Fondements de l'informatique. Examen
Durée: 3h

Sujet proposé par Olivier Bournez
Version 6
(corrigé)

Les 4 parties sont indépendantes, et peuvent être traitées dans un ordre quelconque. On pourra admettre le résultat d'une question pour passer aux questions suivantes. On pourra utiliser tous les résultats et les théorèmes démontrés ou énoncés en cours ou en petite classe ou dans le polycopié sans chercher à les redémontrer.

Il est possible d'avoir la note maximale sans répondre à toutes les questions. La difficulté des questions n'est pas une fonction linéaire ni croissante de leur numérotation.

La ***qualité et clarté de votre argumentation et de votre rédaction*** *sera une partie importante de votre évaluation.*

1 Décidabilité

On considère des machines de Turing qui travaillent sur un alphabet qui contient au moins les chiffres $0, 1, \ldots, 9$, et les lettres de l'alphabet latin, et le symbole _.

Question 1. *Est-ce que l'on peut décider si une machine de Turing M accepte le mot* INF412 *et rejette le mot* CSC_41012_EP *?*

Solution : C'est indécidable par le Théorème de Rice : il y a au moins une machine qui le fait, et au moins une machine qui ne le fait pas, et donc la propriété est non-triviale. □

Question 2. *Est-ce que l'on peut décider si une machine de Turing M termine et refuse sur le mot* CSC_41012_EP *en 41012 étapes, mais accepte et termine toujours à sa 412ième étape sur toute autre entrée ?*

Solution : C'est décidable car il suffit de la simuler. On observera qu'en 412 étapes, la machine ne peut pas lire de toute façon plus que 412 cases de son ruban, et donc qu'il suffit de la simuler sur tous les rubans de taille au plus 412 pour savoir si on est bien dans le second cas. □

Question 3. *On se donne une machine de Turing M qui termine sur toutes ses entrées. Peut-on décider si M termine en temps polynomial sur toutes ses entrées ?*

Solution : C'est une question très similaire à celle de l'examen de 2023, et donc on peut reprendre et adapter la preuve : C'est indécidable. On peut par exemple utiliser le théorème du point fixe pour considérer une machine qui connait son propre code. Si on suppose que l'on sait décider la propriété par une machine de Turing D, la machine appelle D sur son propre code, et si D lui dit qu'elle accepte toutes ses entrées en temps polynomial, alors elle prend un temps 2^n mais fini sur les mots de la forme (disons) 0^n, et sinon, elle accepte immédiatement (et donc en temps polynomial, puisque constant). On obtient alors une contradiction, et donc la machine de Turing D ne peut pas exister.

Voici une preuve alternative. On peut construire une réduction à partir du complément du problème de l'arrêt. Étant donnés une machine M et un mot w, on construit une nouvelle machine M' qui prend en entrée un mot 0^n et simule M pendant n étapes sur w. Si M ne s'arrête pas dans ces n étapes, alors M' accepte. Autrement, M' continue à s'exécuter pendant encore 2^n étapes. Par construction, la machine M' termine sur toutes ses entrées, et M ne s'arrête pas sur w si et seulement si M' termine en temps polynomial sur toutes ses entrées.

□

Question 4. *Le problème de décision de la question précédente est-il récursivement énumérable ?*

Solution : Le complément du problème de l'arrêt n'est pas récursivement énumérable, et la seconde solution de la question précédente produit une réduction à partir de lui vers ce problème.

□

2 Réseaux de neurones à seuil

On note $\mathbf{B}$ pour $\mathbf{B} = \{0, 1\}$: l'intuition est que 0 représente la valeur logique *faux*, et que 1 représente la valeur logique *vrai*. On considère des réseaux de neurones dont la fonction de transition est la *fonction de Heaviside*, c'est-à-dire la fonction $\mathcal{H}(x)$ qui vaut 1 pour $x \geq 0$, 0 sinon.

