Exercice 1 : Question de cours

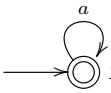
Justifiez soigneusement vos réponses par une preuve ou par un contre-exemple : Une réponse de la forme « oui/non » ne donne pas de point.

Q1. Énumérez les mots de chacun des langages suivants

(1) $\{a,b\} \cdot \{a,e\} = \{aa, ae, ba, be\}$
(2) $\{a,a\} \cdot \{a,\epsilon\} = \{aa, a, aa, a\}$
(3) $\Sigma^* \cdot \{\,\} = \{\,\}$
(4) $\{\,\}^* \cdot \Sigma = \Sigma$
(5) $\{a,\epsilon\}^* = \{a^n \mid n \in \mathbb{N}\}$

Q2. Un automate peut-il reconnaître un langage infini ?

SOLUTION

oui,  *reconnaît* $\{a^n \mid n \in \mathbb{N}\}$ *qui est infini.*

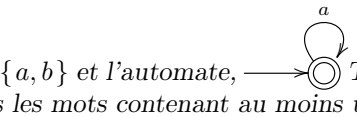
Q3. Décrivez par une phrase en français le langage correspondant à Σ^*. Est-il fini ou infini ?

SOLUTION

Le langage Σ^* *est l'ensemble de* tous *les mots* finis *écrits à l'aide de symbole de l'alphabet* Σ. Σ^* *est infini si* $\Sigma \neq \{\,\}$ *Mais si* $\Sigma = \{\,\}$ *alors* Σ^* *est fini, égal à* $\{\epsilon\}$.

Q4. Un automate dont tous les états sont accepteurs reconnaît-il forcément Σ^* ?

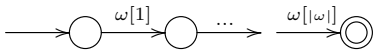
SOLUTION

Pas nécessairement. Considér ons $\Sigma = \{a,b\}$ *et l'automate,*  *Tous ses états sont accepteurs mais comme il n'est pas complet il ne reconnaît pas les mots contenant au moins un b.*

Q5. Soit L un langage fini constitué de n mots $\{\omega_1, \ldots, w_n\}$, existe-il un automate qui le reconnaît ? Si oui *donnez l'automate.* Si non *expliquez pourquoi cela est impossible.*

SOLUTION

Notons $\omega[k]$ *le* k^{ieme} *symbole du mot* ω. *Soit A l'automate constitué des transitions*



pour chaque mot ω *de* $\{\omega_1, \ldots, w_n\}$, *alors* $\mathscr{L}(A) = \{\omega_1, \ldots, w_n\}$.

Q6. Existe-t'il des langages qui ne sont pas reconnaissables par les automates (à nombre) d'états fini ?

SOLUTION

Le langage $\{a^n b^n \mid n \in \mathbb{N}\}$ *et plus généralement le langage des expressions bien parenthésées n'est pas reconnaissable par un* AEF.

Q7. Quels sont les avantages/inconvénients des automates non-déterministes par rapport aux automates déterministes ?

SOLUTION

1. *ils sont plus simples, plus concis*
2. *ils sont moins simples à exécuter (lancement d'une nouvelle exécution concurrente à chaque branchement non-déterministe).*

Q8. Décrivez par une phrase en français le langage correspondant à l'expression régulière suivante « $(a^* \mid b^*) \cdot a$ ». Est-il fini ou infini ?

SOLUTION

— *Ce langage est l'ensemble de tous les mots sur l'alphabet* $\{a, b, c\}$ *formé d'un nombre quelconque de* a *ou d'un nombre quelconque de* b *et terminé par* a

— $L \supseteq \{a^\ell.c \mid \ell \in \mathbb{N}\}$. *Ce langage est donc infini.*

Exercice 2 : Grammaire des expressions arithmétiques

Q9. Donnez une expression régulière *Integer* qui reconnait les entiers de $\mathbb{Z}$ avec des 0 non-significatifs tels que `00123`, `-1`.

$Integer = (0|1|\dots|9) \cdot (0|1|\dots|9)^*$

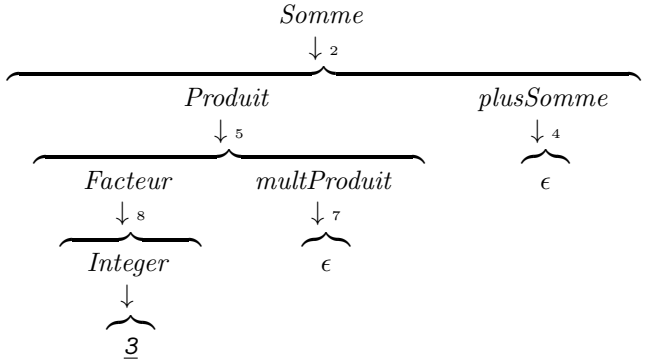
Grammaire des expressions arithmétiques On considère la grammaire constituée des règles suivantes qui définit le langage des expressions arithmétiques :

