{n-1} \max(m, n-m-1) \leq \frac{3n^2-3}{4}$.
6. Déduire des questions précédentes que l'hérédité de notre récurrence est acquise et conclure.

1. On trie le tableau en $O(n \log n)$ via un tri par tas (ni fusion, ni rapide) puis on donne son k-ème élément.
2. On partitionne comme pour le tri rapide et on cherche le k-ème élément d'un des deux côtés du pivot :

```
Quickselect (T,k) :
Choisir uniformément un pivot x ∈ T et partitionner T en M = {y < x} et P = {y ≥ x}
si |M| < (k − 1) alors
  | renvoyer Quickselect (P, k − |M|)
si |M| = k − 1 alors
  | renvoyer x
sinon
  | renvoyer Quickselect (M, k)
```

3. Même pire cas que le tri rapide : si on cherche le plus petit élément et que le pivot choisi est systématiquement le plus grand élément de l'ensemble restant, on obtient une complexité en $O(n^2)$.
4. Les événements $\{Y = m\}$ sont équiprobables donc $P(Y = m) = 1/n$. De plus, $\{Y = m \,|\, m \in [\![0, m-1]\!]\}$ est une partition de l'univers donc la formule de l'espérance totale dit :

$$\mathbb{E}[X_n] = \sum_{m=0}^{n-1} \mathbb{E}[X_n \,|\, Y = m] \times \mathbb{P}(Y = m) = \frac{1}{n}\sum_{m=0}^{n-1} \mathbb{E}[X_n \,|\, Y = m]$$

5. On a :

$$\begin{aligned}\sum_{m=0}^{n-1} &= 2\sum_{m=\lceil (n-1)/2 \rceil}^{n-1} m = 2\left(\sum_{m=0}^{n-1} m - \sum_{m=0}^{\lceil (n-1)/2 \rceil - 1} m\right) \\ &\leq 2\left(\frac{n(n+1)}{2} - \frac{1}{2}\left\lceil \frac{n-1}{2} \right\rceil \left(\left\lceil \frac{n-1}{2} \right\rceil - 1\right)\right) \leq n(n-1) - \frac{(n-1)(n-3)}{4} \leq \frac{3n^2-3}{4}\end{aligned}$$

6. Par HP, $\mathbb{E}[X_m] \leq 4m$ quand $m < n$. Si $Y = m$, le nombre de comparaisons faites vaut en espérance $n-1$ (comparaisons avec le pivot) + le nombre de comparaisons espérées sur un tableau de taille m ou sur un tableau de taille $n-m-1$ selon lequel est le plus grand. Donc $\mathbb{E}[X_n \,|\, Y = m] \leq n-1+4\max(m, n-m+1)$. La question 4 donne alors en majorant la somme grâce à 5 : $\mathbb{E}[X_n] \leq n-1+(3n^2-3)/n \leq 4n$.

Exercice 162 (**) *ABR aléatoire*

On considère l'opération suivante : étant donnés n éléments distincts et comparables, les insérer dans un arbre binaire de recherche (ABR).

1. Quelle est la complexité pire cas d'une recherche dans le pire ABR résultant de cette opération en fonction de n ? Et dans le meilleur ABR ? Donner un ordre d'insertion lorsque les éléments sont ceux de $[\![1, 7]\!]$ pour lequel un des pires ABR est produit puis un ordre pour lequel un des meilleurs est produit.

Afin d'éviter le pire cas, on insère les n éléments en les choisissant dans un ordre aléatoire : on considère que les $n!$ permutations possibles des éléments sont des ordres d'insertion équiprobables. On note H_n la variable aléatoire correspondant à la hauteur de l'arbre obtenu, $X_n = 2^{H_n}$, R_n le rang de la racine de l'arbre parmi les n éléments (c'est-à-dire sa position si les n éléments étaient triés) et $Z_{n,i}$ la variable aléatoire valant 1 si $R_n = i$ et 0 sinon.

2. Montrer que, si $R_n = i$ alors $X_n = 2\max(X_{i-1}, X_{n-i})$.

3. Montrer que $X_n = \sum_{i=1}^{n} Z_{n,i} \times 2\max(X_{i-1}, X_{n-i})$.

4. On admet que $Z_{n,i}$ est indépendante de X_{i-1} et X_{n-i}, montrer que $\mathbb{E}[X_n] \leq \frac{4}{n} \sum_{k=0}^{n-1} \mathbb{E}[X_k]$.

On admet qu'une telle relation de récurrence implique que $\mathbb{E}[X_n] \leq \frac{1}{4}\binom{n+3}{n}$.

5. En utilisant l'inégalité de Jensen déduire de ce qui précède que $\mathbb{E}[H_n] \leq O(\log n)$ et conclure.

1. La complexité pire cas d'une recherche est en $O(n)$ dans le pire ABR et $O(\log n)$ dans le meilleur. Dans le cas où on stocke $[\![1, 7]\!]$, l'ordre croissant donne un pire ABR et l'ordre $4, 2, 6, 1, 3, 5, 7$ un meilleur.
2. Si $R_n = i$, le fils gauche de l'arbre est un ABR généré aléatoirement à $i-1$ éléments et son fils droit un ABR généré aléatoirement à $n-i$ éléments. Donc $H_n = 1 + \max(H_{i-1}, H_{n-i})$ et on passe à la puissance.
3. Il n'y a qu'une racine dans l'arbre donc 2) conclut en ajoutant les termes (nuls) manquants.
4. Par équiprobabilité des permutations, $\mathbb{P}(R_n = i) = 1/n$ donc $\mathbb{E}[Z_{n,i}]$ aussi. Par linéarité de l'espérance, l'indépendance proclamée par l'énoncé et le fait que $X_n \leq \sum_{i=1}^{n} 2Z_{n,i}(X_{i-1} + X_{n-i})$ par 3) :

$$\mathbb{E}[X_n] = 2\sum_{i=1}^{n} \mathbb{E}[Z_{n,i}](\mathbb{E}(X_{i-1}) + \mathbb{E}(X_{n-i})) \leq 2\sum_{i=1}^{n} \frac{1}{n}(\mathbb{E}(X_{i-1}) + \mathbb{E}(X_{n-i})) = \frac{4}{n}\sum_{i=0}^{n-1} \mathbb{E}[X_i]$$

5. Jensen dit que si f est convexe, $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$. Comme $\mathbb{E}[X_n] \leq (1/4)\binom{n+3}{n} \leq Kn^3$ avec K constante indépendante de n et que $x \mapsto 2^x$ est convexe on a donc : $2^{\mathbb{E}[H_n]} \leq \mathbb{E}[2^{H_n}] = \mathbb{E}[X_n] \leq Kn^3$. On prend le logarithme de cette inégalité et $\mathbb{E}[H_n] = O(\log n)$ ce qui veut dire qu'en espérance on est dans le cas favorable d'un ABR équilibré.

Exercice 163 (***) *Classes de complexité probabilistes Monte Carlo*

On définit les classes de complexité probabilistes suivantes :

- BPP est l'ensemble des problèmes de décision tels qu'il existe un algorithme Monte Carlo en temps polynomial dont la probabilité d'erreur (faux positif ou faux négatif) est strictement inférieure à 1/3.
- PP est l'ensemble des problèmes de décision tels qu'il existe un algorithme Monte Carlo en temps polynomial dont la probabilité d'erreur est strictement inférieure à 1/2.
- RP est l'ensemble des problèmes de décision tels qu'il existe un algorithme Monte Carlo en temps polynomial dont la probabilité de faux positif vaut 0 et dont la probabilité de faux négatif est strictement inférieure à 1/2.

1. Montrer la suite d'inclusions suivante : P $\subset$ RP $\subset$ BPP $\subset$ PP.
2. Montrer que RP $\subset$ NP (en s'autorisant à considérer NP comme l'ensemble des problèmes résolvables en temps polynomial par un algorithme non déterministe).
3. Montrer que NP $\subset$ PP. On pourra montrer que `SAT` $\in$ PP.

1. Un algo déterministe est un algo probabiliste qui ne fait pas d'erreur donc P $\subset$ RP. Si A est un problème dans RP via un algo ayant proba de faux négatif $p < 1/2$, répéter deux fois cet algo donne un algo polynomial de proba de faux négatif $\leq p^2 < 1/4 < 1/3$ donc $A \in$ BPP. BPP $\subset$ PP par définition.
2. Si A est un algo polynomial sans faux positif associé à un problème Π dans RP, modifier les choix aléatoires de A en choix non déterministes en fait un algo polynomial B non déterministe qui résout Π :
   - Si x est positive pour Π, la proba que A renvoie oui sur x est au moins 1/2 donc il existe des exécutions de B sur x qui renvoient oui.
   - Si x est négative, A renvoie toujours non donc toutes les exécutions de B aussi.
3. Si $A \in$ NP il se réduit polynomialement à `SAT` donc si `SAT` $\in$ PP, NP $\subset$ PP. Considérons :

Entrée : Une formule φ sur n variables
1 Choisir une valuation v sur n variables aléatoire
2 Si $v(\varphi) =$ vrai, renvoyer oui
3 Choisir $k \in [\![0, 2^{n+1} - 1]\!]$
4 Si $k < 2^n - 1$ renvoyer oui et sinon non

Cet algo est bien polynomial en $|\varphi|$. De plus :

- Si φ est satisfiable la proba que $v(\varphi) = 1$ vaut au moins $1/2^n$ et donc :

$$\begin{aligned}\mathbb{P}(\text{algo dit oui}) &= \overbrace{\mathbb{P}(\text{oui} \,|\, \text{oui est du aux lignes 1-2})}^{=1} \times \mathbb{P}(\text{oui est du aux lignes 1-2}) \\ &\quad + \underbrace{\mathbb{P}(\text{oui} \,|\, \text{oui est du à ligne 4})}_{=Q} \times \mathbb{P}(\text{oui est du à ligne 4}) \\ &= \mathbb{P}(\text{oui est du aux lignes 1-2}) + Q(1 - \mathbb{P}(\text{oui est du aux lignes 1-2})) \\ &= \mathbb{P}(\text{oui est du aux lignes 1-2})(1 - Q) + Q \\ &\geq 1/2^n + (1 - 1/2^n)Q = 1/2^n + (1 - 1/2^n) \times (2^n - 1)/2^{n+1} > 1/2\end{aligned}$$

  Donc la proba d'erreur sur les instances positives est $< 1/2$.

- Si φ n'est pas satisfiable, la proba de renvoyer faux est celle de choisir $k \geq 2^n - 1$ donc vaut $1 - (2^n - 1)/2^{n+1} = 1/2 + 1/2^{n+1} > 1/2$. La proba d'erreur sur les instances négatives est $< 1/2$.

On en déduit que la proba d'erreur sur toute instance est $< 1/2$ et donc cet algo montre que `SAT` $\in$ PP.

Exercice 164 (***) *Classes de complexité probabilistes Las Vegas*

On définit les classes de complexité probabilistes suivantes :

- RP est l'ensemble des problèmes de décision tels qu'il existe un algorithme Monte Carlo en temps polynomial dont la probabilité de faux positif vaut 0 et dont la probabilité de faux négatif est strictement inférieure à 1/2.
- coRP est l'ensemble des problèmes de décision tels qu'il existe un algorithme Monte Carlo en temps polynomial dont la probabilité de faux négatif vaut 0 et la probabilité de faux positif est strictement inférieure à 1/2.
- ZPP est l'ensemble des problèmes de décision tels qu'il existe un algorithme qui les résout dont l'espérance du temps d'exécution est polynomiale.

1. Montrer que P $\subset$ ZPP.
2. Montrer que RP $\cap$ coRP $\subset$ ZPP.
3. L'inclusion réciproque est-elle vraie ?

1. Un algo déterministe polynomial a en particulier une espérance de temps de calcul polynomiale.
2. Si $\Pi \in$ RP $\cap$ coRP, il existe deux algos A_1 et A_2 de complexité polynomiale p_1 et p_2 et respectivement sans faux positif et proba de faux négatif $< 1/2$ / sans faux négatif et proba de faux positif $< 1/2$. Alors notons B l'algo qui sur une entrée x calcule indéfiniment $y_1 = A_1(x)$, $y_2 = A_2(x)$ et qui si $y_1 = y_2$ renvoie y_1. Cet algo résout P : si $x \geq 0$, A_2 renvoie oui donc si $y_1 = y_2$, on renvoie bien oui ; si $x \leq 0$, A_1 renvoie non donc si $y_1 = y_2$ on renvoie bien non.

   De plus $p = \mathbb{P}(y_1 \neq y_2) < 1/2$ car c'est la proba qu'un des deux algos se trompe. Donc en espérance le nombre de boucles faites est $\sum_{k \geq 1} k\mathbb{P}(\text{on fait } k \text{ boucles}) = \sum_{k \geq 1} kp^{k-1}(1 - p) = 1/(1 - p) \leq 2$. Donc la complexité espérée de l'algo est $2(p_1(|x|) + p_2(|x|))$ qui est bien polynomiale en $|x|$.
3. Si X est une variable aléatoire positive et $a > 0$, l'inégalité de Markov dit que $\mathbb{P}(X \geq a) \leq \mathbb{E}[X]/a$.

   La réciproque est vraie. Si A est un algo ZPP pour Π d'espérance de temps de calcul f on considère l'algo B : faire $2f(|x|)$ étapes de calcul de $A(x)$, si le calcul est fini renvoyer le résultat trouvé par A, sinon renvoyer faux. Cet algo est bien polynomial en $|x|$.

Si $x \leq 0$, B renvoie faux donc pas de faux positif. De plus la proba de faux négatif vaut :

$$\mathbb{P}(\text{temps exécution de } A \text{ sur } x \geq 2f(|x|)+1) \leq \frac{\mathbb{E}[\text{temps exécution de } A \text{ sur } x]}{2f(|x|)+1} = \frac{f(|x|)}{2f(|x|)+1} < \frac{1}{2}$$

la première égalité venant de Markov. Donc Π est dans RP. Il est dans coRP avec le même algo mais qui renvoie vrai par défaut si le calcul de A n'a pas terminé sur l'entrée en $2f(|x|)$ étapes.

Exercice 165 (**) *Knuth shuffle*

On suppose qu'un tirage uniforme d'un élément dans un ensemble fini se fait en temps constant. On souhaite concevoir un algorithme permettant de générer une permutation aléatoire d'un tableau uniformément choisie parmi toutes les permutations possibles de ce tableau. Pour ce faire, on propose :

Entrée : Un tableau $T = [t_0, ..., t_{n-1}]$
Sortie : Une permutation de T
pour i allant de 0 à $n-1$
  Choisir k uniformément aléatoirement dans $[\![0, i]\!]$
  Échanger les contenus des cases k et i dans le tableau T
renvoyer T

1. Déterminer la complexité de cet algorithme.
2. Écrire en C une fonction de prototype `knuth_shuffle(int* t, int n)` implémentant cet algorithme.

On note P_i la propriété suivante : si on fixe une permutation σ de $[\![0, i]\!]$, alors les $(i+1)$-èmes premières cases du tableau T sont $t_{\sigma(0)}, t_{\sigma(1)}, ..., t_{\sigma(i)}$ avec probabilité $1/(i+1)!$.

3. Montrer par récurrence que cette propriété est vérifiée au fil des itérations de l'algorithme précédent. Pour l'hérédité, on peut distinguer les cas selon que $\sigma(i+1) = i+1$ ou non. Conclure.

1. Complexité linéaire.
2. Juste faire attention à ne pas écraser de valeur au moment de l'échange.
3. Pour $i = 0$ la seule permutation de $[\![0, i]\!]$ est l'identité et la première case du tableau est bien $t_0 = t_{\sigma(0)}$ avec probabilité 1. Si le résultat tient pour i, soit σ une permutation de $[\![0, i+1]\!]$:
   - Si $\sigma(i+1) = i+1$ alors c'est qu'on a tiré $k = i+1$ ce qui arrive avec proba $1/(i+2)$. Alors la proba cherchée est $1/(i+2)$ fois la proba que la permutation choisie à l'itération précédente soit σ restreinte à $[\![0, i]\!]$, qui vaut $1/(i+1)!$ par hypothèse.
   - Sinon, $\sigma(i+1) = j < i+1$ et donc il existe $k \in [\![0, i]\!]$ tel que $\sigma(k) = i+1$. On note $\widetilde{\sigma}$ la permutation de $[\![0, i]\!]$ qui coïncide avec σ sauf en k où $\widetilde{\sigma}(k) = j$. La proba que les premières cases soient $x_{\sigma(0)}, ..., x_{\sigma(i+1)}$ après l'échange des cases k et $i+1$ vaut la proba d'avoir choisi $\widetilde{\sigma}$ à l'itération précédente (donc $1/(i+1)!$) fois la proba que $\sigma(i+1) = j$ qui vaut $1/(i+2)$.

   On en déduit que la proba que le tableau final représente une permutation donnée est de $1/n!$ donc on a bien une distribution uniforme des permutations obtenues avec cet algo.

Exercice 166 (exo cours) *Complexité du tri rapide randomisé*

On considère l'algorithme du tri rapide randomisé en se restreignant à ne trier que des ensembles dont les éléments sont deux à distincts. Si S est un ensemble de taille n en entrée de cet algorithme, on note $s_1 < s_2 < ... < s_n$ ses éléments et pour tout $1 \leq i < j \leq n$, on note X_{ij} le nombre de fois que les valeurs s_i et s_j sont comparées au cours d'une exécution de l'algorithme.

1. Justifier que pour tout $1 \leq i < j \leq n$, $\mathbb{P}(X_{ij} = 1) = \frac{2}{j-i+1}$.
2. En déduire la complexité espérée du tri rapide randomisé sur une entrée de taille n.

1. On note $E_{ij} = [\![s_i, s_j]\!]$. Si, au cours des tirages, le premier pivot tiré dans E_{ij} est s_i ou s_j alors $X_{ij} = 1$, sinon, le pivot tiré séparera s_i et s_j donc $X_{ij} = 0$. $p_k = \mathbb{P}(X_{ij} = 1 \,|\, \text{le premier pivot tiré dans } E_{ij} \text{ l'est au } k$

ème tirage) est indépendant de k et vaut $2/(j-i+1)$ car c'est la proba de tirer s_i ou s_j dans E_{ij}. Donc :

$$P(X_{ij}=1)=\sum_{k=1}^{\infty} p_k \mathbb{P}(\text{le premier pivot tiré dans } E_{ij} \text{ l'est au } k\text{-ème tirage}) = \frac{2}{j-i+1}\times 1$$

car la proba qu'un pivot soit choisi dans E_{ij} vaut 1 puisque cet ensemble contient deux éléments.

2. Cette complexité est proportionnelle à l'espérance du nombre X de comparaisons effectuées au total :

$$\mathbb{E}[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}[X_{ij}]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1}\leq\sum_{i=1}^{n}\sum_{k=2}^{n}\frac{2}{k}\leq 2n\log n$$

Exercice 167 (exo cours) *Conversion Monte Carlo - Las Vegas*

On suppose disposer d'un algorithme Monte Carlo A résolvant un problème P dont la probabilité d'échec est de p et d'un vérificateur V qui peut vérifier si une sortie de A est correcte ou non.

1. Montrer qu'on peut alors construire un algorithme Las Vegas B résolvant P.
2. Estimer le temps espéré de calcul de B sur une entrée de taille n en fonction des temps de calcul sur une telle entrée pour A et V ainsi que de p.

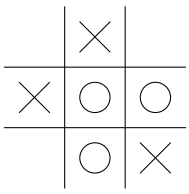
1. Il suffit, indéfiniment, de calculer une solution (potentielle) x de P avec A et de vérifier si x est bien solution avec V. Si oui, on la renvoit. Cet algo est correct mais son temps d'exécution varie.
2. On note $a(n)$ la complexité de A sur une entrée de taille n et $v(n)$ celle de V. Une itération de la boucle implicite ci-dessus s'exécute en temps $a(n)+v(n)$. Il ne reste qu'à déterminer le nombre espéré E de tours de boucles. Or $\mathbb{P}$(on fait k tours de boucle) $=p^{k-1}(1-p)$ car cela revient à échouer $k-1$ fois d'affilée à obtenir un résultat correct avec A et réussir la k-ème fois. Le temps de calcul espéré est :

$$E\times(a(n)+v(n))=(a(n)+v(n))\times\sum_{k=1}^{\infty}kp^{k-1}(1-p)=\frac{a(n)+v(n)}{1-p}$$

9 Algorithmique des jeux

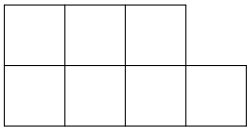
Exercice 168 (*) *Positions gagnantes*

1. En considérant que c'est au joueur X de jouer, déterminer si la configuration suivante au jeu du morpion est une position gagnante pour X, gagnante pour O ou ni l'un ni l'autre.



Reprendre la question si cette configuration appartient au joueur O.

2. On considère le jeu du domineering présenté en cours. En dessinant le graphe des configurations, déterminer pour quel joueur le plateau suivant est gagnant en fonction du joueur qui commence.

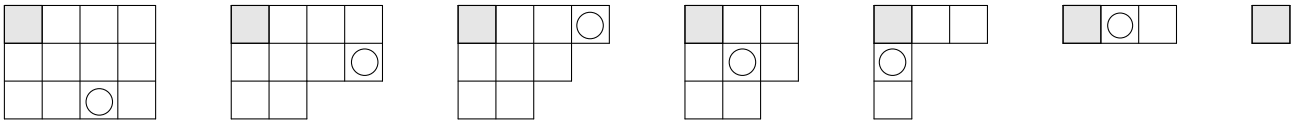


1. La position est gagnante pour X si c'est à lui de jouer (on exhibe une stratégie gagnante pour le montrer). Dessiner le graphe des configurations à partir de la configuration donnée montre en revanche qu'elle n'est pas gagnante pour O.

2. Le plateau est gagnant pour le joueur vertical via la stratégie consistant à jouer un coup au milieu au début. Le plateau est gagnant pour le joueur horizontal via la stratégie consistant à jouer le coup en bas à droite du rectangle 2×3 inclus dans le plateau au début. Ce plateau est donc toujours gagnant pour le joueur qui commence.

Exercice 169 (**) *Stratégie pour le jeu de Chomp*

Le jeu de Chomp a comme support une tablette de chocolat rectangulaire dont le coin supérieur gauche est empoisonné. Chaque joueur choisit tour à tour un carré et le mange, ainsi que tous les carrés situés en dessous et à la droite de celui choisi. Le joueur qui n'a plus d'autre choix que de manger le carré empoisonné a perdu. Voici un exemple de partie dans laquelle le joueur qui a commencé perd (le carré gris représente le carré empoisonné et les cercles les coups des deux joueurs) :



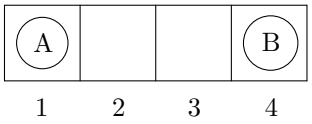
1. En dessinant le graphe des configuration de ce jeu, déterminer quel joueur possède une stratégie gagnante sur une tablette de longueur 3 et de hauteur 2.
2. Le jeu de Chomp est paramétré par la taille (p, q) de la tablette de chocolat. Montrer que si $(p, q) \neq (1, 1)$ alors il existe une stratégie gagnante pour l'un des deux joueurs, qu'on identifiera.

Remarque : La preuve de l'existence d'une stratégie gagnante pour l'un des deux joueurs à la question 2 se fait de manière non constructive. Ce type de preuve est appelé "preuve par vol de stratégie".

1. Le joueur qui commence dispose d'une stratégie gagnante : il suffit pour lui de manger uniquement le carré le plus en bas à droite de la tablette pour gagner quelle que soit la stratégie de l'adversaire.
2. Si $p = 1$ ou $q = 1$, le premier joueur gagne en un coup en mangeant tous les carrés sauf l'empoisonné. Si $p, q > 1$, comme aucune partie nulle n'est possible, il suffit de montrer qu'il existe un premier coup gagnant pour le premier joueur :
   - Si manger le carré en bas à droite est un coup gagnant, c'est le cas.
   - Sinon, le deuxième joueur a un premier coup gagnant et ce dernier aurait pu être choisi par le premier joueur : ce dernier dispose donc encore d'un premier coup gagnant.

Exercice 170 (**) *Min-max avec cycles*

On considère le jeu dont la configuration initiale est présentée ci-dessous. Les joueurs A et B bougent alternativement leur pion dans un couloir de 4 cases. Ils ne peuvent déplacer leur pion que sur une case adjacente vide dans n'importe quelle direction. Si l'adversaire occupe une case adjacente, le joueur peut sauter ce pion et avancer jusqu'à la prochaine case libre, à condition qu'elle existe. Par exemple, si A est en 2 et B en 3, A peut immédiatemment aller en 4. Le joueur A gagne s'il atteint la case 4 avant que le joueur B n'atteigne la case 1, la chose est symétrique du point de vue de B. Le joueur A commence.



