DS4-MPI

Ce sujet est composé de quatre exercices indépendants.

1 Non prouvabilité en logique intuitionniste

On considère $\mathcal{F}$ l'ensemble des formules propositionnelles construites sur un ensemble de variables propositionnelles

$$V = \{x_1, \dots, x_n\},$$

à l'aide des connecteurs logiques $\bot$, $\wedge$, $\vee$ et $\to$. On note $\neg A$ la formule propositionnelle $A \to \bot$.

Pour un séquent $\Gamma \vdash A$, on note :

$$\Gamma \vdash_i A \quad (\text{resp. } \Gamma \vdash_c A)$$

si le séquent est prouvable en logique intuitionniste (resp. classique).

La logique intuitionniste est le système dont les règles sont celles rappelées dans l'annexe B en fin de sujet.

La logique classique est composée du même système de règles auquel on ajoute le Tiers exclu :

$$\frac{}{\vdash \phi \vee \neg\phi}$$

Dans cet exercice, on souhaite montrer qu'il existe des séquents prouvables en logique classique mais pas en logique intuitionniste.

On considère la sémantique suivante, dite *sémantique de Heyting*, définie sur $\mathcal{F}$.

On note $\mathcal{O}(\mathbb{R})$ l'ensemble des ouverts de $\mathbb{R}$ et on définit une valuation comme une fonction

$$\mu : V \to \mathcal{O}(\mathbb{R}).$$

On étend la définition de μ à toutes les formules de $\mathcal{F}$ par :

- $\mu(\bot) = \varnothing$;
- $\mu(A \wedge B) = \mu(A) \cap \mu(B)$;
- $\mu(A \vee B) = \mu(A) \cup \mu(B)$;
- $\mu(A \to B) = \overbrace{\left(\mu(A)^c \cup \mu(B)\right)}^{\circ}$,

où X^c désigne le complémentaire de X et $\mathring{X}$ l'intérieur de X.

Pour $\Gamma \subseteq \mathcal{F}$, on pose

$$\mu(\Gamma) = \bigcap_{A \in \Gamma} \mu(A),$$

avec la convention $\mu(\varnothing) = \mathbb{R}$.

Un séquent $\Gamma \vdash A$ est dit *valide* si

$$\mu(\Gamma) \subseteq \mu(A).$$

Une règle d'inférence est dite valide si, lorsque ses prémisses sont valides, alors sa conclusion est valide.

Q1 Quelle sémantique obtient-on si l'on considère des valuations à valeurs dans $\{\varnothing, \mathbb{R}\}$ au lieu de $\mathcal{O}(\mathbb{R})$?

Q2 Montrer que

$$\vdash_i ((A \vee B) \wedge \neg A) \to B.$$

Q3 Le séquent

$$\vdash ((A \vee B) \wedge \neg A) \to B$$

est-il valide pour la sémantique de Heyting ? Justifier.

Q4 Montrer que

$$\vdash_c (A \to B) \to (\neg A \vee B).$$

Q5 En utilisant la valuation qui pose $\mu(A) = \mathbb{R}^* = \mu(B)$, montrer que le séquent

$$\vdash (A \to B) \to (\neg A \vee B)$$

n'est pas valide pour la sémantique de Heyting.

Q6 Montrer que la règle de déduction $(\to e)$ est valide (on pourra utiliser sans le démontrer que l'intérieur de l'intersection de deux ensembles est égale à l'intersection de leurs intérieurs).

Q7 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.

Q8 Montrer que la règle du tiers exclu n'est pas valide. Que peut-on en déduire concernant la complétude de la logique intuitionniste pour la sémantique booléenne usuelle ?

2 Implémentation d'un tableau associatif par l'utilisation d'un algorithme probabiliste

Nous nous intéressons dans cette section à une manière d'implémenter un tableau associatif, en utilisant une structure de donnée probabiliste : les *skip lists*.

2.1 Principe des skip lists

Une skip list est une liste chaînée optimisée pour accélérer la recherche d'un élément. Pour ce faire, certains nœuds stockent dans un tableau des pointeurs vers d'autres nœuds situés plus loin dans la liste. Chaque case du tableau correspond à un « niveau » et un pointeur de niveau i mène vers un nœud qui possède lui aussi un pointeur de niveau i. Le niveau 0 contient l'ensemble des éléments, classés par ordre alphabétique en fonction de leur clef. Les niveaux supérieurs regroupent un sous-ensemble des éléments, de façon à espacer progressivement les nœuds. Par exemple, idéalement, le niveau 1 contiendrait un nœud sur deux, le niveau 2 un nœud sur quatre, et ainsi de suite (voir Figure 2).