Expr	$\xrightarrow{1}$	*Somme*	*multProduit*	$\xrightarrow{6}$	$\underline{*} \cdot$ *Produit*
Somme	$\xrightarrow{2}$	*Produit* $\cdot$ *plusSomme*	*multProduit*	$\xrightarrow{7}$	ϵ
plusSomme	$\xrightarrow{3}$	$\underline{+} \cdot$ *Somme*	*Facteur*	$\xrightarrow{8}$	*Integer*
plusSomme	$\xrightarrow{4}$	ϵ	*Facteur*	$\xrightarrow{9}$	$\underline{(} \cdot$ *Expr* $\cdot \underline{)}$
Produit	$\xrightarrow{5}$	*Facteur* $\cdot$ *multProduit*			

Q10. Donnez un arbre de dérivation qui produit l'expression « `3` » à partir du germe *Somme* .

SOLUTION

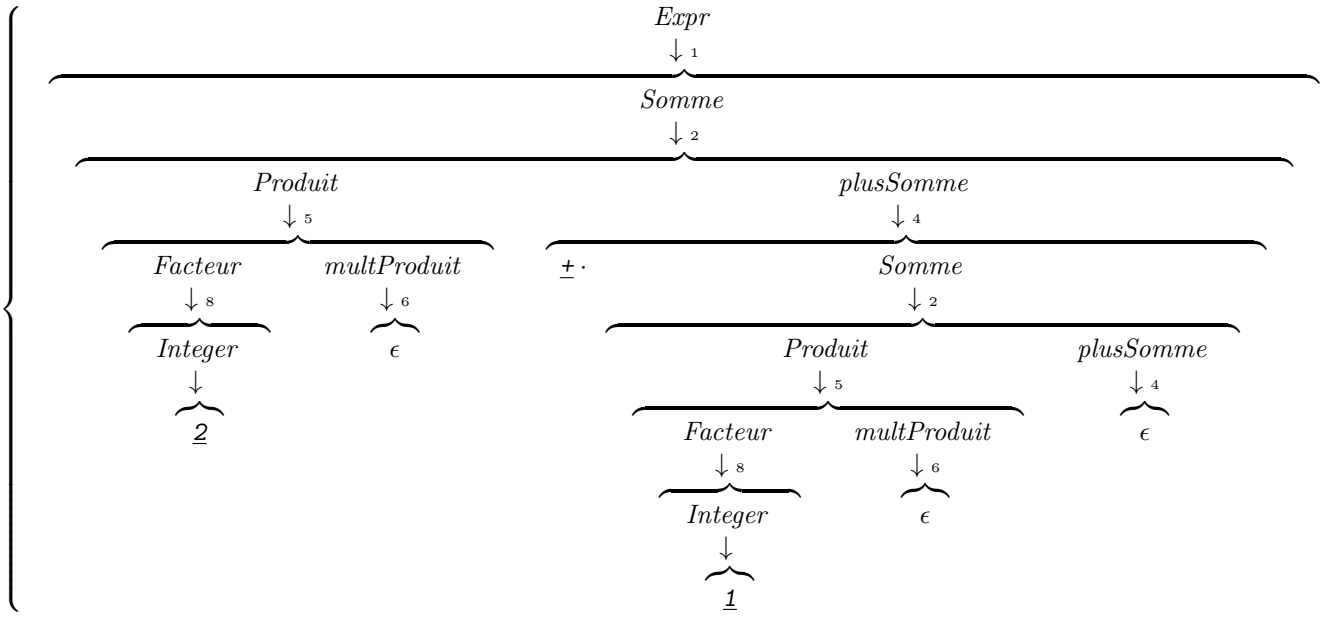
$\Delta_{Q10} =$



Q11. Donnez un arbre de dérivation qui produit l'expression « `2 + 1` » à partir du germe *Expr* .

