Les calculatrices sont interdites.

N.B. : Le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction.
Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

PRÉAMBULE : Les deux parties qui composent ce sujet sont indépendantes et peuvent être traitées par les candidats dans un ordre quelconque.

Partie I : Logique et calcul des propositions

Dans L'Égypte ancienne, la protection de la tombe des Pharaons faisait l'objet d'une grande attention. Le sarcophage était porté dans la salle funéraire par des esclaves accompagnés d'un prêtre. Le couloir y conduisant était définitivement condamné après leur passage. Les esclaves étaient ensuite abandonnés dans la salle funéraire et le prêtre quittait le tombeau par un passage secret. Pour éviter que les esclaves et les éventuels pillards puissent emprunter le même passage, celui-ci était fermé par plusieurs portes dont l'ouverture demandait la réponse à des énigmes. Pour empêcher qu'une autre personne que le prêtre ne puisse trouver la réponse aux énigmes, celles-ci étaient régies par des règles propres à chaque tombe et connues uniquement du prêtre et de son ordre.

Vous faites partie d'une équipe d'archéologues qui explore un tombeau récemment découvert grâce au déchiffrage d'un manuscrit écrit en hiéroglyphes contenant l'emplacement du tombeau et la précieuse règle nécessaire à la résolution des énigmes : « *Chaque énigme sera composée de trois affirmations. Une affirmation parmi les trois sera toujours fausse, les deux autres seront toujours vraies. Attention, ce ne seront pas forcément toujours les mêmes.* »

Votre équipe a réussi à déblayer l'entrée de la salle funéraire et vous vous y êtes précipités trop imprudemment. En effet, la galerie, mal étayée, s'est effondrée après votre passage. Vous ne disposez pas de suffisamment d'air pour attendre que vos collègues dégagent à nouveau le passage. Votre seule chance de survie est d'emprunter le passage secret usuellement réservé au prêtre.

Nous noterons A_1, A_2 et A_3 les propositions associées aux trois affirmations contenues dans les énigmes apparaissant sur chaque porte.

Question I.1 *Représenter la règle sous la forme d'une formule du calcul des propositions dépendant de A_1, A_2 et A_3.*

Une première porte bloque le passage. Devant cette porte se trouvent une dalle blanche et une dalle noire. Les affirmations suivantes sont inscrites sur la porte :

- Si tu poses le pied sur la dalle blanche, alors pose le pied sur la dalle noire !
- Pose les pieds simultanément sur les dalles blanche et noire !
- Pose le pied sur la dalle noire !

Nous noterons B, respectivement N, les variables propositionnelles correspondant au fait de poser le pied sur la dalle blanche, respectivement noire.

Question I.2 *Exprimer A_1, A_2 et A_3 sous la forme de formules du calcul des propositions dépendant de B et de N.*

Question I.3 *En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer la (ou les) dalle(s) sur laquelle (ou lesquelles) vous devez posez les pieds pour ouvrir cette porte.*

La porte s'ouvre, puis se referme après votre passage. Une seconde porte se trouve maintenant face à vous. Sur le côté de la porte se trouvent trois pavés de tailles différentes (petit, moyen, gros). Trois affirmations sont inscrites sur la porte. Malheureusement, un des hiéroglyphes est illisible. Voici les textes que vous arrivez à déchiffrer :

- L'affirmation suivante est fausse : N'appuie sur le petit pavé que si tu as appuyé sur le gros pavé !
- N'appuie pas sur le moyen pavé mais appuie sur le gros pavé !
- N'appuie ni sur le gros pavé, ni sur le ... pavé !

Les hiéroglyphes vous permettent de savoir que ... correspond soit à gros, soit à moyen, soit à petit.

Nous noterons P, M et G les variables propositionnelles correspondant au fait d'appuyer sur le petit, le moyen ou le gros pavé.

Question I.4 *Exprimer A_1, A_2 et A_3 sous la forme de formules du calcul des propositions dépendant de P, M et G.*

Question I.5 *En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer le (ou les) pavé(s) sur lequel (ou lesquels) vous devez appuyer pour ouvrir cette porte.*

Partie II : Algorithmique et programmation en CaML

Cette partie doit être traitée par les étudiants qui ont utilisé le langage CaML dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`for`, `while`, ...) ni de références.

L'explication du comportement d'une fonction (demandée aux questions II.8, II.13, II.16 et II.23) doit au moins contenir :

- L'objectif général ;
- Le rôle des paramètres de la fonction ;
- Les contraintes sur les valeurs des paramètres ;
- Les caractéristiques du résultat renvoyé par la fonction ;
- Le rôle des variables locales à la fonction ;
- Le principe de l'algorithme ;
- Des arguments de terminaison du calcul quelles que soient les valeurs des paramètres.

L'analyse lexicale est la première étape effectuée par les outils d'analyse de texte et, en particulier, de programmes (compilateur, interprète, ...). Elle vise à reconnaître les différents mots composant une phrase sans s'attacher à la structure de la phrase elle-même. La technique la plus couramment employée pour implanter un analyseur lexical repose sur un automate fini dont les transitions sont étiquetées par les caractères composant les mots du vocabulaire. Les états initiaux correspondent aux débuts des mots. Les états terminaux correspondent à la fin des mots. Un mot est reconnu par un parcours de l'automate guidé par les caractères composant ce mot.

Les automates finis peuvent être déterministes ou indéterministes. Ces deux formes présentent chacune des avantages et des inconvénients au niveau des performances :

- le parcours d'un automate déterministe est peu coûteux alors que son occupation mémoire est coûteuse ;
- l'occupation mémoire d'un automate indéterministe est peu coûteuse alors que son parcours est coûteux.

L'outil `lex`, le plus couramment utilisé pour construire des analyseurs de programmes, exploite des automates semi-indéterministes pour obtenir un bon compromis entre les coûts du parcours et de l'occupation mémoire.

Nous allons étudier une catégorie d'automates semi-indéterministes, ses propriétés, ainsi qu'une technique de construction.

1 Représentation et codage d'un automate fini

1.1 Représentation d'un automate fini

Déf. II.1 (Automate fini) *Soit l'alphabet X, un ensemble de symboles, soit ϵ le symbole représentant la transition arbitraire ($\epsilon \notin X$), soit Λ le symbole représentant le mot vide ($\Lambda \notin X$), soit $X^\star$ l'ensemble des mots composés de symboles de X ($\Lambda \in X^\star$); un automate fini sur X est un quintuplet*

$A = (Q,X,I,T,\gamma)$ *composé de :*

- *Un ensemble fini d'états :* Q *;*
- *Un ensemble d'états initiaux :* $I \subseteq Q$ *;*
- *Un ensemble d'états terminaux :* $T \subseteq Q$ *;*
- *Une relation de transition confondue avec son graphe :* $\gamma \subseteq Q \times (X \cup \{\epsilon\}) \times Q$.

Les symboles ϵ et Λ sont souvent confondus. Le premier est spécifique aux automates, il correspond à une transition sans lecture de caractères (le mot lu est donc le mot vide Λ). Le second correspond au mot vide indépendamment des automates.

Pour une transition (o,e,d) donnée, nous appelons o l'origine de la transition, e l'étiquette de la transition et d la destination de la transition.

Remarquons que γ est le graphe d'une fonction totale de transition $\delta : Q \times (X \cup \{\epsilon\}) \to \mathcal{P}(Q)$ dont les valeurs sont définies par :

$$\forall o \in Q, \forall e \in X \cup \{\epsilon\}, \delta(o,e) = \{d \in Q \mid (o,e,d) \in \gamma\}$$

La notation γ est plus adaptée que δ à la formalisation et la construction des preuves.

