RICM3 – Automates et Grammaires

Durée : 2h00, sans documents.

- Tous les appareils électroniques sont interdits à l'exception des montres
- Le barème est donné à titre indicatif
- Le sujet comporte 6 exercices indépendants
- Le sujet est sur 75 mais il suffit d'avoir 50 pour avoir la note maximale, ce qui vous donne le choix et vous permet de faire l'impasse sur 2 exercices.
- Commencez par lire tout le sujet pour repérer les questions faciles

10 *pt*

Exercice 1 : Grammaires des identificateurs (20min)

Dans la plupart des langages de programmation les noms de variables (appelés identificateurs) doivent respecter les contraintes suivantes :

1. les identificateurs sont formés de lettres minuscules (`'a'`,...,`'z'`), de lettres majuscules (`'A'`,...,`'Z'`), de chiffres (`'0'`,...,`'9'`), et du caractère souligné `'_'`
2. un identificateur doit commencer par une lettre minuscule dans `'a'`,...,`'z'`

   Exemple : `x,y` sont des identificateurs, `A,1,_` ne sont pas des identificateurs
3. un identificateur ne doit pas se terminer par un souligné `'_'`

   Exemple : `x_1,y_b` sont des identificateurs, `x_,y_` ne sont pas des identificateurs

Q1. Donnez une grammaire des identificateurs qui respecte ces contraintes.

SOLUTION

Le germe de la grammaire est Id.

$Id \rightarrow Min.SuiteId$
$SuiteId \rightarrow \epsilon \mid Car.SuiteId \mid$ `'_'`$.Car.SuiteId$
$Car \rightarrow Min \mid Maj \mid Digit$
$Min \rightarrow$ `'a'` $\mid ... \mid$ `'z'`
$Maj \rightarrow$ `'A'` $\mid ... \mid$ `'Z'`
$Digit \rightarrow$ `'0'` $\mid ... \mid$ `'9'`

Q2. Donnez une grammaire des identificateurs qui respecte les contraintes précédentes et les contraintes ci-dessous (qui n'existent pas dans les langages de programmation) :

4. un identificateur ne doit pas comporter plusieurs caractères `'_'` consécutifs

   Exemple : `x_A_1,y_B_2` sont des identificateurs, `x__,y__2` ne sont pas des identificateurs

5. un chiffre doit être précédé du caractère souligné `'_'` ou d'un chiffre

   Exemple : `x_1,y_2` sont des identificateurs, `x1,y2` ne sont pas des identificateurs

SOLUTION

Le germe de la grammaire est Id.

$Id \rightarrow Min.SuiteId$
$SuiteId \rightarrow \epsilon \mid Min.SuiteId \mid Maj.SuiteId \mid$ `'_'`$.SuiteSoul.SuiteId$
$SuiteSoul \rightarrow Min \mid Maj \mid Nombre$
$Nombre \rightarrow Digit.SomeDigit$
$SomeDigit \rightarrow \epsilon \mid Nombre$
$Min \rightarrow$ `'a'` $\mid ... \mid$ `'z'`
$Maj \rightarrow$ `'A'` $\mid ... \mid$ `'Z'`
$Digit \rightarrow$ `'0'` $\mid ... \mid$ `'9'`

10 *pt*

Exercice 2 : Grammaire des déclarations de variables en C (20min)

La syntaxe des déclarations de variables en C est définie par une grammaire avec pour germe, *SomeDecl*, et pour règles :

SomeDecl	→	ϵ
	|	*OneDecl SomeDecl*
OneDecl	→	*Type Vars OptAff* `';'`
Type	→	`"int"`
	|	`"float"`
Vars	→	*OneVar MoreVars*
MoreVars	→	ϵ
	|	`','` *Vars*
OneVar	→	grammaire de l'Exercice 1:
OptAff	→	ϵ
	|	`'='` *Value*
Value	→	*Float*
	|	*Int*

Q3. Attributs hérités et synthétisés Ajoutez des attributs à la grammaire ci-dessus afin que le non-terminal *SomeDecl* retourne une liste de déclarations indépendantes des variables, au format du langage Pascal, et avec leur valeur d'initialisation si elle est présente.

