COMPOSITION D'INFORMATIQUE n°6

(Durée : 4 heures)

L'utilisation de la calculatrice **n'est pas autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Le sujet est constitué de trois problèmes indépendants. Il est recommandé d'accorder environ 2h30 à la première partie, 30 minutes à la deuxième partie et 1h à la troisième partie.

1 Inférence d'automates

Dans ce problème, on considère un alphabet Σ. Un **automate fini non déterministe** est un quadruplet $A = (Q, \Delta, q_0, F)$ où Q est un ensemble d'**états**, $q_0 \in Q$ est l'état **initial**, F est l'ensemble des états **finaux** et $\Delta : Q \times \Sigma \to \mathcal{P}(Q)$ est la fonction de **transition**.

On étend naturellement Δ à la fonction $\Delta : \mathcal{P}(Q) \times \Sigma \to \mathcal{P}(Q)$ où pour $X \subseteq Q$ et $a \in \Sigma$, $\Delta(X, a) = \bigcup_{q \in X} \Delta(q, a)$. On définit la fonction de transition **étendue** Δ^* par induction sur les mots par :

- pour $X \subseteq Q$, $\Delta^*(X, \varepsilon) = X$;
- pour $X \subseteq Q$, $u \in \Sigma^*$ et $a \in \Sigma$, $\Delta^*(X, ua) = \Delta(\Delta^*(X, u), a)$.

On dit que $u \in \Sigma^*$ est **reconnu** par A si $\Delta^*(q_0, u) \cap F \neq \emptyset$. On note $L(A)$ l'ensemble des mots reconnus par A.

Enfin, on dit que A est **déterministe** si pour tout $q \in Q$, $a \in \Sigma$, $|\Delta(q, a)| \leqslant 1$.

1.1 Automate inféré

On souhaite créer un modèle de reconnaissance de mot qui reconnait tous les mots parmi un échantillon de mots dits **positifs** et ne reconnait aucun mot parmi un échantillon de mots dits **négatifs**. Un tel modèle peut s'appliquer à la reconnaissance de protéines pour détecter des maladies ou des mutations.

Formellement, on veut résoudre le problème suivant : `AUTOMATE INFÉRÉ` :

* **Instance** : deux langages finis de Σ^* disjoints $L_+ = \{u_1, \ldots, u_m\}$ et $L_- = \{v_1, \ldots, v_n\}$.
* **Solution** : un automate fini inféré par L_+ et L_-, c'est-à-dire un automate fini déterministe A tel que $L_+ \subseteq L(A)$ et $L_- \cap L(A) = \emptyset$.
* **Optimisation** : minimiser le nombre d'états de A.

On admet que ce problème est NP-difficile.

Question 1 À quelle catégorie d'apprentissage automatique appartient un algorithme qui résout ce problème ?

Question 2 Montrer que pour L_+ et L_- des langages finis disjoints, des automates finis déterministes reconnaissant L_+ et $\overline{L_-}$ respectivement sont des automates inférés par L_+ et L_-.

Question 3 Quel problème lié à l'apprentissage automatique peut se poser si on utilise les automates finis déterministes de la question précédente ? Justifier.

Pour $L \subseteq \Sigma$ un langage **fini**, on appelle **automate arborescent** de L, l'automate $T(L)$ défini par induction de la manière suivante :

- on pose $a_1, a_1, \ldots, a_k$ les éléments de Σ tels que $a_i\Sigma^* \cap L \neq \emptyset$;
- on pose $L_i = \{u \in \Sigma^* \mid a_i u \in L\}$ pour tout $i \in [\![1, k]\!]$;

– on pose $A_i = (Q_i, \Delta_i, q_i, F_i) = T(L_i)$ pour tout $i \in [\![1, k]\!]$;
– on pose finalement $T(L) = (Q, \Delta, q_0, F)$ où :
  * $Q = \{q_0\} \cup \bigcup_{i=1}^{k} Q_i$;
  * $F = \begin{cases} \{q_0\} \cup \bigcup_{i=1}^{k} Q_i & \text{si } \varepsilon \in L \\ \bigcup_{i=1}^{k} Q_i & \text{sinon} \end{cases}$;
  * pour $i \in [\![1, k]\!]$, $q \in Q_i$ et $a \in \Sigma$, $\Delta(q, a) = \Delta_i(q, a)$;
  * pour $i \in [\![1, k]\!]$, $\Delta(q_0, a_i) = q_i$

Par exemple, la figure 1 montre un automate $T(L)$ pour $L = \{a, aaa, bba, baaa\}$.

