1 Comment vérifier le bon usage des verroux dans les drivers (60 min) (20 pt)

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) ;
}
```

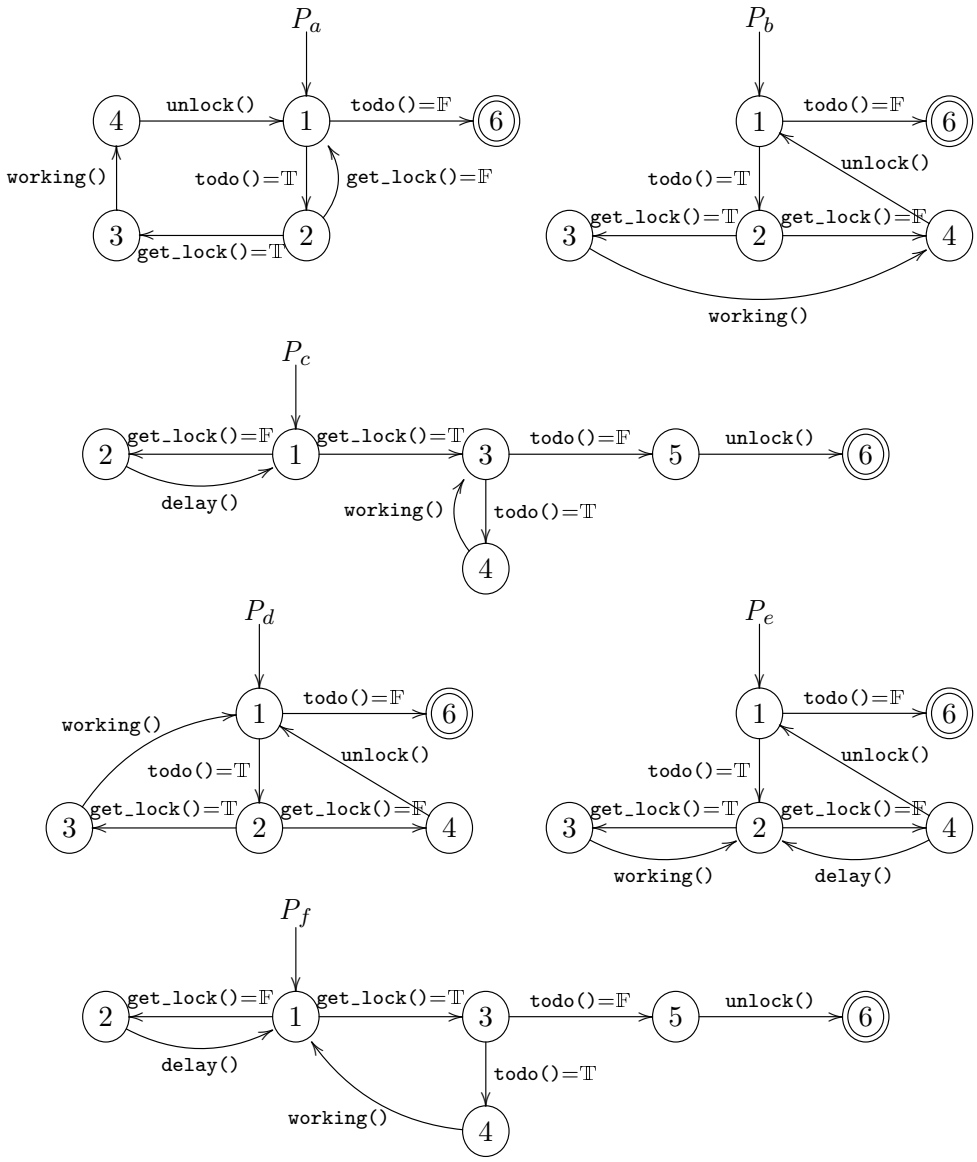

FIGURE 1 – Les automates de la question **Q1**

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()`.

Q1. (1.5 pt) Donnez la correspondance entre les programmes PROGRAM$_1$, PROGRAM$_2$ et PROGRAM$_3$ et les automates de la Figure 1.

Q2. (1.5 pt) Dessinez l'automate correspondant à PROGRAM$_4$.

Q3. (0.5 pt) 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 $\mathscr{L}(P_k)$.

1.1 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()`=$\mathbb{T}$)

Q4. (0.25 pt) Donnez une séquence de 2 instructions, autorisée par la règle (r_2)

Q5. (0.25 pt) **Complétez :** Cette séquence doit donc être ..... par I_2 et ..... par A_2.