Indication :

Vous devez produire une résultat conforme aux exemples ci-dessous.

Vous supposerez que les non-terminaux *Float* et *Int* produisent une chaîne de caractères correspondant à la valeur lue et que le non-terminal *OneVar* produit une chaîne de caractères correspondant au nom de variable qu'il a reconnu.

Vous ne vous préoccuperez pas de détecter les variables définies plusieurs fois.

Exemples :

- *SomeDecl* [⟨ `int x,y,z = 0 ;` ⟩] =
  [`"x : integer = 0"` ; `"y : integer = 0"` ; `"z : integer = 0"`]
- *SomeDecl* [⟨ `int x = 0 ; float x,y =3.14 ;` ⟩] =
  [`"x : integer := 0"` ; `"x : real := 3.14"` ; `"y : real := 3.14"`]

SOLUTION

SomeDecl {*list_decl*}	→	ϵ {*list_decl* := []}
	|	*OneDecl* {*list_decl'*} ; *SomeDecl* {*list_decl''*} ; {*list_decl* := *list_decl'* @ *list_decl''*}
OneDecl {*list_decl*}	→	*Type* {*type*} ; {*type*} *Vars* {*list_var*} ; *OptAff* {*aff*} `';'` ; {*list_decl* := `List.map (fun d → d ^ aff) list_var`}
Type {*type*}	→	`"int"` {*type* := `"integer"`}
	|	`"float"` {*type* := `"real"`}
{*type*} *Vars* {*list_typed_var*}	→	{*type*} *OneVar* {*var*} ; {*type*} *MoreVars* {*list_typed_var'*} ; {*list_typed_var* := *var* :: *list_typed_var'*}
{*type*} *MoreVars* {*list_typed_var*}	→	ϵ {*list_typed_var* := []}
	|	`','` {*type*} *Vars* {*list_typed_var'*} ; {*list_typed_var* := *list_typed_var'*}
{*type*} *OneVar* {*typed_var*}	→	(*grammaire de l'Exercice 1*) {*var*} ; {*typed_var* := *var* ^ `" : "` ^ *type*}
OptAff {*val*}	→	ϵ ; {*aff* := `""`}
	|	`'='` *Value* {*val*} ; {*aff* := `"="` ^ *val*}
Value {*val*}	→	*Float* {*f*} ; {*val* := *f*}
	|	*Int* {*i*} ; {*val* := *i*}

15 pt

Exercice 3 : Transducteurs et addition binaire d'automates (20min)

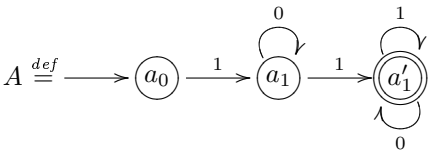
Dans tout l'exercice on considère **des nombres en binaire écrits avec l'unité à gauche.**

Exemples :

- $(1)_{\mathbb{N}} \equiv (1)_2 \equiv (1.0.0.0.0)_2 \equiv (1.0^*)_2$
- $(2)_{\mathbb{N}} \equiv (0.1)_2 \equiv (0.1.0^*)_2$
- $(3)_{\mathbb{N}} \equiv (1.1)_2 \equiv (1.1.0^*)_2$

Q4. Donnez un automate A qui reconnaît les nombres binaires impairs ≥ 3, écrits avec les unités à gauche. Autrement dit, $\{11, 101, 111, 1001\} \in \mathscr{L}(A)$ *et* $\{0, 01, 001\} \notin \mathscr{L}(A)$

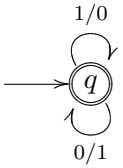
SOLUTION



Q5. Donnez une expression régulière équivalente à A

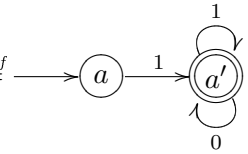
SOLUTION

$$A \equiv 1.0^*.1.(0|1)^*$$

Q6. On considère le transducteur $T_1 \stackrel{def}{=}$ 

Décrire (à chaque fois en une phrase) :

