Devoir d'Informatique n°5

10 mai 2025

Durée de l'épreuve :
3 heures (non tiers-temps)
4 heures (tiers-temps)

L'usage de tout dispositif électronique est interdit.

Consignes

Pour répondre à une question, il vous est permis de réutiliser le résultat d'une question antérieure même si vous n'avez pas réussi à établir ce résultat.

Quand l'énoncé demande de coder une fonction, sauf demande explicite, il n'est pas nécessaire de justifier la correction ou la terminaison de cette fonction, ou de la commenter. Cependant, il est conseillé d'expliquer l'intention de votre code, surtout si celui-ci est long.

Vous attacherez la plus grande importance à la clarté, à la précision et à la concision de la rédaction.

Si vous repérez ce qu'il vous semble être une erreur d'énoncé, signalez-le sur votre copie et poursuivez la composition en expliquant les éventuelles initiatives que vous aurez pris.

Vous devrez traiter les questions de programmation dans le langage OCaml.
Le sujet est constitué de trois parties. Chaque partie est indépendante des deux autres, mais la partie III repose sur des notations introduites au début de la partie II.

I. Exercice de logique

De nombreux travaux sont réalisés en Intelligence Artificielle pour construire un programme qui imite le raisonnement humain et soit capable de réussir le test de Turing, c'est à dire qu'il ne puisse pas être distingué d'un être humain dans une conversation à l'aveugle. Vous êtes chargé.e.s de vérifier la correction des réponses données par un tel programme lors des tests de bon fonctionnement. Dans le scénario de test considéré, le comportement attendu est le respect de la règle suivante : pour chaque question, le programme répondra par **trois** affirmations dont **une seule** sera correcte.

Nous noterons A_1, A_2, A_3 les propositions associées aux affirmations effectuées par le programme.

Q1. Représenter le comportement attendu sous la forme d'une formule du calcul des propositions qui dépend de A_1, A_2, A_3.

Premier cas

Vous demandez au programme : *Quels éléments doivent contenir les aliments que je dois consommer pour préserver ma santé ?*

Il répond les affirmations suivantes :

A_1 : Consommez au moins des aliments qui contiennent des glucides, mais pas de lipides !

A_2 : Si vous consommez des aliments qui contiennent des glucides, alors ne consommez pas d'aliments qui contiennent des lipides !

A_3 : Ne consommez aucun aliment qui contient des lipides !

Nous noterons G, respectivement L, les variables propositionnelles qui correspondent au fait de consommer des aliments qui contiennent des glucides, respectivement des lipides.

Q2. Exprimer A_1, A_2 et A_3 sous la forme de formules du calcul des propositions. Ces formules peuvent dépendre des variables G et L.

Q3. En utilisant le calcul des propositions (simplification avec les formules de De Morgan, associativité, etc...), déterminer ce que doivent contenir les aliments que vous devrez consommer pour préserver votre santé.

Second cas

Vous demandez au programme : *Quelles activités dois-je pratiquer si je veux préserver ma santé ?*

Suite à une coupure de courant, ***la dernière affirmation est interrompue.***

A_1 : Ne faites des activités sportives que si vous prenez également du repos !

A_2 : Si vous ne faites pas d'activité intellectuelle, alors ne prenez pas de repos !

A_3 : Prenez du repos ou faites des activi- !

Nous noterons S, I et R les variables propositionnelles qui correspondent au fait de faire des activités sportives, des activités intellectuelles, et de prendre du repos.

Q4. Exprimer A_1, A_2 et A_3 sous forme de formules du calcul des propositions. Ces formules peuvent dépendre des variables S, I et R.

Q5. En utilisant les tables de vérité, déterminer quelle(s) activité(s) vous devez pratiquer pour préserver votre santé.

II. Résolution

On rappelle qu'en OCaml, la fonction `Array.make: int -> 'a -> 'a array` permet la création de tableau. Elle prend en entrée un entier n et un élément a, et renvoie un tableau de n cases contenant toutes a.

