Partiel 2020 et éléments de réponses

Samuel Mimram, Amaury Pouly, Bruno Salvy

30 septembre 2020

Question 1. Existe-t-il une théorie du premier ordre dont l'unique modèle soit le groupe additif des entiers $\mathbb{Z}$?

— **Correct:** non

— **Incorrect:** oui

Prenons une théorie $\mathcal{T}$ consistante du premier ordre des groupes additifs dont $\mathbb{Z}$ est l'unique modèle. On ajoute à la signature un symbole de constante c et on ajoute à $\mathcal{T}$, pour tout $n \in \mathbb{Z}$, l'axiome $\neg(c = E_n)$ où E_n est une expression désignant l'entier n (par exemple si n est positif et qu'on a une fonction successeur, on peut prendre $E_n = s^n(0)$). Soit $\mathcal{T}'$ la théorie obtenue ainsi. On va maintenant lui appliquer le théorème de compacité : si on prend un ensemble fini d'axiomes de $\mathcal{T}'$ alors elle contient un nombre fini de $\neg(c = E_n)$ donc il suffit de choisir comme modèle le modèle de $\mathcal{T}$ où l'on interprète de plus c comme étant un entier qui n'est pas dans cette liste d'axione. Ainsi par compacité, on aura un modèle de $\mathcal{T}'$ (et donc de $\mathcal{T}$) qui interprète une constante c comme n'étant égale à aucun entier, et donc ne peut pas être $\mathbb{Z}$.

Question 2. Existe-t-il une théorie du premier ordre dont les modèles sont les groupes avec 5 éléments ?

— **Correct:** oui

— **Incorrect:** non

Il suffit de partir de la théorie des groupes donnée en cours et d'ajouter 5 constantes $c_1, \ldots, c_5$, avec les axiomes

— *$c_i \neq c_j$ pour $i, j \in \{1, \ldots, 5\}$ avec $i \neq j$, ce qui assure que tout modèle a au moins 5 éléments,*

— *$\forall x(x = c_1 \vee \ldots \vee x = c_5)$, ce qui assure que tout modèle a au plus 5 éléments.*

Question 3. La formule $\exists X \subseteq \mathbb{N}(x \in X \Rightarrow x = 0)$ est-elle une formule du premier ordre sur la signature $(\{0\}, \{\}, \{\in, =\})$?

— **Correct:** non

— **Incorrect:** oui

La quantification sur un sous-ensemble n'est pas valide.

Question 4. La formule $\forall x(x \in x \Rightarrow x = x)$ est-elle une formule du premier ordre sur la signature $(\{0\}, \{\}, \{\in, =\})$?

— **Correct:** oui
— **Incorrect:** non

Le symbole $\in$ est ici un symbole de relation, la formule est valide.

Question 5. Considérons la théorie des groupes donnée en cours sur la signature $(\{1\}, \{\times\}, \{=\})$. La formule $\exists x(x \times x = x)$ est-elle prouvable ?

— **Correct:** oui
— **Incorrect:** non
— **Incorrect:** ça dépend

L'élément neutre d'un groupe est toujours idempotent ($1 \times 1 = 1$).

Question 6. Considérons la théorie des groupes donnée en cours sur la signature $(\{1\}, \{\times\}, \{=\})$. La formule $\exists y((x \times x) \times y = x)$ est-elle prouvable ?

— **Correct:** oui
— **Incorrect:** non
— **Incorrect:** ça dépend

Le raisonnement classique suivant peut être formalisé, car il n'utilise que des propriétés générales des groupes, c'est-à-dire des axiomes de la théorie des groupes. Considérons $y = x^{-1}$. On a

$$\begin{aligned}
(x \times x) \times y &= x \times (x \times y) && \textit{par associativité de la multiplication} \\
&= x \times 1 && \textit{par définition d'un inverse} \\
&= x && \textit{par propriété des éléments neutres}
\end{aligned}$$

Par transitivité de l'égalité, on a donc $(x \times x) \times x^{-1} = x$ et on en déduit que la formule est vraie.

Question 7. La formule $(\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$ est-elle prouvable en calcul propositionnel ?

— **Correct:** oui
— **Incorrect:** non

C'est vrai classiquement (contraposition) : on peut faire une table de vérité pour s'en convaincre.

Question 8. La formule $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$ est-elle prouvable en calcul propositionnel ?

— **Correct:** oui
— **Incorrect:** non

C'est vrai classiquement (loi de Pierce) : on peut faire une table de vérité pour s'en convaincre.