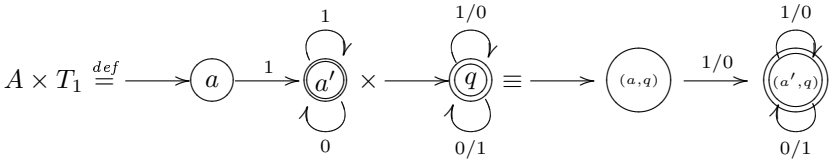
1. le langage reconnu par le transducteur T_1
2. l'effet du transducteur T_1 sur les mots qu'il reconnait
3. le langage de sortie du transducteur T_1

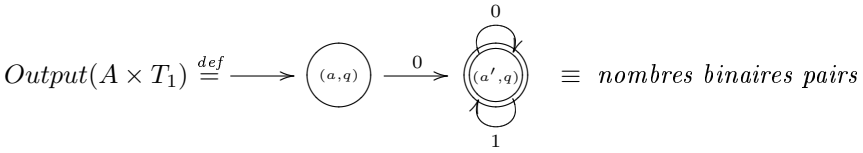
Q7. On considère l'automate $A \stackrel{def}{=}$ 

Décrire en une phrase les nombres binaires (avec unité à gauche) reconnus par l'automate A.

Q8. Construisez le produit de A par le transducteur T_1, puis décrire en une phrase le langage de sortie de l'automate $A \times T_1$.

SOLUTION



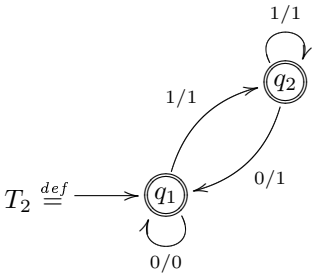


Q9. Donnez un transducteur T_2 qui reconnaît tous les mots et qui remplace les 0 par des 1 lorsque le 0 est précédé d'un 1.

Exemples :

- $T_2(001001) = 001101$
- $T_2(000000) = 000000$
- $T_2(101000) = 111100$

SOLUTION



Rappel sur l'addition en binaire Lorsqu'on veut additionner deux nombres n et p, on complète le plus court des deux par des 0 afin d'obtenir des nombres de même taille ou on leur ajoute même un 0 pour prévoir le cas d'une retenue en fin d'addition.

Exemple :

on considére des nombres binaires écrits avec les unités à gauche
et donc on fait l'addition de gauche à droite :

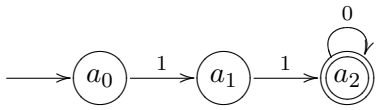
$$\begin{array}{rll} & 111111 & = n \\ + & 11 & = p \\ \hline & 0100001 & \end{array}$$

détail des calculs :

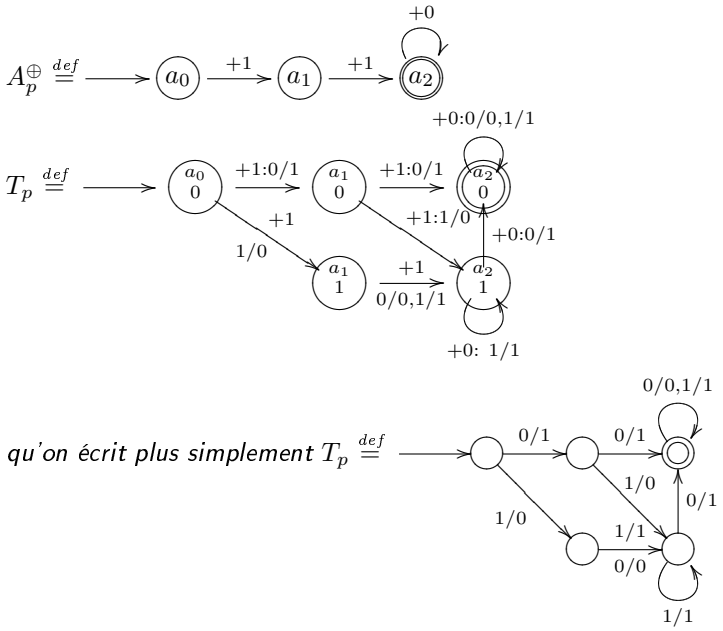
sens du calcul →

$$\begin{array}{rll} & 111111 & \text{retenue} \\ & 1111110 & = n \text{ complété} \\ + & 1100000 & = p \text{ complété} \\ \hline & 0100001 & \end{array}$$

Tout nombre binaire p est équivalent à $p.0^*$: le nombre p auquel on a ajouté des 0 non significatifs.