Un *neurone à n entrées* est décrit par des paramètres $w_1, w_2, \ldots, w_n$ et $h \in \mathbb{Q}$. Chacun des w_i est appelé un *poids*, et h est appelé le *seuil* du neurone. On dit que les poids sont unitaires [1] si $w_1, w_2, \ldots, w_n$ restent dans $\{-1, 0, 1\}$. Un tel neurone calcule une fonction de $\mathbb{Q}^n$ dans $\mathbb{Q}$ de la façon suivante : sur les entrées $x_1, x_2, \ldots, x_n$, il prend la valeur $\mathcal{H}(w_1x_1 + w_2x_2 + \cdots + w_nx_n - h)$. Cette valeur s'appelle *sa valeur d'activation*. On dit qu'une fonction $F : \mathbf{B}^n \to \mathbf{B}$ est une *fonction booléenne linéaire à seuil* si elle correspond à la fonction calculée par un neurone à seuil : dit autrement, il existe des rationnels $w_1, \ldots, w_n$ et h tels que pour tout $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, on a $F(\mathbf{x}) = 1$ si et seulement $\sum_{i=1}^{n} w_i x_i \geq h$.

Par exemple, la fonction ET logique (à n-arguments) que l'on peut nommer $\mathrm{AND}_n(x_1, \ldots, x_n)$, qui vaut 1 si et seulement si chacun des x_i vaut 1, est une fonction booléenne linéaire à seuil (et même à poids unitaires) car $\mathrm{AND}_n(x_1, \ldots, x_n) = 1$ si et seulement si $1x_1 + 1x_2 + \cdots + 1x_n \geq n$.

Question 5. *Montrer que le* OU logique *(à n-arguments)* OR_n*, la* NEGATION logique NOT *(à un argument) sont également des fonctions booléennes linéaires à seuil.*

Solution : On a que $(\underbrace{1, \ldots, 1}_{n}, 1)$, $(-1, 0)$ sont des représentations de ces fonctions : on liste ici les poids, puis le seuil. □

On obtient un réseau de neurones en combinant des neurones en couches comme on s'y attend [2].

1. On n'impose rien sur le seuil. Il peut être quelconque.

2. Si cela aide, formellement : l'architecture d'un réseau est donné par un graphe $G = (V, E)$, où les sommets V se partitionnent en $V_0, V_1, \ldots, V_d$, et avec chaque arête (u, v) de la forme $u \in V_k, v \in V_{k+1}$ pour un certain entier k. Les n sommets de V_0 sont les entrées. Les m sommets de V_d sont les sorties. On associe à chaque sommet n de V_k, pour $1 \leq k < d$, un neurone, avec autant d'entrées que son degré entrant. Un tel réseau calcule une fonction de $\mathbf{B}^n$ dans $\mathbf{B}^m$ en propageant les valeurs des entrées vers les sorties : si on note F_n pour la fonction associée au neurone/sommet n, sa valeur est l'image par F_n des valeurs d'activation de ses entrées, tout cela calculé récursivement des entrées vers les sorties. d est appelé la profondeur du réseau de neurones.

2.1 Sans fixer l'architecture

Question 6. *Montrer que le problème de décision suivant est NP-complet : on se donne un réseau de neurones avec 1 sortie. S'il possède n entrées, on veut savoir si cette sortie est* 1 *sur au moins un élément de* $\mathbf{B}^n$.

Solution : C'est dans NP : la donnée des entrées booléennes constitue un certificat vérifiable en temps polynomial.

Le problème SAT se réduit à se problème, car on peut transformer n'importe quelle formule SAT en un réseau de neurones, en composant les réseaux de la question précédente, pour calculer les $\vee$,$\wedge$ et $\neg$ de la formule. □

Dans la question précédente, on ne cherchait pas à fixer l'architecture. On va montrer que cela reste vrai, même si l'on fixe l'architecture.