Question 9. La formule $(\neg A \Rightarrow A) \Rightarrow A$ est-elle prouvable en calcul propositionnel ?

— **Correct:** oui
— **Incorrect:** non

C'est vrai classiquement :

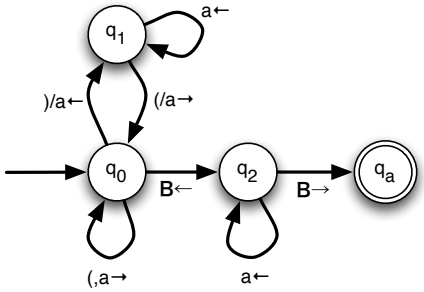
— *si* $A = 0$ *alors* $\neg A = 1$ *donc* $\neg A \Rightarrow A = 0$ *donc* $(\neg A \Rightarrow A) \Rightarrow A = 1$,
— *si* $A = 1$ *alors* $\neg A = 0$ *donc* $\neg A \Rightarrow A = 1$ *donc* $(\neg A \Rightarrow A) \Rightarrow A = 1$.

Question 10. Fixons une théorie du premier ordre inconsistante. Existe-t-il un programme qui prend en entrée une formule F sur la même signature et indique F est prouvable ou non ?

— **Correct:** oui
— **Incorrect:** non

Une théorie inconsistante n'a pas de modèle, toute formule est donc trivialement vraie dans tout modèle, par complétude elle est donc prouvable. Le programme qui renvoit toujours « vrai » convient.

Question 11. Les mots bien parenthésés sur l'alphabet $\{), (\}$ sont reconnus par la machine de Turing suivante :

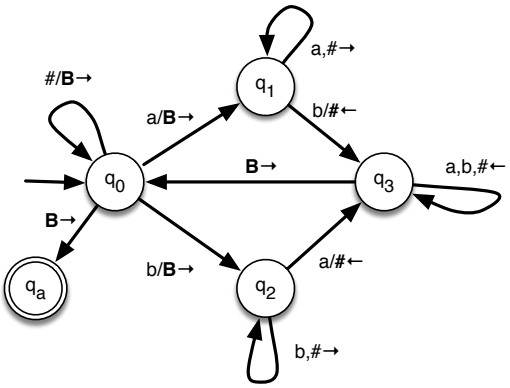


On souhaite maintenant reconnaître des mots bien parenthésés sur l'alphabet $\{), (,], [\}$, de sorte que le mot $([()])()$ est bien parenthésé, mais le mot $(]$ ne l'est pas. En conservant le même principe de fonctionnement pour la machine, combien d'états supplémentaires faut-il lui ajouter au minimum pour reconnaître ce langage ?

— **Correct:** 1
— **Incorrect:** 0
— **Incorrect:** 2
— **Incorrect:** 4

La transition de q_0 vers lui-même devient $(, [, a \rightarrow$, et on ajoute juste un état, q_1', copie de q_1 obtenue en remplaçant les ')' par des ']' dans les transitions vers et depuis q_0.

Question 12. La machine ci-dessous reconnaît les mots sur l'alphabet $\{a, b\}$ qui contiennent autant de a que de b :

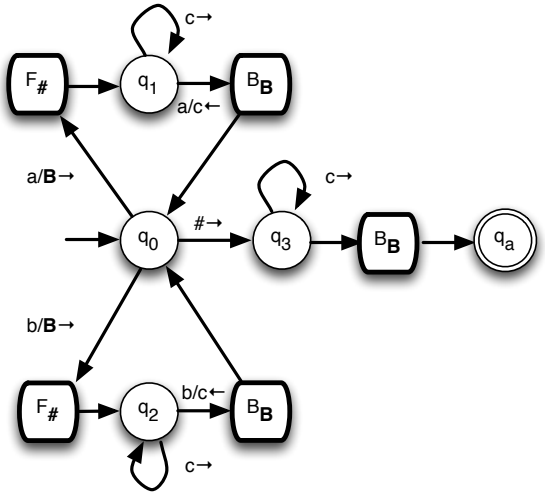


On souhaite maintenant reconnaître les mots sur l'alphabet $\{a, b, c\}$ qui contiennent le même nombre de a, de b et de c. En conservant le même principe de fonctionnement pour la machine, combien d'états supplémentaires faut-il lui ajouter au minimum pour reconnaître ce langage ?