Q10. Construire le transducteur-additionneur T_p associé à l'automate $A_p \stackrel{def}{=}$ 

SOLUTION



qu'on écrit plus simplement $T_p \stackrel{def}{=}$

Q11. On suppose qu'A_n et A_p sont des automates sur $\Sigma = \{0,1\}$. **Expliquez** comment obtenir l'automate qui reconnait le langage $\{n+p \mid n \in \mathscr{L}(A_n),\ p \in \mathscr{L}(A_p)\}$.

SOLUTION

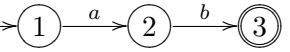
$$\{n+p \mid n \in \mathscr{L}(A_n),\ p \in \mathscr{L}(A_p)\} = \mathscr{L}(Output(A_n \times T_p))$$

15 *pt*

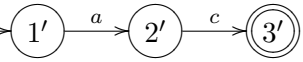
Exercice 4 : Modélisation de protocoles médicaux à l'aide d'AEF (30min)

4.1 Produit d'automates harmonisés

On considère les automates suivants

$A_1 \stackrel{def}{=}$ 

sur l'alphabet $\Sigma_1 = \{a,b\}$

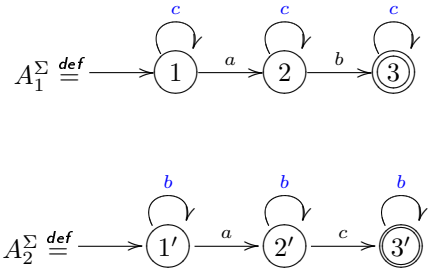
$A_2 \stackrel{def}{=}$ 

sur l'alphabet $\Sigma_2 = \{a,c\}$

Q12. Décrivez le langage $\mathscr{L}(A_1 \times A_2)$ reconnu par l'automate $A_1 \times A_2$. Justifiez avec précision votre réponse.

Harmonisation d'automates Lorsque deux automates n'ont pas le même alphabet (Σ_1 et Σ_2), on peut harmoniser les automates de manière à ce qu'ils aient un alphabet commun ($\Sigma = \Sigma_1 \cup \Sigma_2$). Cela consiste à rendre un automate insensible à un symbole qui n'était pas dans son alphabet, en ajoutant à chaque état de l'automate une boucle sur les nouveaux symboles.

Exemple : A_1^Σ et A_2^Σ ci-dessous sont les versions harmonisées des automates précédents étendus à l'alphabet $\Sigma = \{a, b, c\}$



Q13. Constuire le produit $A_1^\Sigma \times A_2^\Sigma$.

Q14. Donnez le langage reconnu par $A_1^\Sigma \times A_2^\Sigma$.

4.2 Application aux protocoles médicaux

Un protocole médical décrit les actions à effectuer et l'ordre dans lequel les effectuer. Souvent une opération fait appel à plusieurs protocoles médicaux simultanément.
Le but de cet exercice est d'étudier les interactions entre plusieurs protocoles afin de déterminer s'ils sont compatibles ou non. Pour cela on décrit chaque protocole par un automate ou une expression régulière. Son alphabet est l'ensemble des actions du protocole.

4.3 Imagerie médicale

Protocole P_1 On commence par injecter dans le patient un produit radioactif (phase d'injection $= i$) qui permet de voir un organe par analyse du rayonnement, on peut ainsi visualiser l'organe (phase photographie $= p$). Ensuite, on injecte des produits qui neutralise par capture les produits radioactifs (phase neutralisation 1,2 $= n_1, n_2$) qui sont ensuite éliminer par le corps (phase d'élimination $= e$).

Codage du protocole P_1 sous forme d'automate Le protocole P_1 pour l'imagerie médicale est donné par l'expression régulière suivante

$P_1 \stackrel{def}{=} (i.p.n_1)^+.n_2.e$

Q15. Donnez l'alphabet Σ_1 du protocole P_1

Q16. Donnez l'automate correspondant à P_1.