1. Dessiner le graphe des configurations de ce jeu. Que remarque-t-on ?
2. Dessiner le pseudo-arbre des configurations de ce jeu : il s'agit de son arbre dans lequel on ne dessine pas les configurations filles d'une configuration qu'on rencontre pour la deuxième fois.
3. On considère que la valeur min-max d'une feuille de cet arbre vaut 1 si la configuration est gagnante pour A, -1 si la configuration est perdante pour A et ? dans les autres cas.
   a) Calculer la valeur min-max de la racine en expliquant comment gérer les valeurs ?.
   b) La configuration initiale du jeu est-elle une position gagnante pour A ?

4. La version modifiée de l'algorithme min-max utilisée à la question précédente permet-elle d'obtenir une stratégie optimale pour tous les jeux où interviennent des cycles ?

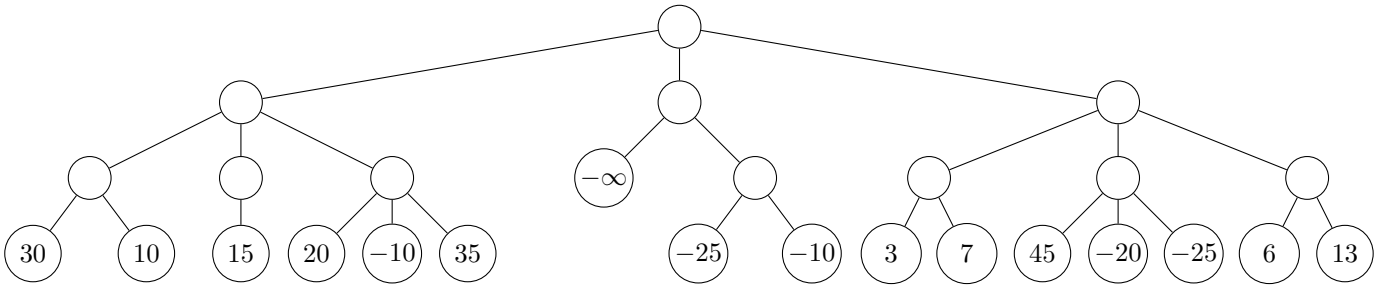
5. Ce jeu à 4 cases peut être généralisé à n cases pour tout $n > 2$. Montrer que pour tout $n > 2$, la configuration initiale est une position gagnante pour l'un des deux joueurs.

1. Il contient des cycles.
2. RAS. Il y a deux configurations qui apparaissent deux fois dans cet arbre : (A en case 1 et B en case 4) et (A en case 2 et B en case 4).
3. a) On applique minmax en considérant que $\max(1, ?) = 1$, $\min(-1, ?) = -1$.

   b) La configuration initiale est gagnante pour A (ce qu'on peut vérifier).
4. Cette version de l'algorithme minmax permet de conclure dans certains cas, mais la comparaison entre ? et ? ou entre ? et une partie nulle n'est pas évidente.
5. On montre par récurrence sur le nombre n de cases que la configuration initiale est gagnante pour A si n est pair et gagnante pour B s'il est impair.

Exercice 171 (*) *Calcul de score avec et sans élagage*

On considère un sous-graphe des configurations d'un jeu donné par l'arbre suivant. La valeur de l'heuristique pour chaque feuille y est indiquée : elle est positive pour le joueur 1 et négative pour le joueur 2.



1. Calculer les scores de chacun des noeuds selon l'algorithme min-max :
   a) En supposant que c'est au joueur 1 de jouer à la racine.
   b) En supposant que c'est au joueur 2 de jouer à la racine.
2. Représenter l'arbre obtenu et les scores des noeuds visités (en supposant qu'il le sont de gauche à droite) lors de l'application de l'algorithme min-max avec élagage alpha-beta sur le graphe des configurations de la première question en supposant que c'est au joueur 1 de jouer à la racine.

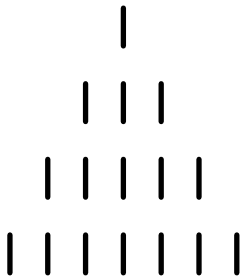
1. RAS. On trouve un score de 15 à la racine dans le premier cas et de -25 dans le second.
2. Les branches contenant de gauche à droite -10 ; 35 ; -25 et -10 ; -20 ; -25 sont élaguées.

Exercice 172 (***) *Théorème de Sprague-Grundy*

Le jeu de Nim est un jeu à deux joueurs dans lequel la configuration initiale consiste en un certain nombre de rangées d'allumettes. Tour à tour, chaque joueur doit choisir une des rangées et en retirer au moins une allumette. Le gagnant est celui qui retire la dernière allumette, autrement dit, celui qui ne peut plus jouer perd. Par convention, le joueur 1 est celui qui commence.

1. Dans sa version la plus simple, il n'y a qu'une seule rangée de n allumettes. Montrer qu'il existe toujours un joueur qui possède une stratégie gagnante pour le jeu de Nim dans ces conditions.

On considère à présent un jeu de Nim à plusieurs rangées. Par exemple, la figure ci-dessous représente le jeu de Nim $(1, 3, 5, 7)$.



2. Si $n \in \mathbb{N}^\star$, montrer qu'il existe une stratégie gagnante pour le joueur 2 pour le jeu de Nim (n, n).
3. Soit $1 \leq m < n$ deux entiers. Montrer qu'il existe une stratégie gagnante pour le premier joueur pour le jeu de Nim (m, n).

Les deux questions précédentes sont des cas particuliers du théorème de Sprague-Grundy qu'on se propose de démontrer dans la suite de l'exercice : il existe une stratégie gagnante pour le premier joueur au jeu de Nim $(x_1, ..., x_n)$ si et seulement si $x_1 \oplus ... \oplus x_n \neq 0$ où $\oplus$ désigne le ou exclusif bit à bit.

On remarque que jouer un coup transforme une instance du jeu de Nim en une nouvelle instance du jeu de Nim. Notons $(x_1, ..., x_n)$ l'état du jeu avant le coup du premier joueur (l'entier x_i est donc le nombre d'allumettes sur la rangée i avant le coup du joueur 1) et $(y_1, ..., y_n)$ l'état du jeu après le coup du premier joueur et définissons de surcroît :

$$x = \bigoplus_{i=1}^{n} x_i \text{ et } y = \bigoplus_{i=1}^{n} y_i$$

4. a) Montrer que si le joueur 1 a enlevé des allumettes dans la rangée k alors $y = x \oplus x_k \oplus y_k$.
   b) Montrer que si $x = 0$ alors $y \neq 0$ quel que soit le coup choisi par le premier joueur.
   c) Montrer que si $x \neq 0$ alors il existe un coup tel que $y = 0$.
   d) En déduire le théorème de Sprague-Grundy.
5. Pour chacune des instances du jeu de Nim suivantes, déterminer si la configuration initiale est gagnante pour le premier joueur et décrire un premier coup gagnant si elle l'est.
   a) $(1, 3, 5, 7)$.
   b) $(1, 3, 5, 7, 9)$.
   c) $(1, 2, 4, ..., 2^n)$ pour $n \geq 1$.
   d) $(1, 3, 7, ..., 2^n - 1)$ pour $n \geq 2$.

TODO

Exercice 173 (exo cours) *Ensembles attracteurs*

On considère une arène G dont les états sont partitionnés en un ensemble S_0 et un ensemble S_1. On cherche à déterminer l'ensemble des positions gagnantes pour le joueur 0 (qui joue dans les états de S_0).

1. Définir une suite récurrente d'ensembles qui converge (on le justifiera) vers une solution au problème.
2. Quelle est la complexité du calcul de l'attracteur du joueur 0 via cette méthode ? On précisera les structures de données utilisées pour aboutir à la complexité avancée.

1. A_0 = ensemble des états finaux gagnants pour 0 et $A_{n+1} = \{s \in S_0 \,|\, \exists t \in A_n, (s,t) \text{ est un coup}\} \cup \{s \in S_1 \setminus \{\text{terminaux}\} \,|\, \forall t \text{ tel que } (s,t) \text{ est un coup } t \in A_n\} \cup A_n$ convient.
2. On représente un ensemble d'états par un tableau de booléens de taille $|S|$ contenant vrai en case i si i est dans l'ensemble et le graphe des configurations par matrice d'adjacence. Alors il suffit de calculer les A_n jusque stabilisation. Vérifier l'atteinte de la stabilisation se fait en $O(|S|)$ et elle arrive en moins de S itérations, chacune pouvant se faire en $O(|A|)$ (pour chaque sommet, on parcourt ses voisins pour savoir s'ils sont dans A_n). Donc $O(|S|(|S| + |A|))$. On peut faire mieux.

Exercice 174 (exo cours) *Algorithme mystère*

On considère l'algorithme suivant :

Entrée : Un graphe $G = (S, A)$ où $S = S_1 \sqcup S_2$; $T \subset S$ un ensemble de sommets de degré sortant nul.
Sortie : ?
$\mathcal{A} \leftarrow \emptyset$
pour tout sommet s
| Calculer le degré sortant n_s de s dans G
Calculer G^t, le graphe transposé de G
`/* Définition d'une fonction auxiliaire interne de parcours */`
Parcours(u) =
si $u \notin \mathcal{A}$ **alors**
| $\mathcal{A} \leftarrow \mathcal{A} \cup \{u\}$
pour tout voisin v de u dans G^t
| $n_v \leftarrow n_v - 1$
| **si** $v \in S_1$ ou $n_v = 0$ **alors**
| | Parcours(v)
pour sommet $u \in T$
| Parcours(u)

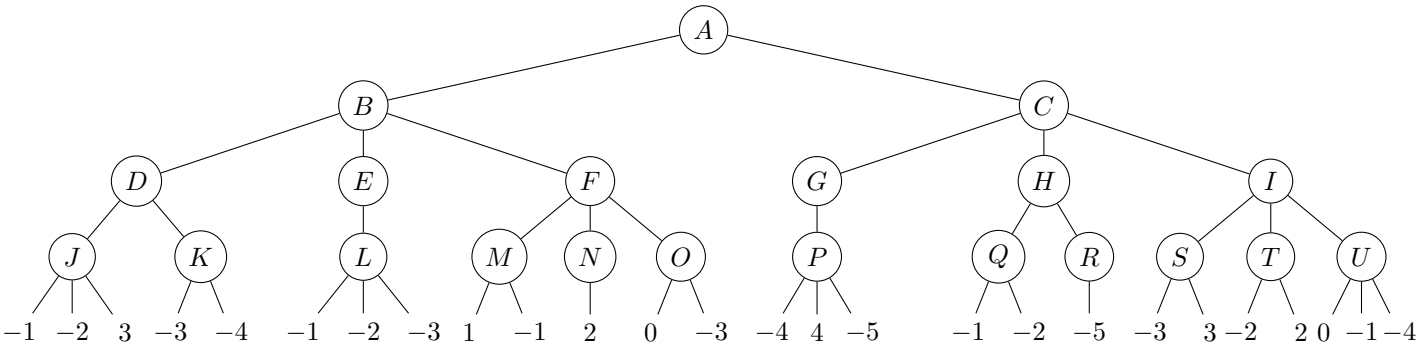
1. Si T est l'ensemble des états finaux gagnants pour le joueur 1 dans un jeu décrit par l'arène G, que calcule cet algorithme ?
2. Estimer la complexité de cet algorthme.

1. Cet algo calcule l'ensemble des positions gagnantes pour le joueur 1.
2. Calcul des degrés = $O(|A|)$. Calcul de G^t et du parcours = $O(|S| + |A|)$.

Exercice 175 (*) *Min-max avec élagage*

Déterminer le score de la racine étant données les valeurs de l'heuristique aux feuilles de l'arbre suivant :

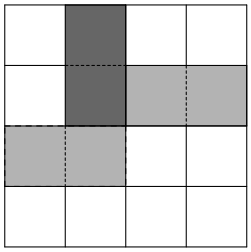
1. En supposant que la racine est un état du joueur qui veut maximiser le score de la racine.
2. En supposant que la racine est un état du joueur qui veut le minimiser.
3. Identifier les branches qui n'auraient pas été explorées avec un élagage alpha-beta dans le premier cas.



1. Le score de la racine vaut -3.
2. Le score de la racine vaut 0.
3. Si on parcourt les fils de l'arbre de gauche à droite, on coupe : le fils droit de K ; les fils N et O de F ; les deux fils à droite de P et les fils H et I de C.

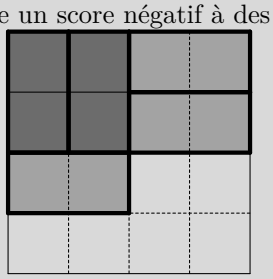
Exercice 176 (**) *Min-max avec heuristique*

On considère la configuration suivante au jeu du domineering : elle correspond à une position où c'est au joueur qui place les dominos verticaux de jouer (par convention ce joueur est le joueur 1, l'autre le 2).



1. Dessiner le sous-graphe des configurations à profondeur 2 depuis cette configuration.
2. Déterminer le score de chaque sommet dans ce graphe via l'algorithme min-max en utilisant l'heuristique suivante : le score d'une configuration vaut $+\infty$ si elle est finale gagnante pour 1, $-\infty$ si elle est finale gagnante pour 2 et vaut le nombre de coups possibles pour le joueur 1 moins celui pour le joueur 2 sinon.
3. En déduire le coup du joueur 1 dans cette position selon l'algorithme min-max avec une profondeur d'exploration de deux. Aurait-on joué le même coup avec une profondeur d'exploration de un ?
4. Pour chaque sommet du sous-graphe précédent, déterminer en justifiant s'il s'agit d'une position gagnante pour le joueur 1 ou pour le joueur 2. Que dire de la qualité de l'heurisique ?

1. On a 3 sommets à l'étage 1 et 10 à l'étage 2.
2. RAS.
3. L'heuristique indique de jouer en troisième colonne aux deux profondeurs.
4. On se contente d'explications orales. L'heuristique est acceptable au sens où elle fait jouer un coup gagnant mais est défaitiste : elle attribue un score négatif à des positions gagnantes pour 1 comme :



Exercice 177 (*) *Jeu des bâtonnets*

On considère un jeu dont le support est une rangée de n bâtonnets. Tour à tour, chaque joueur enlève de la rangée 1, 2 ou 3 bâtonnets. Le perdant est celui qui prend le dernier bâtonnet.

1. Dessiner le graphe des configurations de ce jeu lorsque $n = 7$.
2. Donner une stratégie gagnante pour le premier joueur dans ce cas en justifiant.
3. Montrer que les configurations où le nombre n de bâtonnets est congru à 1 modulo 4 sont perdantes. Si $n = 20$, vaut-il mieux jouer en premier ou en deuxième ? Et si $n = 21$?

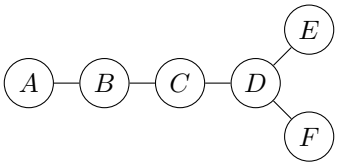
1. RAS.
2. Joueur 1 prend deux batônnets puis :
   - Si joueur 2 en prend 1, il en reste 4 et joueur 1 en prend 3 et gagne.
   - Si joueur 2 en prend 2, il en reste 3 et joueur 1 en prend 2 et gagne.
   - Si joueur 2 en prend 3, il en reste 2 et joueur 1 en prend 1 et gagne.
3. On montre par récurrence sur $k \geq 0$ qu'une config à $4k + 1$ bâtonnets est perdante. Si $n = 20$, il faut commencer et prendre 3 bâtonnets. Si $n = 21$, il vaut mieux joueur en deuxième.

Exercice 178 (**) *Poursuite-évasion*

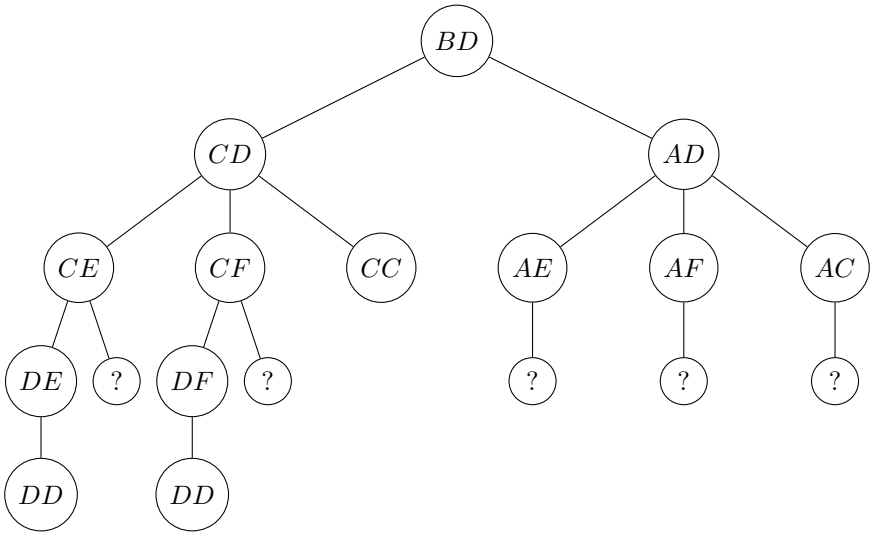
On considère un jeu de poursuite-évasion : deux joueurs, le fugitif et le poursuivant, se déplacent sur un graphe. Tour à tour, ils doivent se déplacer sur un sommet adjacent à celui qu'ils occupent actuellement. Le jeu se

termine dès que les deux joueurs se trouvent sur le même sommet. Le score du poursuivant est l'opposé du nombre de coups joués avant qu'il ait rejoint le fugitif, le fugitif gagne s'il ne se fait jamais attraper.

On considère une instance de ce jeu dans lequel le graphe est le suivant. Initialement, le poursuivant est en B et le fugitif en D. Le poursuivant commence.



On considère un morceau de l'arbre des configurations dans ce cas :



1. Recopier l'arbre et déterminer les scores du poursuivant dans les états terminaux.
2. Indiquer quelle configuration est atteinte à partir des ? dans l'arbre. Expliquer comment trouver des bornes pour la valeur de ces états en se référant au graphe sur lequel se déplacent les joueurs.
3. Déterminer la condition la plus forte possible sur le score de chaque noeud interne de l'arbre précédent (un nombre idéalement, une ou plusieurs inégalités sinon).
4. En supposant que le calcul du score de la racine se fait en évaluant de gauche à droite, indiquer quelles branches il est inutile d'explorer.
5. Prouver que le poursuivant gagne si le graphe de déplacement est un arbre.

TODO

Exercice 179 (*) *Heuristique pour le morpion*

On considère le jeu du morpion sur une grille 3×3. X commence. On note $X_n(s)$ le nombre de colonnes, lignes, diagonales comptant exactement n "X" et aucun "O" dans la configuration s et on définit O_n similairement. Le score d'une position terminale vaut 1 si $X_3 = 1$, -1 si $O_3 = 1$ et 0 sinon. Pour les états s non terminaux, le score est linéairement déterminé par $\text{Score}(s) = 3X_2(s) + X_1(s) - 3O_2(s) - O_1(s)$.

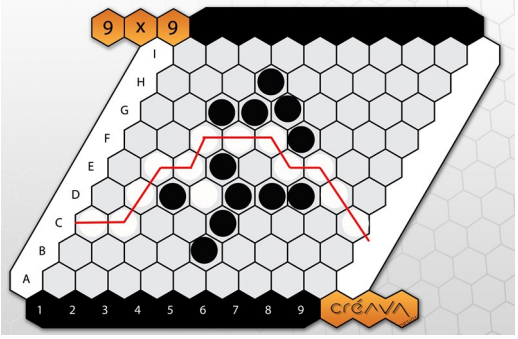
1. Dessiner tout l'arbre de jeu à partir d'une grille vide à profondeur 2 en tenant compte des symétries.
2. Déterminer via l'algorithme min-max le score de la racine de l'arbre précédent et en déduire le coup que jouera le premier joueur en suivant l'heuristique de l'énoncé.
3. Quels sont les scores qui ne seront pas évalués si on applique un élagage alpha-beta à la question précédente en supposant que les appels se font dans l'ordre qui permet d'élaguer le plus ?

1. Si on tient compte des symétries, la grille vide à 3 filles (croix au centre, croix dans un coin, croix sur un bord). Même principe pour le second coup : au deuxième étage on trouve 12 configurations.
2. On devrait trouver le coup qui consiste à joueur au centre (score de la racine = 1).

3. Pour élaguer le plus on commence par parcourir la branche où la première croix est mise au centre (ce qui assure que le score est ≥ 1) puis dans chacune des autres on commence par parcourir un morceau qui aboutit à une feuille dont l'heuristique est négative.

Exercice 180 (**) *Jeu de Hex*

Le jeu de Hex se joue sur un plateau de taille $n \times n$ ($n \geq 2$) à cases hexagonales. Tour à tour, chaque joueur pose un pion de sa couleur (noir ou blanc) sur une des cases libres. Le premier joueur qui arrive à faire un chemin reliant ses deux bords opposés (gauche-droite pour blanc et haut-bas pour noir) gagne. Le joueur noir commence toujours. Le plateau ci-dessous illustre une partie gagnée par le joueur blanc.



1. Montrer qu'il n'est pas possible que le plateau soit rempli avec les deux joueurs gagnants.
2. Montrer qu'il n'est pas possible que le plateau soit rempli sans qu'aucun joueur n'ait gagné.
3. En déduire qu'il existe une stratégie gagnante pour le joueur noir.

1. Si noir a gagné, le plateau est coupé en deux composantes connexes donc blanc n'a pas gagné.
2. On observe la composante connexe C noire adjacente au bas. Si elle contient une tuile qui touche le haut, noir a gagné. Sinon, il y a un chemin blanc qui délimite C et blanc gagne.
3. Comme il n'y a pas de partie nulle (q2) et que le jeu est acyclique, un des joueurs a une stratégie gagnante. Supposons que blanc a une stratégie gagnante. Alors noir aussi : il joue son premier coup arbitrairement en c (un pion n'est jamais désavantageux) puis à chaque coup de la stratégie gagnante de blanc, on intervertit les bords noirs et blancs pour que noir joue le même coup que blanc (en jouant arbitrairement si la stratégie gagnante de blanc demande de jouer en c). Alors noir et blanc gagnent ce qui contredit la question 1.

Exercice 181 (**) *Existence de stratégie gagnante*

Si $G = (S, A)$ est un graphe orienté et acyclique, on dit que $S' \subset S$ est stable si tout sommet de S' n'a aucun successeur dans S' et qu'il est absorbant si tout sommet n'appartenant pas à S' admet un successeur dans S'. On dit que S' est un noyau s'il est à la fois stable et absorbant.

1. Montrer que dans un graphe acyclique il existe un sommet de degré sortant nul.
2. Montrer qu'un graphe orienté et acyclique possède un unique noyau.
3. Donner en pseudo-code un algorithme permettant de calculer ledit noyau dans ces conditions.

On considère le jeu de Chomp. Son support est une tablette de chocolat rectangulaire dont le coin supérieur gauche est empoisonné. Tour à tour, chaque joueur choisit un carré et le mange ainsi que tous les carrés en dessous et à la droite de celui choisi. Celui qui ne peut que choisir le carré empoisonné perd.

4. Calculer le noyau du graphe des configurations associé au jeu de Chomp sur une tablette de longueur 3 et de largeur 2. A quoi correspond ce noyau ?
5. Montrer que dans un jeu dont le graphe des configurations est acyclique et pour lequel toutes les configurations terminales sont perdantes, l'un des deux joueurs a nécessairement une stratégie gagnante.

1. Sinon tout sommet a degré sortant au moins 1 et on peut faire un chemin $(s_0, ...s_{|S|})$ qui contient un cycle car il y a au moins une répétition dans ces $|S| + 1$ sommets.
2. Par récurrence sur $n = |S|$. Soit G un graphe acyclique à $n+1$ sommets et S' un de ses noyaux. Par la

question 1 il admet un sommet sans successeur qui est nécessairement dans S' de G par absorbance. On retire ce sommet et ses prédecesseurs et on a un graphe acyclique qui possède un unique noyau N et $N \cup \{s\}$ est alors l'unique noyau de G.

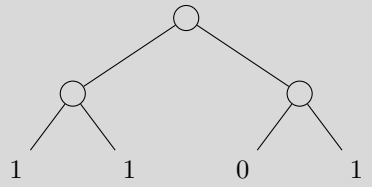
3. Tant qu'il y a des sommets dans G, trouver un sommet de degré sortant nul, l'ajouter au noyau, supprimer s et ses prédecesseurs. On peut demander la complexité en prime.
4. Le noyau est l'ensemble de positions perdantes.
5. Garanti par l'existence d'un noyau.