2.2 En fixant l'architecture

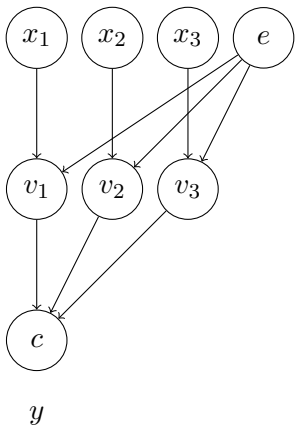
Considérons un réseau de neurones à seuil à n entrées et m sorties. Une *tâche* est un élément de $\mathbf{B}^n \times \mathbf{B}^m$. On dit que le réseau est *compatible* avec la tâche $(x_1, \ldots, x_n, y_1, \ldots, y_m)$ si l'on obtient la sortie $(y_1, \ldots, y_m)$ quand on évalue le réseau sur l'entrée $(x_1, \ldots, x_n)$.

On s'intéresse au problème de décision suivant APPRENTISSAGE : étant donnée une architecture (un graphe) d'un réseau de neurones et une liste finie de tâches, déterminer si l'on peut trouver des poids unitaires et des seuils pour cette architecture qui la rende compatible avec tous les tâches de la liste.

Question 7. *Montrer que le problème est dans la classe NP.*

Solution : Étant donnée une valeur pour chacun des poids, et des seuils, on peut vérifier en temps polynomial que le réseau de neurone obtenu est compatible avec chacune des tâches. Les poids peuvent prendre un nombre fini de valeurs, à savoir -1, 0 ou 1. Pour les seuils, on peut se limiter aussi à un nombre fini de valeurs, puisque les neurones ne prennent que des valeurs booléennes, et ce nombre reste bien polynomial en la taille de la description des entrées. □

Question 8. *On considère le réseau à seuil à* 4 *entrées (à savoir x_1, x_2, x_3, e) et* 1 *sortie (à savoir y) qui se représente graphiquement de la façon suivante :*



On considère les 8 *tâches que l'on obtient en faisant varier $x_1, x_2, x_3 \in \mathbf{B}$, et en ayant fixé $e = 0$, en prenant y qui vaut* 1 *sauf dans l'unique cas $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ où y vaut* 0.

Prouver que cette architecture est compatible avec ces 8 *tâches.*

Solution : L'architecture est compatible, car on peut prendre $v_1 = x_1$, $v_2 = x_2$, et $v_3 = x_3$, et prendre c comme $\neg x_1 \vee x_2 \vee \neg x_3$, avec par exemple $c = \mathcal{H}(-x_1 + x_2 - x_3 + 1)$. □

Question 9. *On suppose qu'on a fixé les seuils et poids de l'architecture de la question précédente de manière à obtenir un réseau de neurones qui, sur les 8 tâches de cette dernière question, produit les sorties attendues. Notons $f_i(x)$ la valeur d'activation du nœud v_i sur l'entrée $(x, 0)$ (c'est-à-dire, celle où $x_i = x$ et $e = 0$). Montrer que f_i est soit la fonction identité, soit la fonction $x \mapsto \mathrm{NOT}(x)$.*

Solution : Cela revient à prouver que la fonction f_i n'est pas fonction contante 0, ni la fonction constante 1, car il n'y a que 4 fonctions possibles de $\mathbf{B}^2$ dans $\mathbf{B}$. Mais, si c'était une fonction constante, on ne pourrait pas avoir la propriété de la question 8, car y ne vaudrait pas 1 dans un unique cas, mais au moins deux, car si on modifie l'entrée x sans modifier les autres entrées, on doit nécessairement obtenir la même sortie y. □

Notons $\overline{x_i}$ pour $f_i(x_i)$.

Question 10. *On suppose toujours donné un réseau de neurones comme à la question précédente.*
Montrer que la valeur d'activation de c, notée y, s'exprime sous la forme $y = \mathrm{NOT}\,(\overline{x_1}) \vee \overline{x_2} \vee \mathrm{NOT}\,(\overline{x_3})$.