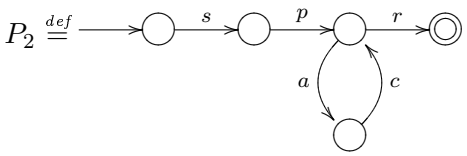
4.4 Intervention chirurgicale

Protocole P_2 On commence par une analyse de sang (phase s) avant toute injection, ensuite on effectue une photographie de l'organe (phase p), ensuite on peut effectue une série d'anesthésie (phase a) suivie d'une intervention chirurgicale (phase c). Enfin on surveille le patient jusqu'à son réveil (phase r)

Codage du protocole P_2 sous forme d'automate

Q17. Donnez l'alphabet Σ_2 du protocole P_2

Q18. Donnez l'expression régulière correspondant à l'automate ci-dessous



4.5 Combinaisons de protocoles

Modélisation de contraintes par des automates Si on souhaite combiner les protocoles P_1 et P_2 il y a deux contraintes à respecter :
- C_1 : L'analyse de sang (phase s) doit avoir lieu avant l'injection de produits radioactifs (phase i)
- C_2 : Après une injection (phase i), on doit faire une neutralisation 1 (phase n_1) avant toute anesthésie (phase a)

Pour modéliser les contraintes C_1 et C_2 vous avez le droit d'utiliser
- des expressions régulières,
- ou des automates (déterministes ou non)
- et les opérateurs ensemblistes ($\cap, \cup, \overline{\ }, \setminus, ...$) **à condition de justifier** comment réaliser cette opération sur les automates ou les expressions régulières.

On ne demande **PAS DE CALCUL** mais de donner une définition précise de chaque contrainte et de justifiez votre modélisation.

Q19. Expliquez comment modéliser la contrainte C_1 sur l'alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$.

Q20. Expliquez comment modéliser la contrainte C_2 sur l'alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$.

Combinaisons des protocoles et des contraintes

Q21. Expliquez (précisement) comment construire un protocole P qui indique comment
- pratiquer une imagerie médicale en respectant le protocole P_1
- et d'effectuer une intervention chirurgicale en suivant le protocole P_2
- tout en respectant les contraintes C_1 et C_2

Q22. Expliquez précisement comment savoir si le protocole P est réalisable ?
Autrement dit, donnez un critère qui permet de savoir s'il est possible de satisfaire simultanément les exigences de P_1, P_2, C_1 et C_2.

Q23. Donnez un algortihme qui permet de savoir si P est réalisable.

Q24. (5 à 10 lignes) Imaginons que P soit réalisable. Proposez un dispositif informatique qui, à partir de l'automate P, permet de savoir à chaque étape du protocole quelles sont les actions autorisés.

Si vous avez répondu sans faute à toutes les questions vous décrocherez un job de chef de projet dans une société grenobloise travaillant sur les systèmes d'information pour le monde médicale et vous sauverez des vies.

Exercice 5 : Inclusions de langages (20min)

10 pt

On considère l'alphabet $\Sigma = \{a, b\}$ et les langages L_1 à L_5 ci-dessous.

Q25. Comparez les langages 2 à 2 et indiquez les relations qu'ils vérifient : $\subseteq$ (inclusion large), $\subset$ (inclusion stricte), $=$ (égalité), ou bien intersection vide. Justifiez chacunes de vos réponses par un raisonnement ou un contre-exemple.
Les justifications comptent pour $\frac{2}{3}$ des points.

- $L_1 \stackrel{def}{=} \mathscr{L}\big(\ \Sigma^*\ \big)$
- $L_2 \stackrel{def}{=} \mathscr{L}\big(\ (b+a)^*\ \big)$
- $L_3 \stackrel{def}{=} \mathscr{L}\big(\ \big((a^*).b + a.(b^*)\big)^*\ \big)$
- $L_4 \stackrel{def}{=} \mathscr{L}\big(\ a^* + b^*\ \big)$
- $L_5 \stackrel{def}{=} \mathscr{L}\big(\ (ab)^*\ \big)$