Exercice 182 (***) *Calcul Las Vegas de positions gagnantes*

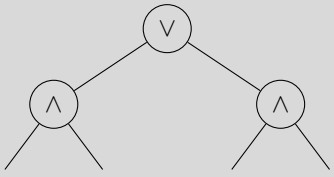
On considère un jeu à deux joueurs acyclique et sans match nul. On suppose que l'arbre de jeu associé est binaire et on connaît le score de chacune de ses feuilles : 1 si la position est gagnante pour le joueur 1 et 0 sinon. On cherche à déterminer le joueur pour qui la racine de cet arbre est gagnante en sachant que la racine est une position du joueur 1 et qu'il reste k coups à jouer pour chacun des deux joueurs.

1. Montrer qu'effectivement l'un des joueurs a nécessairement une stratégie gagnante.
2. En fonction de k, quelle est la complexité temporelle d'un algorithme déterminant le joueur gagnant ?
3. Montrer à l'aide d'un arbre de partie de hauteur 2 bien choisi qu'il n'est pas toujours nécessaire de déterminer l'état de chaque noeud de l'arbre pour déterminer le joueur gagnant.
4. En déduire un algorithme Las Vegas simple permettant de déterminer le joueur gagnant.
5. Montrer par récurrence sur $k \in \mathbb{N}^\star$ que l'espérance du nombre de noeuds explorés par cet algorithme Las Vegas est inférieur à $2 \times 3^k - 1$.
6. En déduire la complexité en espérance de cet algorithme en fonction du nombre n de noeuds de l'arbre et la comparer à celle de la méthode déterministe proposée en question 2.

1. Toute partie termine, et sur la victoire d'un des joueurs. On montre alors par récurrence sur n que toutes les positions à profondeur $2k-n$ sont dans l'attracteur d'un des joueurs. C'est donc en particulier le cas de la racine ce qui montre que l'un des joueurs a une stratégie gagnante.
2. L'arbre est complet de hauteur $2k$ donc a de l'ordre de 4^k noeuds donc une complexité en $O(4^k)$ si on calcule dans quel attracteur se trouve chaque noeud en mémoïsant les résultats.
3. Par exemple dans l'arbre suivant, il suffit d'explorer les 4 premiers noeuds dans l'ordre préfixe pour en déduire que le joueur 1 a une stratégie gagnante (puisqu'il commence) :



4. Pour chaque noeud, on évalue récursivement l'un des fils au hasard. Si c'est au joueur 1 de jouer et que le premier fils est gagnant pour 1 alors son père aussi. Même raisonnement avec le joueur 2. Sinon, on évalue récursivement le second fils pour conclure.
5. Les valeurs aux feuilles étant 0 et 1, un noeud où on veut maximiser le score revient à calculer le $\vee$ du score de ses enfants et un noeud minimisant à en calculer le $\wedge$. Si $k = 1$, on considère donc un arbre de la forme suivante (la racine appartient au joueur 1 : c'est un noeud $\vee$) :



On note N la variable aléatoire comptant le nombre de noeuds explorés et on veut montrer que

$E[N] \leq 5$. Or $E[N] = E[N|\text{la racine renvoie 0}] \times \underbrace{P(\text{la racine renvoie 0})}_{=p} + E[N|\text{la racine renvoie 1}] \times P(\text{la racine renvoie 1})$. Si les deux espérances conditionnelles sont inférieures à 5 alors on aura bien $E[N] \leq 5p + 5(1-p) = 5$. C'est le cas car :

- Si la racine renvoie 0, il faut explorer ses deux fils pour conclure et ceux ci renvoient tous les deux 0. Or ce sont des fils $\wedge$ donc s'il renvoient 0 chacun à une feuille 0. La proba de choisir en premier un fils 0 d'un noeud $\wedge$ est de 1/2 donc pour chaque fils on explore 1 noeud avec proba 1/2 et 2 noeuds avec proba 1/2. Donc en moyenne on explore 3/2 noeuds pour chacun des deux fils + en toutes circonstance les deux noeuds $\wedge$ = 5 noeuds.
- Si la racine renvoie 1 alors l'un des fils renvoie 1. On a une chance sur deux de choisir d'évaluer ce fils en premier (auquel cas on n'évalue pas l'autre) donc avec proba 1/2 on explore ce fils et ses deux feuilles (obligé puisque si un noeud $\wedge$ renvoie 1 il faut avoir parcouru ses deux fils pour le savoir) soit 3 noeuds. Avec proba 1/2 on explore l'autre fils et si on est malchanceux ce fils est un fils qui renvoie 0 auquel cas on explore les deux fils de la racine soit 6 noeuds. Donc on explore en moyenne $3 \times 1/2 + 6 \times 1/2 = 9/2 \leq 5$ noeuds.

Si le résultat est acquis pour k, considérons un arbre complet de hauteur $2(k+1)$ avec alternance d'étages $\wedge$ et $\vee$ et dont l'étage à la racine est un étage $\vee$:

- Si la racine renvoie 0, il faut explorer les deux fils $\wedge$. Pour chacun des fils $\wedge$, il y a proba 1/2 d'explorer en premier un de ses fils 0. Donc pour chaque fils $\wedge$ on explore en moyenne $2 \times 3^k - 1$ noeuds avec proba 1/2 et $2 \times (2 \times 3^k - 1)$ noeuds avec proba 1/2 par HR. Donc en moyenne on explore $(6 \times 3^k - 3)/2$ noeuds pour chaque fils $\wedge$ + 2 noeuds pour compter les deux fils $\wedge$ eux mêmes ce qui fait $6 \times 3^k - 1 = 2 \times 3^{k+1} - 1$ noeuds au total.
- Si la racine renvoie 1, un de ses fils $\wedge$ renvoie 1 et ce fils est choisi avec proba 1/2. Dans ce cas on visite ce noeud + $2 \times 3^k - 1$ noeuds en moyenne dans chacun des fils du fils $\wedge$ par HR. Sinon, avec proba 1/2 on visite les deux fils $\wedge$ et pour chaque fils de ces fils $2 \times 3^k - 1$ noeuds par HR. Donc en moyenne on visite : $(1/2) \times (1 + 2 \times (2 \times 3^k - 1)) + (1/2) \times (2 + 4 \times (2 \times 3^k - 1)) = 6 \times 3^k - 3/2 \leq 2 \times 3^{k+1} - 1$.
- La complexité espérée de l'algorithme Las Vegas est en $O(3^k)$ d'après la question précédente. Or l'arbre à n noeuds qu'on considère est complet de hauteur $2k$ donc k est de l'ordre de $\log_4(n)$ et donc $O(3^k) = O(n^{\log_4(3)}) \simeq O(n^{0.8})$: mieux que le $O(n)$ déterministe.

10 Logique

Exercice 183 (*) *Vocabulaire du premier ordre*

On affirme que les chaînes de caractères ci-dessous sont des formules du calcul des prédicats sur l'ensemble de variables $\{x, y, z\}$ et pour un langage du premier ordre dont la signature est à reconstruire :

$$F_1 = (\forall x\, \exists y\, f(x,y)) \Rightarrow (\exists x\, \forall y\, r(x,y,z))$$
$$F_2 = (\forall x\, p(x) \wedge \forall x\, f(x)) \Rightarrow \forall x\, (p(x) \wedge f(x))$$
$$F_3 = \forall x\, ((\exists x\, g(f(x), a) \vee h(x,x)) \wedge (\forall y\, \exists x\, q(x,y) \vee \exists z\, p(z,y)))$$

Pour chacune de ces trois formules :

- Indiquer quels symboles sont des symboles de fonction / de relation et dans tous les cas leur arité.
- Lister ses termes et ses formules atomiques.
- Dessiner l'arbre syntaxique correspondant à la formule.

Dans F_1, f est un symbole de relation d'arité 2 et r un symbole de relation d'arité 3. Dans F_2, p et f sont des symboles de relation d'arité 2. Dans F_3, q, p, g et h sont des symboles de relation d'arité 2, f un symbole de fonction d'arité 1 et a un symbole de fonction d'arité 0.

RAS sur la suite. Rappel : $\vee$ et $\wedge$ sont prioritaires sur les quantificateurs (mais pas $\Rightarrow$).

Exercice 184 (**) *Liberté et liaisons*

1. Les symboles x, y, z sont des variables. Dans les formules suivantes, pour chaque occurrence de variable, indiquer si c'est une occurrence libre ou liée. Si elle est liée, indiquer son point de liaison.
   a) $\forall x\, p(x) \Rightarrow \exists x\, r(x, y)$
   b) $\forall x\, \exists y\, \forall x (p(x, y) \Rightarrow \exists x\, p(x, y))$
   c) $(\forall x\, p(x) \wedge \forall x\, f(x)) \Rightarrow \forall x\, (p(x) \wedge f(x))$
   d) $\exists z\, p(f(x), z) \Leftrightarrow \forall x\, r(z, z)$
2. Parmi les formules précédentes, lesquelles sont closes ?
3. On se place à présent dans le contexte général. Si φ est une formule du premier ordre, on note $F(\varphi)$ l'ensemble de ses variables libres (free) et $B(\varphi)$ l'ensemble de ses variables liées (bound). Indiquer comment calculer ces ensembles de manière algorithmique étant donnée une formule φ en entrée.

1. a) Premier x lié par le $\forall$, deuxième x lié par le $\exists$, y libre.
   b) Premier x lié par le second $\forall$, second x lié par le $\exists$, les deux y lié par $\exists y$.
   c) Premier x lié par le premier $\forall$, deuxième x par le deuxième, troisième et quatrième par le dernier.
   d) Premier z lié par le $\exists$; toutes les autres occurrences sont libres.
2. Les formules b) et c) sont closes.
3. On procède inductivement :
   - Si φ est atomique $B(\varphi) = \emptyset$ et $F(\varphi) =$ toutes les variables apparaissant dans les termes.
   - Pour la négation : $B(\neg\varphi) = B(\varphi)$ et $F(\neg\varphi) = F(\varphi)$.
   - Pour la disjonction : $B(\varphi \vee \psi) = B(\varphi) \cup B(\psi)$ et $F(\varphi \vee \psi) = F(\varphi) \cup F(\psi)$. Idem pour $\wedge$ et $\Rightarrow$.
   - Pour le pour tout : $B(\forall x\varphi) = B(\varphi) \cup \{x\}$ et $F(\forall x\varphi) = F(\varphi) \setminus \{x\}$. Idem pour $\exists$. Quoique c'est en fait un peu plus compliqué que ça car pour être une variable liée (même raisonnement pour les variables libres), il faut une occurrence de la variable liée dans la formule donc en particulier qu'elle apparaisse. Il faut donc purger les ensembles obtenus des variables qui apparaissent uniquement accolés à un quantificateur mais jamais dans une feuille.

Exercice 185 (***) *Modèles algébriques*

On considère un langage L du premier ordre dont la signature est la suivante : l'ensemble des relations ne contient que l'égalité (d'arité 2) et l'ensemble des fonctions est constitué des symboles suivants :

- un symbole de fonction $\star$ d'arité 2, qu'on s'autorisera à utiliser de manière infixe (autrement dit, si t_1 et t_2 sont des termes, on écrira $t_1 \star t_2$ plutôt que $\star(t_1, t_2)$).
- un symbole de fonction $^{-1}$ d'arité 1, qu'on s'autorisera à utiliser de manière postfixe (autrement dit, si t est un terme, on écrira t^{-1} plutôt que $^{-1}(t)$).
- un symbole de fonction d'arité 0, noté e.

On se donne ensuite quatre formules du premier ordre sur le langage L :

- $A = \forall x\, \forall y\, \forall z\, ((x \star y) \star z = x \star (y \star z))$
- $N = \forall x\, (x \star e = x \wedge e \star x = x)$
- $I = \forall x\, (x \star x^{-1} = e \wedge x^{-1} \star x = e)$
- $G = A \wedge I \wedge N$

1. Donner un exemple de modèle pour la formule G.
2. De manière générale, comment appelle-t-on les structures S qui sont modèles de G ?
3. Expliquer le nom des quatre formules de l'énoncé.
4. Expliquer sans formalisme pourquoi $G \models \forall x\, ((x^{-1})^{-1} = x)$.
5. On considère la formule Ab suivante : $\forall x\, \forall y\, (x \star y = y \star x)$.
   a) A-t-on $G \models Ab$? Justifier sans formalisme.

b) A-t-on $G \models \neg Ab$? Justifier sans formalisme.

1. La structure de domaine $\mathbb{Z}$ dans laquelle e est interprété par 0, $\star$ par l'addition et $^{-1}$ par le passage à l'opposé est un modèle de G.
2. Les modèles de G sont les groupes.
3. A = associativité, N = existence d'un neutre, I = existence d'un inverse, G = axiomes des groupes.
4. Le passage à l'inverse est involutif dans tout groupe.
5. a) Non car il existe des groupes non commutatifs, $(S_3, \circ)$ par exemple.
   b) Non car il existe des groupes commutatifs, $(\mathbb{Z}, +)$ par exemple.

Exercice 186 (**) *Preuves pour le calcul propositionnel*

Montrer en déduction naturelle les séquents du calcul propositionnel suivants. Dans chaque cas, dire si la preuve proposée a été faite en logique minimale, en logique intuitionniste ou en logique classique.

1. $A \Rightarrow B, B \Rightarrow C \vdash A \Rightarrow C$.
2. $A \vdash \neg\neg A$.
3. $\vdash \neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$.
4. $\vdash \neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$. *Indication : le sens direct nécessite la logique classique.*
5. $\vdash (A \vee B) \wedge (A \vee C) \Rightarrow A \vee (B \wedge C)$. *Indication : exploiter la règle elim-$\vee$.*

1. Cette preuve peut se faire en logique minimale, en notant $\Gamma = \{A \Rightarrow B, B \Rightarrow C\}$:
$$\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash A}\,\text{ax} \quad \dfrac{}{\Gamma, A \vdash A \Rightarrow B}\,\text{ax}}{\Gamma, A \vdash B}\,\text{elim-}\Rightarrow \quad \dfrac{}{\Gamma, A \vdash B \Rightarrow C}\,\text{ax}}{\dfrac{\Gamma, A \vdash C}{\Gamma \vdash A \Rightarrow C}\,\text{intro-}\Rightarrow}\,\text{elim-}\Rightarrow$$
2. Cette preuve peut se faire en logique intuitionniste via :
$$\dfrac{\dfrac{\dfrac{}{A, \neg A \vdash \neg A}\,\text{ax} \quad \dfrac{}{A, \neg A \vdash A}\,\text{ax}}{A, \neg A \vdash \bot}\,\text{elim-}\neg}{A \vdash \neg\neg A}\,\text{intro-}\neg$$
3. Cette loi de De Morgan peut se prouver uniquement en logique intuitionniste. Pour le sens direct, on constate déjà qu'on peut dériver le séquent $\Gamma \vdash F \Rightarrow \bot$ du séquent $\Gamma \vdash \neg F$ (fait en cours) :
$$\dfrac{\dfrac{\dfrac{\text{supposé}}{\Gamma, F \vdash \neg F} \quad \dfrac{}{\Gamma, F \vdash F}\,\text{ax}}{\Gamma, F \vdash \bot}\,\text{elim-}\neg}{\Gamma \vdash F \Rightarrow \bot}\,\text{intro-}\Rightarrow$$
Ainsi, on montre que $\vdash \neg(A \vee B) \Rightarrow \neg A$:
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(A \vee B), A \vdash A}\,\text{ax}}{\neg(A \vee B), A \vdash A \vee B}\,\text{intro-}\vee \quad \dfrac{\dfrac{}{\neg(A \vee B), A \vdash \neg(A \vee B)}\,\text{ax}}{\neg(A \vee B), A \vdash (A \vee B) \Rightarrow \bot}\,\text{ci-dessus}}{\neg(A \vee B), A \vdash \bot}\,\text{elim-}\Rightarrow}{\neg(A \vee B) \vdash \neg A}\,\text{intro-}\neg}{\vdash \neg(A \vee B) \Rightarrow \neg A}\,\text{intro-}\Rightarrow$$
Un arbre de preuve similaire permet de montrer que $\vdash \neg(A \vee B) \Rightarrow \neg B$ et l'utilisation de la règle intro-$\wedge$ permet de conclure. Pour le sens réciproque, en notant $\Gamma = \{\neg A \wedge \neg B, A \vee B\}$:

$$\dfrac{\dfrac{}{\Gamma \vdash A \vee B}\,\text{ax} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash \neg A \wedge \neg B}\,\text{ax}}{\Gamma, A \vdash \neg A}\,\text{elim-}\wedge \qquad \dfrac{}{\Gamma, A \vdash A}\,\text{ax}}{\Gamma, A \vdash \bot}\,\text{elim-}\neg \qquad \dfrac{\text{pareil qu'avec } A}{\Gamma, B \vdash \bot}}{\dfrac{\dfrac{\Gamma \vdash \bot}{\neg A \wedge \neg B \vdash \neg(A \vee B)}\,\text{intro-}\neg}{\vdash \neg A \wedge \neg B \Rightarrow \neg(A \vee B)}\,\text{intro-}\Rightarrow}\,\text{elim-}\vee$$

4. Pour cette loi, le sens réciproque peut se faire en logique intuitionniste, en notant $\Gamma = \{\neg A \vee \neg B, A \wedge B\}$:

$$\dfrac{\dfrac{}{\Gamma, \neg A \vee \neg B}\,\text{ax} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \neg A \vdash A \wedge B}\,\text{ax}}{\Gamma, \neg A \vdash A}\,\text{elim-}\wedge \qquad \dfrac{}{\Gamma, \neg A \vdash \neg A}\,\text{ax}}{\Gamma, \neg A \vdash \bot}\,\text{elim-}\neg \qquad \dfrac{\text{pareil qu'avec } A}{\Gamma, \neg B \vdash \bot}}{\dfrac{\dfrac{\Gamma \vdash \bot}{\neg A \vee \neg B \vdash \neg(A \wedge B)}\,\text{intro-}\neg}{\vdash \neg A \vee \neg B \Rightarrow \neg(A \wedge B)}\,\text{intro-}\Rightarrow}\,\text{elim-}\vee$$

Cette preuve est en fait très semblable à la preuve de la réciproque en question 3. Pour le sens direct, et en notant $\Gamma = \{\neg(A \wedge B), \neg(\neg A \vee \neg B)\}$:

$$\dfrac{\dfrac{}{\Gamma \vdash \neg(A \wedge B)}\,\text{ax} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \neg(\neg A \vee \neg B)}\,\text{ax}}{\Gamma \vdash \neg\neg A \wedge \neg\neg B}\,(\star)}{\Gamma \vdash \neg\neg A}\,\text{elim-}\wedge}{\Gamma \vdash A}\,\text{elim-}\neg\neg \qquad \dfrac{\text{pareil qu'avec } A}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\,\text{intro-}\wedge}{\dfrac{\dfrac{\Gamma \vdash \bot}{\neg(A \wedge B) \vdash \neg A \vee \neg B}\,\text{raa}}{\vdash \neg(A \wedge B) \Rightarrow \neg A \vee \neg B}\,\text{intro-}\Rightarrow}\,\text{elim-}\neg$$

La règle $(\star)$ n'en est pas une mais pour dériver le séquent souhaité, il suffit de recopier l'arbre utilisé à la question trois pour le sens direct en instanciant différemment. On peut probablement se restreindre à n'utiliser une règle de la logique classique qu'à un seul endroit.

5. Cette preuve peut se faire en logique minimale. On commence par noter $F = (A \vee B) \wedge (A \vee C)$ et on montre le séquent $F, B \vdash A \vee (B \wedge C)$:

$$\dfrac{\dfrac{\dfrac{}{F, B \vdash F}\,\text{ax}}{F, B \vdash A \vee C}\,\text{elim-}\wedge \qquad \dfrac{}{F, B, A \vdash A \vee (B \wedge C)}\,\text{ax}\,\text{intro-}\vee \qquad \dfrac{\dfrac{}{F, B, C \vdash B}\,\text{ax} \qquad \dfrac{}{F, B, C \vdash C}\,\text{ax}}{F, B, C \vdash B \wedge C}\,\text{intro-}\wedge}{F, B, C \vdash A \vee (B \wedge C)}\,\text{intro-}\vee}{F, B \vdash A \vee (B \wedge C)}\,\text{elim-}\vee$$

Alors en suivant le même principe :

$$\dfrac{\dfrac{\dfrac{}{F \vdash F}\,\text{ax}}{F \vdash A \vee B}\,\text{elim-}\wedge \qquad \dfrac{}{F, A \vdash A}\,\text{ax}\,\text{intro-}\vee \qquad \dfrac{\text{vu la preuve ci-dessus}}{F, B \vdash A \vee (B \wedge C)}}{F \vdash A \vee (B \wedge C)}\,\text{elim-}\vee}{\vdash (A \vee B) \wedge (A \vee C) \Rightarrow A \vee (B \wedge C)}\,\text{intro-}\Rightarrow$$

Exercice 187 (*) *Correction de la déduction naturelle*

Rappel : montrer la correction de la déduction naturelle revient à montrer la correction de ses règles.

1. Montrer la correction de la règle intro-$\wedge$.
2. Reprendre la question précédente pour intro-$\neg$ puis elim-$\neg$.

1. Les prémisses de cette règle sont $\Gamma \vdash A$ et $\Gamma \vdash B$ et sa conclusion $\Gamma \vdash A \wedge B$. Supposons que $\Gamma \models A$ et $\Gamma \models B$ et soit v une valuation qui satisfait Γ. Alors par hypothèse, elle satisfait A et elle satisfait B donc elle satisfait $A \wedge B$ et donc $\Gamma \models A \wedge B$.

2. Supposons que $\Gamma, A \models \bot$ et soit v une valuation qui satisfait Γ. Si v satisfaisait A, elle $\bot$ ce qui est impossible, et on en déduit que v satisfait $\neg A$. Ainsi on a bien $\Gamma \models \neg A$.

   Supposons que $\Gamma \models A$ et $\Gamma \models \neg A$ et supposons par l'absurde qu'il existe une valuation v qui satisfait Γ. alors cette valuation satisfait A et $\neg A$ ce qui est impossible. On en déduit qu'il n'y a aucune valuation qui satisfait Γ et donc $\Gamma \models \bot$ comme attendu.

Exercice 188 (**) *Règle d'affaiblissement*

Il existe deux variantes pour définir l'axiome en déduction naturelle :

- Soit on considère que l'axiome est $\dfrac{}{\Gamma, A \vdash A}\ \text{ax}$. Cela donne lieu à un système de déduction qu'on note S_1.
- Soit on considère que l'axiome est $\dfrac{}{A \vdash A}\ \text{ax}$ et alors on rajoute la règle d'affaiblissement : $\dfrac{\Gamma \vdash A}{\Gamma, B \vdash A}\ \text{aff}$ pour produire un système de déduction qu'on note S_2.

Ces systèmes S_1 et S_2 sont syntaxiquement différents puisqu'ils n'ont pas les mêmes règles mais ils sont équivalents au sens où ils permettent de prouver exactement les mêmes séquents. L'objectif de cet exercice est de montrer cette propriété, qui justifie d'appeler *déduction naturelle* chacun de ces systèmes.

1. On donne ci-dessous une preuve de $\vdash A \Rightarrow (B \Rightarrow A)$ dans S_2 :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\ \text{ax}}{A, B \vdash A}\ \text{aff}}{A \vdash B \Rightarrow A}\ \text{intro-}\Rightarrow}{\vdash A \Rightarrow (B \Rightarrow A)}\ \text{intro-}\Rightarrow$$

   Comment la modifier en une preuve de $\vdash A \Rightarrow (B \Rightarrow A)$ dans S_1 ?

2. Généraliser ce principe en montrant à l'aide d'une induction bien choisie que tout séquent prouvable dans le système S_2 est prouvable dans le système S_1.
3. Montrer réciproquement que tout séquent prouvable dans S_1 est prouvable dans S_2 et conclure.

1. Il suffit d'enlever la racine de l'arbre et d'appliquer directement l'axiome.
2. Il faudrait montrer par induction sur l'ensemble des arbres de preuve que : s'il existe un arbre de preuve pour $\Gamma \vdash A$ dans S_1, alors il existe un arbre de preuve de même hauteur pour $\Gamma, B \vdash A$ dans S_1.
3. Idem, encore une induction.

Exercice 189 (*) *Preuve ou pas preuve ?*

Déterminer parmi les arbres suivants lesquels sont des arbres de preuve (corrects). Dans le cas où la preuve est erronnée, indiquer quelles sont les règles mal utilisées ou inventées.