1.2 Représentation graphique d'un automate

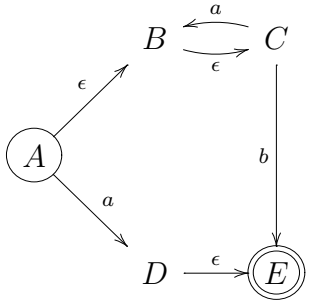
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la relation de transition γ sont représentées par un graphe dont les nœuds sont les états et les arêtes sont les transitions ;
- un état initial est entouré d'un cercle (i) ;
- un état terminal est entouré d'un double cercle $((t))$;
- une arête étiquetée par le symbole $e \in X \cup \{\epsilon\}$ va de l'état o à l'état d si et seulement si $(o,e,d) \in \gamma$.

Exemple II.1 *L'automate* $A = (Q,X,I,T,\gamma)$ *avec :*

$$\begin{aligned} Q &= \{A,B,C,D,E\} \\ X &= \{a,b\} \\ I &= \{A\} \\ T &= \{E\} \\ \gamma &= \{(A,\epsilon,B),(B,\epsilon,C),(C,a,B),(C,b,E),(A,a,D),(D,\epsilon,E)\} \end{aligned}$$

est représenté par le graphe suivant :



Question II.1 *Donner sans la justifier une expression régulière ou ensembliste représentant le langage reconnu par* A.

1.3 Langage reconnu par un automate fini

$X^\star$ est l'ensemble des mots composés des symboles de X. Nous étendons l'opérateur . de construction de mots pour prendre en compte le symbole ϵ apparaissant dans les transitions des automates :

$$\forall m \in X^\star,\ \epsilon.m = m$$

Soit $\gamma^\star$ l'extension de γ à $Q \times X^\star \times Q$ définie par :

$$\begin{cases} \forall q \in Q,\ (q,\Lambda,q) \in \gamma^\star \\ \forall e \in X \cup \{\epsilon\},\ \forall m \in X^\star,\ \forall o \in Q,\ \forall d \in Q,\ (o,e.m,d) \in \gamma^\star \Leftrightarrow \exists q \in Q,\ (o,e,q) \in \gamma \wedge (q,m,d) \in \gamma^\star \end{cases}$$

Le langage sur $X^\star$ reconnu par un automate fini est :

$$L(A) = \{m \in X^\star \mid o \in I,\ d \in T,\ (o,m,d) \in \gamma^\star\}$$

L'analyse d'un mot par un automate peut être caractérisée par le chemin suivi dans l'automate, c'est-à-dire la séquence des états parcourus. En général, dans le cadre d'un automate indéterministe, un même mot peut être reconnu en suivant plusieurs chemins différents dans l'automate.

1.4 Codage en CaML d'un automate fini

Nous considérerons pour les besoins du codage que tous les automates exploitent le même ensemble fini de symboles $X \subset \mathbb{N}$. Nous ne préciserons donc pas cet ensemble dans la suite des fonctions réalisées.

Pour les variables ou les constantes dont le nom est pris dans l'alphabet grec (ϵ, γ, ...), l'identifiant en CaML sera leur nom en toute lettre (`epsilon`, `gamma`, ...).

```
type etat == int;;
```

Un état est représenté par le type de base `int`.

```
type etats == etat list;;
```

Un ensemble d'états est représenté par le type `etats` équivalent à une liste d'états.

```
type etiquette == int;
```

Une étiquette de transition (appartenant à $X \cup \{\epsilon\}$) est représentée par le type de base `int`.

```
let epsilon = -1;;
```

L'étiquette particulière ϵ a la valeur `-1`. Elle est contenue dans la variable `epsilon`.

```
type transition == etat * etiquette * etat;;
```

Une transition est représentée par le type `transition` équivalent à un triplet composé d'un état, d'une étiquette et d'un état.

```
type relation == transition list;;
```

Une relation est représentée par le type `relation` équivalent à une liste de transitions.

```
type paire == relation * relation;;
```

Une paire de relations est représentée par le type `paire`.

```
type automate == etats * etats * etats * relation;;
```

Un automate est représenté par le type `automate` équivalent à un quadruplet composé de trois ensembles d'états et d'une relation.

```
type mot == etiquette list;;
```

Un mot accepté par l'automate est représenté par le type `mot` équivalent à une liste d'étiquettes.

Exemple II.2 *Associons les entiers suivants aux symboles de Q : $A \mapsto 0$,$B \mapsto 1$,$C \mapsto 2$, $D \mapsto 3$,$E \mapsto 4$ et de X : $a \mapsto 0$,$b \mapsto 1$.*

L'automate de l'exemple II.1 sera représenté par la valeur :

```
let Q = [ 0; 1; 2; 3; 4 ] in                (* ensemble d'etats *)
let I = [ 0 ] in                            (* idem             *)
let T = [ 4 ] in                            (* idem             *)

let G = [ (0,-1,1) ; (1,-1,2) ; (2,0,1) ;   (* ensemble de *)
      (2,1,4) ; (0,0,3) ; (3,-1,4) ] in     (* transitions *)

( Q, I, T, G );;                            (* automate *)
```

Exemple II.3 *Le mot $aaab$ du langage reconnu par l'automate donné dans l'exemple II.1, sera représenté par la valeur :*

```
[ 0 ; 0 ; 0 ; 1 ]   (* liste d'etiquettes *)
```

2 Codage des ensembles d'états et de transitions

La représentation d'un automate fini et les algorithmes associés exploitent les structures d'ensemble d'états et de transitions. Dans cette partie, nous allons étudier une représentation des ensembles à base de listes et certains algorithmes de manipulation d'ensembles qui pourront être utilisés par la suite.

2.1 Codage de la structure d'ensemble

Un automate fini exploite deux formes d'ensembles :

– des ensembles d'états : Q, I, et T ;

– un ensemble de transitions : la relation γ.

Nous devons donc disposer d'un codage de la structure d'ensemble pour pouvoir représenter des automates finis. Nous allons utiliser un codage extrêmement simple : un ensemble est une liste contenant exactement un exemplaire de chaque valeur contenue dans l'ensemble. Les opérations de manipulation d'un ensemble devront préserver cette propriété.

Le parcours d'un ensemble sera donc effectué de la même manière que celui d'une liste.

Nous allons maintenant définir plusieurs opérations sur les ensembles qui pourront être utilisées dans la suite du sujet. Ces opérations sont indépendantes de la forme des valeurs rangées dans l'ensemble. Nous allons donc les définir pour un ensemble d'états, les définitions pour une relation seront identiques.

2.2 Opération sur la structure d'ensemble

Nous supposons que toutes les listes passées comme paramètre, qu'elles représentent un ensemble d'états ou de transitions, ne contiennent au plus qu'une fois chaque valeur.

Pour les calculs de complexité, nous noterons $\mid e \mid$ la taille de l'ensemble e, c'est-à-dire le nombre d'éléments qu'il contient.

2.2.1 Cardinalité d'un ensemble

Une première opération consiste à déterminer le nombre d'états appartenant à un ensemble.

Question II.2 *Écrire en CaML une fonction* `cardinal` *de type* `etats -> int` *telle que l'appel* `(cardinal e)` *renvoie le nombre d'états contenus dans l'ensemble* `e`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.3 *Calculer une estimation de la complexité de la fonction* `cardinal` *en fonction de la taille de l'ensemble* `e`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.2 Appartenance d'un état à un ensemble

Une deuxième opération consiste à tester l'appartenance d'un état à un ensemble.

Question II.4 *Écrire en CaML une fonction* `appartient` *de type* `etat -> etats -> boolean` *telle que l'appel* `(appartient v e)` *renvoie la valeur vrai si l'ensemble* `e` *contient l'état* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.5 *Calculer une estimation de la complexité dans le pire cas de la fonction* `appartient` *en fonction de la taille de l'ensemble* `e`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.3 Ajout d'une valeur dans un ensemble

La troisième opération est l'ajout d'un état à un ensemble.