1 Définitions et notations

Soit $\mathcal{Q}$ un ensemble fini de variables propositionnelles. On considère $\mathcal{F}$ l'ensemble des formules construites à partir des variables propositionnelles de $\mathcal{Q}$ et des connecteurs usuels de conjonction $\wedge$, de disjonction $\vee$ et de négation $\neg$. Pour toute formule $F \in \mathcal{F}$, $\mathcal{V}(F)$ désigne l'ensemble des variables propositionnelles qui apparaissent dans F. On note $\bot$ la disjonction vide et $\top$ la conjonction vide.

On appelle valuation toute fonction **totale** (i.e. définie partout) $\sigma : \mathcal{Q} \to \{0, 1\}$. Étant donné $F \in \mathcal{F}$ et σ une valuation, on note $[\![F]\!]^\sigma$ l'interprétation de F dans σ. On rappelle que pour $F, G \in \mathcal{F}$, $F \equiv G$ signifie que pour toute valuation σ, $[\![F]\!]^\sigma = [\![G]\!]^\sigma$.

On appelle **littéral** une variable propositionnelle ou bien la négation d'une variable propositionnelle. Le littéral est dit **positif** dans le premier cas, **négatif** dans le second cas.

On appelle **clause disjonctive** toute formule de la forme $l_1 \vee l_2 \vee \cdots \vee l_q$, où $q \geq 1$ et $l_1, \ldots, l_q$ sont des littéraux. Dans le reste du sujet, le mot **clause** sera utilisé pour désigner les clauses disjonctives. En particulier $\bot$, en tant que disjonction vide, est une clause.

On représente une clause comme un **ensemble** de littéraux. Ainsi, la clause $p \vee \neg p \vee p \vee q$ est représentée par l'ensemble $\{p, \neg p, q\}$, de même que la clause $q \vee p \vee \neg p \vee \neg p$. La clause vide $\bot$ est représentée par l'ensemble vide $\emptyset$.

On dit qu'une clause disjonctive R se déduit de deux clauses disjonctives C_1, C_2 **par résolution sur la variable** p lorsque C_1 est de la forme $\{l_1, \ldots, l_t, p\}$, C_2 de la forme $\{l'_1, \ldots, l'_s, \neg p\}$, et $R = \{l_1, \ldots, l_t, l'_1, \ldots l'_s\}$. On note alors $(C_1, C_2) \xrightarrow{p} R$. On dira que R **se déduit par résolution** de C_1 et C_2 s'il existe une variable p telle que $(C_1, C_2) \xrightarrow{p} R$. À titre d'exemple :

$$(\{a, c\}, \{\neg a, a, \neg b\}) \xrightarrow{a} \{a, \neg b, c\}$$

2 Utilisation de la résolution

Q1. Dans chacun des cas suivants, proposer une clause M rendant la résolution vraie :

(i) $(\{a, \neg b\}, \{\neg a, b, \neg c\}) \xrightarrow{a} M$

(ii) $(\{\neg a, b, \neg c\}, \{a, \neg b\}) \xrightarrow{b} M$

(iii) $(M, \{a, \neg b, c\}) \xrightarrow{b} \{a, c\}$

Q2. Soient deux clauses C_1, C_2 et deux variables distinctes telles qu'il existe deux clauses R_p, R_q telles que $(C_1, C_2) \xrightarrow{p} R_p$ et $(C_1, C_2) \xrightarrow{q} R_q$. Montrer qu'alors $R_p \equiv R_q \equiv \top$.

Q3. Montrer que si $(C_1, C_2) \xrightarrow{p} R$ alors pour tout ensemble $\Gamma \subseteq \mathcal{F}$, on a $\Gamma, C_1, C_2 \models R$.

3 Arbre de preuve

On appelle **séquent** la donnée d'un ensemble Γ de clauses et d'une clause C. On note $\Gamma \vdash C$ un tel séquent, et il se lit "Γ **prouve** C". Pour D une clause, on note $\Gamma, D \vdash C$ pour $\Gamma \cup \{D\} \vdash C$.