1.
$$\dfrac{\dfrac{\dfrac{}{A \vee B \vdash A \vee B}\ \text{ax}}{A \vee B \vdash A}\ \text{elim-}\vee \qquad \dfrac{\dfrac{}{A \vee B \vdash A \vee B}\ \text{ax}}{A \vee B \vdash B}\ \text{elim-}\vee}{A \vee B \vdash A \wedge B}\ \text{intro-}\wedge}{\vdash A \vee B \Rightarrow A \wedge B}\ \text{intro-}\Rightarrow$$

2. On note Δ l'ensemble de formules du calcul propositionnel $\{A \Rightarrow (B \Rightarrow C), A \wedge B\}$.

$$\dfrac{\dfrac{\dfrac{}{\Delta \vdash A \Rightarrow (B \Rightarrow C)}\ \text{ax} \qquad \dfrac{\dfrac{}{\Delta \vdash A \wedge B}\ \text{ax}}{\Delta \vdash A}\ \text{elim-}\wedge}{\Delta \vdash B \Rightarrow C}\ \text{elim-}\Rightarrow \qquad \dfrac{\dfrac{}{\Delta \vdash A \wedge B}\ \text{ax}}{\Delta \vdash B}\ \text{elim-}\wedge}{\Delta \vdash C}\ \text{elim-}\Rightarrow}{\vdash (A \Rightarrow (B \Rightarrow C)) \Rightarrow (A \wedge B \Rightarrow C)}\ \text{intro-}\Rightarrow\ \text{(deux fois)}$$

3. On note $F = \forall x\, A \vee \forall x\, B$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x\, A \vdash \forall x\, A}\,\text{ax}}{\forall x\, A \vdash A}\,\text{elim-}\forall}{\forall x\, A \vdash A \vee B}\,\text{intro-}\vee}{\forall x\, A \vdash \forall x\, (A \vee B)}\,\text{intro-}\forall}{F, \forall x\, A \vdash \forall x\, (A \vee B)}\,\text{aff} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x\, B \vdash \forall x\, B}\,\text{ax}}{\forall x\, B \vdash B}\,\text{elim-}\forall}{\forall x\, B \vdash A \vee B}\,\text{intro-}\vee}{\forall x\, B \vdash \forall x\, (A \vee B)}\,\text{intro-}\forall}{F, \forall x\, B \vdash \forall x\, (A \vee B)}\,\text{aff} \qquad \dfrac{}{F \vdash F}\,\text{ax}$$
$$\dfrac{\dfrac{F \vdash \forall x\, (A \vee B)}{\vdash \forall x\, A \vee \forall x\, B \Rightarrow \forall x\, (A \vee B)}\,\text{intro-}\Rightarrow}{}\,\text{elim-}\vee$$

4. On note F la formule $\exists x\, A \wedge \exists x\, B$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{F \vdash F}\,\text{ax}}{F \vdash \exists x\, A}\,\text{elim-}\wedge \qquad \dfrac{}{F, A \vdash A}\,\text{ax}}{F \vdash A}\,\text{elim-}\exists \qquad \dfrac{\dfrac{\dfrac{}{F \vdash \exists x\, A \wedge \exists x\, B}\,\text{ax}}{F \vdash \exists x\, B}\,\text{elim-}\wedge \qquad \dfrac{}{F, B \vdash B}\,\text{ax}}{F \vdash B}\,\text{elim-}\exists}{F \vdash A \wedge B}\,\text{intro-}\wedge$$
$$\dfrac{\dfrac{F \vdash \exists x\, (A \wedge B)}{\exists x\, A \wedge \exists x\, B \Rightarrow \exists x\, (A \wedge B)}\,\text{intro-}\exists}{}\,\text{intro-}\Rightarrow$$

1. Preuve fausse : règle elim-$\vee$ inventée.
2. Preuve correcte.
3. Preuve correcte.
4. Preuve fausse à cause d'une mauvaise utilisation de elim-$\exists$ car x ne devrait pas être libre ni dans A (branche de gauche) ni dans B (branche de droite) pour pouvoir l'appliquer.

Dans le cas d'arbres faux, on peut s'en rendre compte en observant si le séquent final est valide : si ce n'est pas le cas, la preuve est nécessairement fausse par correction de la déduction naturelle.

Exercice 190 (**) *Preuves pour le calcul des prédicats*

Montrer en déduction naturelle les théorèmes du premier ordre suivants. Dans les questions 1, 2, 4, les formules peuvent dépendre de x. Pour s'aider, on pourra écrire $A(x)$ à la place de A dans les preuves. Pour lesquels avez vous eu besoin d'une règle de la logique classique (c'est-à-dire du tiers exclus ou d'une règle équivalente, comme le raisonnement par l'absurde ou l'élimination d'une double négation) ?

1. $\forall x\, P \Rightarrow \exists x\, P$ où P est un prédicat unaire. On peut considèrer dans un premier temps qu'il existe un symbole de constante x_0 dans le langage du premier ordre considéré pour écrire cette formule puis montrer que même sans cette hypothèse, cette formule reste un théorème.
2. $\exists x\, (A \wedge B) \Rightarrow (\exists x\, A) \wedge (\exists x\, B)$. Peut-on prouver la réciproque ?
3. $\forall x\, \forall y\, \forall z\, ((x = y \wedge y = z) \Rightarrow (x = z))$. Quelle propriété exprime cette formule ?
4. $\neg(\exists x\, P) \Leftrightarrow \forall x\, (\neg P)$.
5. $(\forall x\, A) \vee B \Leftrightarrow \forall x\, (A \vee B)$ si x n'apparaît pas libre dans B.

   Indication : pour la réciproque, se servir du tiers exclus.

1. L'application du elim-$\forall$ est licite car x_0 est bien un terme :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\forall x P(x) \vdash \forall x P(x)}\,\text{ax}}{\forall x P \vdash P(x_0)}\,\text{elim-}\forall}{\forall x P(x) \vdash \exists x P(x)}\,\text{intro-}\exists}{\vdash \forall x P(x) \Rightarrow \exists x P(x)}\,\text{intro-}\Rightarrow$$

   Dans le cas où on ne dispose pas de x_0, il faut changer de stratégie :

$$\dfrac{\dfrac{\dfrac{}{\forall x P \vdash \forall x P}\,\text{ax}}{\forall x P \vdash P}\,\text{elim-}\forall \qquad \dfrac{\dfrac{}{\forall x P, P \vdash P}\,\text{ax}}{\forall x P \vdash \exists x P}\,\text{intro-}\exists}{\forall x P \vdash \exists x P}\,\text{elim-}\exists$$
$$\dfrac{\forall x P \vdash \exists x P}{\vdash \forall x P \Rightarrow \exists x P}\,\text{intro-}\Rightarrow$$

L'élimination du $\exists$ est licite car x n'est libre ni dans $\forall x P$ ni dans $\exists x P$. Intuitivement la formule est vraie même quand on n'a aucun terme par les propriétés de l'ensemble vide.

2. Il suffit de montrer que $\exists x(A \wedge B) \vdash \exists x A$, de faire de même pour $\exists x B$, d'appliquer intro-$\wedge$ puis intro-$\Rightarrow$. Le seul point délicat est la première étape, qu'on prouve en notant $F = \exists x(A \wedge B)$:

$$\dfrac{\dfrac{}{F \vdash \exists x(A \wedge B)}\ \text{ax} \qquad \dfrac{\dfrac{\dfrac{}{F, A \wedge B \vdash A \wedge B}\ \text{ax}}{F, A \wedge B \vdash A}\ \text{elim-}\wedge}{F, A \wedge B \vdash \exists x A}\ \text{intro-}\exists}{F \vdash \exists x A}\ \text{elim-}\exists$$

L'utilisation du elim-$\exists$ est licite car x n'est libre ni dans F ni dans $\exists x A$. Intuitivement on ne pourra pas prouver la réciproque car cette dernière est sémantiquement fausse (et donc non prouvable par correction de la déduction naturelle) : il existe des nombres pairs, il existe des nombres impaires, mais il n'existe pas de nombre à la fois pair et impair.

3. Cette propriété montre que l'égalité est transitive. On note $F = x = y \wedge y = z$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{F \vdash x = y \wedge y = z}\ \text{ax}}{F \vdash x = y}\ \text{elim-}\wedge \qquad \dfrac{\dfrac{}{F \vdash x = y \wedge y = z}\ \text{ax}}{F \vdash y = z}\ \text{elim-}\wedge}{F \vdash x = z}\ \text{elim-}=}{\vdash F \Rightarrow x = z}\ \text{intro-}\Rightarrow}{\text{le séquent voulu}}\ \text{intro-}\forall \times 3$$

Les introductions des $\forall$ sont licites car x, y, z n'apparaissent pas libres dans Γ. La règle elim-$=$ a été utilisée avec la formule $A = (\circ = z)[y/\circ]$.

4. On montre que $\neg(\exists x P) \Rightarrow \forall x(\neg P)$ en formalisant le raisonnement suivant : supposons qu'il existe un x tel que $P(x)$; cela contredit $\neg(\exists x P)$ et donc on a nécessairement $\neg P(x)$ pour tout x :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(\exists x P), P \vdash \neg(\exists x P)}\ \text{ax} \qquad \dfrac{\dfrac{}{\neg(\exists x P), P \vdash P}\ \text{ax}}{\neg(\exists x P), P \vdash \exists x P}\ \text{intro-}\exists}{\neg(\exists x P), P \vdash \bot}\ \text{elim-}\neg}{\neg(\exists x P) \vdash \neg P}\ \text{intro-}\neg}{\neg(\exists x P) \vdash \forall x \neg P}\ \text{intro-}\forall}{\vdash \neg(\exists x P) \Rightarrow \forall x(\neg P)}\ \text{intro-}\Rightarrow$$

L'intro-$\exists$ se fait en substituant x par x. L'intro-$\forall$ est licite car x n'est pas libre dans $\neg(\exists x P)$.

Pour la réciproque, j'utilise une règle de la logique classique, en notant $\Gamma = \{\forall x(\neg P), \neg\neg(\exists x P)\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \neg\neg(\exists x P)}\ \text{ax}}{\Gamma \vdash \exists x P}\ \text{elim-}\neg\neg \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, P \vdash \forall x(\neg P)}\ \text{ax}}{\Gamma, P \vdash \neg P}\ \text{elim-}\forall \qquad \dfrac{}{\Gamma, P \vdash P}\ \text{ax}}{\Gamma, P \vdash \bot}\ \text{elim-}\neg}{\Gamma \vdash \bot}\ \text{elim-}\exists}{\forall x(\neg P) \vdash \neg(\exists x P)}\ \text{raa}}{\forall x(\neg P) \Rightarrow \neg(\exists x P)}\ \text{intro-}\Rightarrow$$

L'elim-$\exists$ est licite car x n'est libre ni dans Γ ni dans $\bot$.

5. Pour le sens direct, pas besoin de la logique classique. On note $F = (\forall x A) \vee B$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{F \vdash F}\ \text{ax} \qquad \dfrac{\dfrac{}{F, B \vdash B}\ \text{ax}}{F, B \vdash A \vee B}\ \text{intro-}\vee \qquad \dfrac{\dfrac{\dfrac{}{F, \forall x A \vdash \forall x A}\ \text{ax}}{F, \forall x A \vdash A}\ \text{elim-}\forall}{F, \forall x A \vdash A \vee B}\ \text{intro-}\vee}{F \vdash A \vee B}\ \text{elim-}\vee}{F \vdash \forall x(A \vee B)}\ \text{intro-}\forall}{\vdash (\forall x A) \vee B \Rightarrow \forall x(A \vee B)}\ \text{intro-}\Rightarrow$$

L'introduction du $\forall$ est licite car x n'est pas libre dans B donc pas non plus dans $(\forall x A) \vee B$. Pour la réciproque, je ne sais pas faire sans logique classique. On formalise le raisonnement suivant : soit on a B etalors on a $(\forall x A) \vee B$, soit on a $\neg B$, mais comme $\forall x (A \vee B)$, cela oblige à avoir $\forall x A$. On montre d'abord le séquent $\forall x (A \vee B), \neg B \vdash (\forall x A) \vee B$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B, \neg B \vdash B}\text{ ax} \quad \dfrac{}{B, \neg B \vdash \neg B}\text{ ax}}{B, \neg B \vdash \bot}\text{ elim-}\neg}{\forall x (A \vee B), B, \neg B \vdash A}\text{ elim-}\bot\text{+aff} \quad \dfrac{}{\forall x (A \vee B), \neg B, A \vdash A}\text{ ax} \quad \dfrac{\dfrac{}{\forall x (A \vee B), \neg B \vdash \forall x (A \vee B)}\text{ ax}}{\forall x (A \vee B), \neg B \vdash A \vee B}\text{ elim-}\forall}{\forall x (A \vee B), \neg B \vdash A}\text{ elim-}\vee}{\forall x (A \vee B), \neg B \vdash \forall x A}\text{ intro-}\forall}{\forall x (A \vee B), \neg B \vdash (\forall x A) \vee B}\text{ intro-}\vee$$

L'introduction du $\forall$ est licite car x n'apparaît pas libre dans $\{\forall x (A \vee B), \neg B\}$ puisque cette variable n'apparaît pas libre dans B. On conclut par utilisation du tiers exclus :

$$\dfrac{\dfrac{}{\forall x (A \vee B) \vdash B \vee \neg B}\text{ TE} \quad \dfrac{\dfrac{}{\forall x (A \vee B), B \vdash B}\text{ ax}}{\forall x (A \vee B), B \vdash (\forall x A) \vee B}\text{ intro-}\vee \quad \dfrac{\text{montré au dessus}}{\forall x (A \vee B), \neg B \vdash (\forall x A) \vee B}}{\forall x (A \vee B) \vdash (\forall x A) \vee B}\text{ elim-}\vee$$
$$\dfrac{}{\vdash \forall x (A \vee B) \Rightarrow (\forall x A) \vee B}\text{ intro-}\Rightarrow$$

Exercice 191 (***) *Paradoxe du buveur*

"Dans toute pièce non vide, il existe une personne ayant la propriété suivante : si cette personne boit, alors tout le monde dans la pièce boit." Cet énoncé, noté E est connu sous le nom de "paradoxe du buveur". Nous allons montrer qu'il n'a en fait rien de paradoxal, puisqu'il exprime en langage naturel une formule du premier ordre qui est universellement vraie (l'équivalent d'une tautologie dans le calcul propositionnel).

1. En s'appuyant sur le tiers exclus, expliquer en français pourquoi l'énoncé E est vrai.
2. En faisant abstraction de "dans toute pièce non vide", écrire une formule F du calcul des prédicats formalisant le paradoxe du buveur. On pourra utiliser un prédicat unaire B (intuitivement, $B(x)$ signifie "x boit").
3. Montrer en déduction naturelle les séquents $\forall x\, B(x) \vdash F$ et $\neg(\forall x\, B(x)) \vdash F$. On suppose qu'on dispose d'une constante x_0 dans le langage (représentant le fait que la pièce n'est pas vide).
4. Conclure.

Remarque : Le paradoxe du buveur est une illustration flagrante du fait qu'en langage naturel, l'implication est souvent comprise comme une causalité, ce qu'elle n'est pas !

1. Soit dans la pièce tout le monde boit et dans ce cas pour toute personne dans la pièce, si elle boit tout le monde boit. Soit il existe une personne qui ne boit pas et dans ce cas si elle boit tout le monde boit vu la table de vérité de l'implication (faux implique vrai est vrai).
2. $F = \exists y (B(y) \Rightarrow \forall x B(x))$.
3. TODO.
4. On conclut par le tiers exclus :

$$\dfrac{\dfrac{}{\vdash \forall x B(x) \vee \neg(\forall x B(x))}\text{ TE} \quad \dfrac{\text{prouvé en 3}}{\forall x B(x) \vdash F} \quad \dfrac{\text{prouvé en 3}}{\neg(\forall x B(x)) \vdash F}}{\vdash F}\text{ elim-}\vee$$

Ainsi, F est bien universellement valide, par correction de la déduction naturelle.

Exercice 192 (***) *Constructivisme et intuitionnisme*

On considère un langage du premier ordre disposant d'un symbole de constante c, d'un symbole de fonction e d'arité 2 et d'un prédicat unaire R. Donner une preuve en déduction naturelle du séquent suivant :

$$\neg R(c), R(e(e(c,c),c) \vdash \exists x \exists y \, (R(e(x,y)) \wedge \neg R(x) \wedge \neg R(y))$$

Remarque : La preuve de ce séquent constitue une preuve non constructive de l'existence de deux nombres x, y irrationnels tels que x^y est rationnel. En voici une version écrite en français (et qui présuppose que l'on sait que $\sqrt{2}$ est irrationnel). De deux choses l'une : soit $\sqrt{2}^{\sqrt{2}}$ est rationnel, soit il ne l'est pas. S'il est rationnel, on conclut immédiatement. Sinon, il est irrationnel, $\sqrt{2}$ aussi et $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = 2$ est rationnel.

Cette preuve (dont vous pouvez vous inspirer pour construire une preuve en logique classique du séquent présenté) montre l'existence de nombres x, y convenables... si on accepte le principe du tiers exclus. Elle ne les construit pas. C'est une des raisons pour laquelle certains mathématiciens (dont Brouwer, père philosophique de la logique intuitionniste) ont refusé et refusent le principe du tiers exclus : accepter le tiers exclus ou une formulation équivalente implique d'accepter des preuves existentielles non constructives.

Il peut paraître loufoque de ne pas vouloir accepter une preuve non constructive, d'autant qu'il est pénible de devoir se passer du tiers exclus (voir le DM7)... jusqu'à ce qu'on souhaite automatiser les preuves ! Il est en effet bien plus facile de concevoir un assistant de preuve dans un cadre constructif. Ce rejet de la logique classique au profit de la logique intuitionniste n'est donc pas qu'une posture philosophique.

TODO

Exercice 193 (exo cours) *Implique et ou*

1. Montrer en déduction naturelle (sans utiliser les règles de la logique classique) le théorème suivant :

$$((C \Rightarrow A) \vee (C \Rightarrow B)) \Rightarrow (C \Rightarrow A \vee B)$$

2. L'implication réciproque est-elle vraie ? Justifier.

1. On note $F = (C \Rightarrow A) \vee (C \Rightarrow B)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{C \Rightarrow A, C \vdash C}\,\text{ax} \quad \dfrac{}{C \Rightarrow A, C \vdash C \Rightarrow A}\,\text{ax}}{C \Rightarrow A, C \vdash A}\,\text{elim-}\Rightarrow}{C \Rightarrow A, C \vdash A \vee B}\,\text{intro-}\vee \qquad \dfrac{\text{même preuve}}{C \Rightarrow B, C \vdash A \vee B}\,\text{intro-}\vee \quad \dfrac{}{F, C \vdash F}\,\text{ax}}{F, C \vdash A \vee B}\,\text{elim-}\vee}{\vdash F \Rightarrow (C \Rightarrow (A \vee B))}\,\text{double intro-}\Rightarrow$$

2. La réciproque est sémantiquement fausse donc non prouvable car la déduction naturelle est correcte.

Exercice 194 (exo cours) *Mi-Morgan*

1. Montrer en déduction naturelle (sans utiliser les règles de la logique classique) le théorème suivant :

$$\neg P \vee \neg Q \Rightarrow \neg(P \wedge Q)$$

2. L'implication réciproque est-elle vraie ? Justifier.

1. On note $F = \neg P \vee \neg Q$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg P, P \wedge Q \vdash P \wedge Q}\,\text{ax}}{\neg P, P \wedge Q \vdash P}\,\text{elim-}\wedge \quad \dfrac{}{\neg P, P \wedge Q \vdash \neg P}\,\text{ax}}{\neg P, P \wedge Q \vdash \bot}\,\text{elim-}\neg \quad \dfrac{\text{même preuve}}{\neg Q, P \wedge Q \vdash \bot}\,\text{elim-}\neg \quad \dfrac{}{F, P \wedge Q \vdash F}\,\text{ax}}{F, P \wedge Q \vdash \bot}\,\text{elim-}\vee\text{+aff}}{F \vdash \neg(P \wedge Q)}\,\text{intro-}\neg}{\vdash \neg P \vee \neg Q \Rightarrow \neg(P \wedge Q)}\,\text{intro-}\Rightarrow$$

2. La réciproque est sémantiquement vraie donc prouvable par complétude de la déduction naturelle.

Exercice 195 (exo cours) *Contraposition*

1. Montrer en déduction naturelle (sans utiliser les règles de la logique classique) le théorème suivant :

$$(P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)$$

2. L'implication réciproque est-elle vraie ? Justifier.

1. On note $\Gamma = \{P \Rightarrow Q, \neg Q, P\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash P}\text{ ax} \quad \dfrac{}{\Gamma \vdash P \Rightarrow Q}\text{ ax}}{\Gamma \vdash Q}\text{ elim-}\Rightarrow \quad \dfrac{}{\Gamma \vdash \neg Q}\text{ ax}}{\Gamma \vdash \bot}\text{ elim-}\neg}{P \Rightarrow Q, \neg Q \vdash \neg P}\text{ intro-}\neg}{\vdash (P \Rightarrow Q) \Rightarrow (\neg Q \Rightarrow \neg P)}\text{ double intro-}\Rightarrow$$

2. La réciproque est sémantiquement vraie donc prouvable par complétude de la déduction naturelle.

Exercice 196 (exo cours) *Ex falso*

1. Montrer en déduction naturelle (sans utiliser les règles de la logique classique) le théorème suivant :

$$((P \Rightarrow Q) \wedge (P \Rightarrow \neg Q)) \Rightarrow \neg P$$

2. L'implication réciproque est-elle vraie ? Justifier.

1. On note $F = (P \Rightarrow Q) \wedge (P \Rightarrow \neg Q)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{F, P \vdash F}\text{ ax}}{F, P \vdash P \Rightarrow Q}\text{ elim-}\wedge \quad \dfrac{}{F, P \vdash P}\text{ ax}}{F, P \vdash Q}\text{ elim-}\Rightarrow \quad \dfrac{\text{même preuve}}{F, P \vdash \neg Q}\text{ elim-}\Rightarrow}{F, P \vdash \bot}\text{ elim-}\neg}{F \vdash \neg P}\text{ intro-}\neg}{\vdash F \Rightarrow \neg P}\text{ intro-}\Rightarrow$$

2. La réciproque est sémantiquement vraie (faire une table de vérité par exemple) donc est prouvable.

Exercice 197 (exo cours) *Réécriture d'une implication*

1. Montrer en déduction naturelle le théorème suivant (sans utiliser les règles de la logique classique) :

$$(\neg P \vee Q) \Rightarrow (P \Rightarrow Q)$$

2. L'implication réciproque est-elle vraie ? Justifier.

1. On note $\Gamma = \{\neg P \vee Q, P, \neg P\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash P}\text{ ax} \quad \dfrac{}{\Gamma \vdash \neg P}\text{ ax}}{\Gamma \vdash \bot}\text{ elim-}\neg}{\Gamma \vdash Q}\text{ elim-}\bot \quad \dfrac{\dfrac{}{\neg P \vee Q, P, Q \vdash Q}\text{ ax} \quad \dfrac{}{\neg P \vee Q, P \vdash \neg P \vee Q}\text{ ax}}{\neg P \vee Q, P \vdash Q}\text{ elim-}\vee}{\vdash (\neg P \vee Q) \Rightarrow (P \Rightarrow Q)}\text{ double intro-}\Rightarrow$$

2. La réciproque est sémantiquement vraie donc prouvable par complétude de la déduction naturelle.

Exercice 198 (exo cours) *Double existence*

1. Montrer en déduction naturelle le théorème suivant :

$$\exists x(A \wedge B) \Rightarrow (\exists x A) \wedge (\exists x B)$$

2. Est-il possible d'en prouver la réciproque ? Justifier.

1. On note $F = \exists x(A \wedge B)$. Alors :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{F, A \wedge B \vdash A \wedge B}\text{ ax}}{F, A \wedge B \vdash A}\text{ elim-}\wedge}{F, A \wedge B \vdash \exists x A}\text{ intro-}\exists \qquad \dfrac{}{F \vdash \exists x(A \wedge B)}\text{ ax}}{F \vdash \exists x A}\text{ elim-}\exists$$

   L'élimination du $\exists$ est licite car x n'est libre ni dans F ni dans $\exists x A$. On procède de même pour $\exists x B$ et on conclut avec un intro-$\wedge$ puis un intro-$\Rightarrow$.

2. La réciproque est sémantiquement fausse donc non prouvable par correction de la déduction naturelle.

Exercice 199 (exo cours) *Négation d'une existence*

1. Montrer en déduction naturelle le théorème suivant :

$$\forall x \neg F \Rightarrow \neg \exists x F$$

2. En vous appuyant sur votre intuition mathématique, l'implication réciproque est-elle vraie ?

1. On note $\Gamma = \{\forall x \neg F, \exists x F\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, F \vdash \forall x \neg F}\text{ ax}}{\Gamma, F \vdash \neg F}\text{ elim-}\forall \qquad \dfrac{}{\Gamma, F \vdash F}\text{ ax}}{\Gamma, F \vdash \bot}\text{ elim-}\neg \qquad \dfrac{}{\Gamma \vdash \exists x F}\text{ ax}}{\dfrac{\Gamma \vdash \bot}{\forall x \neg F \vdash \neg \exists x F}\text{ intro-}\neg}\text{ elim-}\exists}{\vdash \forall x \neg F \Rightarrow \neg \exists x F}\text{ intro-}\Rightarrow$$

   L'élimination du $\exists$ est licite car x n'est libre ni dans Γ ni dans $\bot$.