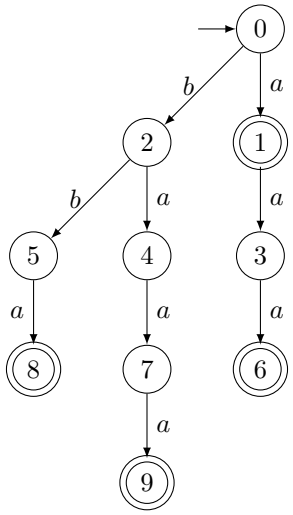


FIGURE 1 – L'automate $T(\{a, aaa, bba, baaa\})$.

Question 4 La définition de $T(L)$ par induction précédente ne comporte pas d'initialisation. Est-ce un problème ? Pourquoi ?

Pour les trois questions suivantes, on considère $L_+ = \{aaa, ab, ba, baa, bab\}$ et $L_- = \{a, b, aba\}$.

Question 5 Construire les automates arborescents de L_+ et L_-.

Question 6 Montrer qu'il existe un automate fini déterministe complet à 3 états inféré par L_+ et L_-.
Indication : on cherchera un automate avec deux états finaux et un état non final.

Question 7 Montrer qu'il n'existe pas d'automate fini déterministe complet à 2 états inféré par L_+ et L_-.

Question 8 Montrer que pour L un langage fini non vide quelconque, $L = L(T(L))$.

1.2 Implémentation en C

On représente Σ en C par $\{0, 1, \ldots, |\Sigma| - 1\}$. Une lettre de Σ sera donc simplement représentée par un objet de type `int` correspondant à sa valeur. On suppose créée une variable globale `N` correspondant à la valeur de $|\Sigma|$.

On représente un état d'un automate fini **déterministe** par le type suivant :

```c
typedef struct Etat {
    bool final;
    struct Etat** Delta;
} etat;
```

tel que si $A = (Q, \Delta, q_0, F)$ est un automate et $q \in Q$ est un objet représenté par `q` de type `etat*`, alors :

- `q->final` est un booléen qui vaut `true` si et seulement si $q \in F$;
- `q->Delta` est un tableau de taille `N` tel que pour $a \in \Sigma$, `q->Delta[a]` est soit un pointeur `NULL` si $\Delta(q, a) = \emptyset$, soit un pointeur vers l'unique état de $\Delta(q, a)$.

Dès lors, un automate fini déterministe est simplement représenté par un pointeur vers son état initial.

Question 9 Écrire une fonction `etat* creer_etat(void)` sans argument qui crée un état non final sans transition sortante.

Question 10 Écrire une fonction `void liberer_etat(etat* q)` qui prend en argument un pointeur vers un état et libère la mémoire allouée pour cet état et **pour tous les états accessibles depuis** q. On supposera que l'automate ne contient pas de cycle et que chaque état ne possède qu'au plus un seul antécédent par la fonction de transition.

Question 11 Quel problème pourrait se produire en faisant appel à la fonction précédente dans un automate qui contient des cycles ? Justifier.

On représente un mot $u \in \Sigma^*$ par un tableau d'entiers de taille $|u|$.

Question 12 Écrire une fonction `bool reconnu(etat* q0, int* u, int n)` qui prend en argument l'état initial q_0 d'un automate A, un mot u et sa taille n et renvoie un booléen qui vaut `true` si et seulement si $u \in L(A)$.

Question 13 Écrire une fonction `etat* arborescent(int** L, int* tailles, int k)` qui prend en argument un tableau de k mots $L = \{u_0, \ldots, u_{k-1}\}$, un tableau de k entiers correspondant à $\{|u_0|, \ldots, |u_{k-1}|\}$ et l'entier k et renvoie $T(L)$.

Question 14 Déterminer la complexité temporelle de la fonction précédente (en fonction des arguments qui seront jugés pertinents pour l'exprimer).

1.3 Fusion déterministe

Pour $A = (Q, \Delta, q_0, F)$ un automate fini (déterministe ou non) dont tous les états sont accessibles et $p, q \in Q$, on appelle **contraction** de p et q dans A, noté $A/\{p, q\}$, l'automate fini non déterministe où les états p et q ont été fusionnés et toutes les transitions partant ou arrivant de l'un ou de l'autre ont été conservées. L'état résultat de la fusion de p et q est final (resp. initial) si et seulement si au moins l'un des deux est final (resp. initial). La figure 2 donne un exemple de la contraction $A/\{0, 2\}$ où A est l'automate de la figure 1.