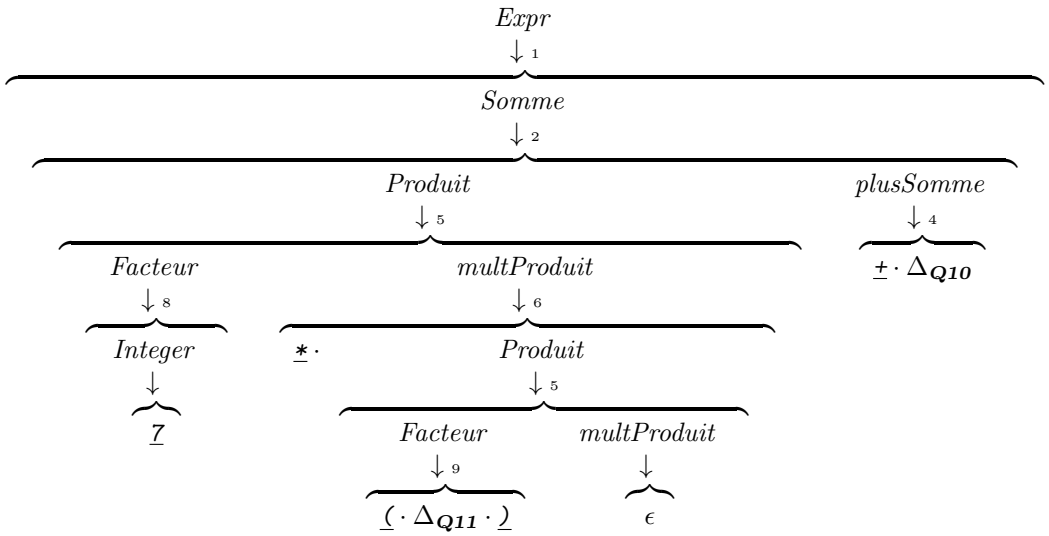
SOLUTION

$\Delta_{Q11} =$



Q12. Donnez un arbre de dérivation qui produit l'expression « `7 * (2 + 1) + 3` » à partir du germe *Expr* .

SOLUTION



Exercice 3 : Déterminisation d'un automate à deux états initiaux

On considère l'automate suivant qui comporte deux états initiaux et un état accepteur :

A	1_i	2	3_i^a
a	$1, 2$	3	
b	2	3	

Q13. Rappelez la définition de l'acceptation d'un mot par un automate non-déterministe.

SOLUTION

Un mot est ***reconnu/accepté*** *par un automate s'il* ***existe*** *une* ***exécution*** *de l'automate*

1. *qui* ***commence/débute*** *dans un état* ***initial***
2. *qui* ***consomme toutes*** *les lettres du mot*
3. *qui se* ***termine*** *dans un état* ***accepteur***

Q14. Donnez l'arbre d'exécution de l'automate sur le mot « *aabb* ». Dessinez toutes les branches, celles qui n'acceptent pas le mot et celles qui l'acceptent.

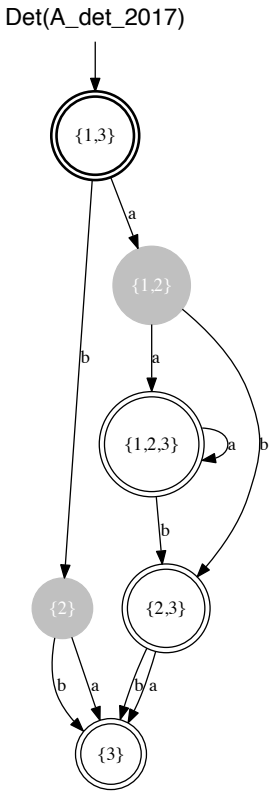
SOLUTION

$$\begin{cases} 1, aabb \xrightarrow{a} 1, abb \xrightarrow{a} 1, bb \xrightarrow{b} 2, b \xrightarrow{b} 3, \epsilon : acc \\ 1, aabb \xrightarrow{a} 1, abb \xrightarrow{a} 2, bb \xrightarrow{b} 3, b \not\to \\ 1, aabb \xrightarrow{a} 2, abb \xrightarrow{a} 3, bb \not\to \end{cases}$$

Q15. Déterminisez l'automate et donnez le résultat sous forme de tableau.

SOLUTION

A^D	$\{1,3\}_i^a$	$\{1,2\}$	$\{1,2,3\}$	$\{2\}$	$\{2,3\}^a$	$\{3\}^a$	$\{\}$
a	$\{1,2\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\{3\}$	$\{3\}$	$\{\}$	$\{\}$
b	$\{2\}$	$\{2,3\}$	$\{2,3\}$	$\{3\}$	$\{3\}$	$\{\}$	$\{\}$



Q16. Étant donné un automate non-déterministe A et A^D sa version déterminisée.
Que représente un état de A^D ?

SOLUTION

*Les **états** de A^D sont les **sous-ensembles** de copies de A s'exécutant en **parallèle** à partir d'un état différent. Par exemple, l'état $\{1,3\}$ de A^D signifie que la suite du mot sera analysée en parallèle par une copie de A dans l'état 1 et une copie de A dans l'état 3.*

Q17. Étant donné un automate non-déterministe A à n états, quel est le nombre maximal d'états que peut avoir la version déterminisée A^D ?

SOLUTION

A^D aura au maximum 2^n ***états*** *car 2^n est le nombre de **sous-ensembles*** différents *qu'on peut construire avec n **éléments/états**.*

Exercice 4 : Minimisation

Pour chacun des automates ci-dessous

1. Calculez les classes d'équivalence des états par l'algorihtme du cours
2. Donnez l'automate miminisé sous forme de tableau

Q18.

A_1	1_i	2^a	3	4^a
a	2	1	4	2
b	4	4	1	

Q19.

A_2	1_i^a	2^a	3^a	4^a
a	2	1	4	2
b	4	4	1	

Q20.