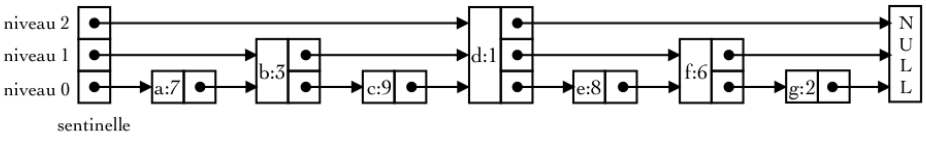

FIGURE 2 – Exemple d'une skip list idéale.

Le premier nœud de la skip list est une *sentinelle*, qui ne contient pas de donnée utile mais possède des pointeurs pour tous les niveaux de la liste.

Pour chercher un élément, on démarre à partir de la sentinelle au niveau le plus élevé. À ce niveau, on suit les pointeurs vers les nœuds suivants tant que la clef du nœud actuel est inférieure à celle recherchée. Dès qu'on rencontre un nœud dont la clef est supérieure, on descend d'un niveau et on répète la même

opération. Lorsque la recherche atteint le niveau 0, elle est terminée et, si l'élément recherché y est présent, il est renvoyé (voir Figure 3).

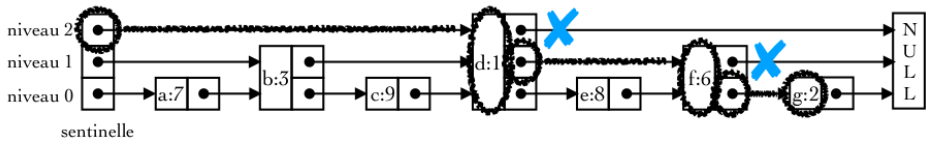

FIGURE 3 – Parcours lors de la recherche de la clef "g".

En pratique, chaque nœud a une probabilité $\frac{1}{2}$ d'être présent dans le niveau supérieur.

2.2 Implémentation

On définit les structures suivantes :

```
struct slNoeud_s {
  char* clef;
  int valeur;
  struct slNoeud_s* suivant_par_niveau[];
};
typedef struct slNoeud_s slNoeud;

struct slListe_s {
  int niveau_actuel;
  slNoeud* sentinelle;
};
typedef struct slListe_s slListe;
```

On définit une constante, `MAX_NIVEAU`, qui correspond au nombre maximal de niveaux qu'un nœud peut avoir. La sentinelle de la skip list est créée comme un nœud particulier, avec une clef vide, une valeur nulle, et un tableau de pointeurs de taille `MAX_NIVEAU`+1 (de 0 à `MAX_NIVEAU` inclus) tous initialisés à `NULL`.

Q9 Écrire une fonction C `sl_rechercher` qui recherche une clef dans la skip list passée en paramètre et renvoie la valeur associée si elle est présente, ou -1 sinon. On rappelle que pour tester l'égalité entre deux chaines de caractères, on utilise la fonction `strcmp`.

La signature de la fonction est : `int sl_rechercher(slListe* liste, char* clef)`

Q10 On considère la fonction suivante :

```
int sl_mystere() {
  int niveau = 0;
  while ((rand() % 2) == 1 && niveau < MAX_NIVEAU) {
    niveau++;
  }
  return niveau;
}
```

Quelles sont les propriétés de la valeur renvoyée par cette fonction ?

On suppose disponible la fonction :

```
slNoeud* sl_creer_noeud(int niveau, char* clef, int valeur);
```

Cette fonction crée un nouveau nœud de niveau niveau avec la clef et la valeur passées en paramètre. Les pointeurs du nouveau nœud sont tous initialisés à `NULL`.

La fonction `sl_ajoute_valeur`, qui ajoute une clef et une valeur dans la skip list passée en paramètre, a la structure suivante :

```
void sl_ajoute_valeur(slListe* liste, char* clef, int valeur) {
   int niveau = sl_mystere();
   slNoeud* nouveau_noeud = sl_creer_noeud(niveau, clef, valeur);
   slNoeud* noeuds_a_mettre_a_jour[MAX_NIVEAU + 1];
   slNoeud* courant = liste->sentinelle;
    // PARTIE A
    /* parcourt l'ensemble des niveaux de la liste pour trouver
    dans chaque niveau le noeud après lequel insérer
    le nouveau noeud, on le stockera dans le tableau noeuds_a_mettre_a_jour */
    // FIN PARTIE A
   if (niveau > liste->niveau_actuel) {
      // PARTIE B
      /* si le niveau du nouveau noeud est plus grand
      que le niveau actuel de la liste
      préparer la sentinelle */
      // FIN PARTIE B
   }

   // PARTIE C
   /* insérer le nouveau noeud */
   // FIN PARTIE C
}
```

Q11 Écrire le code correspondant à la **partie A** de la fonction `sl_ajoute_valeur`.

Q12 Écrire le code correspondant à la **partie B** de la fonction `sl_ajoute_valeur`.