SOLUTION

- $L_2, L_3, L_4, L_5 \subseteq L_1$
  car L_1 est le langage universel donc tout langage sur $\{a, b\}$ est inclu dans L_1 .
- $L_2 = L_1$
  justification (au choix) :
  (a) *en comparant les automates de L_1 et L_2 : l'automate associé à L_2 est le même que l'automate de L_1*
  (b) *$\mathscr{L}(a+b) = \mathscr{L}(a) \cup \mathscr{L}(b) = \{a\} \cup \{b\} = \Sigma$ donc $\mathscr{L}((a+b)^*) = \Sigma^*$*
- $L_2 \subseteq L_3$ *et donc* $L_2 = L_3$
  justification : $\epsilon \in \mathscr{L}(b^*)$ donc $a \in \mathscr{L}(a.(b^*))$ de même $b \in \mathscr{L}((a^*).b)$ donc $\mathscr{L}(b+a) \subseteq \mathscr{L}\big((a^*).b + a.(b^*)\big)$
  d'où $L_2 = \mathscr{L}((b+a)^*) \subseteq L_3 = \mathscr{L}(\big((a^*).b + a.(b^*)\big)^*)$
- $L_3 = \Sigma^*$
  car $\Sigma^* = L_2 \subseteq L_3$ et forcément $L_3 \subseteq \Sigma^*$ puisque L_3 est un langage sur l'alphabet Σ
- $L_4 \subset L_1 = L_2 = L_3$
  car $ab \in L_1$ et $ab \notin L_4$
- $L_5 \subset L_1 = L_2 = L_3$
  car $b \in L_1$ et $b \notin L_5$
- $L_5 \neq L_4$
  car d'une part $ab \in L_5$ et $ab \notin L_4$ donc $L_5 \not\subseteq L_4$
  et d'autres part $a \in L_4$ et $a \notin L_5$ donc $L_4 \not\subseteq L_5$
- $L_4 \cap L_5 = \{\epsilon\}$

15 pt

Exercice 6 : Construction d'automates reconnaisseurs (30min)

On considère l'alphabet $\Sigma = \{a, b\}$. Le but de l'exercice est de construire un automate qui reconnaît le langage L formé des mots qui *contiennent au moins un b* **si** *ils contiennent un nombre pair de a.*

> **Indication :** Les mots suivants appartiennent à L : *a, b, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaab, aaba, aabb, abaa, abab, abba, abbb, baaa, baab, baba, babb, bbaa, bbab, bbba, bbbb, aaaaa, aaaab, aaaba, aaabb, aabaa, aabab, aabba, aabbb, abaaa, abaab, ababa, ababb, abbaa, abbab, abbba, abbbb, baaba, babaa, babab, babba, bbaaa, bbaab, bbaba, bbabb, bbbaa, bbbab, bbbba, aaaaab, ...*

Q26. Complétez (sur votre copie) la formule suivante qui décrit L

$$L \stackrel{def}{=} \{\omega \mid \underbrace{(|\omega|_a \ldots\ldots\ldots\ldots)}_{C_a} \ldots\ldots \underbrace{(|\omega|_b \ldots\ldots\ldots\ldots)}_{C_b}\}$$

où $|\omega|_s$ désigne le nombre de symbole s dans le mot ω

Construction des automates A et B On désigne par A l'automate qui correspond à la contrainte sur les a (accolade C_a) et par B l'automate qui correspond à la contrainte sur les b (accolade C_b).

Q27. Donnez l'automate A sur l'alphabet $\Sigma = \{a, b\}$.

SOLUTION

```
val aut_A : int automaton =
  {name = "H(Pair(a))" ; alphabet = ["a"; "b"]; initial = [1]; accepting = [1];
   transitions = [(1, "a", 2); (2, "a", 1); (1, "b", 1); (2, "b", 2)]}
```

Q28. Donnez l'automate B sur l'alphabet $\Sigma = \{a, b\}$.

SOLUTION

```
val aut_B : int automaton =
  {name = "AuMoinsUn(b)"; alphabet = ["b"; "a"]; initial = [1]; accepting = [2];
   transitions = [ (1, "b", 2); (2, "b", 2); (1, "a", 1); (2, "a", 2)]}
```

Q29. Sachant que $P \Rightarrow Q$ équivaut à $\neg P \vee Q$, donnez l'automate qui reconnaît le langage L **en expliquant les étapes de la construction**.