L'algorithme `DetMerge` (pour *Deterministic Merge*) consiste alors à faire des contractions dans l'automate jusqu'à ce qu'il redevienne déterministe. On la définit par :

Entrée : Automate déterministe $A = (Q, \Delta, q_0, F)$ et $p, q \in Q$.
Début algorithme
  $A' \leftarrow A/\{p, q\}$.
  Tant que $A' = (Q', \Delta', q_0', F')$ n'est pas déterministe **Faire**
    Choisir $(r, a) \in Q' \times \Sigma$ tels que $|\Delta'(r, a)| > 1$.
    Choisir $\{s, t\} \subseteq \Delta'(r, a)$.
    $A' \leftarrow A'/\{s, t\}$.
Renvoyer A'.

En reprenant l'exemple de la figure 2, la figure 3 illustre `DetMerge`$(A, 0, 2)$:

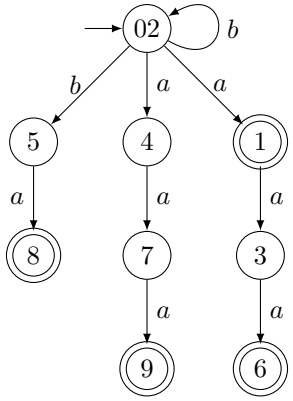


FIGURE 2 – La contraction $A/\{0,2\}$.

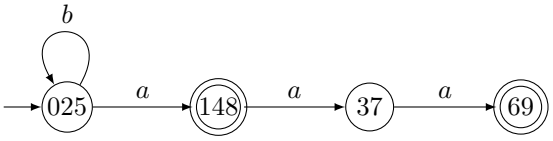


FIGURE 3 – L'automate $\texttt{DetMerge}(A, 0, 2)$.

Question 15 Déterminer $\texttt{DetMerge}(A, 0, 1)$ pour A l'automate de la figure 2. On représentera les contractions successives.

Question 16 Montrer que l'algorithme `DetMerge` termine.

Question 17 Montrer que si $A = (Q, \Delta, q_0, F)$ est un automate non déterministe, et que $(p, q) \in Q^2$, alors $L(A) \subseteq L(A/\{p, q\})$. En déduire que $L(A) \subseteq L(\texttt{DetMerge}(A, p, q))$.

Remarque

On admet que dans notre cadre, l'algorithme `DetMerge` renvoie toujours le même automate, peu importe les choix de (r, a) et $\{s, t\}$ qui sont faits à l'intérieur de la boucle **Tant que**.

1.4 Algorithme RPNI

L'algorithme RPNI (pour *Regular Positive and Negative Inference*) consiste à partir d'un automate reconnaissant L_+ et à effectuer des fusions-déterministes autant que possible, tout en s'assurant qu'aucun mot de L_- n'est reconnu. On peut le décrire par :

Entrée : langages finis et disjoints L_+ et L_-.
Début algorithme
  $A = (Q, \Sigma, \delta, q_0, F) \leftarrow T(L_+)$.
  Pour $\{p, q\} \in \mathcal{P}_2(Q)$ **Faire**
    $A' \leftarrow \texttt{DetMerge}(A, p, q)$.
    Si aucun mot de L_- n'est reconnu par A' **Alors**
      $A \leftarrow A'$.
Renvoyer A

où on suppose que $\texttt{DetMerge}(A, p, q)$ vaut A si p et q sont déjà dans le même état fusionné.

On suppose, pour les deux questions suivantes, que $L_+ = \{a, aaa, bba, baaa\}$ et $L_- = \{aaaa, baab, bbabab\}$. On remarque que l'automate A de la figure 2 est l'automate arborescent $T(L_+)$.

Question 18 Montrer que la fusion-déterministe `DetMerge`$(A, 0, 1)$ ne permet pas de continuer l'algorithme RPNI.

Question 19 Continuer l'exécution de l'algorithme RPNI depuis `DetMerge`$(A, 0, 2)$, l'automate donnée en figure 3. On représentera les automates obtenus après chaque appel à `DetMerge`.