— **Incorrect:** >8
— **Incorrect:** 8
— **Incorrect:** 6
— **Incorrect:** 0
— **Incorrect:** 2
— **Correct:** 4

Une transition supplémentaire de q_0 est utilisée si on rencontre c, puis de là deux autres selon si on rencontre a ou b en premier. Enfin q_3 et ces deux nouveaux états pointent vers un nouvel état duquel on recule jusqu'au blanc du début.

Question 13. La machine de Turing ci-dessous reconnaît les mots de la forme $w\#w$ où w est un mot de l'alphabet $\Sigma = \{a, b\}$:

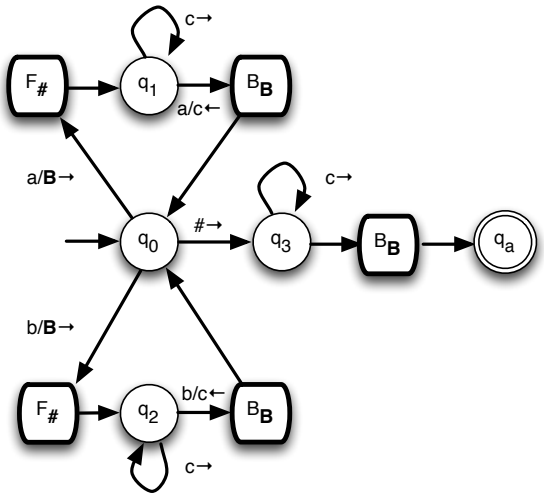


En dehors des caractères blancs (B), que reste-t-il sur le ruban à la fin de son exécution sur le mot $a^{10}\#a^9b$?

— **Incorrect:** $a^{10}\#a^9b$
— **Incorrect:** $c^{19}b$
— **Correct:** $\#c^9b$
— **Incorrect:** $a\#c^9b$

La machine efface progressivement la partie du mot qui précède le caractère # en en remplaçant les lettres par des blancs, tout en remplaçant les lettres correspondantes après le # par une lettre $c \notin \Sigma$. Il ne reste plus à la fin qu'à vérifier qu'il ne reste pas de lettres en trop. Lorsque son entrée est $a^{10}\#a^9b$ elle commence donc par traiter les 9 premiers a, laissant $a\#c^9b$ sur le ruban. Ensuite le a est effacé, la tête de lecture avance au delà des c, mais, en q_1 va dans l'état de rejet en lisant le b. Il reste à ce moment-là $\#c^9b$ sur le ruban.

Question 14. La machine de Turing ci-dessous reconnaît les mots de la forme $w\#w$ où w est un mot de l'alphabet $\Sigma = \{a, b\}$:



En dehors des caractères blancs (B), que reste-t-il sur le ruban à la fin de son exécution sur le mot $a^9b\#a^{10}$?

— **Incorrect:** $a^9b\#a^{10}$
— **Incorrect:** bc^{10}
— **Correct:** $\#c^9a$
— **Incorrect:** $b\#c^9a$

La machine efface progressivement la partie du mot qui précède le caractère # en en remplaçant les lettres par des blancs, tout en remplaçant les lettres correspondantes après le # par une lettre $c \notin \Sigma$. Il ne reste plus à la fin qu'à vérifier qu'il ne reste pas de lettres en trop. Lorsque son entrée est $a^9b\#a^{10}$ elle commence donc par traiter les 9 premiers a, laissant $b\#c^9a$ sur le ruban. Ensuite le b est effacé, la tête de lecture avance au delà des c, mais, en q_2 va dans l'état de rejet en lisant le a. Il reste à ce moment-là $\#c^9a$ sur le ruban.

Question 15. Le problème de déterminer si une machine de Turing M accepte au moins un mot dont la longueur est une puissance de 2, est

— **Correct:** indécidable mais semi-décidable
— **Incorrect:** indécidable et pas semi-décidable
— **Incorrect:** décidable

C'est indécidable par le théorème de Rice : la propriété d'accepter un mot dont la longueur est une puissance de 2 est une propriété du langage, qui est satisfaite par Σ^* mais pas par $\varnothing$. De plus, ce problème est semi-décidable car étant donné M, le langage $L(M)$ des mots qu'elle reconnait est semi-décidable donc récursivement énumérable. Il suffit donc de lister un à un des mots acceptés jusqu'à en trouver un dont la longueur est une puissance de 2.