SOLUTION

```
val aut_cA : int automaton =
  {name = "C(Pair(a))"; alphabet = ["a"; "b"]; initial = [1]; accepting = [2];
   transitions = [(1, "a", 2); (1, "b", 1); (2, "a", 1); (2, "b", 2)]}

val aut : int automaton =
  {name = "C(Pair(a)) + AuMoinsUn(b)";
   alphabet = ["a"; "b"]; initial = [1]; accepting = [5; 3];
   transitions =
    [(4, "a", 5); (4, "b", 4); (5, "a", 4); (5, "b", 5); (2, "b", 3);
     (3, "b", 3); (2, "a", 2); (3, "a", 3); (1, "", 4); (1, "", 2)]}
```

Q30. calculatoire Éliminez les ϵ-transtions.

SOLUTION

```
val aut : int automaton =
  {name = "E*(C(Pair(a) + AuMoinsUn(b))";
   alphabet = ["a"; "b"]; initial = [1]; accepting = [3; 5];
   transitions =
    [(1, "a", 2); (1, "a", 5); (1, "b", 3); (1, "b", 4); (2, "a", 2);
     (2, "b", 3); (3, "a", 3); (3, "b", 3); (4, "a", 5); (4, "b", 4);
     (5, "a", 4); (5, "b", 5)]}
```

Q31. calculatoire Déterminisez l'automate.

SOLUTION

```
val aut : int automaton =
  {name = "D(E*(C(Pair(a) + AuMoinsUn(b)))";
 alphabet = ["a"; "b"]; initial = [[1]];
 accepting = [[2; 5]; [3; 4]; [3; 5]];
 transitions =
  [([3; 5], "b", [3; 5]); ([3; 5], "a", [3; 4]); ([3; 4], "b", [3; 4]);
   ([3; 4], "a", [3; 5]); ([2; 5], "b", [3; 5]); ([2; 5], "a", [2; 4]);
   ([2; 4], "b", [3; 4]); ([2; 4], "a", [2; 5]); ([1], "b", [3; 4]);
   ([1], "a", [2; 5])]}

 Renaming of configurations: 1={1} 2={2,4} 3={2,5} 4={3,4} 5={3,5}

val aut : std automaton =
  {name = "D(E*(C(Pair(a) + AuMoinsUn(b)))";
   alphabet = ["a"; "b"]; initial = [1]; accepting = [3; 4; 5];
   transitions =
    [(5, "b", 5); (5, "a", 4); (4, "b", 4); (4, "a", 5); (3, "b", 5);
     (3, "a", 2); (2, "b", 4); (2, "a", 3); (1, "b", 4); (1, "a", 3)]}
```

Q32. calculatoire Minimisez l'automate.

SOLUTION

```
Minimisation:

 * initial partition = { {1,2,3,4,5} }

 states 1 ~/~ 3 : NOT same accepting status
 So, {1,2,3,4,5}  is splitted into  {1,2} |_| {3,4,5}

 states 3 ~/~ 4 : NOT same behavior on symbol 'a'
 So, {3,4,5}  is splitted into  {3} |_| {4,5}

 * final partition = { {1,2} , {3} , {4,5} }

Renaming of eq. classes: 1={1,2} 2={3} 3={4,5}
```

```
val aut : std automaton =
  {name = "M(D(E*(C(Pair(a) + AuMoinsUn(b))))";
   alphabet = ["a"; "b"]; initial = [1]; accepting = [2; 3];
   transitions =
    [(1, "a", 2); (1, "b", 3); (2, "a", 1); (2, "b", 3); (3, "a", 3);
     (3, "b", 3)]}
```

Q33. calculatoire Donnez une expression régulière équivalente.

Q34. Sachant que $P \Rightarrow Q$ équivaut aussi à $\neg(P \wedge \neg Q)$, proposez une autre méthode de construction de l'automate qui reconnaît le langage L.

On ne demande pas de faire les calculs, mais uniquement de décrire les étapes de la construction.