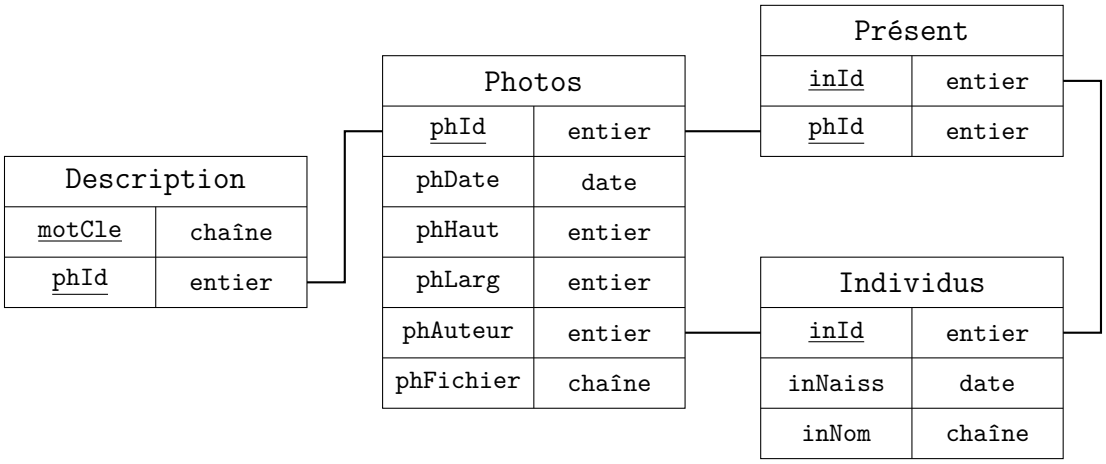
On revient au cas général du problème.

Question 20 Montrer que l'automate obtenu par l'algorithme RPNI est un automate inféré par L_+ et L_-.

Question 21 Déterminer la complexité temporelle de l'algorithme RPNI. On détaillera les choix de structures de données effectués.

2 Base de données photographique

On dispose d'une base de données répertoriant des photos avec diverses informations. La base données est représentée par le schéma ci-dessous :



La clé primaire de chaque table est signalée par un soulignement des attributs concernés.

On a pour chaque table représentant une entité la description de ses attributs :

- `Photos` :
  - `phId` : identifiant de la photo
  - `phDate` : date et heure de la photo (on supposera le type `date` compatible avec les opérations de comparaison, le minimum, le maximum)
  - `phHaut`, `phLarg` : dimensions de la photo en pixels
  - `phAuteur` : identifiant de la personne ayant pris la photo
  - `phFichier` : nom du fichier contenant la photo
- `Individus` :
  - `inId` : identifiant de l'individu
  - `inNaiss` : date de naissance de l'individu
  - `inNom` : nom de l'individu

Les tables `Présent` et `Description` représentent des relations entre les entités. `Présent` indique les personnes apparaissant sur les photos, et `Description` indique les mots-clé associés aux photos.

Question 22 Proposer un diagramme entité-association illustrant les informations présentes dans la base. On précisera la cardinalité des associations.

Question 23 En justifiant à l'aide des clés primaires, déterminer

1. si une personne peut apparaitre sur plusieurs photos,
2. si plusieurs personnes peuvent apparaitre sur la même photo.

Question 24 Écrire une requête donnant les identifiants des photos dont le format est **exactement** 16/9.

Question 25 Écrire une requête donnant les noms des individus apparaissant dans au moins un *selfie*, c'est-à-dire une photo dont ils sont l'auteur.

Question 26 Écrire une requête donnant les identifiants des photos sur lesquelles personne n'apparait.

Question 27 Écrire une requête donnant les noms des fichiers contenant des photos décrites par au moins 5 mots-clé.

3 Logique

3.1 Quelques séquents à prouver

Pour chacune des questions suivantes, utiliser les règles de la déduction naturelle (rappelées en annexe) pour construire un arbre de preuve ayant le séquent spécifié comme conclusion.

Question 28 $\varphi \vdash \neg\neg\varphi$

Question 29 $\neg(\varphi \vee \psi) \vdash \neg\varphi \wedge \neg\psi$

Question 30 $\forall x.\varphi \vdash \neg(\exists x.\neg\varphi)$

3.2 Logique et typage

On s'intéresse au typage des expressions dans un sous-ensemble du langage OCaml. On note $\vdash e : \tau$ pour dire que l'expression e admet τ comme type (on parle alors de **jugement de typage**). On a donc par exemple $\vdash$ `123` : `int` et $\vdash$ `1 :: [3; 4]` : `int list`.

Question 31 Quel jugement de typage obtiendrait-on pour l'expression `List.map (fun x -> x + 1)` ?