A_3	1_i	2	3	4
a	2	1	4	2
b	4	4	1	

SOLUTION

— *On commence par compléter l'automate A_1 avec un état puit 0. On calcule les classes d'équivalence*

```
Minimisation:

* initial partition = { {0,1,2,3,4} }

 states 0 ~/~ 2 : NOT same accepting status
 So, {0,1,2,3,4}  is splitted into  {0,1,3} |_| {2,4}

 states 0 ~/~ 1 : NOT same behavior on symbol 'a'
 So, {0,1,3}  is splitted into  {0} |_| {1,3}

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

 states 1 ~/~ 3 : NOT same behavior on symbol 'a'
 So, {1,3}  is splitted into  {1} |_| {3}

* final partition = { {0} , {1} , {2} , {3} , {4} }
```

L'automate A_1 est déjà minimal.

— *On commence par compléter l'automate A_2 avec un état puit 0. On calcule les classes d'équivalence*

```
Minimisation:

* initial partition = { {0,1,2,3,4} }

   states 0 ~/~ 1 : NOT same accepting status
   So, {0,1,2,3,4}  is splitted into  {0} |_| {1,2,3,4}

   states 1 ~/~ 4 : NOT same behavior on symbol 'b'
   So, {1,2,3,4}  is splitted into  {1,2,3} |_| {4}

   states 1 ~/~ 3 : NOT same behavior on symbol 'a'
   So, {1,2,3}  is splitted into  {1,2} |_| {3}

* final partition = { {0} , {1,2} , {3} , {4} }
```

Finalement, l'automate minimisé est

A_2m	$Eq\{0\}$	$Eq\{1,2\}^a{}_i$	$Eq\{4\}^a$
a	$Eq\{0\}$	$Eq\{1,2\}$	$Eq\{1,2\}$
b	$Eq\{0\}$	$Eq\{4\}$	$Eq\{0\}$

— *Aucun état accepteur donc le langage de* A_3 *est* $\mathscr{L}(A_3) = \emptyset$ *et l'automate minimal qui reconnaît* $\emptyset$ *sur* $\Sigma = \{a, b\}$ *est*

A_3m	$Eq\{1,2,3,4\}_i$
a	$Eq\{1,2,3,4\}$
b	$Eq\{1,2,3,4\}$

Q21. Complétez

— Deux automates A et A' sont équivalents si $\mathscr{L}(A) = \mathscr{L}(A')$

— Deux états q et q' sont équivalents si $\mathscr{L}(q) = \mathscr{L}(q')$

Exercice 5 : Comment vérifier le bon usage des verroux dans les drivers

L'exercice comporte 23 questions relativement indépendantes.

Implantation d'un verrou Un verrou (*lock* en anglais) sert à resteindre l'accès à une ressource partagée (une imprimante ou un port USB par exemple) à un seul processus utilisateur afin de pouvoir utiliser la ressource sans risque d'interférence avec d'autres processus.
L'accès est controlé au moyen d'une variable privée `owner` (*propriétaire*) associée à la ressource. Cette variable est gérée par les deux fonctions `get_lock()` et `unlock()` ci-après.

```
bool get_lock(){
  int pid = caller() ;
  atomic{ if (owner<0) { owner = pid } } ;
  return (owner == pid)
}
```

```
void unlock(){ owner = -1 }
```

L'instruction `caller()` retourne l'identifiant du processus (*pid* de type `integer`) qui demande de la ressource. L'instruction `get_lock()` demande à acquérir le droit d'utiliser la ressource. Si le verrou est déjà pris par un autre processus l'instruction `get_lock()` retourne `false`. Si au contraire la ressource est libre, l'instruction `get_lock()` rend le processus appelant *(caller)* propriétaire *(owner)* de la ressource et retourne `true`. L'instruction `unlock()` libère la ressource.

Exemple d'utilisation On considère 4 programmes qui utilisent un verrou pour effectuer des tâches (`working()`) tant qu'il y a du travail à faire (`todo() == true`).
Le but de cet exercice est de déterminer si ces programmes utilisent correctement le verrou.

PROGRAM$_1$

```
1 while( todo() == true ){
2   if ( get_lock() == true ){
3     working() ;
4     unlock()
5   }
6 }
```

PROGRAM$_2$

```
1 while( get_lock() == false ){
2   delay() } ;
3 while( todo() == true ){
4   working() } ;
5 unlock()
```

PROGRAM$_3$

```
1 while( todo() == true ){
2   if ( get_lock() == true ){
3     working()
4   } ;
5   unlock()
6 }
```

PROGRAM$_4$

```
1 if ( todo() == true ){
2   while( get_lock() == false ){
3     delay() } ;
4   while( todo() == true ){
5     working() } ;
6   unlock() }
```

À titre d'illustration, on donne un exemple très simple de ce que pourraient être les fonctions `todo()` et `working()`.

```
void working(){
  utiliser la ressource partagee pour effectuer la tache t ;
  t = t-1 ;
}
```

```
bool todo(){
  return (t>0) ;
}
```