Question II.6 *Écrire en CaML une fonction* `ajout` *de type* `etat -> etats -> etats` *telle que l'appel* `(ajout v e)` *renvoie un ensemble contenant les mêmes états que l'ensemble* `e` *ainsi que l'état* `v` *s'il ne figurait pas dans* `e`*. L'ensemble renvoyé contiendra exactement une fois l'état* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.4 Comparaison du contenu de deux ensembles

La quatrième opération est la comparaison du contenu de deux ensembles.

Question II.7 *Écrire en CaML une fonction* `egalite` *de type* `etats -> etats -> boolean` *telle que l'appel* `(egalite e1 e2)` *renvoie la valeur vraie si les deux ensembles* `e1` *et* `e2` *contiennent exactement les mêmes états. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.8 *Expliquer la fonction* `egalite` *proposée pour la question précédente.*

Question II.9 *Calculer une estimation de la complexité dans le pire cas de la fonction* `egalite` *en fonction des tailles des ensembles* `e1` *et* `e2`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.5 Union de deux ensembles

La cinquième opération étudiée est la construction d'un ensemble résultant de l'union de deux autres ensembles.

Question II.10 *Écrire en CaML une fonction* `union` *de type* `etats -> etats -> boolean` *telle que l'appel* `(union e1 e2)` *renvoie un ensemble contenant les états contenus dans* `e1` *et les états contenus dans* `e2`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.6 Intersection de deux ensembles

La dernière opération étudiée est la construction d'un ensemble résultant de l'intersection de deux autres ensembles.

Question II.11 *Écrire en CaML une fonction* `intersection` *de type* `etats -> etats -> etats` *telle que l'appel* `(intersection e1 e2)` *renvoie un ensemble contenant les états contenus à la fois dans* `e1` *et dans* `e2`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3 Exploitation des relations de transition

3.1 Suivants dans une relation

La construction d'automates semi-indéterministes repose sur la manipulation des relations de transition et, en particulier, sur la construction de fermetures transitives. Nous allons définir des opérations de parcours des relations qui pourront être utilisées par la suite.

Déf. II.2 (Suivants dans une relation) *Les suivants $\mathcal{S}(O,\gamma)$ d'un ensemble d'états O selon une relation γ sont les ensembles des destinations des transitions de γ dont les états de O sont les origines, indépendamment des étiquettes.*

$$\mathcal{S}(O,\gamma) = \{d \in Q \mid \exists o \in O, \exists e \in X \cup \{\epsilon\}, (o,e,d) \in \gamma\}$$

Exemple II.4 *Dans l'automate de l'exemple II.1, nous avons par exemple :*

$$\begin{array}{l}\mathcal{S}(\{A\},\gamma) = \{B,D\}\\ \mathcal{S}(\{C\},\gamma) = \{B,E\}\\ \mathcal{S}(\{B,D\},\gamma) = \{C,E\}\\ \mathcal{S}(\{E\},\gamma) = \emptyset\end{array}$$

Nous allons implanter une fonction calculant $\mathcal{S}(O,\gamma)$.

Question II.12 *Écrire en CaML une fonction* `suivants` *de type* `etats -> relation -> etats` *telle que l'appel* `(suivants O gamma)` *renvoie un ensemble d'états contenant les mêmes états que $\mathcal{S}(O,\gamma)$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.13 *Expliquer la fonction* `suivants` *proposée pour la question précédente.*

3.2 Etats accessibles

Nous allons étendre les opérations précédentes pour obtenir l'ensemble des états accessibles à partir d'un ensemble d'états donné en suivant plusieurs transitions sans prendre en compte les étiquettes.

Déf. II.3 (États accessibles) *L'ensemble des états accessibles $\mathcal{A}(O,\gamma)$ depuis les états de O selon la relation γ est défini par :*

$$\mathcal{A}(O,\gamma) = \{d \in Q \mid \exists o \in O, \exists m \in X^\star, (o,m,d) \in \gamma^\star\}$$

Exemple II.5 *Dans l'automate de l'exemple II.1, nous avons par exemple :*

$$\begin{array}{l}\mathcal{A}(\{A\},\gamma) = \{A,B,C,D,E\}\\ \mathcal{A}(\{C\},\gamma) = \{B,C,E\}\\ \mathcal{A}(\{B,D\},\gamma) = \{B,C,D,E\}\\ \mathcal{A}(\{E\},\gamma) = \{E\}\end{array}$$

Nous allons implanter une fonction calculant $\mathcal{A}(O,\gamma)$.

Question II.14 *Soient $O \subseteq Q$ un ensemble d'états, γ une relation sur Q, la suite (A_i) d'ensembles d'états ($A_i \subseteq Q$) est définie par :*

$$\left\{\begin{array}{l} A_0 = O\\ A_{i+1} = A_i \cup \mathcal{S}(A_i,\gamma)\end{array}\right.$$

1. *Montrer que la suite (A_i) est croissante pour la relation d'inclusion $\subseteq$;*
2. *Montrer qu'il existe un entier $k \in \mathbb{N}$ tel que $A_k = A_{k+1}$;*
3. *Montrer que $\forall i \in \mathbb{N},\ \mathcal{A}(A_i,\gamma) = \mathcal{A}(A_{i+1},\gamma)$;*
4. *Montrer que $\forall E \subseteq Q, \mathcal{S}(E,\gamma) \subseteq E \Rightarrow \mathcal{A}(E,\gamma) \subseteq E$;*
5. *Montrer que $\mathcal{A}(O,\gamma) = A_k$.*

Question II.15 *Écrire en CaML une fonction* `accessibles` *de type* `etats -> relation -> etats` *telle que l'appel* `(accessibles O gamma)` *renvoie un ensemble d'états contenant les mêmes états que $\mathcal{A}(O,\gamma)$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.16 *Expliquer la fonction* `accessibles` *proposée pour la question précédente.*

3.3 Construction de familles de transitions

Les opérations de construction des automates semi-indéterministes étudiées par la suite engendrent de nouvelles transitions. Pour simplifier leurs définitions, nous pouvons utiliser des fonctions de construction de familles de transitions similaires (par exemple, de transitions ayant la même origine, la même étiquette et plusieurs destinations distinctes).

Question II.17 *Écrire en CaML une fonction* `prefixe` *de type* `etat -> etiquette -> etats -> relation` *telle que l'appel* `(prefixe o e ds)` *renvoie une relation contenant les transitions d'origine* `o`*, d'étiquette* `e` *et de destination les états contenus dans* `ds`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4 Automate fini semi-indéterministe

En général, dans le cadre d'un automate fini, un même mot du langage accepté par un automate peut être reconnu par plusieurs parcours distincts de l'automate.

L'algorithme naturel de parcours de l'automate pour analyser un mot est donc indéterministe car il existe plusieurs possibilités de parcours.

4.1 Causes d'indéterminisme

Trois causes d'indéterminisme peuvent apparaître dans un automate fini :

- L'existence de plusieurs états initiaux ;
- L'existence de plusieurs transitions avec une même étiquette et une même origine ;
- L'existence de transitions arbitraires (d'étiquette ϵ).

Les algorithmes indéterministes de parcours sont souvent plus coûteux, il est donc pertinent d'envisager la déterminisation de l'automate d'analyse.

La construction d'un automate déterministe reconnaissant le même langage qu'un automate indéterministe donné peut faire apparaître un grand nombre de nouveaux états. L'occupation mémoire est donc souvent beaucoup plus coûteuse.