Solution :
On regarde la table de vérité, et on observe que cela est nécessairement cela. □

Lorsque x est une variable, on note $x[0]$ pour x, et $x[1]$ pour sa négation. Si on pose $d_i = 0$ lorsque f_i est la fonction identité, et $d_i = 1$ lorsque f_i est la fonction $\mathrm{NOT}(x)$, on a donc $\overline{x_i} = x_i[d_i]$.

Question 11. *On se donne une formule du calcul propositionnel $\phi(x_1, \ldots, x_n)$ avec les variables $x_1, \ldots, x_n$. Montrer que les deux assertions suivantes sont équivalentes :*

- *la formule $\phi(x_1, \ldots, x_n)$ est satisfiable ;*
- *on peut choisir $d_1, d_2, \ldots, d_n \in \{0, 1\}$, tels que si l'on regarde la formule $\phi'(x_1, \ldots, x_n, e)$ avec $n+1$ variables, donnée[3] par $\phi'(x_1, \ldots, x_n, e) = \phi(x_1[d_1], \ldots, x_n[d_n])$ pour tout x_1,..., x_n, $e \in \mathbf{B}$, on a que*

$$\phi'(0, \ldots, 0, 1)$$

est satisfaite.

Solution : Supposons que ϕ soit satisfaite pour les valeurs $x_1, \ldots, x_n \in \mathbf{B}$. On considère $d_1 = x_1$,..., $d_n = x_n$. Alors $0[d_1] = x_1$. On prend la formule $\phi'(x_1, \ldots, x_n, e)$ qui vaut $\phi(x_1[d_1], \ldots, x_n[d_n])$. Elle vérifie exactement les propriétés énoncées.

Réciproquement, supposons que l'on aie cette propriété. Pour $x_1 = x_2 = \cdots = x_n = 0$, on doit avoir $\phi'(0, \ldots, 0, 1) = \phi(0[d_1], \ldots, 0[d_n]) = 1$. Ca veut donc dire que ϕ est satisfiable pour les entrées $0[d_1], \ldots, 0[d_n]$. □

Question 12. *Montrer que le problème* APPRENTISSAGE*, même limité aux réseaux de neurones de profondeur[4] 2, est NP-complet. Même si dans les exemples précédents, les réseaux de neurones n'avaient qu'une sortie, rappelons que les réseaux de neurones utilisés peuvent avoir plusieurs sorties.*

3. Nous confirmons que la valeur de ϕ' ne dépend pas de son argument e.
4. Voir la note en bas de page de la page précédente pour une définition formelle de la profondeur. Par exemple, le réseau de neurones de la figure de la question 8 est de profondeur 2.

Solution : Supposons que l'on se donne une instance de 3-SAT, et donc une conjonction des clauses $C_1, C_2, \dots C_m$. On suppose que la clause C_j utilise les variables x_{j_k} pour $1 \leq k \leq 3$. En particulier, on peut écrire $C_j = (x_{j_1}[\alpha_j] \vee x_{j_2}[\beta_j] \vee x_{j_3}[\gamma_j])$ pour $\alpha_j, \beta_j, \gamma_j \in \mathbf{B}$ (en utilisant la notation où $x[0]$ est x, et $x[1]$ est sa négation).

On considère une architecture A construite comme suit. Il y a une entrée d'entraînement e. Pour chaque variable $x_i, 1 \leq i \leq n$, il y a une entrée x_i et une porte v_i. Pour chaque clause $C_j, 1 \leq i \leq m$, il y a un sommet c_j. Chaque v_i est connectée à x_i et e, et chaque c_j est connectée à v_{j_k}, pour $1 \leq i \leq n$, $1 \leq j \leq m$, et $1 \leq k \leq 3$. Il y a une sortie pour chaque sommet c_j.