2. La réciproque est vraie, on ne demande pas de justification formelle.

Exercice 200 (exo cours) *Correction classique*

1. Rappeler les règles du tiers exclus et du raisonnement par l'absurde et en montrer la correction.
2. La règle suivante est-elle correcte ?

$$\dfrac{\Gamma \vdash \neg B \Rightarrow \neg A}{\Gamma \vdash A \Rightarrow B}$$

1. Pour le tiers exclus $\dfrac{}{\Gamma \vdash A \vee \neg A}\text{ TE}$. Pour toute formule A, toute valuation satisfait $A \vee \neg A$: c'est en particulier vrai pour les valuations qui satisfont Γ donc $\Gamma \models A \vee \neg A$.

   Pour le raisonnement par l'absurde, la règle à étudier est :

$$\dfrac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}\text{ raa}$$

   Supposons qu'on ait $\Gamma, \neg A \models \bot$ et montrons $\Gamma \models A$. Soit donc v une valuation v qui satisfait Γ et supposons par l'absurde que $v(A) = 0$. Alors $v(\neg A) = 1$ donc v satisfait $\Gamma, \neg A$ donc v satisfait $\bot$ ce qui est impossible. On en déduit que toute valuation qui satisfait Γ satisfait A.

2. Oui. Soit v une valuation satisfaisant Γ. Par hypothèse, v satisfait $\neg B \Rightarrow \neg A$ mais cette formule est équivalente à $\neg\neg B \vee \neg A \equiv \neg A \vee B \equiv A \Rightarrow B$. Donc v satisfait $A \Rightarrow B$ ce qui conclut.

Exercice 201 (***) *Théorème de compacité*

On montre dans cet exercice une version faible du théorème de compacité dont l'énoncé suit. Soit Γ un ensemble dénombrable de formules du calcul propositionnel. L'ensemble Γ est satisfiable si et seulement tout sous ensemble fini de Γ est satisfiable.

1. Montrer que ce théorème est vrai lorsque Γ est un ensemble fini.

Ce cas traité, on suppose à présent que Γ est infini dénombrable. On suppose de plus que que tout sous ensemble fini de Γ est satisfiable et on souhaite montrer que dans ces conditions Γ est satisfiable.

2. Montrer que l'ensemble V des variables intervenant dans les formules de Γ est dénombrable.

On numérote donc les variables intervenant dans Γ : $V = \{x_0, x_1, ...\}$. Pour montrer le résultat souhaité, on construit une suite $(\varepsilon_n)_{n\in\mathbb{N}} \in \{0,1\}^{\mathbb{N}}$ telle que la valuation $v_\Gamma : x_n \mapsto \varepsilon_n$ satisfait Γ. Pour ce faire, on montre par récurrence sur $n \in \mathbb{N}$ la propriété $P(n)$ suivante en parallèle de la construction de la suite ε : pour toute partie finie $\Gamma' \subset \Gamma$, il existe une valuation v telle que v satisfait Γ' et pour tout $i \in [\![0, n]\!]$, $v(x_i) = \varepsilon_i$.

3. On définit ε_0 de la façon suivante. Si pour toute partie finie $\Gamma' \subset \Gamma$ il existe une valuation v qui satisfait Γ' et telle que $v(x_0) = 0$, alors on fixe $\varepsilon_0 = 0$. Sinon, on fixe $\varepsilon_0 = 1$. Montrer $P(0)$.

4. Supposons $\varepsilon_0, ..., \varepsilon_n$ définis et $P(n)$ vraie. Construire ε_{n+1} de telle sorte à ce que $P(n+1)$ soit vraie aussi et montrer qu'elle l'est bien.

5. Montrer enfin que la valuation v_Γ ainsi construite satisfait Γ et conclure.

1. Γ est un sous ensemble fini de lui-même dans ces conditions.
2. Chaque formule de Γ est finie donc fait intervenir un nombre fini de variables donc :

$$\{\text{variables intervenant dans } \Gamma\} = \bigcup_{F\in\Gamma} \{\text{variables intervenant dans } F\}$$

est une union dénombrable (puisque Γ l'est par hypothèse) d'ensembles finis donc est dénombrable.

3. $P(0)$ est évidemment vraie dans le premier cas. Dans le second cas, il existe une partie finie $\Gamma_0 \in \Gamma$ telle que toute valuation v satisfaisant Γ_0 est telle que $v(x_0) = 1$. Soit Γ' une partie finie de Γ. Alors $\Gamma' \cup \Gamma_0$ est aussi une partie finie de Γ donc est satisfiable par une valuation v (par hypothèse). Alors v satisfait Γ_0 donc par hypothèse sur Γ_0, $v(x_0) = \varepsilon_0 = 1$ et v satisfait Γ' aussi. Ainsi, tout sous ensemble fini de Γ est satisfiable par une valuation v telle que $v(x_0) = \varepsilon_0$ et c'est ce qu'on voulait montrer.
4. On construit ε_{n+1} selon le même principe : si pour toute partie finie $\Gamma' \subset \Gamma$, il existe une valuation v satisfaisant Γ' telle que $v(x_i) = \varepsilon_i$ pour tout $i \in [\![0, n]\!]$ et vérifiant $v(x_{n+1}) = 0$, alors on pose $\varepsilon_{n+1} = 0$, sinon, on pose $\varepsilon_{n+1} = 1$.

   Dans le premier cas, $P(n+1)$ est vraie. Dans le second, il existe $\Gamma_0 \subset \Gamma$ une partie finie telle que pour toute valuation v satisfaisant Γ_0 et telle que $v(x_i) = \varepsilon_i$ pour tout $i \in [\![0, n]\!]$, on a $v(x_{n+1}) = 1$. Soit Γ' une partie finie de Γ. Alors $\Gamma' \cup \Gamma_0$ est une partie finie de Γ donc par hypothèse de récurrence, il existe une valuation v satisfaisant $\Gamma' \cup \Gamma_0$ et telle que pour tout $i \in [\![0, n]\!]$ on ait $v(x_i) = \varepsilon_i$. Comme v satisfait en particulier Γ_0, $v(x_{n+1}) = 1 = \varepsilon_{n+1}$ par hypothèse du deuxième cas. De plus v satisfait Γ'.
5. Soit $F \in \Gamma$ et montrons que v_Γ satisfait F. On note V_F l'ensemble des variables intervenant dans F. Comme V_F est fini, il existe $k \geq 0$ tel que $V_F \subset \{x_0, ..., x_k\}$. D'après $P(k)$, il existe une valuation v satisfaisant $\{F\}$ et telle que pour tout $i \in [\![0, k]\!]$, $v(x_i) = \varepsilon_i$. Comme v et v_Γ coïncident sur toutes les variables de F, v_Γ satisfait F. On en déduit que v_Γ satisfait Γ ce qui conclut le sens réciproque du théorème, le sens direct étant évident.

Exercice 202 (***) *Esquisse pour la complétude de la déduction naturelle*

Dans cet exercice, on donne les grandes lignes d'une preuve de la complétude de la logique classique pour le calcul propositionnel. Pour faciliter la construction d'arbres de preuves, on considère que les règles d'affaiblissement et d'introduction du vrai (ci-dessous) font partie des règles de ce système de déduction :

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ \text{aff} \qquad\qquad \frac{}{\Gamma \vdash \top}\ \text{intro-}\top$$

On note $V = \{x_1, ..., x_n\}$ l'ensemble des variables propositionnelles manipulées. Pour toute formule φ et toute valuation v on note par ailleurs :

$$|\varphi|_v = \begin{cases} \varphi \text{ si } v \text{ satisfait } \varphi \\ \neg\varphi \text{ sinon} \end{cases}$$

1. Soit φ une formule et v une valuation. Dans cette question, $\Gamma = \{|x_1|_v, ..., |x_n|_v\}$. Montrer par induction structurelle sur φ que $\Gamma \vdash |\varphi|_v$. Pour les cas inductifs, on ne traitera que les cas impliquant les connecteurs $\neg$ et $\wedge$ et on admettra que des raisonnements similaires sont possibles pour $\vee$ et $\Rightarrow$.

2. Soit x une variable, φ une formule et Γ un ensemble quelconque de formules. Montrer que si $\Gamma, x \vdash \varphi$ et $\Gamma, \neg x \vdash \varphi$ alors $\Gamma \vdash \varphi$.

3. Soit φ une tautologie. Montrer que pour tout $i \leq n$ et toute valuation v sur $\{x_1, ..., x_i\}$ on a $\Gamma_{v,i} = \{|x_1|_v, ..., |x_i|_v\} \vdash \varphi$. En déduire que toute tautologie est un théorème.

4. En déduire la complétude de la logique classique.

---

1. Si $\varphi = \top$ alors $|\varphi|_v = \top$ et $\dfrac{}{\Gamma \vdash \top}$ intro-$\top$ en est une preuve.

   Si $\varphi = \bot$ alors $|\varphi|_v = \neg\bot$ ce qui se prouve via $\dfrac{\dfrac{}{\Gamma, \bot \vdash \bot}\ \text{ax}}{\Gamma \vdash \neg\bot}\ \text{intro-}\neg$

   Si $\varphi = x_i$ est une variable alors on en a une preuve depuis Γ par axiome.

   Si $\varphi = \neg\psi$, on sait par hypothèse inductive que $\Gamma \vdash |\psi|_v$. Alors :

   - Si $v(\varphi) = 1$ alors $|\varphi|_v = \varphi = \neg\psi = |\psi|_v$ donc immédiatement $\Gamma \vdash |\varphi|_v$.
   - Si $v(\varphi) = 0$ alors $|\psi|_v = \psi$ et $|\varphi|_v = \neg\varphi = \neg\neg\psi$ et on obtient $\Gamma \vdash |\varphi|_v$ via :

   $$\dfrac{\dfrac{\dfrac{\dfrac{\text{par hypothèse inductive}}{\Gamma \vdash \psi}\ \text{aff}}{\Gamma, \neg\psi \vdash \psi} \qquad \dfrac{}{\Gamma, \neg\psi \vdash \psi}\ \text{ax}}{\Gamma, \neg\psi \vdash \bot}\ \text{elim-}\neg}{\Gamma \vdash \neg\neg\psi = |\varphi|_v}\ \text{intro-}\neg$$

   Si $\varphi = \psi \wedge \sigma$, on a par hypothèse inductive $\Gamma \vdash |\psi|_v$ et $\Gamma \vdash |\sigma|_v$. Alors :

   - Si $v(\varphi) = 1$ alors $|\varphi|_v = \psi \wedge \sigma$ et vu la sémantique de $\wedge$, $v(\psi) = v(\sigma) = 1$ dont $|\psi|_v = \psi$ et $|\sigma|_v = \sigma$. Il suffit donc d'introduire un $\wedge$ pour obtenir la preuve souhaitée : $\dfrac{\Gamma \vdash \psi \qquad \Gamma \vdash \sigma}{\Gamma \vdash \psi \wedge \sigma = \varphi}\ \text{intro-}\wedge$
   - Si $v(\varphi) = 0$, $|\varphi|_v = \neg(\psi \wedge \sigma)$ et l'un parmi $v(\psi)$ et $v(\sigma)$ vaut 0 : on traite le cas où $v(\psi) = 0$.

   $$\dfrac{\dfrac{\dfrac{\dfrac{\text{par hypothèse inductive}}{\Gamma \vdash |\psi|_v = \neg\psi}\ \text{aff}}{\Gamma, \psi \wedge \sigma \vdash \neg\psi} \qquad \dfrac{\dfrac{}{\Gamma, \psi \wedge \sigma \vdash \psi \wedge \sigma}\ \text{ax}}{\Gamma, \psi \wedge \sigma \vdash \psi}\ \text{elim-}\wedge}{\Gamma, \psi \wedge \sigma \vdash \bot}\ \text{elim-}\neg}{\Gamma \vdash \neg(\psi \wedge \sigma) = |\varphi|_v}\ \text{intro-}\neg$$

2. Il suffit d'utiliser le tiers exclus : $\dfrac{\Gamma, x \vdash \varphi \qquad \Gamma, \neg x \vdash \varphi \qquad \dfrac{}{\Gamma \vdash x \vee \neg x}\ \text{TE}}{\Gamma \vdash \varphi}\ \text{elim-}\vee$

3. Il s'agit de montrer que si $\models \varphi$ alors $\vdash \varphi$. On montre par récurrence descendante sur i que pour toute valuation v sur $\{x_1, ..., x_i\}$, $\Gamma_{v,i} = \{|x_1|_v, ..., |x_i|_v\} \vdash \varphi$. C'est vrai pour $i = n$ car, si v est une valuation elle satisfait φ (puisque c'est une tautologie) donc par la question 1, $\Gamma_{v,n} \vdash |\varphi|_v = \varphi$.

   Si le résultat est établi pour i, soit v une valuation sur $V_i = \{x_1, ..., x_{i-1}\}$. On note v_0 la valuation qui étend v par $v_0(x_i) = 0$ et de même v_1 telle que $v_1(x_i) = 1$. Par hypothèse, $\Gamma_{v_0,i} \vdash \varphi$ et $\Gamma_{v_1,i} \vdash \varphi$. Mais $\Gamma_{v_0,i} = \Gamma_{v,i-1} \cup \{\neg x_i\}$ et $\Gamma_{v_1,i} = \Gamma_{v,i-1} \cup \{x_i\}$ donc la question 2 donne $\Gamma_{v,i-1} \vdash \varphi$.

   Pour $i = 0$ on a en particulier $\Gamma_{v,0} = \emptyset \vdash \varphi$ ce qui conclut.

4. On suppose que $\Gamma = \{\varphi_1, ..., \varphi_k\} \models \varphi$ (et on veut montrer que $\Gamma \vdash \varphi$). De cette conséquence sémantique on déduit que $\bigwedge_{i=1}^{k} \varphi_i \Rightarrow \varphi$ est une tautologie donc un théorème par la question 3. On montre en utilisant la règle intro-$\wedge$ à répétition que $\Gamma \vdash \bigwedge_{i=1}^{k} \varphi_i$. On en déduit que :

   $$\dfrac{\Gamma \vdash \bigwedge_{i=1}^{k} \varphi_i \qquad \dfrac{\vdash \bigwedge_{i=1}^{k} \varphi_i \Rightarrow \varphi}{\Gamma \vdash \bigwedge_{i=1}^{k} \varphi_i \Rightarrow \varphi}\ \text{aff}}{\Gamma \vdash \varphi}\ \text{elim-}\Rightarrow$$

Exercice 203 (*) *Ensembles minimalement insatisfiables*

Dans cet exercice, une *clause* est une clause disjonctive.

1. Redémontrer que toute formule du calcul propositionnel est équivalente à une conjonction de clauses.

Soit S un ensemble de clauses. Si S est non satisfiable, on dit que S est *minimalement insatisfiable* si pour tout $R \subsetneq S$, l'ensemble de clauses R est satisfiable.

2. L'ensemble $\{p \vee \neg q, p \vee q, \neg p\}$ est-il insatisfiable ? Est-il minimalement insatisfiable ?

On dit qu'un littéral intervenant dans S est *pur* si son opposé n'intervient pas dans S.

3. Soit S un ensemble insatisfiable de clauses. Montrer que si S est minimalement insatisfiable alors il ne contient pas de littéral pur.
4. La réciproque de la question précédente est-elle vraie ? Justifier.

1. Si F est une formule, on détermine la table de vérité de $\neg F$ d'où on déduit une formule équivalente à F sous forme normale disjonctive puis une forme normale conjonctive pour $\neg\neg F = F$ via De Morgan.
2. Oui aux deux.
3. Supposons par l'absurde que S contient un littéral pur L. Notons C l'ensemble de clauses de S faisant intervenir L. Alors $S' = S \setminus C$ est satisfiable par une valuation v' par minimale insatisfiabilité de S. Si $v'(L) = 0$, modifier v' en v telle que $v(L) = 1$ produit v qui continue de satisfaire S' puisque L est pur. Alors v satisfait S' et C donc S : contradiction.
4. Non : $\{a, \neg a, b, \neg b\}$ est un ensemble insatisfiable qui ne contient pas de littéral pur mais qui n'est pas minimalement insatisfiable car $\{a, b, \neg b\}$ en est un sous ensemble strict non satisfiable.

Exercice 204 (**) *Tigres de papier*

D'après Le livre qui rend fou *par Raymond Smullyan.*

Le roi d'une contrée lointaine s'inquiète du surpeuplement de ses prisons. Il décide de faire passer des épreuves aux prisonniers : un prisonnier qui les réussit toutes est libre. Chaque épreuve se déroule ainsi : le prisonnier est amené devant deux pièces fermées. Chacune des pièces peut contenir soit un tigre, soit être vide : au pire des cas, les deux contiennent des tigres, au meilleur, les deux sont vides. Le prisonnier doit ouvrir l'une des deux pièces : si un tigre s'y trouve, il finit dévoré, sinon, il est libre d'affronter l'épreuve suivante. Le roi décide de favoriser les prisonniers disposant d'un sens logique robuste en fixant sur chacune des deux portes une pancarte donnant une information partielle sur le contenu des pièces.

1. Lors de la première épreuve, le prisonnier peut lire sur la porte 1 "Au moins une pièce est vide" et sur la porte 2 "Il y a un tigre dans l'autre pièce". Le roi - qui est incapable de mentir - affirme au prisonnier que les deux pancartes disent soit toutes les deux la vérité, soit mentent toutes les deux.
   a) Traduire les informations données par les pancartes et le roi en formules du calcul propositionnel qui utilisent deux variables propositionnelles dont on précisera la signification.
   b) En dressant une table de vérité, indiquer si possible le contenu des deux pièces.
2. Lors de la deuxième épreuve, les deux pancartes affirment "Les deux pièces sont vides". Le roi - toujours incapable de mentir - indique que la pancarte de la porte 1 dit la vérité si la pièce 1 est vide (et ment sinon) et que la pancarte de la porte 2 dit la vérité si la pièce 2 contient un tigre (et ment sinon).
   a) Traduire ces informations en formules du calcul propositionnel.
   b) Simplifier par équivalences la formule qui permet de déduire le contenu des pièces. Indiquer si possible le contenu des pièces. Est-il possible de ne pas finir dévoré ?
3. Lors de la dernière épreuve, le roi donne exactement la même information - véridique - au prisonnier que dans la question 2. Sur la porte 1 on lit "Choisis bien quelle pièce ouvrir, ça a de l'importance !" et sur la porte 2 "Tu ferais mieux de choisir l'autre pièce !"
   a) Modéliser à nouveau cette énigme par des formules du calcul propositionnel.
   b) Le prisonnier peut-il retrouver la liberté ? Si oui, dire comment.

1. a) On note v_1 la proposition "La pièce numéro 1 est vide" et v_2 la proposition "La pièce numéro 2 est vide". On note P_1 (respectivement P_2) la formule correspondant à l'information donnée par la pancarte 1 (respectivement 2) et R la formule correspondant à l'information donnée par le roi :

$$P_1 = v_1 \vee v_2 \ , \ P_2 = \neg v_1 \text{ et } R = (v_1 \vee v_2) \Leftrightarrow \neg v_1.$$

b) Comme le roi ne ment pas, on sait que R est vraie. On dresse la table de vérité de cette formule :

v_1	v_2	$v_1 \vee v_2$	R
0	0	0	0
0	1	1	1
1	0	1	0
1	1	1	0

La seule valuation que rend R vraie est celle qui attribue la valeur 0 à v_1 et 1 à v_2 : on en déduit qu'il y a un tigre dans la pièce 1 et que la pièce 2 est vide.

2. a) On reprend les notations de la question précédente. On a donc :

$$P_1 = P_2 = v_1 \wedge v_2 \text{ et } R = (v_1 \Leftrightarrow (v_1 \wedge v_2)) \wedge (\neg v_2 \Leftrightarrow (v_1 \wedge v_2))$$

b) Comme le roi ne ment pas, on sait que R est vraie. Or :

$$\begin{aligned} R &\equiv (\neg v_1 \vee (v_1 \wedge v_2)) \wedge (\neg v_1 \vee \neg v_2 \vee v_1) \wedge (v_2 \vee (v_1 \wedge v_2)) \wedge (\neg v_1 \vee \neg v_2 \vee \neg v_2) \\ &\equiv (\neg v_1 \wedge v_2) \wedge \top \wedge (v_2 \vee v_1) \wedge v_2 \wedge (\neg v_1 \vee \neg v_2) \\ &\equiv \neg v_1 \wedge (v_2 \vee \neg v_2) \wedge (v_2 \vee v_1) \wedge v_2 \\ &\equiv \neg v_1 \wedge v_2 \wedge (v_2 \vee v_1) \end{aligned}$$

La seule valuation v qui peut satisfaire R est donc telle que $v(v_2) = 1$ et $v(v_1) = 0$ et comme elle convient on sait que la pièce 1 contient un tigre et la pièce 2 non : il faut la choisir.

3. a) Toujours avec les mêmes notations :

$$P_1 = (v_1 \wedge \neg v_2) \vee (v_2 \wedge \neg v_1) \ , \ P_2 = \neg v_2 \text{ et } R = (v_1 \Leftrightarrow (\neg v_1 \vee \neg v_2)) \wedge (\neg v_2 \Leftrightarrow \neg v_2)$$

La formule pour P_1 mérite une courte explication : si le choix du prisonnier a de l'importance, c'est que les deux pièces ne contiennent pas la même chose (sinon le choix du prisonnier n'aurait aucune importance sur la suite des événements). Pour P_2, on peut interpréter la pancarte comme disant "il n'y a pas de tigre dans la pièce 1" (v_1) plutôt que "il y a un tigre dans la pièce 2".

b) Le roi ne ment toujours pas donc R doit être vraie. On simplifie R en remarquant que $R \equiv (v_1 \Leftrightarrow (v_1 \wedge \neg v_2) \vee (v_2 \wedge \neg v_1))$ puis on dresse sa table de vérité :

v_1	v_2	$(v_1 \wedge \neg v_2) \vee (v_2 \wedge \neg v_1)$	R
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Deux cas sont possibles : soit les deux pièces contiennent des tigres (première ligne), soit la première pièce est vide et l'autre contient un tigre (troisième ligne). Le prisonnier ne peux donc choisir que la pièce 1 s'il veut avoir une chance de retrouver la liberté.

Exercice 205 (*) *Contingence*

On dit qu'une formule F est contingente si elle est satisfiable sans être une tautologie.

1. Les formules suivantes sont-elles contingentes ?
   a) $F = (p \wedge q) \Leftrightarrow (p \Rightarrow \neg q)$.
   b) $G = (p \Rightarrow q) \Rightarrow p$.
2. Soit F et G deux formules du calcul propositionnel. Dire en justifiant quelles affirmations sont vraies :

a) Si F est contingente alors $\neg F$ l'est aussi.

b) Si F et G sont contingentes alors $F \vee G$ et $F \wedge G$ aussi.

c) Si $F \vee G$ est insatisfiable, alors F et G sont insatisfiables.

d) Si $F \vee G$ est une tautologie alors F et G sont des tautologies.

e) Si $F \vee G$ est une tautologie alors F est une tautologie ou G est une tautologie.

1. a) est une contradiction donc non contingente ; b) est contingente.
2. a) Vrai en utilisant la définition.

   b) Faux : p et $\neg p$ sont contingentes mais $p \vee \neg p$ est tautologique et $p \wedge \neg p$ antilogique.

   c) Vrai d'après la sémantique du $\vee$.

   d) Faux : ni p ni $\neg p$ ne sont des tautologies alors que $p \vee \neg p$ si.

   e) Faux avec le même contre exemple que d).

Exercice 206 (**) *Preuves fonctionnelles*

TODO : Reprendre.

1. Ici, f est un symbole de fonction unaire et T un symbole de relation binaire. Montrer que $A_1, A_2, A_3 \vdash A$ où :
   - $A_1 = \forall x\,(\exists y\, T(x,y) \Rightarrow T(x, f(x)))$.
   - $A_2 = \forall x\, \exists y\, T(x,y)$.
   - $A_3 = \exists x\, T(f(f(x)), x)$.
   - $A = \exists x\, \exists y\, \exists z\, (T(x,y) \wedge T(y,z) \wedge T(z,x))$.
2. Formaliser et montrer formellement que si f et g sont des fonctions unaires données alors :

   a) Si f est une involution alors c'est une bijection.

   b) Si f et g sont injectives alors $f \circ g$ l'est aussi.

   c) Si $f \circ g$ est injective et g est surjective alors f est injective.