Intuitivement, un séquent $\Gamma \vdash C$ signifie qu'en supposant vraie les formules de Γ, on peut montrer C en raisonnant par résolution.

Une **preuve** valide d'un séquent est un arbre, que l'on définit inductivement :

— Pour Γ un ensemble de clauses et C une clause telle que $C \in \Gamma$, la feuille $A(\Gamma \vdash C)$ est une preuve valide de $\Gamma \vdash C$, appelée un **axiome**.

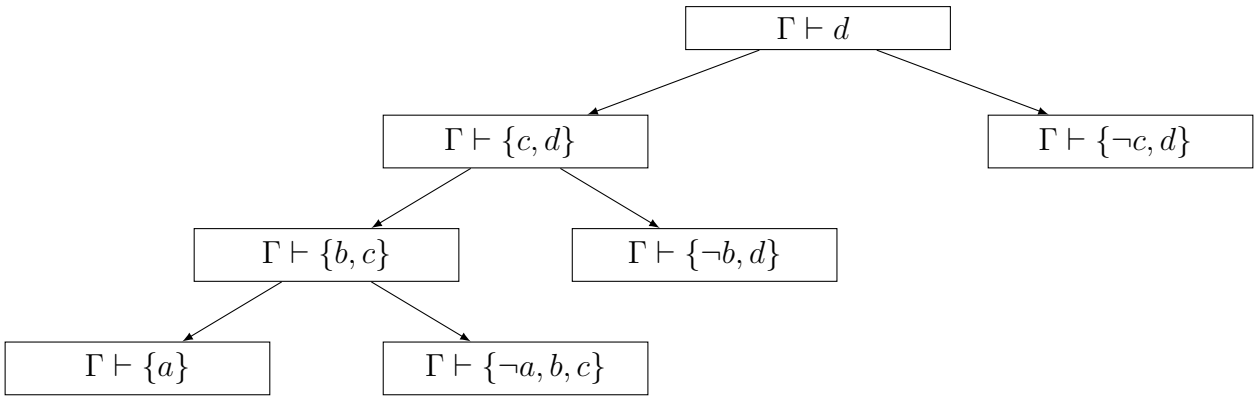
— Soient Γ un ensemble de clauses, C_1, C_2, R des clauses, p une variable, et Π_1, Π_2 deux preuves, tels que :

- $(C_1, C_2) \xrightarrow{p} R_p$;
- Π_1 est une preuve valide de $\Gamma \vdash C_1$;
- Π_2 est une preuve valide de $\Gamma \vdash C_2$.

Alors $N(\Gamma \vdash R, p, \Pi_1, \Pi_2)$ est une preuve valide de $\Gamma \vdash R$.

Un arbre de preuve est donc l'enregistrement de toutes les résolutions à appliquer à partir d'un ensemble Γ pour montrer une clause C.

Par exemple, pour $\Gamma = \{\{a\}, \{\neg a, b, c\}, \{\neg b, d\}, \{\neg c, d\}\}$, l'arbre suivant est un arbre de preuve valide (les variables utilisées pour la résolution ont été omises pour alléger le schéma) :



Q4. Pour $\Gamma = \{\{a\}, \{a, d\}, \{\neg a, b\}, \{\neg a, \neg b, c\}, \{\neg c, e\}, \{\neg d, e\}\}$, donner un arbre de preuve valide du séquent $\Gamma \vdash e$.

Q5. Montrer que si un séquent $\Gamma \vdash C$ admet un arbre de preuve valide, alors $\Gamma \models C$.

Q6. Montrer qu'il existe un séquent $\Gamma \vdash C$ n'admettant pas d'arbre de preuve, mais tel que $\Gamma \models C$.

Les deux questions précédentes montrent que le calcul par résolution est correct, c'est à dire que toute formule prouvable est vraie, mais qu'il n'est pas complet, c'est à dire qu'il existe une formule vraie mais pas démontrable.