Dans un objectif de compromis entre occupation mémoire et temps de parcours, nous allons nous intéresser à des automates semi-indéterministes qui ne font pas apparaître la troisième forme d'indéterminisme. Nous étudierons par la suite un algorithme de construction qui n'introduit pas de nouveaux états.

4.2 Automate fini semi-indéterministe

Nous définissons une restriction des automates finis qui ne contient pas de transitions arbitraires (étiquette ϵ). Ces automates sont appelés semi-indéterministes car un degré d'indéterminisme est levé. Les définitions précédentes sont toujours valides en éliminant les transitions arbitraires.

Déf. II.4 (Automate semi-indéterministe) *Soit* $A = (Q,X,I,T,\gamma)$ *un automate fini,* A *est semi-indéterministe si et seulement si* $\gamma \subseteq Q \times X \times Q$.

Nous allons décomposer la relation de transition d'un automate fini quelconque en une relation arbitraire et une relation non-arbitraire que nous recomposerons ensuite pour obtenir un automate semi-indéterministe.

4.2.1 Relation de transition arbitraire

Déf. II.5 (Relation de transition arbitraire) *Soit Q un ensemble d'états, nous appelons relation de transition arbitraire γ_ϵ, une relation dont toutes les transitions sont étiquetées par ϵ, donc $\gamma_\epsilon \subseteq Q \times \{\epsilon\} \times Q$.*

4.2.2 Relation de transition non-arbitraire

Déf. II.6 (Relation de transition non-arbitraire) *Soient Q un ensemble d'états et X un ensemble de symboles, nous appelons relation de transition non-arbitraire γ_X, une relation dont toutes les transitions sont étiquetées par des symboles de X (donc différent de ϵ), donc $\gamma_X \subseteq Q \times X \times Q$.*

Nous noterons γ_X une relation de transition non-arbitraire car elle ne peut pas faire apparaître la troisième forme d'indéterminisme.

4.2.3 Décomposition d'une relation de transition

Théorème II.1 (Décomposition d'une relation de transition) *Soit l'automate fini $A = (Q,X,I,T,\gamma)$, la relation γ peut donc se décomposer de manière unique en deux sous-relations disjointes :*

- *une relation arbitraire γ_ϵ ;*
- *une relation non-arbitraire γ_X.*

Donc,

$$\gamma = \gamma_\epsilon \cup \gamma_X$$

Exemple II.6 *La relation de transition de l'automate de l'exemple II.1 se décompose en :*

$$\gamma_\epsilon = \{(A,\epsilon,B),(B,\epsilon,C),(D,\epsilon,E)\}$$
$$\gamma_X = \{(C,a,B),(C,b,E),(A,a,D)\}$$

Nous allons définir une fonction pour décomposer la relation de transition γ de l'automate en sous-relations arbitraire γ_ϵ et non-arbitraire γ_X.

Question II.18 *Écrire en CaML une fonction* `decompose` *de type* `relation -> paire` *telle que l'appel* `(decompose gamma)` *renvoie une paire contenant deux relations composées des transitions de la relation γ. La première contient les transitions arbitraires (d'étiquette ϵ) de γ. La seconde contient les transitions de γ d'étiquettes différentes de ϵ. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4.2.4 Fermeture d'une relation arbitraire

Déf. II.7 (Fermeture transitive d'une relation arbitraire) *Nous noterons $\vec{\gamma_\epsilon}$ la fermeture transitive de γ_ϵ définie par :*

$$\begin{cases} \forall q \in Q, (q,\epsilon,q) \in \vec{\gamma_\epsilon} \\ \forall q \in Q, \forall q' \in Q, (q,\epsilon,q') \in \vec{\gamma_\epsilon} \Leftrightarrow \exists q'' \in Q, (q,\epsilon,q'') \in \gamma_\epsilon \wedge (q'',\epsilon,q') \in \vec{\gamma_\epsilon} \end{cases}$$

Question II.19 *Construire $\vec{\gamma_\epsilon}$ à partir de la relation γ_ϵ de l'exemple II.6.*

Question II.20 *Calculer une borne supérieure de la taille de $\vec{\gamma_\epsilon}$ en fonction de la taille de γ_ϵ.*

Question II.21 *Montrer que :*

$$\vec{\gamma_\epsilon} = \{(o,\epsilon,d) \mid o \in Q, d \in \mathcal{A}(\{o\},\gamma_\epsilon)\}$$

Question II.22 *Écrire en CaML une fonction* `fermeture` *de type* `relation -> relation` *telle que l'appel* `(fermeture gamma)` *renvoie une relation contenant toutes les transitions engendrées par la fermeture transitive $\vec{\gamma}$ des transitions de la relation arbitraire γ. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.23 *Expliquer la fonction* `fermeture` *proposée pour la question précédente.*

4.2.5 Composition de relations arbitraires et non-arbitraires

Déf. II.8 (Opérateur de composition $\triangleright$) *Soient γ_ϵ une relation de transition arbitraire, γ_X une relation de transition non-arbitraire, l'opérateur de composition $\gamma_X \triangleright \gamma_\epsilon$ est défini par :*

$$\gamma_X \triangleright \gamma_\epsilon = \{(o,e,d) \mid \exists q \in Q, (o,e,q) \in \gamma_X, (q,\epsilon,d) \in \gamma_\epsilon\}$$

Question II.24 *Calculer une borne supérieure de la taille de la relation $\gamma_X \triangleright \gamma_\epsilon$ en fonction des tailles de γ_X et de γ_ϵ.*

Question II.25 *Construire la composition $\gamma_X \triangleright \vec{\gamma_\epsilon}$ de la relation γ_X et de la fermeture transitive de la relation γ_ϵ données dans l'exemple II.6.*

Question II.26 *Écrire en CaML une fonction* `compose` *de type* `relation -> relation -> relation` *telle que l'appel* `(compose gamma1 gamma2)` *renvoie une relation résultant de la composition $\gamma_1 \triangleright \gamma_2$ de la relation non-arbitraire γ_1 et de la relation arbitraire γ_2. Nous supposons que γ_1 ne contient que des transitions non-arbitraires et que γ_2 ne contient que des transitions arbitraires. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4.2.6 Construction d'un automate semi-indéterministe

Déf. II.9 (Automate semi-indéterministe) *Soit l'automate fini $A = (Q,X,I,T,\gamma)$ avec $\gamma = \gamma_X \cup \gamma_\epsilon$, l'automate semi-indéterministe associé $\mathcal{E}(A)$ est défini par :*

$$\mathcal{E}(A) = (Q,X,I \cup \mathcal{S}(I,\vec{\gamma_\epsilon}),T,\gamma_X \cup (\gamma_X \triangleright \vec{\gamma_\epsilon}))$$

Question II.27 *Montrer que l'automate semi-indéterministe associé reconnaît le même langage que l'automate initial, c'est-à-dire que $L(A) = L(\mathcal{E}(A))$.*

Question II.28 *Calculer une borne supérieure de la taille de l'automate semi-indéterministe associé en fonction de la taille de l'automate initial. Comparer cette valeur avec celle de l'automate résultant de la déterminisation de l'automate initial.*

Question II.29 *Donner la représentation graphique de l'automate semi-indéterministe associé à l'automate de l'exemple II.1.*

Question II.30 *Écrire en CaML une fonction* `semi` *de type* `automate -> automate` *telle que l'appel* `(semi A)` *renvoie l'automate semi-indéterministe associé à l'automate A. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Partie II : Algorithmique et programmation en PASCAL

Cette partie doit être traitée par les étudiants qui ont utilisé le langage PASCAL dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (`for`, `while`, `repeat`, ...).