Exercice 207 (***) *Paradoxe du barbier*

1. Montrer à l'aide d'une preuve en déduction naturelle que $(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$ est un théorème.
2. En déduire une preuve en déduction naturelle du séquent suivant : $P \Leftrightarrow \neg P \vdash \bot$.
3. On considère un langage du premier ordre disposant d'un symbole de relation R d'arité 2. On note
$$F = \exists y\, \forall x\, (R(x,y) \Leftrightarrow \neg R(x,x))$$
Montrer en déduction naturelle que $\neg F$ est un théorème.

Le paradoxe du barbier, s'énonce ainsi : le maire donne l'ordre au barbier du village de raser toutes les personnes qui ne se rasent pas elles-mêmes et seulement celles-ci. Qui rase le barbier ? S'il se rase lui même, alors il contredit la règle car cette dernière ne lui permet de raser que les personnes qui ne se rasent pas elles-mêmes. S'il ne se rase pas lui-même, il est à nouveau en tort puisqu'il doit raser les personnes qui ne se rasent pas elles-mêmes...

4. A l'aide de la question 3, expliquer pourquoi il n'y a rien de paradoxal dans le paradoxe du barbier ! Quelle particularité du langage naturel fait tout d'abord penser à un paradoxe ?

1. Il s'agit d'exhiber une preuve en déduction naturelle pour le séquent $\vdash (P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$. Or :
$$\dfrac{\dfrac{\dfrac{}{\neg P, \neg(\neg P \vee Q) \vdash \neg(\neg P \vee Q)}\,\text{ax} \quad \dfrac{\dfrac{}{\neg P, \neg(\neg P \vee Q) \vdash \neg P}\,\text{ax}}{\neg P, \neg(\neg P \vee Q) \vdash \neg P \vee Q}\,\text{intro-}\vee}{\neg P, \neg(\neg P \vee Q) \vdash \bot}\,\text{elim-}\neg}{\neg(\neg P \vee Q) \vdash P}\,\text{ra}$$

On en déduit l'arbre de preuve suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(\neg P \vee Q) \vdash P}\text{ prouvé}}{\neg(\neg P \vee Q), P \Rightarrow Q \vdash P}\text{ aff} \quad \dfrac{}{\neg(\neg P \vee Q), P \Rightarrow Q \vdash P \Rightarrow Q}\text{ ax}}{\neg(\neg P \vee Q), P \Rightarrow Q \vdash Q}\text{ elim-}\Rightarrow}{\neg(\neg P \vee Q), P \Rightarrow Q \vdash \neg P \vee Q}\text{ intro-}\vee \quad \dfrac{}{\neg(\neg P \vee Q), P \Rightarrow Q \vdash \neg(\neg P \vee Q)}\text{ ax}}{\neg(\neg P \vee Q), P \Rightarrow Q \vdash \bot}\text{ elim-}\neg}{P \Rightarrow Q \vdash \neg P \vee Q}\text{ ra}}{\vdash (P \Rightarrow Q) \Rightarrow (\neg P \vee Q)}\text{ intro-}\Rightarrow$$

Cette preuve formalise l'intuition sémantique suivante. On souhaite montrer $\neg P \vee Q$ sachant $P \Rightarrow Q$. Supposons par l'absurde que $\neg P \vee Q$ est faux. Dans ce cas la sémantique du $\vee$ nous indique que Q est faux et P est vrai, mais dans ce cas, l'implication $P \Rightarrow Q$ est fausse et c'est une contradiction.

2. Notons $F = A \Leftrightarrow \neg A$ (attention, F désignera une autre formule à la question 3 !). On se rappelle que $A \Leftrightarrow \neg A$ n'est qu'un raccourci pour signifier $(A \Rightarrow \neg A) \wedge (\neg A \Rightarrow A)$. On a d'une part :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash (A \Rightarrow \neg A) \Rightarrow (\neg A \vee \neg A)}\text{ prouvé en 1}}{F \vdash (A \Rightarrow \neg A) \Rightarrow (\neg A \vee \neg A)}\text{ aff} \quad \dfrac{\dfrac{}{F \vdash F}\text{ ax}}{F \vdash A \Rightarrow \neg A}\text{ elim-}\wedge}{F \vdash \neg A \vee \neg A}\text{ elim-}\Rightarrow \quad \dfrac{}{F, \neg A \vdash \neg A}\text{ ax} \quad \dfrac{}{F, \neg A \vdash \neg A}\text{ ax}}{F \vdash \neg A}\text{ elim-}\vee$$

Et de manière très similaire (avec pour seule différence notable l'utilisation de elim-$\neg\neg$) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash (\neg A \Rightarrow A) \Rightarrow (\neg\neg A \vee A)}\text{ prouvé en 1}}{F \vdash (\neg A \Rightarrow A) \Rightarrow (\neg\neg A \vee A)}\text{ aff} \quad \dfrac{\dfrac{}{F \vdash F}\text{ ax}}{F \vdash \neg A \Rightarrow A}\text{ elim-}\wedge}{F \vdash \neg\neg A \vee A}\text{ elim-}\Rightarrow \quad \dfrac{}{F, A \vdash A}\text{ ax} \quad \dfrac{\dfrac{}{F, \neg\neg A \vdash \neg\neg A}\text{ ax}}{F, \neg\neg A \vdash A}\text{ elim-}\neg\neg}{F \vdash A}\text{ elim-}\vee$$

En utilisant la règle elim-$\neg$, on déduit de ces deux arbres une preuve de $F \vdash \bot$ comme attendu.

3. On note G la formule $\forall x\,(R(y,x) \Leftrightarrow \neg R(x,x))$. Voici une preuve du séquent $\vdash \neg F$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{F \vdash F}\text{ ax} \quad \dfrac{\dfrac{\dfrac{}{F, G \vdash G}\text{ ax}}{F, G \vdash R(y,y) \Leftrightarrow \neg R(y,y)}\text{ elim-}\forall \quad \dfrac{\dfrac{}{R(y,y) \Leftrightarrow \neg R(y,y) \vdash \bot}\text{ prouvé en 2}}{F, G \vdash (R(y,y) \Leftrightarrow \neg R(y,y)) \Rightarrow \bot}\text{ intro-}\Rightarrow + \text{ aff}}{F, G \vdash \bot}\text{ elim-}\Rightarrow}{F \vdash \bot}\text{ elim-}\exists}{\vdash \neg F}\text{ intro-}\neg$$

L'utilisation de la règle elim-$\exists$ ci-dessus est licite puisque la variable y n'est libre ni dans F ni dans $\bot$. On en déduit bien que $\neg F$ est un théorème. Cette preuve formalise l'intuition sémantique suivante : par l'absurde, si on avait F, il y aurait un y tel que $\forall x\,(R(y,x) \Leftrightarrow \neg R(x,x))$. En particulier, on aurait $R(y,y) \Leftrightarrow \neg R(y,y)$ ce qui est faux : c'est donc que $\neg F$.

4. Si on interprète le prédicat R de façon à ce que $R(x,y)$ signifie "x rase y", la preuve obtenue à la question 3 et la correction de la déduction naturelle pour le calcul des prédicats prouvent qu'il n'existe pas d'individu y tel que pour tout individu x, y rase x si et seulement si x ne se rase pas lui même. Autrement dit, sous les contraintes du paradoxe du barbier, le barbier n'existe pas et donc le paradoxe disparaît.

   L'impression de paradoxe provient du fait qu'en langage naturel, lorsqu'on dit "le barbier rase les personnes qui ne se rasent pas elles-mêmes et uniquement celles-ci", on présuppose implicitement l'existence d'un barbier.

Exercice 208 (**) *Définitions équivalentes de la logique classique*

L'objectif de cet exercice est de montrer que l'on peut construire la logique classique à partir de la logique

intuitionniste de différentes façons, toutes équivalentes en un sens précisé par la suite. On rappelle ci-dessous les règles de la logique intuitionniste dont on note $\mathcal{R}$ l'ensemble :

$$\frac{}{\Gamma, A \vdash A}\ \text{ax} \qquad \frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ \text{aff} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \text{intro-}\Rightarrow \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \text{elim-}\Rightarrow$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \text{intro-}\wedge \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \text{elim-}\wedge \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \text{elim-}\wedge$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \text{intro-}\vee \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \text{intro-}\vee \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \text{elim-}\vee$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \text{elim-}\bot \qquad \frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \text{intro-}\neg \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \text{elim-}\neg$$

On introduit par ailleurs les règles suivantes :

- Axiome du tiers-exclus : $\dfrac{}{\Gamma \vdash A \vee \neg A}$ te
- Règle d'élimination de la double négation : $\dfrac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}$ elim-$\neg\neg$
- Raisonnement par l'absurde : $\dfrac{\Gamma, \neg A \vdash \bot}{\Gamma \vdash A}$ ra
- Contraposition : $\dfrac{\Gamma \vdash \neg B \Rightarrow \neg A}{\Gamma \vdash A \Rightarrow B}$ ctr
- Règle de Peirce : $\dfrac{}{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$ Peirce

On dit qu'une règle est *dérivable* dans un système de déduction si on peut construire sa conclusion avec les règles du système de déduction en supposant qu'on a déjà une preuve de ses prémisses. Par exemple, la règle suivante — dite de la coupure — est dérivable en logique intuitionniste :

$$\frac{\Gamma, A \vdash B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \text{cut}$$

En effet, on peut la prouver en n'utilisant que les règles de la logique intuitionniste via cet arbre :

$$\frac{\dfrac{\dfrac{\text{supposé}}{\Gamma, A \vdash B}\ \text{intro-}\Rightarrow}{\Gamma \vdash A \Rightarrow B} \qquad \dfrac{\text{supposé}}{\Gamma \vdash A}}{\Gamma \vdash B}\ \text{elim-}\Rightarrow$$

1. a) On considère le système de déduction S_1 dont les règles sont celles de $\mathcal{R}$ auxquelles on rajoute le tiers exclus. Montrer que l'élimination de la double négation est dérivable dans S_1.

   b) On considère le système de déduction S_2 dont les règles sont celles de $\mathcal{R}$ auxquelles on rajoute l'élimination de la double négation. Montrer que le raisonnement par l'absurde y est dérivable.

   c) On considère le système de déduction S_3 dont les règles sont celles de $\mathcal{R}$ auxquelles on rajoute le raisonnement par l'absurde. Montrer qu'on peut y dériver le tiers exclus. On pourra dans un premier temps prouver en logique intuitionniste les séquents $\neg(A \vee B) \vdash \neg A$ et $\neg(A \vee B) \vdash \neg B$.

On dit que deux systèmes de déduction S et S' sont équivalents si les séquents prouvables dans S sont exactement les mêmes que les séquents prouvables dans S'.

2. Montrer que S_1 (qui correspond à la logique classique), S_2 et S_3 sont équivalents.

Ce résultat explique pourquoi on ajoute indifféremment soit le tiers exclus, soit l'élimination de la double négation, soit le raisonnement par l'absurde à la logique intuitionniste pour construire la logique classique : les trois options aboutissent au "même" système de déduction au sens où les trois systèmes obtenus permettent de prouver exactement les mêmes choses. Ainsi, pour chaque $i \in [\![1, 3]\!]$ il fait sens de dire que S_i "est" la logique classique. Il existe en fait d'autres façons de la construire :

3. Montrer que le système de déduction S_4 dont les règles sont celles de $\mathcal{R}$ auxquelles on ajoute la contraposition est équivalent à la logique classique. On expliquera précisément la démarche.

4. De même, montrer que le système de déduction S_5 dont les règles sont celles de $\mathcal{R}$ auxquelles on ajoute la règle de Peirce est équivalent à la logique classique.

5. Au vu des résultats précédents, expliquer cette citation attribuée à David Hilbert :

Priver le mathématicien du tiers exclus serait enlever son téléscope à l'astronome, son poing au boxeur.

1. a) La règle elim-$\neg\neg$ est dérivable dans S_1 car l'arbre ci-dessous n'utilise que des règles de S_1 :

$$\dfrac{\dfrac{}{\Gamma \vdash A \vee \neg A}\ \text{te} \qquad \dfrac{\dfrac{}{\Gamma, A \vdash A}\ \text{ax} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, \neg A \vdash \neg A}\ \text{ax} \qquad \dfrac{\dfrac{\text{supposé}}{\Gamma \vdash \neg\neg A}}{\Gamma, \neg A \vdash \neg\neg A}\ \text{aff}}{\Gamma, \neg A \vdash \bot}\ \text{elim-}\neg}{\Gamma, \neg A \vdash A}\ \text{elim-}\bot}{\Gamma \vdash A}\ \text{elim-}\vee$$

Pour trouver cette preuve, on peut tenir le raisonnement suivant : a priori on doit utiliser le tiers exclus donc introduire (probablement) $A \vee \neg A$. Notre objectif est d'obtenir A. Si on a $A \vee \neg A$, soit on a A, soit on a $\neg A$. Dans le premier cas, on a directement obtenu A. Dans le second, on se rappelle qu'on a une hypothèse : $\Gamma \vdash \neg\neg A$. Or, si on a $\neg A$ et $\neg\neg A$, notre intuition sémantique nous dit qu'on a une contradiction. Ce qui tombe bien car d'une contradiction, on peut déduire ce qu'on veut, par exemple... A !

b) Le raisonnement par l'absurde est dérivable dans S_2 puisque l'arbre ci-dessous n'utilise que les règles de la logique intuitionniste et l'élimination de la double négation :

$$\dfrac{\dfrac{\dfrac{\text{supposé}}{\Gamma, \neg A \vdash \bot}}{\Gamma \vdash \neg\neg A}\ \text{intro-}\neg}{\Gamma \vdash A}\ \text{elim-}\neg\neg$$

c) On suit les indications. L'arbre suivant est un arbre de preuve en logique intuitionniste :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(A \vee B), A \vdash A}\ \text{ax}}{\neg(A \vee B), A \vdash A \vee B}\ \text{intro-}\vee \qquad \dfrac{}{\neg(A \vee B), A \vdash \neg(A \vee B)}\ \text{ax}}{\neg(A \vee B), A \vdash \bot}\ \text{elim-}\neg}{\neg(A \vee B) \vdash \neg A}\ \text{intro-}\neg$$

Ceci prouve le séquent $\neg(A \vee B) \vdash \neg A$. Un raisonnement symétrique permet de prouver $\neg(A \vee B) \vdash \neg B$. En particulier, cela implique qu'on peut prouver en logique intuitionniste les séquents $\neg(A \vee \neg A) \vdash \neg A$ et $\neg(A \vee \neg A) \vdash \neg\neg A$ en spécialisant les résultats précédents avec $B = \neg A$.

Ainsi, le tiers exclus est dérivable dans S_3 via l'arbre suivant (qui n'utilise que les règles de S_3) :

$$\dfrac{\dfrac{\dfrac{\text{prouvé}}{\neg(A \vee \neg A) \vdash \neg A}}{\Gamma, \neg(A \vee \neg A) \vdash \neg A}\ \text{aff} \qquad \dfrac{\dfrac{\text{prouvé}}{\neg(A \vee \neg A) \vdash \neg\neg A}}{\Gamma, \neg(A \vee \neg A) \vdash \neg\neg A}\ \text{aff}}{\Gamma, \neg(A \vee \neg A) \vdash \bot}\ \text{elim-}\neg$$
$$\dfrac{}{\Gamma \vdash A \vee \neg A}\ \text{ra}$$

2. La question 1 montre qu'ajouter à $\mathcal{R}$ n'importe laquelle des règles parmi le tiers exclus, le raisonnement par l'absurde et l'élimination de la double négation permet de dériver les deux autres. Toute preuve dans un système $S \in \{S_1, S_2, S_3\}$ peut donc se réécrire en une preuve dans $S' \in \{S_1, S_2, S_3\}$. (démonstration par induction sur l'arbre de preuve : il suffit de remplacer les morceaux de l'arbre qui utilisent une règle absente dans le système S' par les morceaux d'arbre idoines n'utilisant que des règles de S', et on peut). Ainsi, les séquents prouvables dans S_1, S_2 et S_3 sont les mêmes.

3. Pour montrer que S_4 est équivalent à la logique classique, il suffit de montrer deux choses : qu'on peut, à partir de S_4, dériver l'une des règles parmi le tiers exclus, l'élimination de la double négation et le raisonnement par l'absurde et réciproquement, qu'on peut dériver la contraposition d'une de ces trois règles (pas forcément la même que pour la première dérivation). On choisit par exemple de montrer qu'on peut dériver la contraposition du raisonnement par l'absurde et le raisonnement par l'absurde de la contraposition :

L'arbre suivant n'utilise que les règles de $\mathcal{R}$ et le raisonnement par l'absurde, on en déduit qu'on peut dériver la contraposition de S_3 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{supposé}}{\Gamma \vdash \neg B \Rightarrow \neg A}}{\Gamma, A, \neg B \vdash \neg B \Rightarrow \neg A}\,\text{aff} \quad \dfrac{\dfrac{}{\Gamma, \neg B \vdash \neg B}\,\text{ax}}{\Gamma, A, \neg B \vdash \neg B}\,\text{aff}}{\Gamma, A, \neg B \vdash \neg A}\,\text{elim-}\Rightarrow \quad \dfrac{}{\Gamma, A, \neg B \vdash A}\,\text{ax}}{\Gamma, A, \neg B \vdash \bot}\,\text{elim-}\neg}{\Gamma, A \vdash B}\,\text{ra}}{\Gamma \vdash A \Rightarrow B}\,\text{intro-}\Rightarrow$$

Cette preuve est la formalisation du raisonnement sémantique suivant. On suppose qu'on a $\neg B \Rightarrow \neg A$ et on veut montrer $A \Rightarrow B$. Supposons donc A et montrons B. Par l'absurde, on a $\neg B$. Dans ce cas, comme $\neg B \Rightarrow \neg A$, on a $\neg A$, ce qui contredit A.

Réciproquement, l'arbre suivant n'utilise que les règles de $\mathcal{R}$ et la contraposition. On en déduit que le raisonnement par l'absurde est dérivable dans S_4 et ceci conclut notre preuve :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{supposé}}{\Gamma, \neg A \vdash \bot}}{\Gamma, \neg A, \bot \Rightarrow \bot \vdash \bot}\,\text{aff}}{\Gamma, \neg A \vdash \neg(\bot \Rightarrow \bot)}\,\text{intro-}\neg}{\Gamma \vdash \neg A \Rightarrow \neg(\bot \Rightarrow \bot)}\,\text{intro-}\Rightarrow}{\Gamma \vdash (\bot \Rightarrow \bot) \Rightarrow A}\,\text{ctr} \quad \dfrac{\dfrac{}{\Gamma, \bot \vdash \bot}\,\text{ax}}{\Gamma \vdash \bot \Rightarrow \bot}\,\text{intro-}\Rightarrow}{\Gamma \vdash A}\,\text{elim-}\Rightarrow$$

Cette preuve est la formalisation du raisonnement sémantique suivant. On suppose que $\neg A \Rightarrow \bot$ et on veut montrer A. Comme $\neg A \Rightarrow \neg \top$, par contraposition, l'implication $\top \Rightarrow A$ est vraie. Mais cette implication ne peut être vraie que si A l'est vu la sémantique standard de $\Rightarrow$. On s'aide du fait que $\bot \Rightarrow \bot$ est un raccourci pour $\top$.

4. La méthode est similaire à celle utilisée dans la question 4 : on montre que la loi de Peirce est dérivable du raisonnement par l'absurde puis que le raisonnement par l'absurde est dérivable de la loi de Peirce. Cela montre l'équivalence des systèmes S_5 et S_3 et conclut d'après la question 2.

   L'arbre ci-dessous montre qu'on peut dériver la loi de Peirce de S_3 (F désigne la formule $(A \Rightarrow B) \Rightarrow A$) :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, F \vdash (A \Rightarrow B) \Rightarrow A}\,\text{ax} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \neg A, A \vdash \neg A}\,\text{ax} \quad \dfrac{}{\Gamma, \neg A, A \vdash A}\,\text{ax}}{\Gamma, \neg A, A \vdash \bot}\,\text{elim-}\neg}{\Gamma, \neg A, A \vdash B}\,\text{elim-}\bot}{\Gamma, \neg A \vdash A \Rightarrow B}\,\text{intro-}\Rightarrow}{\Gamma, F, \neg A \vdash A}\,\text{elim-}\Rightarrow + \text{aff} \quad \dfrac{}{\Gamma, F, \neg A \vdash \neg A}\,\text{ax}}{\Gamma, F, \neg A \vdash \bot}\,\text{elim-}\neg}{\Gamma, F \vdash A}\,\text{ra}}{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}\,\text{intro-}\Rightarrow$$

   Cette preuve est la formalisation du raisonnement suivant. On veut montrer $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$. Pour ce faire, on suppose $(A \Rightarrow B) \Rightarrow A$ et on veut montrer A. Supposons par l'absurde que $\neg A$. Sémantiquement, A est faux, donc l'implication $A \Rightarrow B$ est vraie de part la sémantique de $\Rightarrow$. En utilisant le fait que $(A \Rightarrow B) \Rightarrow A$ par hypothèse on en déduit A et c'est une contradiction.

   Réciproquement, l'arbre ci-dessous montre qu'on peut dériver le raisonnement par l'absurde de S_5 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, \neg A \vdash \bot}\,\text{supposé}}{\Gamma \vdash \neg\neg A}\,\text{elim-}\bot}{\Gamma, \neg A \vdash A}\,\text{elim-}\neg}{\Gamma, \neg A \Rightarrow A}\,\text{intro-}\Rightarrow}{\Gamma, A \Rightarrow \bot \vdash \neg A \Rightarrow A}\,\text{aff} \quad \dfrac{\dfrac{}{A, A \Rightarrow \bot \vdash A \Rightarrow \bot}\,\text{ax} \quad \dfrac{}{A, A \Rightarrow \bot \vdash A}\,\text{ax}}{\dfrac{A, A \Rightarrow \bot \vdash \bot}{A \Rightarrow \bot \vdash \neg A}\,\text{intro-}\neg}\,\text{elim-}\Rightarrow}{\Gamma, A \Rightarrow \bot \vdash A}\,\text{aff}}{\Gamma \vdash (A \Rightarrow \bot) \Rightarrow A}\,\text{intro-}\Rightarrow \quad \dfrac{}{\Gamma \vdash ((A \Rightarrow \bot) \Rightarrow A) \Rightarrow A}\,\text{Peirce}}{\Gamma \vdash A}\,\text{elim-}\Rightarrow$$

Pour avoir une idée amenant à cette preuve, on peut tenir le raisonnement suivant. On veut montrer A sachant que $\Gamma, \neg A \vdash \bot$ et qu'a priori il faudra utiliser la loi de Peirce. En se rappelant que $\neg A \Leftrightarrow (A \Rightarrow \bot)$ est un théorème en logique intuitionniste, on suppute que l'instance de la loi de Peirce qu'il nous faudra utiliser est $((A \Rightarrow \bot) \Rightarrow A) \Rightarrow A$. Pour montrer A à partir de cette formule, il suffit de montrer $(A \Rightarrow \bot) \Rightarrow A$. Notre intuition sémantique nous dit qu'on devrait pouvoir y arriver puisque par hypothèse, $\neg A$ est faux, donc l'implication $(A \Rightarrow \bot)$ est fausse, et de faux, on peut déduire n'importe quoi, par exemple... A.

5. On comprend Hilbert ! En effet, les questions 1 à 4 montrent que s'interdire d'utiliser le tiers exclus dans ses raisonnements revient à s'interdire l'utilisation du raisonnement par l'absurde, de la contraposition, de l'élimination de la double négation et de la loi de Peirce. Si cette dernière n'est pas couramment utilisée en mathématiques, il va tout autrement des trois premiers raisonnements !

Exercice 209 (***) *Incomplétude de la logique intuitionniste*

On note $\mathcal{F}$ l'ensemble des formules propositionnelles sur un ensemble de variables propositionnelles $\mathcal{V} = \{x_1, ..., x_n\}$ et sur les connecteurs logiques $\bot$, $\wedge$, $\vee$, $\rightarrow$. On note $\neg A$ la formule propositionnelle $A \rightarrow \bot$. Pour tout séquent $\Gamma \vdash A$, on note $\Gamma \vdash_i A$ (respectivement, $\Gamma \vdash_c A$) si le séquent est prouvable en logique intuitionniste (respectivement, en logique classique). L'objectif de l'exercice est de montrer qu'il existe des séquents prouvables en logique classique qui ne le sont pas en logique intuitionniste.