Si c'est plus clair : si on note V les portes, X les entrées, Y les sorties, et E les arcs du graphe, on a :

$$\begin{aligned}
V &= \{v_i \mid 1 \leq i \leq n\} \cup \{c_i \mid 1 \leq i \leq m\} \\
X &= \{x_i \mid 1 \leq i \leq n\} \cup \{e\} \\
Y &= \{c_i \mid 1 \leq i \leq m\} \\
E &= \{(x_i, v_i), (e, v_i) \mid 1 \leq i \leq n\} \cup \{(v_{j_k}, c_j) \mid 1 \leq j \leq m, 1 \leq k \leq 3\}
\end{aligned}$$

L'idée est de construire un ensemble de tâches qui force A à se comporter comme suit. Quand $e = 0$, v_i calcule soit x_i soit son complément, pour chaque $1 \leq i \leq n$. Lorsque $e = 1$, v_i sera libre et pourra prendre n'importe quelle valeur, qui sera interprétée comme une valeur pour x_i, pour $1 \leq i \leq n$. Chaque porte c_j évaluera la clause C_j sur la valeur de $x_{j_1}, x_{j_2}, x_{j_3}$ produite à partir de $v_{j_1}, v_{j_2}, v_{j_3}$, respectivement.

Pour cela, on utilise 8 tâches pour chaque clause C_j, $1 \leq j \leq m$. Elles sont construite comme dans les questions précédentes pour $e = 0$, en fixant toutes les variables qui n'apparaissent pas dans la clause à 0 (en généralisant simplement au cas de la clause qu'on cherche à produire, en variant l'unique y pour lequel on a un exemple négatif).

Plus formellement, on défini $\delta : \mathbb{N} \times \mathbf{B}^3 \to \mathbf{B}^n$ comme suit :

$$\delta(j, \alpha\beta\gamma) = b_1 \cdots b_n$$

où $b_{j_1} = \alpha, b_{j_2} = \beta, b_{j_s} = \gamma$, et $b_i = 0$ pour $i \neq j_1, j_2, j_3$. Soit $v(j, \alpha\beta\gamma)(i)$ la valeur de la clause j lorsque toutes les variables sont mises à 0 sauf $x_{j_1}, x_{j_2}, x_{j_3}$ que l'on fixe aux valeurs $\alpha, \beta, \gamma \in \mathbf{B}$ respectivement. Pour chaque clause $C_j, 1 \leq j \leq m$, on a 8 tâches

$$(\delta(j, \alpha\beta\gamma)0, t(j, \alpha\beta\gamma))$$

avec $t(j, \alpha\beta\gamma)$, pour $\alpha\beta\gamma \in \{000, 001, 010, 011, 100, 101, 110, 111\}$, définies par :

$$t(j, \alpha\beta\gamma) = (v(j, \alpha\beta\gamma)(0), v(j, \alpha\beta\gamma)(1) \cdots v(j, \alpha\beta\gamma)(m)).$$

Il y a également une tâche de calcul :

$$(\underbrace{0 \cdots 0}_{n} 1, \underbrace{1 \cdots 1}_{m}).$$

qui vise à dire que lorsque $e = 1$ on veut toutes les clauses satisfaites.

Clairement, on peut produire ce circuit en temps polynomial à partir de la description d'une formule 3-SAT.

Supposons que C soit satisfiable. Par le raisonnement de la question 8, le réseau de neurone est compatible. Plus formellement, si $b_1, \dots, b_n \in \mathbf{B}^n$ est une valeur de vérité qui satisfait C, pour $1 \leq i \leq n$, pour $e = 0, v_i$ produit x_i, et quand $e = 1, v_i$ produit b_i. For $1 \leq j \leq m, c_j$ calcule $(v_{j_1}[\alpha_j] \vee v_{j_2}[\beta_j] \vee v_{j_3}[\gamma_j])$.