L'explication du comportement d'une fonction (demandée aux questions II.8, II.13, II.16 et II.23) doit au moins contenir :

- L'objectif général ;
- Le rôle des paramètres de la fonction ;
- Les contraintes sur les valeurs des paramètres ;
- Les caractéristiques du résultat renvoyé par la fonction ;
- Le rôle des variables locales à la fonction ;
- Le principe de l'algorithme ;
- Des arguments de terminaison du calcul quelles que soient les valeurs des paramètres.

L'analyse lexicale est la première étape effectuée par les outils d'analyse de texte et, en particulier, de programmes (compilateur, interprète, ...). Elle vise à reconnaître les différents mots composant une phrase sans s'attacher à la structure de la phrase elle-même. La technique la plus couramment employée pour implanter un analyseur lexical repose sur un automate fini dont les transitions sont étiquetées par les caractères composant les mots du vocabulaire. Les états initiaux correspondent aux débuts des mots. Les états terminaux correspondent à la fin des mots. Un mot est reconnu par un parcours de l'automate guidé par les caractères composant ce mot.

Les automates finis peuvent être déterministes ou indéterministes. Ces deux formes présentent chacune des avantages et des inconvénients au niveau des performances :

- le parcours d'un automate déterministe est peu coûteux alors que son occupation mémoire est coûteuse ;
- l'occupation mémoire d'un automate indéterministe est peu coûteuse alors que son parcours est coûteux.

L'outil `lex`, le plus couramment utilisé pour construire des analyseurs de programmes, exploite des automates semi-indéterministes pour obtenir un bon compromis entre les coûts du parcours et de l'occupation mémoire.

Nous allons étudier une catégorie d'automates semi-indéterministes, ses propriétés, ainsi qu'une technique de construction.

1 Représentation et codage d'un automate fini

1.1 Représentation d'un automate fini

Déf. II.1 (Automate fini) *Soit l'alphabet X, un ensemble de symboles, soit ϵ le symbole représentant la transition arbitraire ($\epsilon \notin X$), soit Λ le symbole représentant le mot vide ($\Lambda \notin X$), soit $X^\star$ l'ensemble des mots composés de symboles de X ($\Lambda \in X^\star$); un automate fini sur X est un quintuplet*

$A = (Q,X,I,T,\gamma)$ *composé de :*

- *Un ensemble fini d'états :* Q *;*
- *Un ensemble d'états initiaux :* $I \subseteq Q$ *;*
- *Un ensemble d'états terminaux :* $T \subseteq Q$ *;*
- *Une relation de transition confondue avec son graphe :* $\gamma \subseteq Q \times (X \cup \{\epsilon\}) \times Q$.

Les symboles ϵ et Λ sont souvent confondus. Le premier est spécifique aux automates, il correspond à une transition sans lecture de caractères (le mot lu est donc le mot vide Λ). Le second correspond au mot vide indépendamment des automates.

Pour une transition (o,e,d) donnée, nous appelons o l'origine de la transition, e l'étiquette de la transition et d la destination de la transition.

Remarquons que γ est le graphe d'une fonction totale de transition $\delta : Q \times (X \cup \{\epsilon\}) \to \mathcal{P}(Q)$ dont les valeurs sont définies par :

$$\forall o \in Q, \forall e \in X \cup \{\epsilon\}, \delta(o,e) = \{d \in Q \mid (o,e,d) \in \gamma\}$$

La notation γ est plus adaptée que δ à la formalisation et la construction des preuves.

1.2 Représentation graphique d'un automate

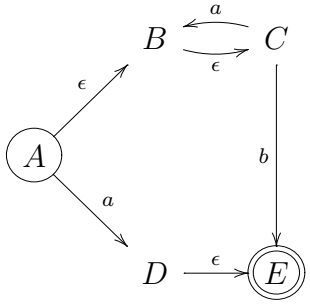
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la relation de transition γ sont représentées par un graphe dont les nœuds sont les états et les arêtes sont les transitions ;
- un état initial est entouré d'un cercle (i) ;
- un état terminal est entouré d'un double cercle $((t))$;
- une arête étiquetée par le symbole $e \in X \cup \{\epsilon\}$ va de l'état o à l'état d si et seulement si $(o,e,d) \in \gamma$.

Exemple II.1 *L'automate* $A = (Q,X,I,T,\gamma)$ *avec :*

$$\begin{aligned} Q &= \{A,B,C,D,E\} \\ X &= \{a,b\} \\ I &= \{A\} \\ T &= \{E\} \\ \gamma &= \{(A,\epsilon,B),(B,\epsilon,C),(C,a,B),(C,b,E),(A,a,D),(D,\epsilon,E)\} \end{aligned}$$

est représenté par le graphe suivant :



Question II.1 *Donner sans la justifier une expression régulière ou ensembliste représentant le langage reconnu par* A.

1.3 Langage reconnu par un automate fini

$X^\star$ est l'ensemble des mots composés des symboles de X. Nous étendons l'opérateur . de construction de mots pour prendre en compte le symbole ϵ apparaissant dans les transitions des automates :

$$\forall m \in X^\star,\ \epsilon.m = m$$

Soit $\gamma^\star$ l'extension de γ à $Q \times X^\star \times Q$ définie par :

$$\begin{cases} \forall q \in Q,\ (q,\Lambda,q) \in \gamma^\star \\ \forall e \in X \cup \{\epsilon\},\ \forall m \in X^\star,\ \forall o \in Q,\ \forall d \in Q,\ (o,e.m,d) \in \gamma^\star \Leftrightarrow \exists q \in Q,\ (o,e,q) \in \gamma \wedge (q,m,d) \in \gamma^\star \end{cases}$$

Le langage sur $X^\star$ reconnu par un automate fini est :

$$L(A) = \{m \in X^\star \mid o \in I,\ d \in T,\ (o,m,d) \in \gamma^\star\}$$

L'analyse d'un mot par un automate peut être caractérisée par le chemin suivi dans l'automate, c'est-à-dire la séquence des états parcourus. En général, dans le cadre d'un automate indéterministe, un même mot peut être reconnu en suivant plusieurs chemins différents dans l'automate.

1.4 Codage en PASCAL d'un automate fini

Nous considérerons pour les besoins du codage que tous les automates exploitent le même ensemble fini de symboles $X \subset \mathbb{N}$. Nous ne préciserons donc pas cet ensemble dans la suite des fonctions réalisées.

Pour les variables ou les constantes dont le nom est pris dans l'alphabet grec (ϵ, γ, ...), l'identifiant en PASCAL sera leur nom en toute lettre (`epsilon`, `gamma`, ...).

Un état est représenté par le type de base `INTEGER`.

Un ensemble d'états est représenté par le type de base `ETATS` correspondant à une liste d'états.

Une étiquette de transition (appartenant à $X \cup \{\epsilon\}$ est représentée par le type de base `INTEGER`.

L'étiquette particulière ϵ a la valeur `-1`. Elle est contenue dans la constante `epsilon`.

Une transition est représentée par le type de base `TRANSITION` correspondant à un triplet composé d'un état, d'une étiquette et d'un état.

Une relation est représentée par le type de base `RELATION` correspondant à une liste de transitions.

Une paire de relations est représentée par le type de base `PAIRE`.

Un automate est représenté par le type de base `AUTOMATE` équivalent à un quadruplet composé de trois ensembles d'états et d'une relation.

Un mot accepté par l'automate est représenté par le type `MOT` équivalent à une liste d'étiquettes.