Représentation des programmes par des automates On considère l'alphabet à 7 symboles

$$\Sigma = \{\texttt{delay()},\ \texttt{get\_lock()}{=}\mathbb{T},\ \texttt{get\_lock()}{=}\mathbb{F},\ \texttt{todo()}{=}\mathbb{T},\ \texttt{todo()}{=}\mathbb{F},\ \texttt{unlock()},\ \texttt{working()}\}$$

représentant les instructions possibles, où $\mathbb{T}$, respectivement $\mathbb{F}$ sont des abbréviations pour `true`, respectivement `false`.
À partir d'un programme la compilation produit un automate équivalent dans lequel les structures de contrôle while et if sont représentées par la structure de l'automate et les instructions sont placées sur les transitions. Un branchement conditionnel en fonction de la valeur de `get_lock()` se traduit par deux transitions $\xrightarrow{\texttt{get\_lock()}=\mathbb{T}}$ et $\xrightarrow{\texttt{get\_lock()}=\mathbb{F}}$. Même chose pour un test sur la valeur de `todo()`.

Q22. Donnez la correspondance entre les programmes PROGRAM$_1$, PROGRAM$_2$ et PROGRAM$_3$ et les automates de la Figure 1.

SOLUTION

$$\text{PROGRAM}_1 = P_a,\quad \text{PROGRAM}_2 = P_c,\quad \text{PROGRAM}_3 = P_b$$

Q23. Dessinez l'automate correspondant à PROGRAM$_4$.

SOLUTION

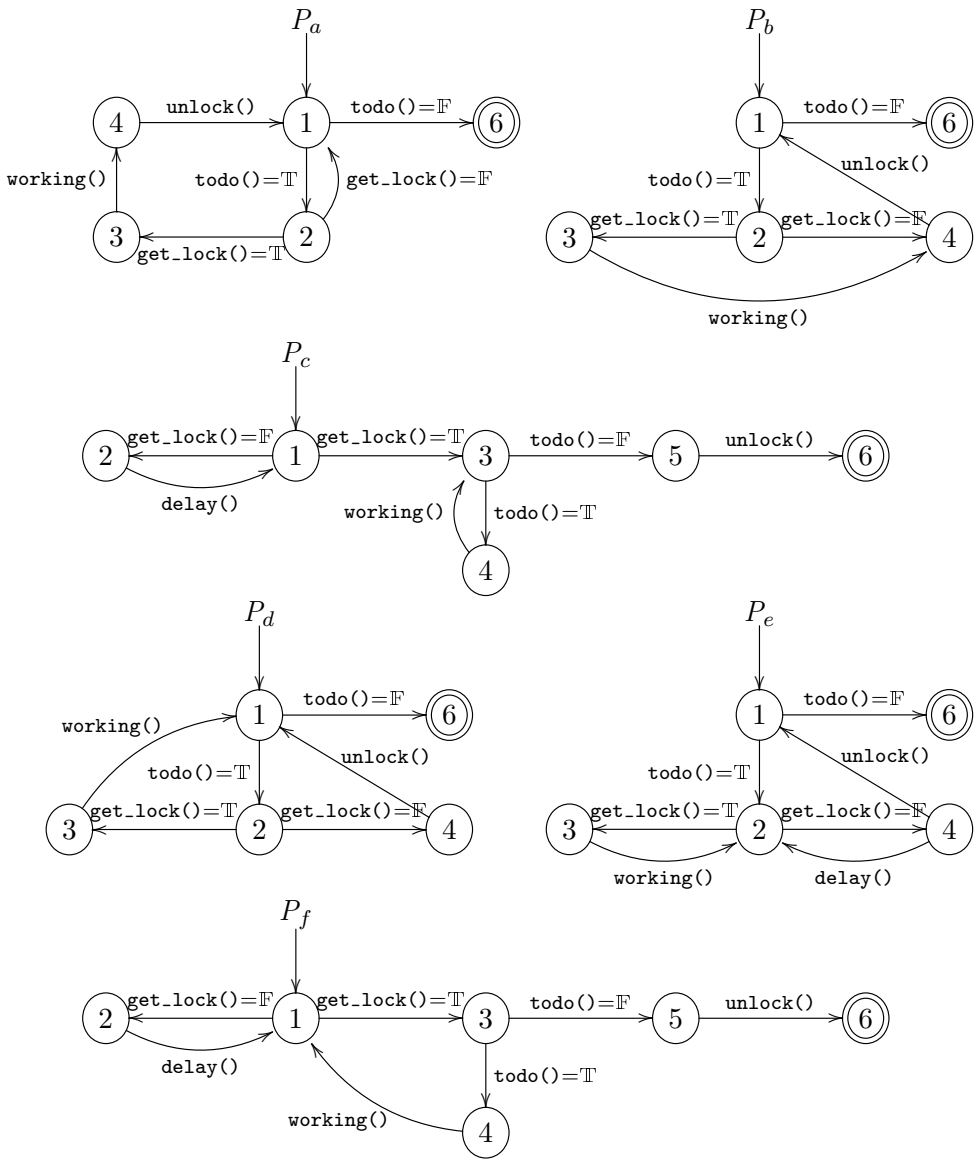
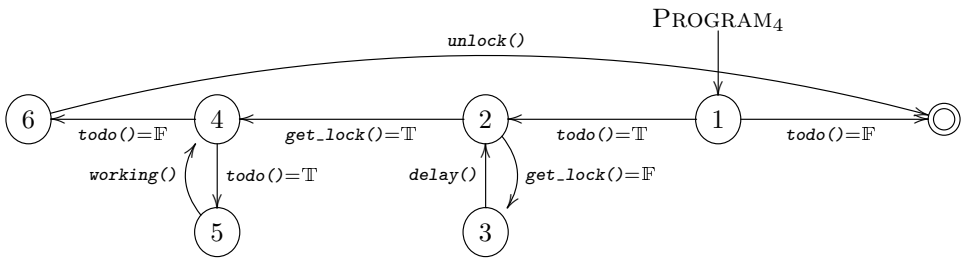