4 Implémentation de la résolution en OCaml

On s'intéresse à l'implémentation d'un prouveur automatique par résolution.

Quitte à renommer les variables, on suppose dans la suite que $\mathcal{Q} = [\![0, n-1]\!]$. On suppose également que dans tout le code, on dispose d'une constante globale `n` contenant le nombre de variables propositionnelles disponibles. Ainsi, on se munit du type suivant pour représenter les variables propositionnelles :

```
let n = ...
type var_prop = int (* Supposés être entre 0 et n-1 *)
```

Dans tous les exemples qui suivent, on choisira $n = 4$, et plutôt que d'utiliser des entiers pour les littéraux, on utilisera les lettres a, b, c, d. Ainsi on n'écrira pas dans les exemples $\{0, \neg 1, 2, \neg 0\}$ mais $\{a, \neg b, c, \neg a\}$.

On remarque qu'une clause contenant un littéral et sa négation est logiquement équivalente à $\top$. Ainsi, il y a de multiples représentations de $\top$ (par exemple $\{\neg a, a, b\}$ ou encore $\{b, \neg b\}$). Afin d'éviter ces multiples représentations dans l'implémentation, on représentera toute formule logiquement équivalente à $\top$ par un constructeur `Top`.

Au vu des remarques précédentes, chaque clause non équivalente à $\top$ contient au plus une occurrence de chaque variable propositionnelle. Ainsi, une telle clause C peut être représentée par la donnée, pour chaque variable propositionnelle p, de la manière dont p apparaît dans C : soit positivement, soit négativement, soit pas du tout. On représentera alors une clause par le type suivant :

```
type clause =
  | Top
  | NTop of bool option array
```

Une clause est alors définie par disjonction de cas. Si c'est une clause équivalente à $\top$, on la représente par `Top`. Sinon, on la représente par `NTop(t)`, où `t` est un tableau de taille n donnant, pour chaque case d'indice $p \in [\![0, n-1]\!]$, la manière dont la variable p apparaît dans la clause.
Par exemple :

- La clause $\{a, \neg a, b\}$ est décrite par la valeur `Top`
- La clause $\{\neg a, b, c\}$ est décrite par la valeur `NTop [|Some false; Some true; Some true; None|]`.
- La clause $\{a, \neg b, d\}$ est décrite par la valeur `NTop [|Some true; Some false; None; Some true|]`.

Q7. Écrire une fonction `polarites: clause -> (int * int * int)` qui, étant donné une clause, renvoie un triplet (pos, neg, abs) avec pos le nombre de variables apparaissant positivement, neg le nombre de variables apparaissant négativement, et abs le nombre de variables n'apparaissant pas. La fonction renverra le triplet $(0, 0, 0)$ sur la clause $\top$.

5 Listes triées

On représente un ensemble de clauses Γ, que l'on appellera un **contexte**, par une liste de clauses. De plus, afin de faciliter la manipulation, on supposera les listes sans doublons, et triées selon l'ordre natif d'OCaml.

Q8. Définir une fonction `mem_triee: 'a list -> 'a -> bool` déterminant si un élément apparaît dans une liste triée par ordre croissant. Cette fonction doit utiliser le fait que la liste est triée.

Q9. Définir une fonction `insert_triee: 'a list -> 'a -> 'a list` insérant un élément dans une liste triée sans doublons. La fonction devra être récursive terminale.

Dans la suite, on utilisera donc le type suivant pour les contextes :

```
type contexte = clause list (* supposées triées sans doublons *)
```

6 Règle de résolution

Q10. Définir une fonction `is_bot: clause -> bool` déterminant si une clause est logiquement équivalente à $\bot$.

Q11. Définir une fonction `resolution: var_prop -> clause -> clause -> clause option` prenant en argument une variable p et deux clauses C_1, C_2, et renvoyant :

- `Some R` dans le cas où il existe une clause R telle que $(C_1, C_2) \xrightarrow{p} R$;
- `None` s'il n'existe pas de telle clause R.