Q13 Écrire le code correspondant à la **partie C** de la fonction `sl_ajoute_valeur`.

Q14 L'algorithme des skip lists est-il de type *Monte-Carlo* ou *Las Vegas* ? Justifier.

Q15 En supposant une skip list idéale comme dans la Figure 2, où chaque niveau possède exactement la moitié des éléments du niveau inférieur idéalement répartis, quelle est la complexité en temps de la recherche d'un élément dans une skip list de taille n ? On acceptera une explication de haut niveau.

On considère désormais une skip list non idéale, où les éléments sont répartis de manière aléatoire avec une probabilité $p = \frac{1}{2}$ d'être présent dans le niveau supérieur.

Q16 Quelle est l'espérance du nombre d'éléments au niveau k ?

Q17 Quelle est l'espérance du niveau le plus haut d'une skip list de taille n ? (question très difficile).

On acceptera dans la suite le résultat suivant : cette espérance est proportionnelle à $\log_2(n)$.

Q18 Quel est en moyenne le nombre de nœuds parcourus horizontalement dans un niveau k lors d'une recherche dans une skip list de taille n ?

Q19 En déduire la complexité moyenne de la recherche dans une skip list de taille n.

3 Analyse syntaxique et tableaux associatifs

On souhaite construire une fonction `analyse` qui prend en paramètre une chaîne de caractères décrivant un tableau associatif sous la forme de couples `clef:valeur`.

Exemple :

```
analyse("{id:5,valeur:12,ok:0}")
```

Le format de la chaîne est défini par les règles suivantes :

- la chaîne commence par `{` et se termine par `}` ;
- chaque couple est de la forme `clef:valeur` ;
- les couples sont séparés par des virgules ;
- une clef est une suite non vide de lettres minuscules ;
- une valeur est un entier non signé sans zéro non significatif.

On considère les règles de production suivantes :

$$N \to 0 \mid \cdots \mid 9 \qquad M \to 1 \mid \cdots \mid 9 \qquad L \to a \mid \cdots \mid z$$

Q20 Écrire la règle de production C correspondant à la description d'une clef.

Q21 Écrire la règle de production V correspondant à la description d'une valeur.

On considère la grammaire suivante :

$$\begin{aligned} T &\to \{S\} \\ S &\to K \mid K, S \\ K &\to C : V \end{aligned}$$

Q22 Le mot `{id:5,valeur:12,ok:0}` appartient-il au langage engendré par cette grammaire ? Même question pour `{james:007}`.

Q23 Décrire un automate déterministe sans transition vide reconnaissant le langage décrit par la règle T.

Q24 Vérifier que les séquences `{code:102}`, `{v:0}` et `{a:1,b:2}` sont reconnues par l'automate. Dans quel état se trouve l'automate pour `{james:007}` ?

4 Grammaires et décidabilité

Dans cette partie, on considère Σ un alphabet fini. Le but est de prouver l'indécidabilité de plusieurs problèmes de décision sur les grammaires non contextuelles.

4.1 Le Problème de Correspondance de Post

Définition 1.1 (PCP). On appelle *domino* un couple de mots $(u, v) \in (\Sigma^\star)^2$, noté $\binom{u}{v}$.

Le *Problème de Correspondance de Post (PCP)* est le problème de décision suivant :

PCP

- **Entrée :** une liste $\left\langle \begin{pmatrix} u_1 \\ v_1 \end{pmatrix}, \ldots, \begin{pmatrix} u_n \\ v_n \end{pmatrix} \right\rangle$ de dominos ;
- **Question :** existe-t-il $k \in \mathbb{N}^\star$ et $(i_1, \ldots, i_k) \in [\![1, n]\!]^k$ tels que

$$u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k} \,?$$

Exemple 1.1. Considérons la liste de dominos :

$$\begin{pmatrix} bc \\ ca \end{pmatrix}, \begin{pmatrix} a \\ ab \end{pmatrix}, \begin{pmatrix} ca \\ a \end{pmatrix}, \begin{pmatrix} abc \\ c \end{pmatrix}.$$

Cette instance est positive. Une solution est donnée par la suite $(2, 1, 2, 4)$.

Q25 L'instance suivante possède-t-elle une solution ?

$$\begin{pmatrix} a \\ baa \end{pmatrix}, \begin{pmatrix} ab \\ aa \end{pmatrix}, \begin{pmatrix} bba \\ bb \end{pmatrix}.$$

Q26 Même question pour l'instance :

$$\begin{pmatrix} abc \\ ab \end{pmatrix}, \begin{pmatrix} ca \\ a \end{pmatrix}, \begin{pmatrix} acc \\ ba \end{pmatrix}.$$

Q27 Montrer que si une instance de PCP possède une solution, alors elle possède une infinité de solutions.