Q6. (0.25 pt) Donnez une séquence de 2 instructions, interdite par la règle (r_2).

Q7. (0.25 pt) **Complétez :** Cette séquence doit donc être ..... par I_2 et ..... par A_2.

Q8. (1.5 pt) Dessinez un automate I_2 à 3 états, *déterministe et complet*, qui reconnaît les séquences *interdites* par la règle (r_2).

Q9. (1 pt) Dessinez un automate A_2, *déterministe et minimal*, qui reconnaît les séquences *autorisées* par la règle (r_2).

Q10. (0.5 pt) Quel lien y a t'il entre I_2 et A_2 ?

Q11. (0.5 pt) Donnez une séquence de 2 instructions, interdite par la règle (r_1)

Q12. (0.5 pt) Donnez une séquence de 3 instructions, autorisée par la règle (r_1)

Q13. (1.5 pt) Dessinez un automate I_1 à 3 états, *déterministe et complet*, qui reconnaît les séquences *interdites* par la règle (r_1).

Q14. (0.5 pt) Donnez une séquence de 2 instructions, autorisée par la règle (r_5).

Q15. (0.5 pt) Donnez une séquence de 3 instructions, interdite par la règle (r_5).

Q16. (1.5 pt) Parmi les automates de la figure 2, indiquez ceux qui reconnaîssent les séquences d'instructions *autorisées* par la règle (r_5).

Q17. (1.5 pt) 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 $Aut' = \ldots$ (.75 pt) et justifiez votre construction (.75 pt)

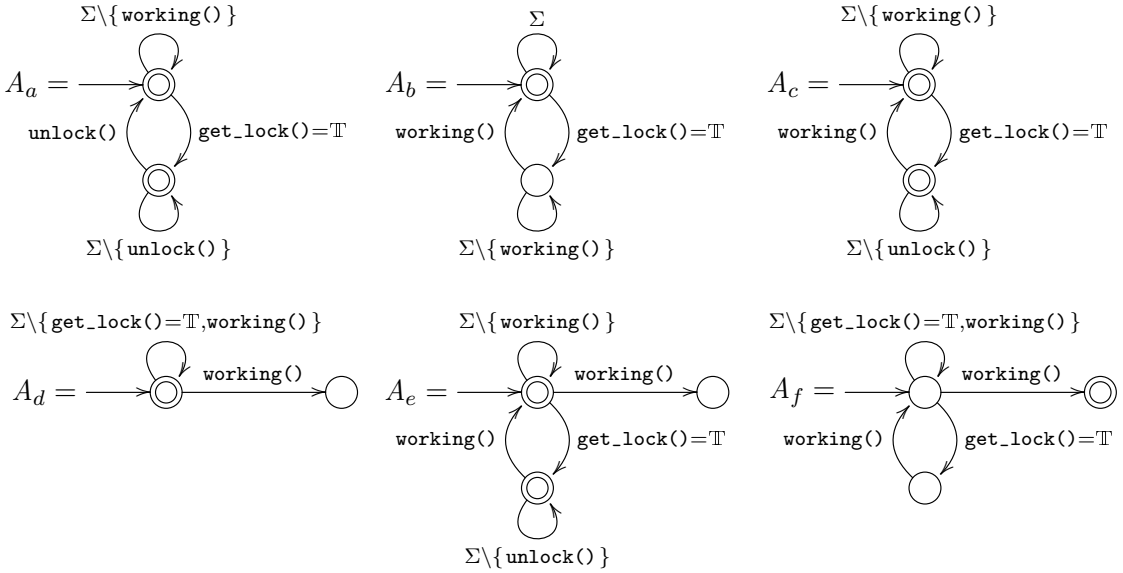

FIGURE 2 – Les automates de la question **Q16**

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	

Q18. (0.75 pt) Dessinez l'automate Aut.

Q19. (1.5 pt) 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 ...

Vérification d'un programme

Q20. (0.5 pt) Rappelez les étapes de construction de l'automate A^C qui reconnaît le langage complémentaire d'un automate A.

Q21. (0.5 pt) Construire l'automate Aut^C sous forme de tableau.

Q22. (0.25 pt) Donnez une interprétation (en une phrase en français) de ce que représente $\mathscr{L}(Aut^C)$.

Q23. (2 pt) Détaillez les étapes qui permettent de s'assurer qu'un programme PROGRAM_k respecte les règles de bon usage du verrou imposées par Aut.