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `NIL` représente la liste vide d'états, la liste vide de transitions et la liste vide d'étiquettes ;
- `FUNCTION LE_Ajouter(e:INTEGER;l:ETATS):ETATS;` renvoie une liste d'états composée d'un premier état `e` et du reste de la liste contenu dans `l` ;
- `FUNCTION LE_Premier(l:ETATS):INTEGER;` renvoie le premier état de la liste `l` ;

- `FUNCTION LE_Reste(l:ETATS):ETATS;` renvoie le reste de la liste `l` privée de son premier état ;
- `FUNCTION LE_Juxtaposer(l1:ETATS;l2:ETATS):ETATS;` renvoie la liste composée de la liste `l1` suivie de la suite `l2` ;
- `FUNCTION T_Creer(o:INTEGER;e:INTEGER;d:INTEGER):TRANSITION;` renvoie une transition dont l'origine et la destination sont les états `o` et `d` et l'étiquette est `e` ;
- `FUNCTION T_Origine(t:TRANSITION):INTEGER;` renvoie l'état origine de la transition `t`,
- `FUNCTION T_Destination(t:TRANSITION):INTEGER;` renvoie l'état destination de la transition `t` ;
- `FUNCTION T_Etiquette(t:TRANSITION):INTEGER;` renvoie l'étiquette de la transition `t` ;
- `FUNCTION LT_Ajouter(t:TRANSITION;l:RELATION):RELATION;` renvoie une liste de transitions composée d'une première transition `t` et du reste de la liste contenu dans `l` ;
- `FUNCTION LT_Premier(l:RELATION):TRANSITION;` renvoie la première transition de la liste `l` ;
- `FUNCTION LT_Reste(l:RELATION):RELATION;` renvoie le reste de la liste `l` privée de sa première transition ;
- `FUNCTION LT_Juxtaposer(l1:RELATION;l2:RELATION):RELATION;` renvoie la liste composée de la liste `l1` suivie de la suite `l2` ;
- `FUNCTION A_Creer(q:ETATS;i:ETATS;t:ETATS;g:RELATION):AUTOMATE;` renvoie un automate dont l'ensemble des états est `q`, l'ensemble des états initiaux et terminaux sont `i` et `t` et la relation de transition est `g` ;
- `FUNCTION A_Etats(a:AUTOMATE):ETATS;` renvoie l'ensemble des états de l'automate `a` ;
- `FUNCTION A_Initiaux(a:AUTOMATE):ETATS;` renvoie l'ensemble des états initiaux de l'automate `a` ;
- `FUNCTION A_Terminaux(a:AUTOMATE):ETATS;` renvoie l'ensemble des états terminaux de l'automate `a` ;
- `FUNCTION A_Relation(a:AUTOMATE):RELATION;` renvoie la relation de transition de l'automate `a` ;
- `FUNCTION P_Creer(g1:RELATION;g2:RELATION):PAIRE;` renvoie une paire dont les éléments sont `g1` et `g2` ;
- `FUNCTION P_Un(p:PAIRE):RELATION;` renvoie le premier élément de `p` ;
- `FUNCTION P_Deux(p:PAIRE):RELATION;` renvoie le second élément de `p` ;
- `FUNCTION M_Ajouter(e:INTEGER;m:MOT):MOT;` renvoie le mot composé d'une première étiquette `e` et du reste du mot contenu dans `m` ;
- `FUNCTION M_Premier(m:MOT):INTEGER;` renvoie la première étiquette du mot `m` ;
- `FUNCTION M_Reste(m:MOT):MOT;` renvoie le reste du mot `m` privé de sa première étiquette ;
- `FUNCTION M_Juxtaposer(m1:MOT;m2:MOT):MOT;` renvoie le mot composé du mot `m1` suivie du mot `m2`.

Exemple II.2 *Associons les entiers suivants aux symboles de* Q *:* $A \mapsto 0$,$B \mapsto 1$,$C \mapsto 2$, $D \mapsto 3$,$E \mapsto 4$ *et de* X *:* $a \mapsto 0$,$b \mapsto 1$.

L'automate de l'exemple II.1 sera représenté par la valeur :

```
CONST
EPSILON = -1;
VAR
Q, I, T : ETATS;                (* ensemble d'etats *)
G       : RELATION;             (* ensemble de transitions *)
A       : AUTOMATE;             (* automate *)
BEGIN
Q := LE_Ajouter( 0, LE_Ajouter( 1, LE_Ajouter( 2,
     LE_Ajouter( 3, LE_Ajouter( 4, NIL)))));
I := LE_Ajouter( 0, NIL);
T := LE_Ajouter( 4, NIL);

G := LT_Ajouter( T_Creer(0,-1,1),
     LT_Ajouter( T_Creer(1,-1,2),
     LT_Ajouter( T_Creer(2,0,1),
     LT_Ajouter( T_Creer(2,1,4),
     LT_Ajouter( T_Creer(0,0,3),
     LT_Ajouter( T_Creer(3,-1,4), NIL))))));

A := A_Creer( Q, I, T, G );
END.
```

Exemple II.3 *Le mot $aaab$ du langage reconnu par l'automate donné dans l'exemple II.1, sera représenté par la valeur :*

```
M_Ajouter( 0,
M_Ajouter( 0,
M_Ajouter( 0,
M_Ajouter( 1, NIL))))
```

2 Codage des ensembles d'états et de transitions

La représentation d'un automate fini et les algorithmes associés exploitent les structures d'ensemble d'états et de transitions. Dans cette partie, nous allons étudier une représentation des ensembles à base de listes et certains algorithmes de manipulation d'ensembles qui pourront être utilisés par la suite.

2.1 Codage de la structure d'ensemble

Un automate fini exploite deux formes d'ensembles :

- des ensembles d'états : Q, I, et T ;
- un ensemble de transitions : la relation γ.

Nous devons donc disposer d'un codage de la structure d'ensemble pour pouvoir représenter des automates finis. Nous allons utiliser un codage extrêmement simple : un ensemble est une liste conte-

nant exactement un exemplaire de chaque valeur contenue dans l'ensemble. Les opérations de manipulation d'un ensemble devront préserver cette propriété.

Le parcours d'un ensemble sera donc effectué de la même manière que celui d'une liste.

Nous allons maintenant définir plusieurs opérations sur les ensembles qui pourront être utilisées dans la suite du sujet. Ces opérations sont indépendantes de la forme des valeurs rangées dans l'ensemble. Nous allons donc les définir pour un ensemble d'états, les définitions pour une relation seront identiques.

2.2 Opération sur la structure d'ensemble

Nous supposons que toutes les listes passées comme paramètre, qu'elles représentent un ensemble d'états ou de transitions, ne contiennent au plus qu'une fois chaque valeur.

Pour les calculs de complexité, nous noterons $\mid e \mid$ la taille de l'ensemble e, c'est-à-dire le nombre d'éléments qu'il contient.

2.2.1 Cardinalité d'un ensemble

Une première opération consiste à déterminer le nombre d'états appartenant à un ensemble.

Question II.2 *Écrire en PASCAL une fonction* `cardinalE(e:ETATS):INTEGER;` *telle que l'appel* `cardinalE(e)` *renvoie le nombre d'états contenus dans l'ensemble* `e`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.3 *Calculer une estimation de la complexité de la fonction* `cardinalE` *en fonction de la taille de l'ensemble* `e`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.2 Appartenance d'un état à un ensemble

Une deuxième opération consiste à tester l'appartenance d'un état à un ensemble.

Question II.4 *Écrire en PASCAL une fonction* `appartientE(v:ETAT;e:ETATS):BOOLEAN;` *telle que l'appel* `appartientE(v,e)` *renvoie la valeur vrai si l'ensemble* `e` *contient l'état* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.5 *Calculer une estimation de la complexité dans le pire cas de la fonction* `appartient` *en fonction de la taille de l'ensemble* `e`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.3 Ajout d'une valeur dans un ensemble

La troisième opération est l'ajout d'un état à un ensemble.