FIGURE 1 – Les automates de la question **Q22**



Q24. Soit P_k l'automate de la Figure 1 associé au programme PROGRAM$_k$. Expliquez par une phrase en français ce que représente le langage $\mathscr{L}(P_k)$.

SOLUTION

Le langage $\mathscr{L}(P_k)$ *est l'ensemble des* séquences d'instructions *de* Σ *qui correspondent à des exécutions* finies *du programme* PROGRAM$_k$.

5.1 Les règles de bon usage du verrou

On considère que les 3 instructions qui prennent du temps sont : (1) `working()` (car on effectue une tâche), (2) `delay()` (car on force à attendre) et (3) `todo() == false` (car on devra attendre de recevoir un requête de travail). Par contre, `unlock()`, `todo() == true` et `get_lock()` (peu importe son résultat) ne prennent pas de temps.

Le but de cette section est de construire un automate Aut' qui reconnaît les séquences d'instructions autorisées par les règles de bon usage du verrou.

— on nommera I_k l'automate qui reconnaît les *séquences interdites* par la règle (r_k)

— on nommera A_k l'automate qui reconnaît les *séquences autorisées* par la règle (r_k)

Les 6 règles de bon usage du verrou

(r_1) Il est interdit de faire un appel à `get_lock()` (peu importe le résultat $\mathbb{T}$ ou $\mathbb{F}$) après un `get_lock()`=$\mathbb{T}$ sans avoir fait un `unlock()` à un moment entre les deux `get_lock()`.

(r_2) Il est interdit de faire un `unlock()` sans avoir eu auparavant un `get_lock()`=$\mathbb{T}$

(r_3) Il est interdit de faire plusieurs appel à `get_lock()` sans avoir fait au moins une instruction intermédiaire qui prend du temps (`delay()`, `todo()`=$\mathbb{F}$ ou `working()`) (c'est pourquoi les programmes contiennent des instructions `delay()` à certains endroits)

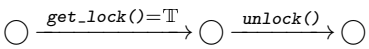
(r_4) Il est interdit de faire un `get_lock()` (peu importe le résultat $\mathbb{T}$ ou $\mathbb{F}$) après un `unlock()` sans au moins une instruction intermédiaire qui prend du temps

(r_5) Les instructions `working()` sont autorisées uniquement lorsque le verrou sur la ressource est acquis (`get_lock()`=$\mathbb{T}$ non suivi d'un `unlock()`).

(r_6) Il est interdit d'attendre (instruction `delay()`) lorsqu'on a acquis la ressource (`get_lock()`=$\mathbb{T}$)

Q25. Donnez une séquence de 2 instructions, autorisée par la règle (r_2)

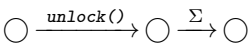
SOLUTION



Q26. **Complétez :** Cette séquence doit donc être rejetée par I_2 et acceptée par A_2.

Q27. Donnez une séquence de 2 instructions, interdite par la règle (r_2).

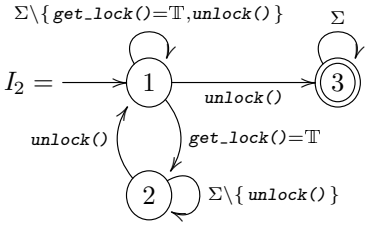
SOLUTION



Q28. **Complétez :** Cette séquence doit donc être acceptée par I_2 et rejetée par A_2.

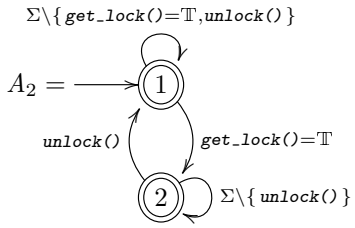
Q29. Dessinez un automate I_2 à 3 états, *déterministe et complet*, qui reconnaît les séquences *interdites* par la règle (r_2).

SOLUTION



Q30. Dessinez un automate A_2, *déterministe et minimal*, qui reconnaît les séquences *autorisées* par la règle (r_2).

SOLUTION



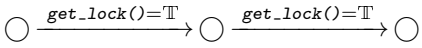
Q31. Quel lien y a t'il entre I_2 et A_2 ?