Question 16. Le problème de déterminer si une machine de Turing M et un mot w sont tels que la machine accepte l'entrée w en utilisant (=écrivant) au plus $|w|^{42}$ cases, est

— **Incorrect:** indécidable mais semi-décidable
— **Incorrect:** indécidable et pas semi-décidable
— **Correct:** décidable

Le théorème de Rice ne s'applique pas car il s'agit d'une propriété de machine et non de langage. Ce problème est en fait décidable car étant donné w, on a une borne sur l'espace de travail de la machine, on peut donc simuler la machine jusqu'à ce qu'elle accepte/rejette, boucle ou qu'elle utilise plus de cases qu'elle n'en avait le droit. On peut détecter lorsqu'elle boucle en se souvenant des configurations.

Question 17. Le problème de déterminer si une machine de Turing M est telle qu'il existe une constante A_M telle que sur toute entrée, elle s'arrête en au plus A_M étapes, est

— **Correct:** indécidable mais semi-décidable
— **Incorrect:** indécidable et pas semi-décidable
— **Incorrect:** décidable

Le théorème de Rice ne s'applique pas, mais ce problème est indécidable par réduction depuis le problème de l'arrêt : soit T une machine, on construit T' qui sur l'entrée w simule T sur l'entrée vide pendant au plus $|w|$ étapes. Alors T s'arrête sur l'entrée vide si et seulement si T' fonctionne en temps constant. Le problème est semi-décidable car étant donné une constante A, on peut décider si la machine s'arrête en temps A sur toutes les entrées. En effet, si elle s'arrête en temps A alors seules les A premières cases peuvent êtres lues (les autres n'influent pas sur l'arrêt). On peut énumérer toutes les entrées de taille A et vérifier si la machine s'arrête en A étapes. Ainsi, on itérant sur des A de plus en plus grand, on va finir par en trouver un qui marche (s'il existe).

Question 18. Le problème de déterminer si une machine de Turing M est telle que $L(M)$ est reconnu par une (autre) machine de Turing ayant un nombre pair d'états, est

— **Incorrect:** indécidable mais semi-décidable

— **Incorrect:** indécidable et pas semi-décidable
— **Correct:** décidable

Pour toute machine M, on peut ajouter un état inutile si nécessaire pour s'assurer qu'il y a un nombre pair d'états, la réponse est donc toujours oui.

Question 19. Le problème de déterminer si une machine de Turing M accepte un nombre infini d'entrées, est

— **Incorrect:** indécidable mais semi-décidable
— **Correct:** indécidable et pas semi-décidable
— **Incorrect:** décidable

Il s'agit clairement une propriété de langage non triviale donc c'est indécidable par Rice. On va montrer que ce n'est pas semi-décidable en réduisant depuis le problème "étant donné une machine M et un mot w, décider si M ne s'arrête pas sur w" (ce problème n'est pas semi-décidable d'après le cours). Si on a M et w, on construit la machine M' qui sur l'entrée u va simuler M sur l'entrée w pendant $|u|$ étapes et accepte seulement si la machine n'a ***pas*** *terminé après $|u|$ étapes. On vérifie que si M ne s'arrête pas sur w alors M' accepte toutes les entrées (infini) donc M' est est accepté ; par contre si M s'arrête sur w alors M' va accepter seulement un nombre fini donc M' est réfusée.*

Question 20. Le problème de déterminer si une machine de Turing M accepte un nombre fini d'entrées, est

— **Incorrect:** indécidable mais semi-décidable
— **Correct:** indécidable et pas semi-décidable
— **Incorrect:** décidable

La preuve est la même qu'à la Question 19 mais cette fois M' accepte seulement si la machine a terminé avec $|u|$ étapes.

Question 21. Le problème de déterminer si une machine de Turing M accepte au moins 42 mots, est

— **Correct:** indécidable mais semi-décidable
— **Incorrect:** indécidable et pas semi-décidable
— **Incorrect:** décidable

C'est clairement une propriété de langage non triviale donc c'est indécidable par Rice. C'est semi-décidable car le langage $L(M)$ des mots qu'elle reconnait est semi-décidable donc récursivement énumérable. Il suffit donc de lister un à un des mots acceptés jusqu'à en trouver 42. Si jamais la machine accepte moins de 42 mots alors notre simulation va durer infiniment longtemps.

Question 22. Le problème de déterminer si une machine de Turing M accepte au plus 42 mots, est

— **Incorrect:** indécidable mais semi-décidable
— **Correct:** indécidable et pas semi-décidable
— **Incorrect:** décidable

C'est clairement une propriété de langage non triviale donc c'est indécidable par Rice. Ce n'est pas semi-décidable car son complémentaire est semi-décidable par la Question 21. En effet, si ce langage et son complémentaire étaient semi-décidable alors le langage serait décidable, mais on vient de montrer que ce n'est pas le cas.