Réciproquement, on suppose que l'architecture est compatible. Par le raisonnement de la question 9, les tâches pour la clause C_j forcent v_i a être soit x_i ou un complément. Définissons $value(v_i)$ comme x_i dans le premier cas, et sa négation dans le second. Toujours par un raisonnement similaire à la question 9, les tâches pour la clause C_j forcent la porte c_j à calculer

$$value(v_{j_1})[\alpha_j] \vee value(v_{j_2})[\beta_j] \vee value(v_{j_3})[\gamma_j]$$

Autrement dit, si on interprète les sorties des portes $v_1, \ldots, v_n$ comme une valeur pour les variables $x_1, \ldots, x_n$ de C (si ce n'est que la valeur de certaines variables peut être listée comme le complément de la faqn dont elles apparaissent vraiment), alors c_j calcule la valeur de C_j pour cette valeur de variables. Puisqu'on doit satisfaire la tâche avec $e = 1$, chacune des clauses doit être satisfaite (selon le principe de la question 10). Par conséquent, il doit y avoir une valeur pour les variables qui satisfait C. □

3 Sous-shifts et motifs interdits

Une configuration est une fonction de $\mathbb{Z}^2$ dans $\{0,1\}$: une configuration c colore chaque point du plan soit par 0 soit par 1. Un sous-ensemble fini $\mathbb{U} \subset \mathbb{Z}^2$ est appelé une fenêtre. Un motif est une fonction de $\mathbb{U}$ dans $\{0,1\}$, où $\mathbb{U}$ est une fenêtre, que l'on appelle le support du motif. Etant donnée une configuration c, on écrit $c_{\mathbb{U}}$ pour le motif que l'on voit dans la fenêtre $\mathbb{U}$: formellement, c'est la restriction de la fonction c à $\mathbb{U}$. On dit que ce motif est dans la configuration.

Etant donné $(d_1, d_2) \in \mathbb{Z}^2$, la fonction décalage $\sigma^{(d_1,d_2)}$ est la fonction qui envoie la configuration c sur la configuration c' avec $c'(i,j) = c(i - d_1, j - d_2)$. Elle envoie un motif m sur un motif m' avec $m'(i,j) = m(i - d_1, j - d_2)$: son support $\mathbb{U}'$ est donné par l'ensemble des (i,j) tels que $(i - d_1, j - d_2) \in \mathbb{U}$.

On se fixe un ensemble de motifs $\mathcal{F}$. On s'autorise à ce que cet ensemble soit fini ou infini.

On note $T(\mathcal{F})$ pour l'ensemble des configurations où l'on s'interdit les motifs de $\mathcal{F}$: c'est l'ensemble des configurations c privé de tous les décalages des motifs de $\mathcal{F}$.

Question 13. *Montrer qu'il existe une théorie $\mathcal{T}$ du calcul propositionnel qui possède un modèle si et seulement $T(\mathcal{F})$ est non-vide.*

Solution : Notons $\Sigma = \{0,1\}$. Pour chaque $i, j \in \mathbb{Z}^2$, on introduit une variable propositionnelle $c_{i,j}$ qui vise à coder le fait que la configuration vaut 1 à la position i, j. On considère pour chaque motif p de $\mathcal{F}$, et chaque $(d_1, d_2) \in \mathbb{Z}^2$ le motif $p' = \sigma^{(d_1,d_2)}(p)$. On écrit une formule qui dit que l'on a pas le motif p' dans la configuration c de la forme $\neg(\bigwedge_{(u,v)\in support(p'):p'(u,v)=1} c_{u,v} \wedge \bigwedge_{(u,v)\in support(p'):p'(u,v)=0} \neg c_{u,v})$. La théorie est l'union de toutes ces formules.

Par construction et définition, elle possède un modèle si et seulement si $T(\mathcal{F})$ est non-vide. □

On dit qu'une suite $\left(c^i\right)_{i\in\mathbb{N}}$ de configurations converge vers la configuration c si pour tout $(n_1, n_2) \in \mathbb{Z}^2$ il y a un rang $k \in \mathbb{N}$ tel que pour tout $i \geq k$ on a $c^i(n_1, n_2) = c(n_1, n_2)$: autrement dit, une suite converge si à chaque position, à partir d'un moment, un seul élément de $\{0,1\}$ apparaît dans cette suite à cette position. On dit qu'un ensemble de configurations C est *fermé* si pour toute suite à valeur dans C qui converge vers une certaine limite, est telle que cette limite est dans C.