SOLUTION

Ils sont complémentaires : $A_2 = (I_2)^C$

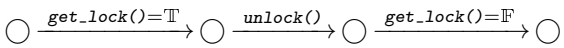
Q32. Donnez une séquence de 2 instructions, interdite par la règle (r_1)

SOLUTION



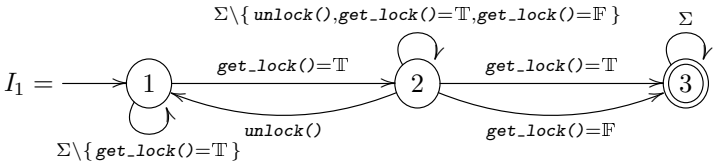
Q33. Donnez une séquence de 3 instructions, autorisée par la règle (r_1)

SOLUTION



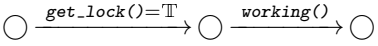
Q34. Dessinez un automate I_1 à 3 états, *déterministe et complet*, qui reconnaît les séquences *interdites* par la règle (r_1).

SOLUTION



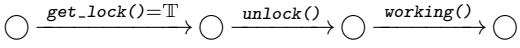
Q35. Donnez une séquence de 2 instructions, autorisée par la règle (r_5).

SOLUTION



Q36. Donnez une séquence de 3 instructions, interdite par la règle (r_5).

SOLUTION



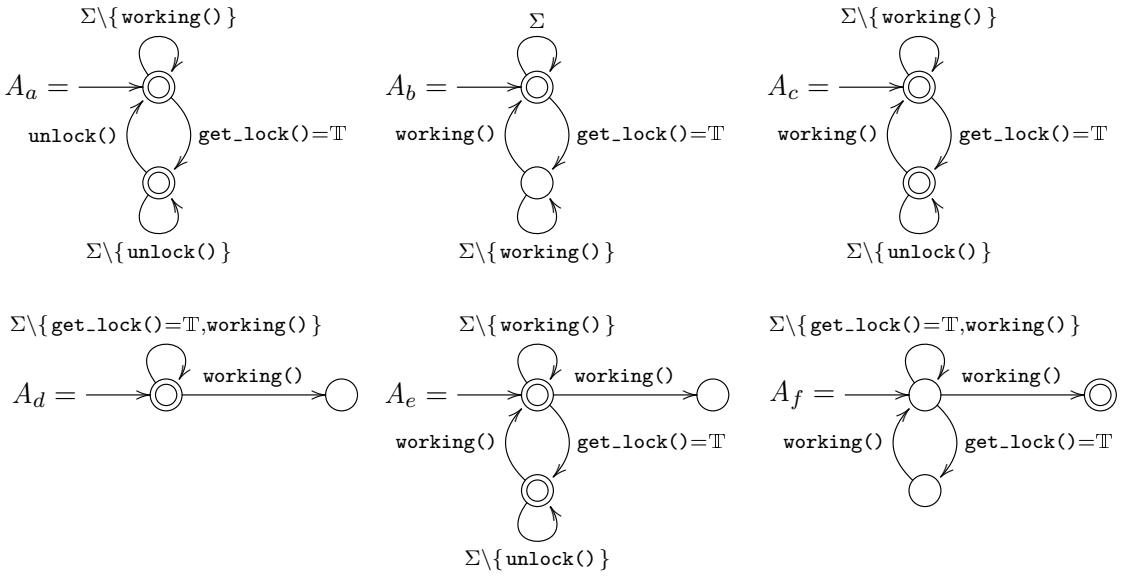


FIGURE 2 – Les automates de la question **Q37**

Q37. Parmi les automates de la figure 2, indiquez ceux qui reconnaîssent les séquences d'instructions *autorisées* par la règle (r_5).

SOLUTION

Seul l'automate A_a reconnaît les séquences autorisées *par la règle (r_5).*

Q38. Donnez une expression, constituée des automates $(I_k,\ A_k)$ et des opérations suivantes : $\times, +$, (concaténation : $\bullet$), (itération de Kleene : $\ldots^*$), (complémentaire : $\ldots^C$), (déterminisation : $\ldots^D$), (minimisation : $\ldots^M$) qui permet d'obtenir l'automate Aut' *déterministe et minimal* qui reconnaît les séquences d'instructions *qui respectent toutes les règles de bon usage du verrou.*

> **Indication :** L'opération (privé de : « − ») n'est pas autorisée. Vous prendrez soin d'indiquer les déterminisations nécessaires. Donnez votre réponse sous la forme d'une équation $Aut' = \ldots$ et justifiez votre construction

SOLUTION

De manière générale $A_k = (I_k^D)^C$ et inversement $I_k = (A_k^D)^C$. On veut construire un automate qui reconnaît les séquences autorisées par la conjonction des règles 1 à 5. Le langage $\mathscr{L}(Aut')$ est l'intersection des langages $\mathscr{L}(A_k)$. Aut' est donc le produit des automates A_k.