Notons que d'après la Q2, si la clause R de la question précédente existe et n'est pas équivalente à $\top$, alors c'est la **seule** clause pouvant être obtenue par résolution à partir de C_1 et C_2.

Q12. Écrire une fonction `trouve_resolution: clause -> clause -> (clause * var_prop) option` prenant en entrée deux clauses C_1, C_2 et renvoyant :

- `Some (R, p)` dans le cas où il existe une (unique) clause R non équivalente à $\top$ et une variable p telle que $(C_1, C_2) \xrightarrow{p} R$;
- `None` s'il n'en existe pas.

On représente les arbres de preuve par le type suivant :

```
type preuve =
  | A of clause
  | N of clause * var_prop * preuve * preuve)
```

Notons que puisque le contexte reste constant au cours d'une preuve, il n'est pas utile de le faire apparaître dans les nœuds de l'arbre.

Q13. Écrire une fonction `est_preuve_valide: contexte -> preuve -> bool` prenant en entrée un contexte Γ et un arbre de preuve Π, et déterminant si Π est valide.

7 Saturation du contexte

On s'intéresse dorénavant à l'écriture d'une fonction qui génère un arbre de preuve valide lorsqu'il en existe un.

On définit, sur l'espace des ensembles de clauses, la fonction $\mathcal{C}$ par :

$$\mathcal{C}(\Gamma) = \Gamma \cup \{R \text{ clause } \mid \exists p \in \mathcal{Q}, \exists (C_1, C_2) \in \Gamma^2, (C_1, C_2) \xrightarrow{p} R\}$$

Autrement dit, $\mathcal{C}(\Gamma)$ est l'ensemble Γ auquel on a rajouté toutes les clauses pouvant être obtenues par résolution depuis Γ.

On note $\mathcal{C}^{(k)}$ la composée k fois de la fonction $\mathcal{C}$, avec $\mathcal{C}^{(0)}$ l'identité. On définit alors l'ensemble $\Gamma^\uparrow$ comme suit :

$$\Gamma^\uparrow = \bigcup_{k \in \mathbb{N}} \mathcal{C}^{(k)}(\Gamma)$$

Q14. Montrer que pour tout contexte Γ, $\mathcal{C}(\Gamma^\uparrow) = \Gamma^\uparrow$.

Q15. Montrer que pour tout contexte Γ et pour toute clause C, $\Gamma \vdash C$ admet un arbre de preuve si et seulement si $C \in \Gamma^\uparrow$.

8 Preuve automatique

L'algorithme permettant de trouver une preuve pour un séquent $\Gamma \vdash C$ va alors consister à calculer $\Gamma^\uparrow$ puis à tester si C en est un élément.

Q16. Définir une fonction `grand_c: contexte -> contexte` qui, étant donné Γ, calcule $\mathcal{C}(\Gamma)$.

Q17. Définir une fonction `saturation: contexte -> contexte` qui, étant donné Γ, calcule $\Gamma^\uparrow$. On justifiera **soigneusement** que cette fonction termine.

Q18. (Question ouverte) Définir une fonction `construit_preuve: contexte -> clause -> preuve option` qui, étant donné un contexte Γ et une clause C, construit un arbre de preuve du séquent $\Gamma \vdash C$ (ou renvoie None s'il n'existe pas de tel arbre de preuve). Il est conseillé de s'aider de fonctions intermédiaires qui seront dûment décrites. Les traces de recherche, même partielles, seront évaluées.

9 Complétude par réfutation

Q19. Étant donné Γ un contexte et G une formule quelconque, montrer que $\Gamma \models G$ si et seulement si $\Gamma, \neg G \models \bot$.

La question précédente justifie qu'au lieu d'essayer de démontrer $\Gamma \models G$ en exhibant une preuve de $\Gamma \vdash G$, on peut essayer d'obtenir une preuve de $\Gamma, \neg G \models \bot$. On s'intéresse dans cette partie à la démonstration du théorème suivant :

Théorème 1. Soit Γ un ensemble de clauses tel que $\Gamma \models \bot$. Alors $\Gamma \vdash \bot$.