On considère pour cela une sémantique particulière, dite *sémantique de Heyting*. Une valuation dans cette sémantique est définie comme étant une fonction $\mu : \mathcal{V} \rightarrow \mathcal{O}(\mathbb{R})$ où $\mathcal{O}(\mathbb{R})$ est l'ensemble des ouverts de $\mathbb{R}$. On étend la définition d'une valuation μ à $\mathcal{F}$ par induction via :

- $\mu(\bot) = \emptyset$.
- $\mu(A \wedge B) = \mu(A) \cap \mu(B)$.
- $\mu(A \vee B) = \mu(A) \cup \mu(B)$.
- $\mu(A \rightarrow B) = \overset{\circ}{\overbrace{\mu(A)^c \cup \mu(B)}}$ où X^c désigne le complémentaire de X et $\overset{\circ}{X}$ l'intérieur de X.

Pour tout $\Gamma \subset \mathcal{F}$, on note $\mu(\Gamma) = \bigcap_{A \in \Gamma} \mu(A)$ avec comme convention que $\mu(\emptyset) = \mathbb{R}$.

Un séquent $\Gamma \vdash A$ est dit valide si pour toute valuation μ, $\mu(\Gamma) \subset \mu(A)$. Une règle d'inférence est dite valide si, lorsque ses prémisses sont valides, alors sa conclusion est valide.

1. Quelle sémantique obtient-on si on considère des valuations à valeurs dans $\{\emptyset, \mathbb{R}\}$ plutôt que $\mathcal{O}(\mathbb{R})$?
2. Le séquent $\vdash ((A \vee B) \wedge \neg A) \rightarrow B$ est-il valide pour la sémantique de Heyting ? Justifier.
3. Montrer que $\vdash_c (A \rightarrow B) \rightarrow (\neg A \vee B)$.
4. Le séquent $\vdash (A \rightarrow B) \rightarrow (\neg A \vee B)$ est-il valide pour la sémantique de Heyting ? Justifier.
5. Montrer que la règle intro-$\rightarrow$ est valide. Faire de même pour la règle elim-$\rightarrow$.
6. On admet que les autres règles de la logique *intuitionniste* sont valides. En déduire que la logique intuitionniste est correcte pour la sémantique de Heyting.
7. Montrer que la règle du tiers exclus n'est pas valide. Que peut-on déduire concernant la complétude de la logique intuitionniste pour la sémantique booléenne standard ?

1. On retrouve la sémantique standard en assimilant $\emptyset$ à 0 et $\mathbb{R}$ à 1.
2. Ce séquent est valide. Notons F la conclusion du séquent. Montrer qu'il est valide revient à montrer

que pour toute valuation μ, $\mu(F) = \mathbb{R}$. Or :

$$\mu((A \vee B) \wedge \neg A) = (\mu(B) \cup \mu(A)) \cap \overbrace{\mu(A)^c \cup \mu(\bot)}^{\circ} = \mu(B) \cap \overbrace{\mu(A)^c}^{\circ}$$

car l'intérieur de $\mu(A)^c$ est inclus dans $\mu(A)^c$ donc est d'intersection vide avec $\mu(A)$. Donc :

$$\mu(((A \vee B) \wedge \neg A) \to B) = \overbrace{\left(\mu(B) \cap \overbrace{\mu(A)^c}^{\circ}\right)^c \cup \mu(B)}^{\circ} = \mathbb{R}$$

En effet, $\mu(B) \cap \overbrace{\mu(A)^c}^{\circ} \subset \mu(B)$ donc $\mu(B)^c \subset \left(\mu(B) \cap \overbrace{\mu(A)^c}^{\circ}\right)^c$, donc l'union de ce dernier ensemble avec $\mu(B)$ fait bien tout $\mathbb{R}$ (qui est égal à son intérieur).

3. Voici un arbre de preuve convenable (qui utilise le tiers exclus) :

$$\dfrac{\dfrac{}{A \to B \vdash A \vee \neg A}\ \text{TE} \qquad \dfrac{\dfrac{\dfrac{}{A \to B, A \vdash A}\ \text{ax} \quad \dfrac{}{A \to B, A \vdash A \to B}\ \text{ax}}{A \to B, A \vdash B}\ \text{elim-}\to}{A \to B, A \vdash \neg A \vee B}\ \text{intro-}\vee \qquad \dfrac{\dfrac{}{A \to B, \neg A \vdash \neg A}\ \text{ax}}{A \to B, \neg A \vdash \neg A \vee B}\ \text{intro-}\vee}{\dfrac{A \to B \vdash \neg A \vee B}{\vdash (A \to B) \to (\neg A \vee B)}\ \text{intro-}\to}\ \text{elim-}\vee$$

4. Ce séquent n'est pas valide. En effet, si on considère la formule $\varphi = (x \to y) \to (\neg x \vee y)$ et la valuation telle que $\mu(x) = \mu(y) = \mathbb{R}^\star$ alors :

   - $\mu(x \to y) = \overbrace{\mu(x)^c \cup \mu(y)}^{\circ} = \mathbb{R}$.
   - $\mu(\neg x \vee y) = \overbrace{\mu(x)^c}^{\circ} \cup \mu(y) = \overset{\circ}{\{0\}} \cup \mathbb{R}^\star = \mathbb{R}^\star$.
   - $\mu(\varphi) = \overbrace{\mu(x \to y)^c \cup \mu(\neg x \vee y)}^{\circ} = \overbrace{\emptyset \cup \mathbb{R}^\star}^{\circ} = \mathbb{R}^\star \neq \mathbb{R}$.

5. Pour la règle intro-$\to$, on suppose que $\Gamma, A \vdash B$ est valide et on veut montrer que $\Gamma \vdash A \to B$ l'est aussi. Soit donc μ une valuation. Par hypothèse, $\mu(\Gamma, A) \subset \mu(B)$, c'est-à-dire $\mu(\Gamma) \cap \mu(A) \subset \mu(B)$. Or, on peut partitionner $\mu(\Gamma)$ en $\mu(\Gamma) \cap \mu(A)$ d'une part et $\mu(\Gamma) \cap \mu(A)^c$ d'autre part. Comme :

   - $\mu(\Gamma) \cap \mu(A) \subset \mu(B) \subset \mu(A)^c \cup \mu(B)$ par hypothèse et,
   - $\mu(\Gamma) \cap \mu(A)^c \subset \mu(A)^c \subset \mu(A)^c \cup \mu(B)$,

   on a bien $\mu(\Gamma) \subset \mu(A)^c \cup \mu(B)$. De plus $\mu(\Gamma)$ est un ouvert donc égal à son intérieur et ainsi :

$$\mu(\Gamma) = \overset{\circ}{\mu(\Gamma)} \subset \overbrace{\mu(A)^c \cup \mu(B)}^{\circ} = \mu(A \to B)$$

   Pour la règle elim-$\to$, on suppose que $\Gamma \vdash A \to B$ et $\Gamma \vdash A$ sont valides et on veut montrer que $\Gamma \vdash B$ l'est. Soit donc une valuation μ. On a alors :

$$\begin{aligned}
\mu(\Gamma) &\subset \mu(A) \cap \overbrace{\mu(A)^c \cup \mu(B)}^{\circ} \text{ car } \mu(\Gamma) \text{ est inclus dans ces deux ensembles par hypothèse} \\
&= \overset{\circ}{\mu(A)} \cap \overbrace{\mu(A)^c \cup \mu(B)}^{\circ} \text{ car } \mu(A) \text{ est un ouvert} \\
&= \overbrace{\mu(A) \cap (\mu(A)^c \cup \mu(B))}^{\circ} \text{ car l'intérieur d'une intersection vaut l'intersection des intérieurs} \\
&= \overbrace{\mu(A)^c \cap \mu(B)}^{\circ} \subset \mu(B)
\end{aligned}$$

Donc le séquent $\Gamma \vdash B$ est bien valide.

6. Il s'agit de montrer que si $\Gamma \vdash_i A$ alors $\Gamma \vdash A$ est un séquent valide. On montre ce résultat par induction sur l'arbre de preuve du séquent $\Gamma \vdash A$. Si ce séquent est un axiome, ce séquent est valide par validité de la règle ax. Sinon, on sait par hypothèse inductive que les prémisses de la dernière règle R appliquée sont valides et par validité de R (montrée en question 5) sa conclusion l'est aussi.

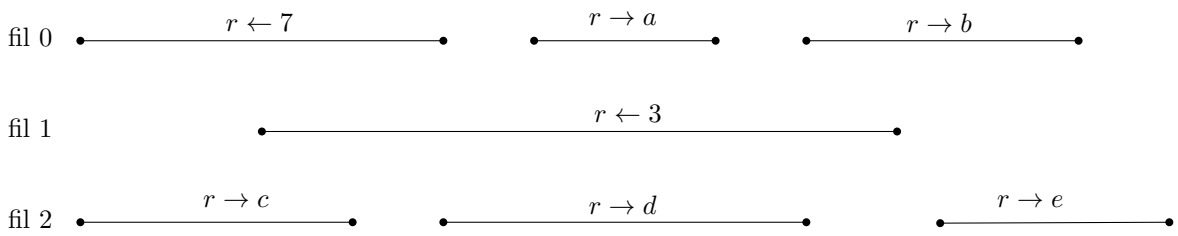
7. On note $A = x \vee \neg x$. Avec la valuation telle que $\mu(x) = \mathbb{R}^\star$, $\mu(A) = \mu(x) \cup \overbrace{\mu(x)^c}^{\circ} = \mathbb{R}^\star \cup \overset{\circ}{\{0\}} = \mathbb{R}^\star \neq \mathbb{R}$. On en déduit que la logique intuitionniste est incomplète pour la sémantique standard. En effet, elle est sémantique vraie, mais elle n'est pas prouvable en logique intuitionniste, sinon, la correction de la logique intuitionniste pour la sémantique de Heyting montrée à la question précédente assurerait que $\vdash x \vee \neg x$ est valide (vis à vis de la sémantique de Heyting) et ce n'est pas le cas.

11 Concurrence

Exercice 210 (*) *Entrelacement*

On considère trois fils partageant une variable entière r initialement nulle. Ces fils écrivent et lisent la variable r : on note $r \leftarrow x$ si le fil écrit x dans r et $r \rightarrow x$ si le fil lit la variable r pour la stocker dans x.

On suppose que chaque lecture ou écriture est atomique et prend place à un moment au cours d'un certain intervalle de temps (bornes incluses) ; ces derniers étant symbolisés sur le diagramme suivant :



1. Que peut valoir c ?
2. Que peut valoir a ?
3. Que peut valoir e ?
4. Si c vaut 7, que peut valoir d ?
5. Si d vaut 0, que peut valoir c ?
6. Si a vaut 3, que peut valoir e ?

1. 0, 3 ou 7.
2. 3 ou 7.
3. 3 ou 7 (car le fil 0 peut avoir écrit après le fil 1).
4. 3 ou 7.
5. Nécessairement 0 puisque cela signifie qu'aucune des deux écritures n'a encore pris place.
6. Nécessairement 3 car $a = 3$ signifie que l'écriture de 3 a pris place après l'écriture de 7.

Exercice 211 (**) *A en perdre le compte*

On considère dans un premier temps le pseudo code suivant d'une fonction `incremente` dans laquelle compteur est une variable globale partagée par tous les fils qui l'exécutent et qui initialement vaut 0 :

```
i ← 0
tant que i < 10
    i ← i + 1
    compteur ← compteur +1
```

On suppose que deux fils, notés A et B exécutent la fonction `incremente` en parallèle.

1. Montrer qu'en fin d'exécution le compteur peut valoir 20.

2. Montrer qu'il peut valoir 10.
3. Montrer qu'il peut valoir 2.

On modifie le code précédent de sorte à ce que la variable i soit une variable globale partagée par tous les fils (comme compteur) et on fait exécuter cette nouvelle version d'`incremente` en parallèle par deux fils.

4. Quelle est la plus petite valeur possible que peut contenir compteur en fin d'exécution ? Et la plus grande ? *Remarque : Venez me proposer vos réponses pour vérification, cette question est fourbe.*

1. On obtient 20 par exemple si le fil A fait tous ses incréments puis le fil B fait tous les siens.
2. On obtient 10 si le fil A lit le compteur (qui vaut 0), puis le fil B fait tous ses incréments, puis le fil A fait tous ses incréments (qui sont donc fait à partir de 0).
3. On obtient 2 avec la trace suivant : A lit le compteur (qui vaut 0), B fait 9 incréments, A écrit dans le compteur la valeur qu'il a lue plus 1, c'est-à-dire 1, B lit le compteur (qui vaut donc 1), A fait ses 9 incréments restants puis B écrit la valeur qu'il a lue dans le compteur (à savoir 1) plus 1.
4. Au minimum, on peut obtenir 1 avec cette trace : A lit $i = 0$ puis ajoute 1 à i puis lit le compteur (qui vaut 0). Ensuite, B fait toute son exécution ; en particulier, i vaudra 10. Ensuite, A écrit dans le compteur la valeur 1 (puisqu'il avait lu compteur = 0) et s'interrompt puisque $i < 10$ n'est plus vérifié.

   Au maximum, on peut obtenir 65 avec la trace suivante :

   - A lit $i = 0$
   - B fait 10 incréments et donc compteur vaut 10
   - A écrit $i = 1$
   - B lit $i = 1$
   - A fait 10 incréments et donc compteur vaut 20
   - B écrit $i = 2$
   - A lit $i = 2$
   - B fait 9 incréments et donc compteur vau 29
   - A écrit $i = 3$
   - B lit $i = 3$ ...

   In fine on aura compteur qui vaut $10 + 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 65$.

Exercice 212 (*) *Autour de l'algorithme de Peterson*

On considère les algorithmes suivants destinés à implémenter un mutex dans le cas où on a deux fils. On utilise pour ce faire un tableau `want` de booléens indiquant en case i si le fil i souhaite acquérir le verrou et d'une variable `turn` indiquant quel fil a la priorité.

```
create_lock ( ) =
  renvoyer {turn = 0, want = [false, false]}
```

```
lock(mutex, thread) =
  mutex.want[thread] ← true
  mutex.turn ← thread
  other ← 1-thread
  tant que mutex.turn = other et mutex.want[other] ( )
```

```
unlock(mutex, thread) =
  mutex.turn ← 1-thread
  mutex.want[thread] ← false
```

1. Montrer qu'un verrou ainsi implémenté ne vérifie pas l'exclusion mutuelle.

2. Cet algorithme est très similaire à l'algorithme de Peterson qui, lui, garantit l'exclusion mutuelle. Où se trouve la différence et pourquoi est-elle importante ?

Dans la suite, les fonctions `create_lock`, `lock` et `unlock` sont celles de l'algorithme de Peterson.

3. Dans chacun des cas suivants, justifier que l'affirmation est vraie ou fournir une trace d'exécution qui montre qu'elle ne l'est pas :

   (A) Si le fil T_0 rentre dans `lock` avant le fil T_1 (c'est-à-dire exécute la première instruction de `lock` avant que T_1 ne le fasse), alors T_0 obtiendra le verrou avant T_1.

   (B) Si T_0 exécute toutes les instructions qui précèdent la boucle tant que de `lock` avant que T_1 n'exécute la première instruction de `lock` alors T_0 obtiendra le verrou avant T_1.

   (C) Si T_0 et T_1 cherchent tous les deux à acquérir le verrou, alors aucun des deux fils ne peut l'obtenir deux fois de suite.

4. Que vous inspirent les réponses à la question précédente ?

1. La trace suivant provoque la présence des deux fils en section critique :
   - Le fil 0 écrit `true` dans `want`, écrit 0 dans `turn`, lit 0 dans `turn` donc entre en section critique.
   - Le fil 1 écrit `true` dans `want`, écrit `1` dans `turn`, lit 1 dans `turn` donc entre en section critique.
2. La différence avec l'algorithme de Peterson est la politesse : dans cet algorithme, quand un fil veut entrer en section critique, il laisse la priorité à l'autre fil ce qui n'est pas le cas ici.
3. (A) Faux avec la trace suivante : fil 0 écrit `true` dans `want[0]`, fil 1 écrit `true` dans `want[1]`, fil 1 écrit 0 dans `turn`, fil 0 écrit 1 dans `turn` et ainsi le fil 1 entre en section critique.

   (B) Vrai en dessinant le graphe le graphe de dépendance des instrcutions.

   (C) Vrai aussi. On est dans la situation où le fil 0 a écrit `true` dans `want[0]` et le fil 1 a fait de même dans `want[1]`. L'un des deux va modifier la variable `turn` en premier ; sans perte de généralité, c'est le fil 0 qui écrit donc 1 dans `turn`. Ensuite :
   - Le fil 0 ne peut donc plus progresser et donc c'est le fil 1 qui fera l'instruction suivante ; à savoir écrire 0 dans `turn`.
   - Le fil 1 ne peut alors plus progresser et donc l'instruction suivante sera faite par le fil 0, qui peut prendre le verrou et donc le fait. Le fil 1 ne peut toujours pas progresser, donc le fil 0 excéute la section critique puis en sort donc écrit `false` dans `want[0]`.
   - Si le fil 0 tente immédiatement de reprendre le verrou, il écrit donc `true` dans `want[0]` et 1 dans `turn` : il est alors bloqué et donc le fil 1 prend le verrou (et il peut).
4. L'algorithme de Peterson ne garantit pas un comportement "premier arrivé / premier servi" d'après (A), mais garantit tout de même qu'un fil ne peut pas monopoliser le verrou d'après (C).

Exercice 213 (**) *Interblocage et sémaphores*

On note P (= `acquire` = `wait`) la demande sur un sémaphore et V (= `release` = `post`) sa libération. Dans chacune des situations suivantes, dire si il est possible d'obtenir un (inter)-blocage :

1. a et b sont deux sémaphores dont les compteurs valent respectivement 1 et 0 au début. Deux fils exécutent de manière parallèle les instructions suivantes :
   - Fil 1 : P(a), V(a), P(b), V(b).
   - Fil 2 : P(a), V(a), P(b), V(b).
2. a,b et c sont trois sémaphores dont les compteurs valent initialement tous 1. Trois fils exécutent de manière parallèle les instructions suivantes :
   - Fil 1 : P(a), P(b), V(b), P(c), V(c), V(a).
   - Fil 2 : P(c), P(b), V(b), V(c), P(a), V(a).
   - Fil 3 : P(c), V(c), P(b), P(a), V(a), V(b).
3. a,b et c sont trois sémaphores dont les compteurs valent initialement tous 1. Deux fils exécutent de manière parallèle les instructions suivantes :

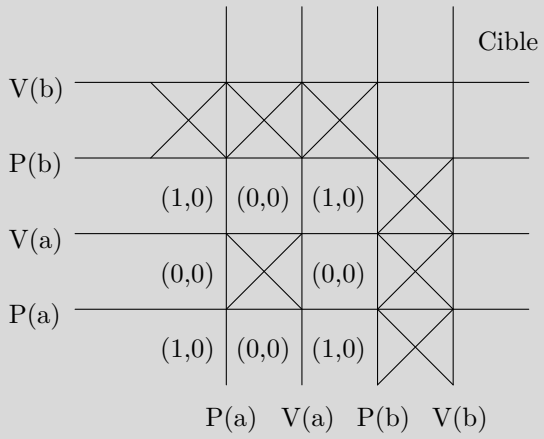
- Fil 1 : P(a), P(b), V(b), P(c), V(c), V(a).
- Fil 2 : P(c), P(b), V(b), V(c).

1. En fait, chacun des deux fils est bloqué au moment de faire P(b) puisque le compteur de ce sémaphore est à 0 et n'est jamais incrémenté avant que les fils n'essaient d'appeler P. On peut s'en convaincre en dessinant un diagramme de transition qui représente tous les entrelacments possibles dans un tableau bidimensionnel. En abscisses se touvent les actions du fil 1 et en ordonnées celles du fil 2. Dans chaque case se trouvent dans cet ordre les valeurs des compteurs des sémaphores a et b.

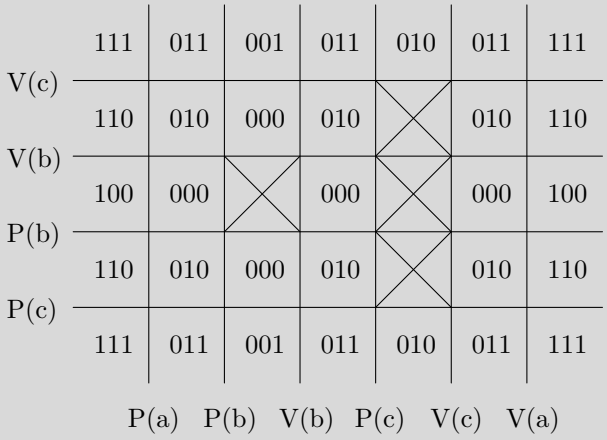
   Passer d'une case à sa voisine de droite symbolise l'évolution de ces compteurs après une action du fil 1 et passer d'une case à sa voisine du haut symbolise celle après une action du fil 2. Chaque chemin de la case en bas à gauche (= O = origine) à celle en haut à droite (= C = cible) en ne se déplaçant que vers la droite ou vers le haut symbolise un entrelacement possible. Alors :

   - Si depuis toute case il existe un chemin vers C alors il n'y a jamais d'interblocage.
   - S'il existe des cases accessibles depuis O pour lesquelles il est impossible d'atteindre C alors il est possible d'avoir un interblocage.
   - Si pour toute case accessible depuis O il est impossible d'atteindre C, alors toutes les exécutions mènent à un interblocage.

   Ici, on est dans ce troisième cas de figure d'après le diagramme de transition :



2. Ici il est possible d'avoir interblocage (mais ce n'est pas systématique) avec la trace suivante : fil 1 fait P(a), fil 3 fait P(c), V(c) et P(b), fil 2 fait V(c). Alors :
   - Le fil 1 est bloqué dans P(b), car le compteur de b est à 0 à cause de fil 3.
   - Le fil 2 est bloqué dans P(b) pour la même raison.
   - Le fil 3 est bloqué dans P(a), car le compteur de a est à 0 à cause du fil 1.
3. Il est impossible d'avoir un interblocage comme le montre le diagramme de transition suivant (dans l'ordre, on indique le compteur pour a, pour b et pour c) ; En effet depuis toute case, il existe un chemin menant vers C dans lequel on ne se déplace que vers le haut ou la droite :



Remarque : s'il est possible d'obtenir un interblocage, il n'y a en fait pas besoin de dessiner le diagramme de transition puisqu'exhiber un entrelacement problématique suffit (même si cela peut aider à le trouver). En revanche, quand il n'y a pas d'interblocage, le diragramme permet de le montrer en listant de manière exhaustive tous les entrelacements. Cette technique n'est en revanche applicable que lorsqu'on a deux fils.

Exercice 214 (**) *Rendez-vous*

On considère le problème du rendez-vous. C'est un problème classique de synchronisation dans lequel deux fils doivent, en un point de leur exécution, attendre avant de continuer que l'autre fils en soit lui-même arrivé à un certain point de son exécution. Plus formellement, les fils A et B exécutent :

```
Faire a_1
Faire a_2
```

Algorithme 2 - Code pour le fil A

```
Faire b_1
Faire b_2
```

Algorithme 3 - Code pour le fil B

et on souhaite garantir que b_2 ne peut arriver que après a_1 et que a_2 ne peut arriver que après b_1.

1. À l'aide de sémaphores modifier les codes des fils A et B de sorte à garantir la propriété souhaitée.

On souhaite généraliser la résolution de ce problème lorsqu'un nombre n de fils doivent se synchroniser au cours de leur exécution (donc pas une jointure puisque cette opération synchronise les fils après la fin de leur exécution). Pour cela on implémente une *barrière de synchronisation* c'est-à-dire deux fonctions :

- `creer`, qui prend en entrée un entier n et créé une barrière de synchronisation pour n fils.
- `rendez_vous`, qui prend en entrée une barrière de synchronisation et attend qu'un total de n fils aient fait appel à cette fonction pour autoriser les fils à poursuivre leur exécution. Autrement dit, les $n-1$ premiers fils qui exécutent cette fonction avec une même barrière B doivent être bloqués et ne seront autorisés à continuer leur calcul que lorsqu'un n-ème fil appellera `rendez_vous`(B).

On choisit d'implémenter cette barrière de synchronisation à l'aide d'un compteur (le nombre de fils qu'il faut encore attendre), d'un mutex pour protéger le compteur et d'un sémaphore. La fonction `creer` est :

```
creer(n) =
  renvoyer {compteur = n, m = creer_mutex ( ), s = creer_semaphore(0)}
```

Pour la fonction `rendez_vous` on propose les codes suivants :

```
Entrée : Une barrière B à n fils
Verrouiller B.m
B.compteur ← B.compteur −1
Déverrouiller B.m
si B.compteur = 0 alors
  | Signaler B.s
Attendre B.s
```

Algorithme 4 - Version gauche

```
Entrée : Une barrière B à n fils
Verrouiller B.m
B.compteur ← B.compteur −1
si B.compteur = 0 alors
  | Signaler B.s
Attendre B.s
Signaler B.s
Déverouiller B.m
```

Algorithme 5 - Version droite

2. Montrer que la version gauche est problématique.
3. Montrer que la version droite est aussi problématique.
4. En s'inspirant de ces versions erronées, implémenter une fonction `rendez_vous` correcte.
5. Avec votre version correcte de `rendez_vous`, après que les n fils aient franchi la barrière de synchronisation B, quelle est la valeur du compteur de $B.s$? Peut-on réutiliser cette barrière pour synchroniser nos n fils une seconde fois ?
6. (bonus) Proposer une façon d'obtenir une barrière réutilisable.