SOLUTION

$$\mathscr{L}(C_a \Rightarrow C_b) \equiv \mathscr{L}(\neg(C_a \wedge \neg C_b)) \equiv \overline{\mathscr{L}(A) \cap \overline{\mathscr{L}(B)}} \equiv \mathscr{L}((A \times B^C)^C)$$

Donc pour construire l'automate qui reconnaît L

1. *on déterminise B et on prend son complémentaire*
2. *on fait le produit de A avec le complémentaire de B. On obtient une automate déterministe car le produit de deux automates déterministes donne un automate déterministe.*
3. *On prend le complémentaire de l'automate obtenu*
4. *On le minimise*

PAS DEMANDÉ

```
(* construction de aut_cB *)

let aut_cB = explained_complementary aut_B ;;
let aut_cB = normalize aut_cB ;;

val aut_cB : int automaton =
  {name = "N(C(FS(D(H(AuMoinsUn(b))))))"; alphabet = ["a"; "b"];
   initial = [1]; accepting = [1];
   transitions = [(1, "b", 2); (1, "a", 1); (2, "b", 2); (2, "a", 2)]}

(* verification de aut_cB *)

print_words_such_that length_leq 7 (normalize aut_cB) ;;

L(N(N(C(FS(D(H(AuMoinsUn(b)))))))) s.t. |w|<=7 = { w in L1 s.t. |w|<=7 } =
 {$,a,aa,aaa,aaaa,aaaaa,aaaaaa,aaaaaaa}

(* construction de aut *)

let aut = product aut_A aut_cB ;;

val aut : (int * int) automaton =
  {name = "H(Pair(a))_x_N(C(FS(D(H(AuMoinsUn(b))))))"; alphabet = ["a"; "b"];
```

```
    initial = [(1, 1)]; accepting = [(1, 1)];
    transitions =
     [((1, 1), "a", (2, 1)); ((1, 1), "b", (1, 2)); ((1, 2), "a", (2, 2));
      ((1, 2), "b", (1, 2)); ((2, 1), "a", (1, 1)); ((2, 1), "b", (2, 2));
      ((2, 2), "a", (1, 2)); ((2, 2), "b", (2, 2))]}

let aut = complementary_of_deterministic_automaton aut ;;

val aut : (int * int) set automaton =
  {name = "C(FS(H(Pair(a))_x_N(C(FS(D(H(AuMoinsUn(b))))))))";
   alphabet = ["a"; "b"]; initial = [[(1, 1)]];
   accepting = [[]; [(1, 2)]; [(2, 1)]; [(2, 2)]];
   transitions =
    [([], "a", []); ([], "b", []); ([(1, 1)], "a", [(2, 1)]);
     ([(1, 1)], "b", [(1, 2)]); ([(1, 2)], "a", [(2, 2)]);
     ([(1, 2)], "b", [(1, 2)]); ([(2, 1)], "a", [(1, 1)]);
     ([(2, 1)], "b", [(2, 2)]); ([(2, 2)], "a", [(1, 2)]);
     ([(2, 2)], "b", [(2, 2)])]}


let aut = normalize aut ;;

val aut : int automaton =
  {name = "N(C(FS(H(Pair(a))_x_N(C(FS(D(H(AuMoinsUn(b))))))))";
   alphabet = ["a"; "b"]; initial = [1]; accepting = [2; 3; 4];
   transitions =
    [(1, "a", 3); (1, "b", 2); (2, "a", 4); (2, "b", 2); (3, "a", 1);
     (3, "b", 4); (4, "a", 2); (4, "b", 4)]}


let aut = explained_minimization aut ;;


Minimisation:

 * initial partition = { {1,2,3,4} }

 states 1 ~/~ 2 : NOT same accepting status
 So, {1,2,3,4}  is splitted into  {1} |_| {2,3,4}

 states 2 ~/~ 3 : NOT same behavior on symbol 'a'
 So, {2,3,4}  is splitted into  {2,4} |_| {3}

 * final partition = { {1} , {2,4} , {3} }

Renaming of eq. classes: 1={1} 2={2,4} 3={3}

val aut : std automaton =
  {name = "Induced(N(C(FS(H(Pair(a))_x_N(C(FS(D(H(AuMoinsUn(b))))))))))";
   alphabet = ["a"; "b"]; initial = [1]; accepting = [2; 3];
   transitions =
    [(1, "a", 3); (1, "b", 2); (2, "a", 2); (2, "b", 2); (3, "a", 1);
     (3, "b", 2)]}

```