$$Aut' \stackrel{\text{def}}{=} \left((A_1 \times A_2 \times A_3 \times A_4 \times A_5 \times A_6)^D\right)^M = \left(((I_1 + I_2 + I_3 + I_4 + I_5 + I_6)^D)^C\right)^M$$

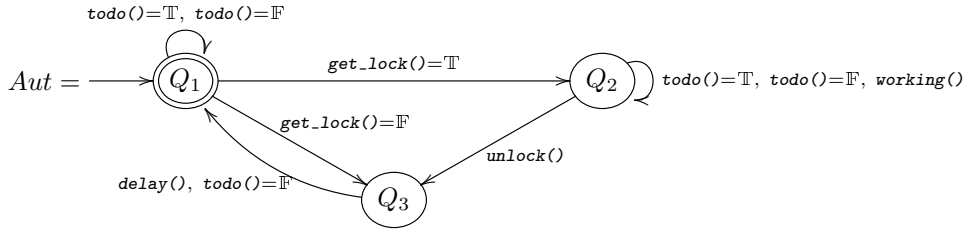
Définition des règles de bonne usage directement sous forme d'automate, sans description en français L'automate Aut' défini par les règles précédentes a trop d'états pour un sujet d'examen (436 avant minimisation, 8 après), donc nous poursuivrons avec l'automate Aut ci-dessous qui définit

les séquences d'instructions qui respectent les règles de bon usage du verrou.

Aut	$\searrow Q_1^{acc}$	Q_2	Q_3
`delay()`			Q_1
`get_lock()`$=\mathbb{T}$	Q_2		
`get_lock()`$=\mathbb{F}$	Q_3		
`todo()`$=\mathbb{T}$	Q_1	Q_2	
`todo()`$=\mathbb{F}$	Q_1	Q_2	Q_1
`unlock()`		Q_3	
`working()`		Q_2	

Q39. Dessinez l'automate Aut.

SOLUTION



Q40. Pour les deux automates P_b et P_c de la Figure 1 donnez l'exécution la plus courte (de l'état initial 1 à l'état de sortie 6) qui n'est pas acceptée par Aut.

Indication : Donnez vos réponse sous la forme : l'exécution $1 \xrightarrow{\cdots} \ldots \xrightarrow{\cdots} 6$ de P_k n'est pas acceptée par Aut car ...

SOLUTION

$P_b : 1 \xrightarrow{\texttt{todo()=T}} 2 \xrightarrow{\texttt{get\_lock()=F}} 4 \xrightarrow{\texttt{unlock()}} 1 \xrightarrow{\texttt{todo()=F}} 6$ *car* $\xrightarrow{\texttt{unlock()}}$ *n'est pas possible en* Q_3

$P_c : 1 \xrightarrow{\texttt{get\_lock()=T}} 3 \xrightarrow{\texttt{todo()=F}} 5 \xrightarrow{\texttt{unlock()}} 6$ *termine dans l'état* Q_3 *non accepteur*

Vérification d'un programme

Q41. Rappelez les étapes de construction de l'automate A^C qui reconnaît le langage complémentaire d'un automate A.

SOLUTION

1. *on déterminise l'automate A si nécessaire*
2. *on complète l'automate A si nécessaire*
3. *on inverse le statut accepteurs/non-accepteurs des états*

Q42. Construire l'automate Aut^C sous forme de tableau.

SOLUTION

Aut^C	$\searrow Q_1$	Q_2^{acc}	Q_3^{acc}	X^{acc}
`delay()`	X	X	Q_1	X
`get_lock()`$=\mathbb{T}$	Q_2	X	X	X
`get_lock()`$=\mathbb{F}$	Q_3	X	X	X
`todo()`$=\mathbb{T}$	Q_1	Q_2	X	X
`todo()`$=\mathbb{F}$	Q_1	Q_2	Q_1	X
`unlock()`	X	Q_3	X	X
`working()`	X	Q_2	X	X

Q43. Donnez une interprétation (en une phrase en français) de ce que représente $\mathscr{L}(Aut^C)$.

SOLUTION

Le langage $\mathscr{L}(Aut^C)$ est l'ensemble des séquences d'instructions qui ne respectent pas les règles de bonnes usages du verrou.

Q44. Détaillez les étapes qui permettent de s'assurer qu'un programme PROGRAM$_k$ respecte les règles de bon usage du verrou imposées par Aut.

SOLUTION

On construit l'automate P_k associé au programme PROGRAM$_k$ *par compilation. On cherche ensuite à savoir si $\mathscr{L}(P_k) \stackrel{?}{\subseteq} \mathscr{L}(Aut)$, ce qui équivaut à $\mathscr{L}(P_k) \cap \overline{\mathscr{L}(Aut)} \stackrel{?}{=} \emptyset$, soit encore à $\mathscr{L}(P_k \times Aut^C) \stackrel{?}{=} \emptyset$.*
Le programme PROGRAM$_k$ *respecte les règles de bon usage du verrou imposées par Aut si le langage de l'automate $P_k \times Aut^C$ est vide, c'est à dire si cet automate n'a pas d'état accepteur accessible depuis un état initial.*