1. On note a_fini et b_fini deux sémaphores initialisés à 0. Alors on propose le code suivant pour le fil A (similaire pour le fil B en évhangeant a_fini et b_fini) :

```
1 faire a_1
2 signaler a_fini
3 attendre b_fini
4 faire a_2
```

   Les lignes 2 et 3 peuvent être interverties mais uniquement pour l'un des deux fils, autrement on pourrait avoir un interblocage.

2. La version gauche provoquera des interblocages. En effet, seul le dernier fil signale le sémaphore s et donc seul un fil pourra l'acquérir et poursuivre son exécution, les autres resteront bloqués à la barrière.
3. La version droite provoque un interblocage dès le premier fil. En effet, ce dernier prend le mutex, interdisant à tous les autres fils de le faire. Il est donc le seul à pouvoir poursuivre son exécution :
   - Il décrémente donc le compteur, et constate qu'il n'est pas nul puisque $n > 1$.
   - Puis il attend indéfiniment le sémaphore s puisqu'il est initialisé à 0 et qu'aucun fil ne peut l'incrémenter (ni lui, ni aucun autre puisque tous attendent en vain de pouvoir prendre le mutex).
4. Voici une version qui devrait être correcte :

```
1 Verrouiller B.m
2 B.compteur ← B.compteur −1
3 Déverrouiller B.m
4 si B.compteur = 0 alors
5 |  Signaler B.s
6 Attendre B.s
7 Signaler B.s
```

   Le test des lignes 4 et 5 pourrait être protégé par le mutex mais ce n'est pas nécessaire. Le principe est de ne protéger que la décrémentation par le mutex puis, dès que n fils sont arrivés, l'un d'entre eux signale s puis chaque fil qui décrémente s l'incrémente sitôt avoir réussi à prendre le sémaphore : c'est le principe d'un *tourniquet*.
5. À la fin du rendez-vous, le dernier fil à passer a signalé s et donc le compteur de ce sémaphore vaut 1 (au lieu de 0) : on ne peut donc pas réutiliser la barrière.
6. Une solution (cf Little Book of Semaphores p 41) est d'utiliser deux tourniquets T_1 et T_2. Au début, T_1 est fermé (compteur à 0) et T_2 est ouvert (compteur à 1). Le fil qui fait passer le nombre de fils à la barrière à n attend T_2 pour le fermer et signale T_1 ; puis tous les fils passent T_1. Derrière, on redécrémente le compteur, protégé par le mutex et le fil qui fait passer le compteur à 0 attend T_1 (qui passe donc à 0) et signale T_2 ; puis tous les fils passent T_2.

   En fin de ce code, T_1 a un compteur qui vaut 0 et T_2 un compteur qui vaut 1 et c'est bien la situation qu'on avait au début, ce qui permet de réutiliser la barrière :

```
Verrouiller B.m
B.compteur ← B.compteur +1
si B.compteur = n alors
    Attendre B.T_2
    Signaler B.T_1
Déverouiller B.m

Attendre B.T_1
Signaler B.T_1

Verrouiller B.m
B.compteur ← B.compteur -1
si B.compteur = 0 alors
    Attendre B.T_1
    Signaler B.T_2
Dévrouiller B.m

Attendre B.T_2
Signaler B.T_2
```

Exercice 215 (**) *Lecteurs-rédacteurs*

Le problème des lecteurs-rédacteurs est le suivant. On dispose d'une base de données à laquelle deux types d'utilisateurs ont accès. Les lecteurs ne font que lire des informations dans la base. Les rédacteurs peuvent lire ou écrire dans la base. On impose les contraintes suivantes :

- Plusieurs lecteurs peuvent consulter la base en même temps.
- Si un rédacteur est en train d'utiliser la base, aucun autre utilisateur (ni lecteur, ni un autre rédacteur) ne peut l'utiliser en même temps.

On propose d'utiliser le code suivant pour régir le comportement des lecteurs et des rédacteurs : tous partagent deux sémaphores `lecteur` et `redacteur` initialisés à 1 et une variable `nb_lecteurs` valant initialement 0. Puis :

```
1  Attendre(lecteur)
2  nb_lecteurs ← nb_lecteurs +1
3  si nb_lecteurs = 1 alors
4  |  Attendre(redacteur)
5  Libérer(lecteur)
6  Faire des lectures
7  Attendre(lecteur)
8  nb_lecteurs ← nb_lecteurs −1
9  si nb_lecteurs = 0 alors
10 |  Libérer(redacteur)
11 Libérer(lecteur)
```

Algorithme 6 - Code pour un lecteur

```
1 Attendre(redacteur)
2 Faire des lecteures et écritures
3 Libérer(redacteur)
```

Algorithme 7 - Code pour un rédacteur

1. Justifier que lorsqu'un rédacteur accède à la base il en a l'accès exclusif.
2. Justifier que plusieurs lecteurs peuvent accéder à la base en même temps.
3. Le comportement des fils serait-il impacté si on utilisait un mutex plutôt qu'un sémaphore binaire pour `lecteur` ? Même question pour `redacteur`.
4. Montrer qu'avec ce code il peut y avoir famine pour les rédacteurs.
5. Écrire un pseudo-code corrigeant ce problème de famine en utilisant un sémaphore supplémentaire.

1. On remarque que le compteur de `redacteur` varie entre 0 et 1. Ainsi, si un rédacteur parvient à prendre le sémaphore `redacteur`, il ne peut pas y avoir d'autre rédacteur dans la base. De plus, il ne peut pas y avoir de lecteur non plus car on est dans un des cas suivants :

- Aucun lecteur n'est dans la base et dans ce cas les lecteurs ne peuvent pas avoir accès à la base car le premier d'entre eux doit aussi prendre `redacteur` ; or son compteur est à 0.
- Il y avait un lecteur dans la base au moment de la prise de `redacteur`, mais c'est impossible car le premier d'entre eux a pris `redacteur` et le compteur de ce sémaphore n'a pas été incrémenté puisque seul le dernier lecteur quittant la base peut le faire : le rédacteur n'aurait donc pas pu accéder à la base et c'est contradictoire.

2. C'est du au fait que les lecteurs ne monoplisent `lecteur` que le temps de modifier le nombre de lecteurs.
3. Pour `lecteur`, on peut le remplacer par un mutex (même si on rappelle qu'un mutex et un sémaphore initialisé à 1 se comportent différemment). Pour `redacteur` ce n'est pas possible car le fil qui libère `redacteur` dans le code des lecteurs n'est pas forcément celui qui l'avait pris. Ainsi, on ne pourrait plus avoir plusieurs lecteurs dans la base en même temps.
4. On pourrait avoir deux lecteurs qui alternent sans jamais laisser le rédacteur décrément `redacteur`.
5. Une solution est d'ajouter une sémaphore `prio` initialisé à 1 et modifier le code ainsi (à gauche pour les lecteurs, à droite pour les rédacteurs) :

Attendre prio Bloc entre les lignes 1 et 5 Signaler prio Faire les lectures Bloc entre les lignes 7 et 11	Attendre prio Attendre redacteur Signaler prio Faire les écritures Signaler redacteur

De cette façon, dès qu'un rédacteur annonce son intention d'écrire, il pourra accéder à la base une fois que tous les lecteurs qui y sont au moment du souhait en seront sortis. En effet, les lecteurs qui souhaitent lire après ce souhait ne pourront accéder à la base qu'après le rédacteur ; à condition que le sémaphore `prio` ait un comportement premier arrivé - premier servi.

Exercice 216 (exo cours) *Concurrence bancaire*

Deux agences d'une banque, l'une à Nancy, l'autre à Tours veulent mettre à jour le même compte bancaire. Pour ce faire, l'agence de Nancy effectue les opérations suivantes :

1. courant ← récupérer le montant sur le compte $18A3$
2. nouveau ← courant $+1000$
3. montant sur le compte $18A3$ ← nouveau

L'agence de Tours quant à elle effectue les opérations suivantes :

A. courant ← récupérer le montant sur le compte $18A3$
B. nouveau ← courant -1000
C. montant sur le compte $18A3$ ← nouveau

1. Montrer qu'il est possible que le montant sur le compte $18A3$ ait augmenté ou diminué de 1000.
2. Si on suppose que l'agence de Nancy commence en premier à exécuter son code, quel est le montant sur le compte $18A3$ à la fin des deux transactions ?
3. Comment garantir qu'à la suite de ces deux transactions le montant sur le compte n'ait pas changé ?

1. La trace d'exécution A-1-B-C-2-3 augmente le montant de 1000 et A-1-2-3-B-C le fait diminuer.
2. Le même qu'au début, augmenté de 1000 ou diminué de 1000 : on ne peut pas savoir.
3. Il faut protéger l'intégralité des opérations par un mutex commun.

Exercice 217 (exo cours) *Algorithme de Peterson*

1. Rappeler les trois primitives dont on dispose pour un mutex.
2. Donner le pseudo-code de ces primitives dans le cadre de l'algorithme de Peterson.
3. Montrer que cet algorithme implémente un mutex pour lequel il n'y a jamais famine.

1. Création, verrouilllage, déverrouillage.
2. want indique quels fils veulent aller en section critique et prio quel fil a la priorité pour y entrer.

```
creer () = {want = [faux, faux], prio = 0}
```

```
deverouiller (m,fil) = m.want[fil] ← faux
```

```
verrouiller (m, fil) =
m.want[fil] ← vrai
autre ← 1-fil
prio ← autre
tant que m.want[autre] et prio = autre
  | attendre
```

3. Supposons par l'absurde et sans perte de généralité que le fil 0 soit en famine. Cela signifie qu'on a indéfiniment want[1] = vrai et prio = 1. Une fois que le fil 0 est indéfiniment bloqué :
   - Fil 1 ne commence plus d'appel à verrouiller sinon il fixe prio à 0 et cette valeur ne sera pas modifiée d'ici à la prochaine lecture de prio par le fil 0 qui entrera alors en section critique.
   - Fil 1 ne peut pas terminer un appel à verrouiller sinon après avoir exécuté la section critique il fixerait want[1] à faux et comme il ne fait ensuite plus d'appel à verrouiller d'après ce qui précède, cette valeur ne serait plus modifiée, libérant fil 0.
   - Fil 1 a commencé un appel à verrouiller avant que le fil 0 soit bloqué sinon on aurait want[1] = faux et le fil 0 ne serait pas bloqué.

   On en déduit que fil 1 est bloqué dans l'appel à verrouiller et donc prio = 0 ce qui est faux.

Exercice 218 (exo cours) *Algorithme de Lamport*

1. Rappeler en pseudo-code ou en langage naturel l'algorithme de la boulangerie de Lamport.
2. Montrer que cet algorithme garantit l'exclusion mutuelle.

1. n est le nombre de fils :

```
creer (n) =
{want = [faux,...,faux], ticket = [0,...,0]}
```

```
deverouiller (m,t) =
m.want[t] ← faux
```

```
verrouiller (m,t) =
m.want[t] ← vrai
m.ticket[t] ← max(ticket) +1
tant que il existe j tel que m.want[j] et
 (ticket[j], j) <lex (ticket[i], i)
 | attendre
```

2. Pour tout fil i, ticket[i] croît au fil du temps. Par l'absurde les fils i et j sont tous les deux en section critique. Sans perte de généralité, on a ($\star$) : (ticket[i], i) $<_{\text{lex}}$ (ticket[j], j) à cet instant. Comme j est entré en section critique alors que i y était on a soit :
   - (ticket[j], j) $<_{\text{lex}}$ (ticket[i], i) mais c'est impossible car j a soit lu la bonne valeur de ticket[i] et ($\star$) donne une contradiction, soit une valeur ancienne qui est encore plus petite.
   - want[i] = faux donc j était déjà en section d'attente quand i a commencé à calculer son ticket donc ticket[i] $\geq$ ticket[j] +1 ce qui contredit ($\star$).

Exercice 219 (exo cours) *Trace d'exécution*

On considère les fonctions suivantes ; x et y étant deux variables globales initialement nulles :

```
1 f1( ) =
2 si x = 0 alors
3 |   x ← x + 1
4 |   y ← y + 1
```

```
1 f2( ) =
2 si x = 0 alors
3 |   x ← x + 1
4 |   y ← y + 2
```

L'instruction $t \leftarrow t + a$ où $a \in \mathbb{N}$ consiste à lire la valeur de t et la stocker dans une variable v locale au fil, ajouter a à v et écraser t avec cette valeur. On suppose que f_1 et f_2 sont exécutées de manière concurrente par deux fils. En fin d'exécuction, est-il possible...

1. ... que $x = 0$?
2. ... que $x = 2$?
3. ... que $(x, y) = (1, 3)$?

1. Non car l'un des deux fils au moins incrémente x et il n'est jamais décrémenté.
2. Oui avec la trace (fil 1 lit x = 0), (fil 2 lit x = 0), (fil 1 stocke v = 0), (fil 1 écrit v+1 = 1 dans x), (fil 2 lit 1 dans x), (fil 2 stocke v = 1), (fil 2 écrit v+1 = 2 dans x).
3. Oui avec la trace (fil 1 lit 0 dans x), (fil 2 lit 0 dans x), (fil 1 stocke v = 0), (fil 2 stocke v = 0), (fil 1 écrit 1 dans x), (fil 2 écrit 1 dans x), (fil 1 incrémente y), (fil 2 ajoute 2 à y).

Exercice 220 (exo cours) *Excès de politesse*

On cherche à implémenter un mutex dans le cadre de deux fils d'exécution (0 et 1) à l'aide d'un tableau de booléens indiquant en case i si le fil numéro i souhaite entrer en section critique. On propose d'utiliser l'implémentation suivante :

```
1 creer () =
2 renvoyer [faux, faux]
```

```
1 deverrouiller (m,t) =
2 m[t] ← faux
```

```
1 verrouiller (m,t) =
2 m[t] ← vrai
3 tant que m[1-t]
4 |   Attendre
```

1. Montrer que cette implémentation ne respecte pas l'absence d'interblocage.
2. Cette implémentation respecte-t-elle l'exclusion mutuelle ? Justifier votre réponse.

1. La trace suivante conduit à un interblocage : (fil 0 écrit vrai dans want[0]), (fil 1 écrit vrai dans want[1]). Aucun des deux fils ne peut alors progresser.
2. Oui. Supposons par l'absurde que les deux fils sont en section critique. Alors le fil 0 a écrit vrai dans want[0] (1) puis lu faux dans m[1] (2) et de même le fil 1 a écrit vrai dans want[1] (3) puis lu faux dans m[0] (4). On doit avoir (1) qui se passe avant (2), (3) qui se passe avant (4), (2) qui se passe avant (3) et (4) qui se passe avant (1). Donc (4) se passe avant (4) : absurde.

Exercice 221 (**) *Organisation de tâches à l'aide de sémaphores*

On considère l'exécution parallèle de trois processus P1, P2, P3 ayant les caractéristiques suivantes :

- P1 exécute en boucle les tâches A puis B.
- P2 exécute en boucle les tâches U puis V.
- P3 exécute en boucle la tâche X.

De plus, les contraintes suivantes doivent être respectées :

(1) Les tâches B et U sont en exclusion mutuelle.

(2) La tâche A produit un élément (qui sera consommé) nécessaire à la complétion de la tâche X.

1. On note dA_i le début de la i-ème occurrence de la tâche A et fA_i sa fin (similairement pour les autres tâches). Les traces d'exécution suivantes sont elles possibles ? Si non, indiquer tous les problèmes.
   a) $dA_1, fA_1, dX_1, dB_1, dU_1, fX_1, fU_1, fB_1, dA_2, dX_2, dV_1, fV_1, fX_2, fA_2, dU_2, dB_2, fU_2, fB_2$.
   b) $dA_1, fA_1, dX_1, dB_1, fB_1, dA_2, dU_1, fA_2, fX_1, fU_1, dX_2, dV_1, fV_1, fX_2, dU_2, fU_2, dB_2, fB_2$.
2. Expliquer comment utiliser un sémaphore pour garantir la contrainte (1).
3. Expliquer comment utiliser un sémaphore pour garantir la contrainte (2).
4. Peut-on utiliser le même sémaphore pour garantir en même temps les conditions (1) et (2) ? Justifier.

1. La première est impossible : U et B ne sont pas en exclusion mutuelle et dX_2 arrive avant fA_2 ce qui contredit (2). La seconde est possible.
2. On introduit un sémaphore S_{BU} initialisé à 1. B et U doivent commencer par acquérir S_{BU} et terminer en le relâchant (ce sémaphore est utilisé comme un mutex).
3. On initialise un sémaphore S_{AX} à 0. A termine en relâchant S_{AX} et X commence en prenant S_{AX}.
4. Non. Par l'absurde, utilisons le même sémaphore S pour les deux. Si sa valeur initiale est ≥ 1, X pourrait s'exécuter immédiatement ce qui contredit (2). Si elle est nulle, le seul processus qui peut s'exécuter est P1 qui incrémente S à la fin de A. Si à cet instant P3 prend la main, X peut s'exécuter en bloquant au passage B et U. Puis on arrive à un interblocage : P3 ne peut pas faire X car il faudrait que P1 fasse A mais avant P1 doit faire B qui ne peut pas s'exécuter.

Exercice 222 (*) *Implémentation d'un sémaphore*

On souhaite implémenter un sémaphore en OCaml à l'aide d'un mutex. On rappelle que les primitives sur un tel objet sont implémentées respectivement par `Mutex.create`, `Mutex.lock` et `Mutex.unlock`. On propose d'utiliser le type suivant :

```
type semaphore = {mutable compteur : int; mutex : Mutex.t}
```

1. A priori, à quoi servira le mutex dans cette implémentation ?
2. Écrire une fonction `creer : int -> semaphore` prenant en argument un entier naturel k et renvoyant un sémaphore dont le compteur est initialisé à k.

Dans notre implémentation, on procède naïvement en faisant attendre activement les fils : un fil qui souhaite acquérir un sémaphore dont le compteur est négatif tourne en boucle. Cela implique que lors d'une incrémentation du compteur, il n'est pas nécessaire de réveiller un fil en attente de décrémentation.

3. Écrire une fonction `liberer : semaphore -> unit` qui relâche un sémaphore.
4. Écrire une fonction `acquerir : semaphore -> unit` qui tente d'accéder à un sémaphore. On attendra activement que le compteur soit positif pour pouvoir décrementer le compteur.

1. Il servira à protéger les modifications du compteur du sémaphore (en particulier les décrémentations).
2.
```
let creer (k:int) :semaphore =
    {compteur = k; mutex = Mutex.create ()}
```
3. Il suffit d'incrémenter le compteur après l'avoir verrouillé :
```
let liberer (sem:semaphore) :unit =
    Mutex.lock sem.mutex;
    sem.compteur <- sem.compteur + 1;
    Mutex.unlock sem.mutex
```
4. Le compteur peut être redevenu négatif entre le moment où il a été testé positif et le moment où on souhaite le décrémenter. Donc on vérifie qu'on peut toujours décrémenter après prise du mutex.
```
let rec attendre (sem:semaphore) : unit =
    if sem.compteur <= 0 then attendre sem;
    Mutex.lock sem.mutex;
    if sem.compteur > 0 then begin
        sem.compteur <- sem.compteur - 1;
        Mutex.unlock sem.mutex
    end else begin
        Mutex.unlock sem.mutex;
        attendre sem
    end
```
Si l'élève propose de commencer par lock le mutex, on lui propose la première ligne du code et on lui

demande comment compléter. On peut aussi demander des défauts de cette implémentation.

Exercice 223 (*) *Interrupteur*

Un interrupteur est un objet de synchornisation basé sur le principe suivant : lorsque quelqu'un est le premier à entrer dans une pièce, il allume la lumière et lorsque quelqu'un est le dernier à sortir, il éteint la lumière. On veut implémenter les primitives suivantes :

- `creer(s)` créé un interrupteur associé à un sémaphore `s`.
- `entrer(I)` permet d'entrer dans la pièce en réquisitionnant `s` si on est le premier à entrer.
- `sortir(I)` permet de sortir de la pièce en libérant `s` si on est le dernier à partir.

Le sémaphore `s`, initialisé à 1, permettra ainsi de savoir si la lumière est allumée ou non. On utilisera par ailleurs un compteur correctement protégé pour garder trace du nombre de personnes dans la pièce.

1. Expliquer pourquoi un sémaphore initialisé à 1 n'est pas équivalent à un mutex.
2. Proposer une implémentation des trois primitives requises en pseudo-code.

1. Un sémaphore peut être libéré par un autre fil que le dernier qui l'a acquis.
2. `creer(s)` renvoie un enregistrement contenant un champ compteur initialement nul, un champ m contenant un mutex pour protéger compteur et un champ s contenant `s`. Puis :

```
entrer(I) =
Verrouiller I.m
si I.compteur = 0 alors
| Attendre I.s
I.compteur ← I.compteur +1
Déverouiller I.m
```

```
sortir(I) =
Verrouiller I.m
I.compteur ← I.compteur -1
si I.compteur = 0 alors
| Libérer I.s
Déverouiller I.m
```

On peut demander à l'élève comment utiliser un interrupteur pour implémenter un système lecteurs-rédacteurs (en rappelant ce problème) et poursuivre à l'oral sur ce thème.

Exercice 224 (***) *L'échoppe du barbier*

La boutique du barbier est constituée d'une salle d'attente comportant n chaises et du salon, où le barbier rase les gens un par un. Lorsque le barbier a fini de raser un client, il fait entrer le client suivant dans le salon ; sinon, il fait la sieste dans la salle d'attente. Si un client trouve le barbier endormi, il le réveille. Sinon, s'il reste de la place dans la salle d'attente, il s'installe et si elle est pleine, il rentre chez lui.

On considère que :

- Le barbier rase un client via la fonction `raser ()`.
- Un client se fait raser par le barbier via la fonction `se_faire_raser ()`.
- Un client quitte la boutique via la fonction `partir ()`.

1. Écrire le code d'un client et celui du barbier sans se préoccuper des problèmes de synchronisation.
2. Montrer qu'il est possible d'avoir strictement plus de clients dans la boutique que ce qu'elle peut en contenir au maximum avec cette implémentation. Proposer une correction.
3. À l'aide de sémaphores dont on explicitera la signification, modifier les codes de sorte à garantir que lorsque le barbier appelle `raser ()` il y ait exactement un client qui appelle `se_faire_raser ()` de manière concurrente.
4. Pourquoi n'y a-t-il pas besoin d'introduire une fonction `faire_sieste ()` pour le barbier ?

1. On se donne une variable globale `nb_clients` qui compte le nombre de client dans la boutique (et qui peut valoir au plus $n+1$: n en attente et 1 qui se fait raser). Puis le code du barbier consiste à appeler `raser ()` en boucle. Le code d'un client est :

```
si nb_clients = n + 1 alors
  | partir ()
sinon
  | nb_clients ← nb_clients +1
se_faire_raser ()
nb_clients ← nb_clients -1
```

2. L'incrémentation de `nb_clients` n'est pas atomique : deux clients pourraient constater qu'il n'y a que n clients dans la boutique et entrer tous les deux en salle d'attente. On protège les modifications de `nb_clients` derrière le mutex `m`.
3. On introduit un sémaphore `barbier` et un sémaphore `client` : le barbier attend `client` et quand un client arrive, il attend `barbier` (pour qu'il soit disponible). De même, on introduit `client_fini` et `barbier_fini` qui permettent de synchroniser le départ d'un client. Tous sont initialisés à 0. Puis :

```
Attendre client
Signaler barbier
raser ()
Signaler barbier_fini
Attendre client_fini
```

Code du barbier

```
Verrouiller m
si nb_clients = n + 1 alors
  | Déverouiller m
  | partir ()
sinon
  | nb_clients ← nb_clients +1
  | Déverouiller m

Signaler client
Attendre barbier
se_faire_raser ()
Attendre barbier_fini
Signaler client_fini

Verrouiller m
nb_clients ← nb_clients -1
Déverouiller m
```

Code d'un client

4. Faire la sieste consiste à attendre le sémaphore `client` : lorsque ce dernier est incrémenté par un client, le barbier est réveillé d'après le fonctionnement d'un sémaphore.