Question II.6 *Écrire en PASCAL une fonction* `ajoutE(v:ETAT;e:ETATS):ETATS;` *telle que l'appel* `ajoutE(v,e)` *renvoie un ensemble contenant les mêmes états que l'ensemble* `e` *ainsi que l'état* `v` *s'il ne figurait pas dans* `e`*. L'ensemble renvoyé contiendra exactement une fois l'état* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.4 Comparaison du contenu de deux ensembles

La quatrième opération est la comparaison du contenu de deux ensembles.

Question II.7 *Écrire en PASCAL une fonction* `egaliteE(e1:ETATS;e2:ETATS):BOOLEAN;` *telle que l'appel* `egaliteE(e1,e2)` *renvoie la valeur vraie si les deux ensembles* `e1` *et* `e2` *contiennent exactement les mêmes états. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.8 *Expliquer la fonction* `egaliteE` *proposée pour la question précédente.*

Question II.9 *Calculer une estimation de la complexité dans le pire cas de la fonction* `egaliteE` *en fonction des tailles des ensembles* `e1` *et* `e2`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

2.2.5 Union de deux ensembles

La cinquième opération étudiée est la construction d'un ensemble résultant de l'union de deux autres ensembles.

Question II.10 *Écrire en PASCAL une fonction* `unionE(e1:ETATS;e2:ETATS):ETATS;` *telle que l'appel* `unionE(e1,e2)` *renvoie un ensemble contenant les états contenus dans* `e1` *et les états contenus dans* `e2`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.6 Intersection de deux ensembles

La dernière opération étudiée est la construction d'un ensemble résultant de l'intersection de deux autres ensembles.

Question II.11 *Écrire en PASCAL une fonction* `intersectionE(e1:ETATS;e2:ETATS):ETATS;` *telle que l'appel* `intersectionE(e1,e2)` *renvoie un ensemble contenant les états contenus à la fois dans* `e1` *et dans* `e2`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Nous supposons également définies les opérations suivantes sur les relations (ensemble de transitions) :

- `cardinalR(g:RELATION):INTEGER;`
- `appartientR(t:TRANSITION;g:RELATION):BOOLEAN;`
- `egaliteR(r1:RELATION;r2:RELATION):RELATION;`
- `ajouterR(t:TRANSITION;g:RELATION):RELATION;`
- `unionR(g1:RELATION;g2:RELATION):RELATION;`
- `intersectionR(g1:RELATION;g2:RELATION):RELATION;`

3 Exploitation des relations de transition

3.1 Suivants dans une relation

La construction d'automates semi-indéterministes repose sur la manipulation des relations de transition et, en particulier, sur la construction de fermetures transitives. Nous allons définir des opérations de parcours des relations qui pourront être utilisées par la suite.

Déf. II.2 (Suivants dans une relation) *Les suivants $\mathcal{S}(O,\gamma)$ d'un ensemble d'états O selon une relation γ sont les ensembles des destinations des transitions de γ dont les états de O sont les origines, indépendamment des étiquettes.*

$$\mathcal{S}(O,\gamma) = \{d \in Q \mid \exists o \in O, \exists e \in X \cup \{\epsilon\}, (o,e,d) \in \gamma\}$$

Exemple II.4 *Dans l'automate de l'exemple II.1, nous avons par exemple :*

$$\begin{array}{l}\mathcal{S}(\{A\},\gamma) = \{B,D\}\\ \mathcal{S}(\{C\},\gamma) = \{B,E\}\\ \mathcal{S}(\{B,D\},\gamma) = \{C,E\}\\ \mathcal{S}(\{E\},\gamma) = \emptyset\end{array}$$

Nous allons implanter une fonction calculant $\mathcal{S}(O,\gamma)$.

Question II.12 *Écrire en PASCAL une fonction* `suivants(o:ETATS;g:RELATION):ETATS;` *telle que l'appel* `suivants(O,gamma)` *renvoie un ensemble d'états contenant les mêmes états que $\mathcal{S}(O,\gamma)$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.13 *Expliquer la fonction* `suivants` *proposée pour la question précédente.*

3.2 Etats accessibles

Nous allons étendre les opérations précédentes pour obtenir l'ensemble des états accessibles à partir d'un ensemble d'états donné en suivant plusieurs transitions sans prendre en compte les étiquettes.

Déf. II.3 (États accessibles) *L'ensemble des états accessibles $\mathcal{A}(O,\gamma)$ depuis les états de O selon la relation γ est défini par :*

$$\mathcal{A}(O,\gamma) = \{d \in Q \mid \exists o \in O, \exists m \in X^\star, (o,m,d) \in \gamma^\star\}$$

Exemple II.5 *Dans l'automate de l'exemple II.1, nous avons par exemple :*

$$\begin{array}{l}\mathcal{A}(\{A\},\gamma) = \{A,B,C,D,E\}\\ \mathcal{A}(\{C\},\gamma) = \{B,C,E\}\\ \mathcal{A}(\{B,D\},\gamma) = \{B,C,D,E\}\\ \mathcal{A}(\{E\},\gamma) = \{E\}\end{array}$$

Nous allons implanter une fonction calculant $\mathcal{A}(O,\gamma)$.

Question II.14 *Soient $O \subseteq Q$ un ensemble d'états, γ une relation sur Q, la suite (A_i) d'ensembles d'états ($A_i \subseteq Q$) est définie par :*

$$\begin{cases} A_0 = O \\ A_{i+1} = A_i \cup \mathcal{S}(A_i,\gamma) \end{cases}$$

1. *Montrer que la suite (A_i) est croissante pour la relation d'inclusion $\subseteq$;*
2. *Montrer qu'il existe un entier $k \in \mathbb{N}$ tel que $A_k = A_{k+1}$;*
3. *Montrer que $\forall i \in \mathbb{N},\ \mathcal{A}(A_i,\gamma) = \mathcal{A}(A_{i+1},\gamma)$;*
4. *Montrer que $\forall E \subseteq Q, \mathcal{S}(E,\gamma) \subseteq E \Rightarrow \mathcal{A}(E,\gamma) \subseteq E$;*
5. *Montrer que $\mathcal{A}(O,\gamma) = A_k$.*

Question II.15 *Écrire en PASCAL une fonction* `accessibles(o:ETATS;g:RELATION):ETATS;` *telle que l'appel* `accessibles(O,gamma)` *renvoie un ensemble d'états contenant les mêmes états que $\mathcal{A}(O,\gamma)$. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.16 *Expliquer la fonction* `accessibles` *proposée pour la question précédente.*

3.3 Construction de familles de transitions

Les opérations de construction des automates semi-indéterministes étudiées par la suite engendrent de nouvelles transitions. Pour simplifier leurs définitions, nous pouvons utiliser des fonctions de construction de familles de transitions similaires (par exemple, de transitions ayant la même origine, la même étiquette et plusieurs destinations distinctes).

Question II.17 *Écrire en PASCAL une fonction* `prefixe(o:ETAT;e:ETIQUETTE;ds:ETATS):RELATION;` *telle que l'appel* `prefixe(o,e,ds)` *renvoie une relation contenant les transitions d'origine* `o`*, d'étiquette* `e` *et de destination les états contenus dans* `ds`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4 Automate fini semi-indéterministe

En général, dans le cadre d'un automate fini, un même mot du langage accepté par un automate peut être reconnu par plusieurs parcours distincts de l'automate.

L'algorithme naturel de parcours de l'automate pour analyser un mot est donc indéterministe car il existe plusieurs possibilités de parcours.

4.1 Causes d'indéterminisme

Trois causes d'indéterminisme peuvent apparaître dans un automate fini :

– L'existence de plusieurs états initiaux ;

– L'existence de plusieurs transitions avec une même étiquette et une même origine ;

– L'existence de transitions arbitraires (d'étiquette ϵ).

Les algorithmes indéterministes de parcours sont souvent plus coûteux, il est donc pertinent d'envisager la déterminisation de l'automate d'analyse.