4.2 Problèmes de décision sur les grammaires non contextuelles

On considère les problèmes **InterVide**, **Ambig**, **Univ** et **Égal** définis classiquement sur les grammaires non contextuelles.

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

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

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

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

On admet que le problème PCP est indécidable.

Q28 Soit $f_1 : E_1 \to \mathbb{B}$ et $f_2 : E_2 \to \mathbb{B}$ deux problèmes de décision. Montrer que si $f_1 \leq f_2$ et si f_1 est indécidable, alors f_2 est indécidable.

Soit

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix}, \ldots, \begin{pmatrix} u_n \\ v_n \end{pmatrix}.$$

une instance de PCP. On se donne $\Delta = \{d_1, \ldots, d_n\}$ un ensemble de nouvelles lettres ($\Delta \cap \Sigma = \emptyset$) représentant nos n dominos, et on va travailler avec des grammaires sur l'alphabet $\Sigma' = \Sigma \cup \Delta$. Pour $L \subset (\Sigma')^*$, on note $\bar{L} = (\Sigma')^* \setminus L$ le complémentaire de L dans $(\Sigma')^*$. On définit les langages suivants sur Σ' : $L_u = \{u_{i_1} \ldots u_{i_k} a_{i_k} \ldots a_{i_1} | k \in \mathbb{N}^*, (i_1, \ldots, i_k) \in \{1, \ldots, n\}^k\}$
$L_v = \{v_{i_1} \ldots v_{i_k} a_{i_k} \ldots a_{i_1} | k \in \mathbb{N}^*, (i_1, \ldots, i_k) \in \{1, \ldots, n\}^k\}$

Q29 Donner une grammaire non contextuelle engendrant le langage L_u.

Q30 Si l'instance de PCP est positive, que dire de L_u et L_v ? La réciproque est-elle vraie ?

Q31 En déduire que PCP $\leq$ **InterVide**.

Q32 Quel langage est engendré par la grammaire suivante ?

$S \to U|V$
$U \to u_1 U a_1 | \ldots | u_n U a_n$
$U \to u_1 a_1 | \ldots | u_n a_n$
$V \to v_1 V a_1 | \ldots | v_n V a_n$
$V \to v_1 a_1 | \ldots | v_n a_n$

Q33 Montrer que PCP $\leq$ **Ambig**.

On considère les langages suivants : $L'_u = \{w a_{i_k} \ldots a_{i_1} | k \in \mathbb{N}^*$ et $w \in \Sigma^*, w \neq u_{i_1} \ldots u_{i_k}\}$
$L'_v = \{w a_{i_k} \ldots a_{i_1} | k \in \mathbb{N}^*$ et $w \in \Sigma^*, \neq v_{i_1} \ldots v_{i_k}\}$

Q34 Calculer $L'_u \cup \overline{\Sigma^\star \Delta^\star} \cup \{\varepsilon\}$.

Q35 Donner une grammaire engendrant $\overline{\Sigma^\star \Delta^\star}$.

Q36 Donner une grammaire engendrant L'_u.

Q37 En déduire que PCP $\leq$ **Univ**.

Q38 Montrer que **Égal** est indécidable.

B. Annexe : règles de la déduction naturelle

Dans les tableaux suivants, la lettre Δ désigne un ensemble de formules de logique ; les lettres A, B et C désignent des formules de logique.

Axiome
$\dfrac{}{\Delta, A \vdash A}\ (\text{ax})$

	Introduction	Élimination
$\to$	$\dfrac{\Delta, A \vdash B}{\Delta \vdash A \to B}\ (\to\text{i})$	$\dfrac{\Delta \vdash A \qquad \Delta \vdash A \to B}{\Delta \vdash B}\ (\to\text{e})$
$\wedge$	$\dfrac{\Delta \vdash A \qquad \Delta \vdash B}{\Delta \vdash A \wedge B}\ (\wedge\text{i})$	$\dfrac{\Delta \vdash A \wedge B}{\Delta \vdash A}\ (\wedge\text{e})$ $\dfrac{\Delta \vdash A \wedge B}{\Delta \vdash B}\ (\wedge\text{e})$
$\vee$	$\dfrac{\Delta \vdash A}{\Delta \vdash A \vee B}\ (\vee\text{i})$ $\dfrac{\Delta \vdash B}{\Delta \vdash A \vee B}\ (\vee\text{i})$	$\dfrac{\Delta \vdash A \vee B \qquad \Delta, A \vdash C \qquad \Delta, B \vdash C}{\Delta \vdash C}\ (\vee\text{e})$
$\neg$	$\dfrac{\Delta, A \vdash B \qquad \Delta, A \vdash \neg B}{\Delta \vdash \neg A}\ (\neg\text{i})$	$\dfrac{\Delta \vdash A \qquad \Delta \vdash \neg A}{\Delta \vdash B}\ (\neg\text{e})$