Un ensemble de configurations C est stable par décalage, si pour tout $c \in C$, pour toute valeur possible de (d_1, d_2), on a que $\sigma^{(d_1,d_2)}(c) \in C$.

On appelle *sous-shift* un ensemble de configurations qui est fermé et qui est stable par décalage.

Question 14. *Prouver que $T(\mathcal{F})$ est un sous-shift.*

Solution : Par définition, c'est stable par décalage (la composition de deux décalages est un décalage). C'est fermé : en effet, si l'on prend une suite de configurations qui évite les configurations de $\mathcal{F}$ qui converge vers une configuration c, alors pour tout motif $p \in \mathcal{F}$, si on considère le support $\mathbb{U}$ de p, ce support étant fini, à partir d'un certain rang, tous les éléments de la suite ont le même contenu sur $\mathcal{U}$, et donc évite bien p. □

C'est même une caractérisation des sous-shifts :

Question 15. *Montrer que pour tout sous-shift T, on peut trouver un ensemble de motifs $\mathcal{F}$ tel que $T = T(\mathcal{F})$.*

Solution : On note $\mathbb{U}_n = [-n, n]^d$ et $\mathcal{L}_{\mathbb{U}_n}(T) = \{p \in \{0,1\}^{\mathbb{U}_n} : \exists y \in T$ tel p est un motif de $y\}$.

Soit T un sous-shift. On considère l'ensemble des motifs $\mathcal{F}$ donné par le complément du langage de T, c'est-à-dire $\mathcal{F} = \cup_{n\in\mathbb{N}}\{0,1\}^{\mathbb{U}_n} \setminus \mathcal{L}_{\mathbb{U}_n}(T)$.

— Si $c \in T$, tout motif de c est un motif de $\mathcal{L}(T)$ donc $c \in T(\mathcal{F})$.

— Si $c \in T(\mathcal{F})$ alors pour tout $n \in \mathbb{N}$ on a $c_{[-n,n]^d} \in \mathcal{L}(T)$ c'est-à-dire (par définition de $\mathcal{L}(T)$), il y a un $y^n \in T$ tel que $c_{[-n,n]^d} = y^n_{[-n,n]^d}$. On a donc $\lim_{n\to\infty} y^n = x$ et puisque T est fermé, $c \in T$.

□

Question 16. *On se fixe un ensemble de motifs $\mathcal{F}$.*

Montrer que $T(\mathcal{F})$ est non-vide si et seulement si pour tout entier n, on peut trouver un motif de support $\mathbb{U}_n = [-n, n]^2$ qui ne contient aucun motif de $\mathcal{F}$.

Solution : $\Rightarrow$: Si $c \in T(\{0,1\}, d, \mathcal{F}) \cap C$ alors toutes ses restrictions $c_{\mathbb{U}_n}$ est un motif de support $\mathbb{U}_n$, qui ne contient aucun motif de $\mathcal{F}$.

$\Leftarrow$: On sait par la question 13 qu'il suffit de montrer que la théorie associée est non-vide. Par le théorème de compacité, il suffit de le montrer pour toutes les parties finies de la théorie. Une telle partie finie mentionne au plus un nombre fini de variables propositionnelles. Toutes celle-ci sont includes dans $\mathbb{U}_n$ pour n suffisemment grand. Par hypothèse, on sait que l'on peut paver ce $\mathbb{U}_n$ et donc cela implique, vu leur définition, que l'on peut satisfaire la partie finie. □

4 Systèmes de Post

Un système [5] de Post est défini par un alphabet fini Σ, et un nombre fini de règles de récriture de la forme $g \to d$, où g, d sont des mots de Σ^*.