Pour pouvoir typer des expressions contenant des variables libres, il est nécessaire d'avoir un contexte donnant le type de chacune de ces variables. On étend donc la notion de jugement pour inclure un contexte Γ, qui est un ensemble de jugements de la forme $x : \tau$ où les x sont des **variables distinctes** et τ un type. Par exemple :

$$\{x : \texttt{int}, l : \texttt{int list}, f : \texttt{int -> int}\} \vdash \texttt{((f x) :: l)} : \texttt{int list}$$

On définit alors un système d'inférence pour les jugements de typage, constitué de :

– pour chaque constante c du langage de type τ, on dispose d'un axiome :

$$\frac{}{\Gamma \vdash c : \tau}\ \text{ax}$$

par exemple, $\vdash$ `true` : `bool` et $\vdash$ `2.14` : `float` sont des axiomes ;

– une autre règle d'axiome, que l'on notera (C) pour « contexte »

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ \text{C}$$

– une règle $(\to_i)$

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash (\texttt{fun}\ x \to e) : \sigma \to \tau}\ \to_i$$

– une règle $(\to_e)$

$$\frac{\Gamma \vdash f : \sigma \to \tau \qquad \Gamma \vdash e : \sigma}{\Gamma \vdash f\ e : \tau}\ \to_e$$

Question 32 On considère l'environnement de typage $\Gamma = \{f : \alpha \to (\beta \to \gamma), g : \beta \to \alpha\}$. Dériver le jugement suivant : $\Gamma, x : \beta \vdash g\ x : \alpha$.

Question 33 En déduire que, pour le même Γ, on a $\Gamma \vdash (\texttt{fun}\ x \to f(g\ x)\ x) : \beta \to \gamma$.

On souhaite à présent prouver que l'expression $e = (\texttt{fun}\ h \to h\ 1\ 2)(\texttt{fun}\ x \to 3)$ n'est pas typable, c'est-à-dire qu'il n'existe pas de contexte Γ et de type τ tels que $\Gamma \vdash e : \tau$. On procède par l'absurde et on suppose qu'il existe un tel contexte.

Question 34 Justifier qu'on a nécessairement $\Gamma \vdash (\texttt{fun}\ h \to h\ 1\ 2) : \alpha \to \tau$ et $\Gamma \vdash (\texttt{fun}\ x \to 3) : \alpha$ pour un certain type α.

Question 35 En considérant la dernière étape de la dérivation de $\Gamma \vdash (\texttt{fun}\ x \to 3) : \alpha$, dire quelle forme doit avoir le type α.

Question 36 En s'intéressant au jugement $\Gamma \vdash (\texttt{fun}\ h \to h\ 1\ 2) : \alpha \to \tau$, conclure (on admettra que `int` ne peut pas se mettre sous la forme $\gamma \to \delta$).

$\star\star\star$

Annexe : règles de la déduction naturelle

Introduction et élimination

Opérateur	Introduction	Élimination
Implication $\to$	$\dfrac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi} \to_i$	$\dfrac{\Gamma \vdash \varphi \to \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi} \to_e$
Conjonction $\wedge$	$\dfrac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi} \wedge_i$	$\dfrac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi} \wedge_e^g$ et $\dfrac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi} \wedge_e^d$
Disjonction $\vee$	$\dfrac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi} \vee_i^g$ et $\dfrac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi} \vee_i^d$	$\dfrac{\Gamma \vdash \varphi \vee \psi \quad \Gamma, \varphi \vdash \sigma \quad \Gamma, \psi \vdash \sigma}{\Gamma \vdash \sigma} \vee_e$
Négation $\neg$	$\dfrac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi} \neg_i$	$\dfrac{\Gamma \vdash \varphi \quad \Gamma \vdash \neg\varphi}{\Gamma \vdash \bot} \neg_e$
Atomiques $\top$ et $\bot$	$\dfrac{}{\Gamma \vdash \top} \top_i$	$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash \varphi} \bot_e$
Existentiel $\exists$	$\dfrac{\Gamma \vdash \varphi[x := t]}{\Gamma \vdash \exists x \varphi} \exists_i$	$\dfrac{\Gamma \vdash \exists x\, \varphi \quad \Gamma, \varphi \vdash \psi \quad x \notin V_F(\Gamma) \cup V_F(\psi)}{\Gamma \vdash \psi} \exists_e$
Universel $\forall$	$\dfrac{\Gamma \vdash \varphi \quad x \notin V_F(\Gamma)}{\Gamma \vdash \forall x\, \varphi} \forall_i$	$\dfrac{\Gamma \vdash \forall x\, \varphi}{\Gamma \vdash \varphi[x := t]} \forall_e$

Autres règles

Nom de la règle	Description
Axiome	$\dfrac{}{\Gamma, \varphi \vdash \varphi}$ ax
Affaiblissement	$\dfrac{\Gamma \vdash \varphi}{\Gamma, \psi \vdash \varphi}$ aff
Raisonnement par l'absurde	$\dfrac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}$ raa