La construction d'un automate déterministe reconnaissant le même langage qu'un automate indéterministe donné peut faire apparaître un grand nombre de nouveaux états. L'occupation mémoire est donc souvent beaucoup plus coûteuse.

Dans un objectif de compromis entre occupation mémoire et temps de parcours, nous allons nous intéresser à des automates semi-indéterministes qui ne font pas apparaître la troisième forme d'indéterminisme. Nous étudierons par la suite un algorithme de construction qui n'introduit pas de nouveaux états.

4.2 Automate fini semi-indéterministe

Nous définissons une restriction des automates finis qui ne contient pas de transitions arbitraires (étiquette ϵ). Ces automates sont appelés semi-indéterministes car un degré d'indéterminisme est levé. Les définitions précédentes sont toujours valides en éliminant les transitions arbitraires.

Déf. II.4 (Automate semi-indéterministe) *Soit $A = (Q,X,I,T,\gamma)$ un automate fini, A est semi-indéterministe si et seulement si $\gamma \subseteq Q \times X \times Q$.*

Nous allons décomposer la relation de transition d'un automate fini quelconque en une relation arbitraire et une relation non-arbitraire que nous recomposerons ensuite pour obtenir un automate semi-indéterministe.

4.2.1 Relation de transition arbitraire

Déf. II.5 (Relation de transition arbitraire) *Soit Q un ensemble d'états, nous appelons relation de transition arbitraire γ_ϵ, une relation dont toutes les transitions sont étiquetées par ϵ, donc $\gamma_\epsilon \subseteq Q \times \{\epsilon\} \times Q$.*

4.2.2 Relation de transition non-arbitraire

Déf. II.6 (Relation de transition non-arbitraire) *Soient Q un ensemble d'états et X un ensemble de symboles, nous appelons relation de transition non-arbitraire γ_X, une relation dont toutes les transitions sont étiquetées par des symboles de X (donc différent de ϵ), donc $\gamma_X \subseteq Q \times X \times Q$.*

Nous noterons γ_X une relation de transition non-arbitraire car elle ne peut pas faire apparaître la troisième forme d'indéterminisme.

4.2.3 Décomposition d'une relation de transition

Théorème II.1 (Décomposition d'une relation de transition) *Soit l'automate fini $A = (Q,X,I,T,\gamma)$, la relation γ peut donc se décomposer de manière unique en deux sous-relations disjointes :*

- *une relation arbitraire γ_ϵ ;*
- *une relation non-arbitraire γ_X.*

Donc,

$$\gamma = \gamma_\epsilon \cup \gamma_X$$

Exemple II.6 *La relation de transition de l'automate de l'exemple II.1 se décompose en :*

$$\gamma_\epsilon = \{(A,\epsilon,B),(B,\epsilon,C),(D,\epsilon,E)\}$$
$$\gamma_X = \{(C,a,B),(C,b,E),(A,a,D)\}$$

Nous allons définir une fonction pour décomposer la relation de transition γ de l'automate en sous-relations arbitraire γ_ϵ et non-arbitraire γ_X.

Question II.18 *Écrire en PASCAL une fonction* `decompose(g:RELATION):PAIRE;` *telle que l'appel* `decompose(gamma)` *renvoie une paire contenant deux relations composées des transitions de la relation γ. La première contient les transitions arbitraires (d'étiquette ϵ) de γ. La seconde contient les transitions de γ d'étiquettes différentes de ϵ. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4.2.4 Fermeture d'une relation arbitraire

Déf. II.7 (Fermeture transitive d'une relation arbitraire) *Nous noterons $\vec{\gamma_\epsilon}$ la fermeture transitive de γ_ϵ définie par :*

$$\begin{cases} \forall q \in Q,\ (q,\epsilon,q) \in \vec{\gamma_\epsilon} \\ \forall q \in Q, \forall q' \in Q,\ (q,\epsilon,q') \in \vec{\gamma_\epsilon} \Leftrightarrow \exists q'' \in Q,\ (q,\epsilon,q'') \in \gamma_\epsilon \wedge (q'',\epsilon,q') \in \vec{\gamma_\epsilon} \end{cases}$$

Question II.19 *Construire $\vec{\gamma_\epsilon}$ à partir de la relation γ_ϵ de l'exemple II.6.*

Question II.20 *Calculer une borne supérieure de la taille de $\vec{\gamma_\epsilon}$ en fonction de la taille de γ_ϵ.*

Question II.21 *Montrer que :*

$$\vec{\gamma_\epsilon} = \{(o,\epsilon,d) \mid o \in Q, d \in \mathcal{A}(\{o\},\gamma_\epsilon)\}$$

Question II.22 *Écrire en PASCAL une fonction* `fermeture(g:RELATION):RELATION;` *telle que l'appel* `fermeture(gamma)` *renvoie une relation contenant toutes les transitions engendrées par la fermeture transitive $\vec{\gamma}$ des transitions de la relation arbitraire γ. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question II.23 *Expliquer la fonction* `fermeture` *proposée pour la question précédente.*

4.2.5 Composition de relations arbitraires et non-arbitraires

Déf. II.8 (Opérateur de composition $\triangleright$) *Soient γ_ϵ une relation de transition arbitraire, γ_X une relation de transition non-arbitraire, l'opérateur de composition $\gamma_X \triangleright \gamma_\epsilon$ est défini par :*

$$\gamma_X \triangleright \gamma_\epsilon = \{(o,e,d) \mid \exists q \in Q, (o,e,q) \in \gamma_X, (q,\epsilon,d) \in \gamma_\epsilon\}$$

Question II.24 *Calculer une borne supérieure de la taille de la relation $\gamma_X \triangleright \gamma_\epsilon$ en fonction des tailles de γ_X et de γ_ϵ.*

Question II.25 *Construire la composition $\gamma_X \triangleright \vec{\gamma_\epsilon}$ de la relation γ_X et de la fermeture transitive de la relation γ_ϵ données dans l'exemple II.6.*

Question II.26 *Écrire en PASCAL une fonction* `compose(g1:RELATION;g2:RELATION):RELATION;` *telle que l'appel* `compose(gamma1,gamma2)` *renvoie une relation résultant de la composition $\gamma_1 \triangleright \gamma_2$ de la relation non-arbitraire γ_1 et de la relation arbitraire γ_2. Nous supposons que γ_1 ne contient que des transitions non-arbitraires et que γ_2 ne contient que des transitions arbitraires. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

4.2.6 Construction d'un automate semi-indéterministe

Déf. II.9 (Automate semi-indéterministe) *Soit l'automate fini $A = (Q,X,I,T,\gamma)$ avec $\gamma = \gamma_X \cup \gamma_\epsilon$, l'automate semi-indéterministe associé $\mathcal{E}(A)$ est défini par :*

$$\mathcal{E}(A) = (Q,X,I \cup \mathcal{S}(I,\vec{\gamma}_\epsilon),T,\gamma_X \cup (\gamma_X \triangleright \vec{\gamma}_\epsilon))$$

Question II.27 *Montrer que l'automate semi-indéterministe associé reconnaît le même langage que l'automate initial, c'est-à-dire que $L(A) = L(\mathcal{E}(A))$.*

Question II.28 *Calculer une borne supérieure de la taille de l'automate semi-indéterministe associé en fonction de la taille de l'automate initial. Comparer cette valeur avec celle de l'automate résultant de la déterminisation de l'automate initial.*

Question II.29 *Donner la représentation graphique de l'automate semi-indéterministe associé à l'automate de l'exemple II.1.*

Question II.30 *Écrire en PASCAL une fonction* `semi(a:AUTOMATE):AUTOMATE;` *telle que l'appel* `semi(A)` *renvoie l'automate semi-indéterministe associé à l'automate A. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Fin de l'énoncé