Sur un mot $w \in \Sigma^*$ sur l'alphabet Σ, on applique l'instruction suivante :

1. pour chaque règle $g \to d$, si le mot commence par le mot g, on efface ce prefixe, et on ajoute le mot d à sa fin : autrement dit, si le mot est de la forme $w = gw'$, il devient $w'd$.

On fait cela tant que c'est possible. Si ce processus mène à un mot auquel plus aucune règle ne s'applique, on dit que le mot initial w est accepté. Notons qu'il est possible que sur un mot donné il y ait plusieurs possibilités d'évolution. Il est aussi possible de considérer les règles $g \to d$ dans n'importe quel ordre.

Question 17. *Montrer que l'ensemble des mots acceptés par un système de Post est récursivement énumérable.*

Solution : On peut l'énumérer en testant de façon systématique sur toutes les paires (w, t) les dérivations en t étapes sur w. □

On dira qu'on transforme un mot w en un mot w' s'il y a un nombre fini d'étapes tel qu'en partant du mot w on fini par atteindre le mot w'.

Question 18. *On considère un mot de la forme $I_j 1^{n_1} K_j 1^{n_2}$: c'est-à-dire avec le symbole I_j, puis $n_1 \geq 0$ fois le symbole 1, puis le symbole K_j, puis $n_2 \geq 0$ fois le symbole 1.*

Proposer des règles qui transforment ce mot en un mot similaire, mais où on a ajouté 1 à n_1, remplacé I_j par I_{j+1}, et remplacé K_j par K_{j+1}.

5. C'est en fait ce que l'on appelle un système de post en forme normale. On ne considère que des systèmes de Post en forme normale dans ce sujet.

Solution : On considère $I_j \to I_{j+1}1$, $1 \to 1$, $K_j \to K_{j+1}$. □

Question 19. *Que fait le système de Post avec les règles* $1 \to 1$, $I_jK_j \to I_{j'}K_{j'}$, $I_j1 \to I_{j+1}$, $K_j \to K_{j+1}$ *sur un mot de la forme* $I_j1^{n_1}K_j1^{n_2}$ *?*

Solution : Il transforme ce mot en un mot de la même forme, c'est à dire avec le symbole I_{j^+}, puis $n_1^+ \geq 0$ fois le symbole 1, puis le symbole K_{j^+}, puis $n_2^+ \geq 0$ fois le symbole 1.

On a j^+ qui vaut j' si $n_1 = 0$, et $j+1$ sinon. Par ailleurs $n_1^+ = max(n_1 - 1, 0)$, $n_2^+ = n_2$.

Autrement dit, il simule l'instruction d'une machine à 2-compteurs qui teste si le premier compteur vaut 0, et si oui va à l'instruction $j^+ = j'$, sinon à l'instruction suivante. □

Question 20. *Démontrer qu'il est indécidable de déterminer si un système de Post accepte un mot donné.*

Solution : Les règles de la question 18 ajoute 1 au premier compteur, et passent à l'instruction suivante. Les règles $I_j \to I_{j+1}$, $1 \to 1$, $K_j \to K_{j+1}1$, font de même sur le second compteur. Les règles de la question 19 simule l'instruction d'une machine à 2-compteur qui teste si le premier compteur vaut 0, et si oui va à l'instruction j^+, sinon à l'instruction suivante. On peut faire de même sur le second compteur, par exemple avec :

$I_j \to I'_j$, $K_iI'_j \to K'_jI_{j'}$, $K'_j \to K_{j'}$, $K_j1 \to K_{j+1}$, $I_{j'} \to I_{j+1}$.

On peut donc simuler n'importe quel programme d'une machine à deux compteurs avec des règles d'un système de Post.

Le problème de savoir si une machine de Turing accepte un mot donné est indécidable (c'est le problème L_{univ} du cours). Par les constructions du cours simulant une machine de Turing par une machine à deux compteurs, L_{univ} se réduit au problème de savoir si une machine à deux compteurs accepte un mot donné. Par ce qui précède, cela se réduit au problème de la question, et donc c'est indécidable. □