On dit alors que le calcul par résolution est **complet par réfutation**.

Dans le reste de cet exercice, on manipule des valuations partielles, i.e. n'étant pas nécessairement définies sur tout $\mathcal{Q}$. Pour σ une telle valuation partielle, on note $\mathbf{def}(\sigma)$ son ensemble de définition. Lorsque σ_1 et σ_2 sont deux valuations telles que $\mathbf{def}(\sigma_1) \cap \mathbf{def}(\sigma_2) = \emptyset$, on note $\sigma_1 \uplus \sigma_2$ la valuation définie par :

$$\begin{array}{rcl} \mathbf{def}(\sigma_1) \cup \mathbf{def}(\sigma_2) & \longrightarrow & \{0,1\} \\ p & \longmapsto & \begin{cases} \sigma_1(p) & \text{si } p \in \mathbf{def}(\sigma_1) \\ \sigma_2(p) & \text{si } p \in \mathbf{def}(\sigma_2) \end{cases} \end{array}$$

Pour $p_1, \ldots, p_k \in \mathcal{Q}$ et $b_1, \ldots, b_k \in \{0,1\}$, on note $(p_1 \mapsto b_1, \ldots, p_k \mapsto b_k)$ la valuation associant à chaque p_i le b_i correspondant. En particulier, on note $()$ l'unique valuation vide, telle que $\mathbf{def}(()) = \emptyset$.

On souhaite donner un sens à l'interprétation d'une formule dans une valuation partielle. On se munit du symbole ? qui désigne une interprétation indéterminée. L'interprétation d'une formule ne sera alors plus à valeur dans $\{0,1\}$ mais dans $\{0,1,?\}$. Étant donné H une formule et σ une valuation partielle, on note $[H]^\sigma$ l'interprétation de H dans σ, définie par :

$$[H]^\sigma = \begin{cases} [\![H]\!]^\sigma & \text{si } \mathcal{V}(H) \subseteq \mathbf{def}(\sigma) \\ ? & \text{sinon} \end{cases}$$

On prendra garde à ne pas écrire $[\![H]\!]^\sigma$ lorsque $\mathcal{V}(H) \not\subseteq \mathbf{def}(\sigma)$.
Pour Γ un ensemble de clauses, on note $[\Gamma]$ l'ensemble des valuations partielles σ telles que pour tout $H \in \Gamma$, $[H]^\sigma \in \{?, 1\}$.

Q20. Soit Γ un ensemble de clauses. Montrer que $[\Gamma] = \emptyset$ si et seulement si $\bot \in \Gamma$.

Q21. Soit Γ un ensemble de clauses tel que :

- $() \in [\Gamma]$
- Pour toute valuation $\sigma \in [\Gamma]$, pour toute variable propositionnelle $p \in \mathcal{Q} \setminus \mathbf{def}(\sigma)$, $\sigma \uplus (p \mapsto 1) \in [\Gamma]$ ou $\sigma \uplus (p \mapsto 0) \in [\Gamma]$

Montrer qu'alors, Γ est satisfiable.

Q22. Soit C une clause, σ une valuation partielle, et p une variable sur laquelle σ n'est pas définie, tels que $[C]^\sigma = ?$ et $[C]^{\sigma \uplus (p \mapsto 0)} = 0$. Montrer qu'alors $p \in C$.

Q23. Soit Γ un ensemble insatisfiable de clauses, tel que $[\Gamma] \neq \emptyset$. Montrer qu'il existe trois clauses C_1, C_2, R, une valuation partielle σ et une variable $p \notin \mathbf{def}(\sigma)$ tels que :

- $p \in C_1$
- $\neg c \in C_2$
- $(C_1, C_2) \xrightarrow{p} R$
- $[\Gamma \cup \{R\}] \subsetneq [\Gamma]$

Q24. Montrer que le calcul par résolution est complet par réfutation.