SCIENCES SORBONNE UNIVERSITÉ

Examen Réparti 2 ASTRE
Master 2 Informatique Fév 2021

UE 5IN455

Année 2020-2021

2 heures – **Documents manuscrits autorisés**
Tout appareil de communication électronique interdit (téléphones. . .)

Le sujet est composé de sections indépendantes que l'on pourra traiter dans l'ordre souhaité. On vous demande de rédiger les deux parties sur deux copies séparées.

Part I.
Ordonnancement

1. Ordonnancement multi-processeurs temps réel (2 points)

Dans cet exercice, nous cherchons à ordonnancer sur un processeur à deux coeurs le système suivant de tâches synchrones périodiques à échéances implicites :

Tâches	Budget	Période
T1	1	4
T2	1	5
T3	18	20

Question 1. (1 point) Rappeler le fonctionnement de Global-EDF. Ce système est-il ordonnançable en utilisant Global-EDF ?

> En faisant le chronogramme sur l'hyper-période de 20, on trouve que le système est ordonnançable.

Question 2. (1 point) On augmente la période de T1 de 4 à 5. Ce système est-il ordonnançable en utilisant Global-EDF ? Commenter les résultats des deux questions de la section ?

> En faisant le chronogramme sur l'hyper-période de 20, on trouve que le système n'est pas ordonnançable. T1 et T2 s'exécutent par trois fois au moins simultanément ce qui retarde T3 de 3 unités de temps pour lui faire rater son échéance.
>
> Le second système est moins chargé que le premier. Pourtant, le premier est ordonnançable alors que le second ne l'est pas. Il s'agit d'un phénomène connu sous le nom d'anomalie d'ordonnancement.

2. Couverture de code (2 points)

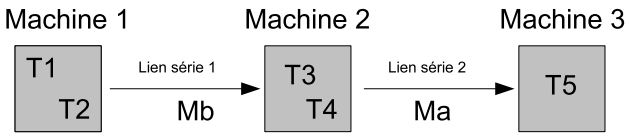
Indiquez ce que doit faire un banc de tests qui cherche à garantir la couverture de code selon les approches 1) Statement Coverage, 2) Decision Coverage et 3) Modified Condition/Decision Coverage. Illustrez chacune de ces approches en fournissant des valeurs possibles de a, b et c pour les tests à fournir. Une attention particulière sera accordée aux explications.

$$if((a == 1)||((b! = 3)||(c == 4)))F_1(a, b, c); F_2(a, b, c);$$

- pour 1) il faut 1 test avec la décision vraie
- pour 2) il faut 2 tests avec la décision vraie et fausse
- pour 3) il faut faire 4 tests en faisant varier les conditions de sorte que lorsqu'une seule condition varie, deux autres restant fixes, la décision varie.

	$a == 1$	$b! = 3$	$c == 4$	Decision
test1	False	False	False	False
test2	True	False	False	True
test3	False	True	False	True
test4	False	False	True	True

3. Ordonnancement temps réel réparti (4 points)



On vous demande d'assister un ingénieur qui doit analyser une chaîne de production de casseroles. Cette chaîne est constituée de trois tâches, hébergées sur 3 machines (cf. figure ci-dessus):

1. Sur la machine 1, la tâche T1 produit par emboutissage les fonds de casseroles. 2. Puis, la tâche T3 de la machine 2 soude les manches aux fonds de casseroles. 3. Enfin, la machine 3 emballe le produit terminé grâce à la tâche T5.

Sur les machines hébergeant T1 et T3, d'autres tâches existent, mais elles ne participent pas à la fabrication des casseroles. Enfin les trois machines sont connectées par deux liens séries. Les liens séries ne sont pas partagés, ainsi: 1. Le message Mb émis par la tâche T1 sur le lien série entre la machine 1 et la machine 2 prévient la tâche T3 qu'elle peut commencer son exécution. 2. Le message Ma émis par la tâche T3 sur le lien série entre la machine 2 et la machine 3 prévient la tâche T5 qu'elle peut commencer son exécution.

L'ingénieur souhaite connaître le pire temps de fabrication d'une casserole: c'est à dire le délai entre le réveil de la tâche T1 et la terminaison de la tâche T5. On vous demande de l'aider. Toutes les tâches du système sont à échéances implicites.

Question 1. (1 point) Il est possible d'utiliser l'approche holistique pour déterminer le pire temps de fabrication des casseroles. Expliquez à l'ingénieur comment fonctionne la méthode holistique. Si vous ne savez plus en quoi consiste la méthode holistique, décrivez comment intuitivement vous procéderiez.

Pour calculer le temps de réponse d'une tâche sur mono-processeur avec priorité fixe, on calcule itérativement le temps de calcul de la tâche plus tous les temps de calcul des travaux des tâches plus prioritaires pendant ce temps de réponse (comme vu en M1 et rappeler au début du cours de M2). Ce calcul peut prendre en compte un retard ou gigue J. Dans le cas de tâches dépendantes, on utile comme gigue le maximum des temps de réponse des tâches dont on dépend. On procède de la manière itérative comme pour un système non réparti. Formellement, si $hp(i)$ désigne les tâches plus prioritaires que la tâche i:

$$R_i^{n+1} = J_i + C_i + \Sigma_{j \in hp(i)} \lceil \frac{J_j + R_i^n}{T_j} \rceil \times C_j$$

Question 2. (1.5 points) Vous avez effectué quelques mesures sur la chaîne de production et vous en avez déduit que:

1. Les messages Ma et Mb ont un temps de communication égal à 1 seconde. Ces temps de communication incluent les temps de transmission sur le lien série, les temps de propagation ainsi que les temps de traversée des couches logicielles et matérielles.
2. Les tâches sont caractérisées par le tableau ci-dessous dans lequel les mesures sont **en seconde**. Notez que la priorité 100 est le niveau de priorité le plus fort et 90 un niveau de priorité plus faible.

Tâche	Priorité	Budget	Période
T1	90	15	160
T2	100	10	160
T3	100	5	160
T4	90	20	160
T5	90	100	160

Appliquez l'approche holistique au jeu de tâches ci-dessus afin de calculer le pire temps de production d'une casserole, c'est à dire le pire temps de réponse de T5.

Sans utiliser la formule, on peut faire le raisonnement suivant. Comme la période est de 160, compte tenu des budgets, nous n'aurons qu'un travail pour chacune des tâches. R(T2)=10 d'où R(T1)=10+15. R(T4)=20 d'où R(T3)=10+15+1+5 (T4 sera terminée lorsque T3 s'active). R(15)=10+15+1+5+1+100=132.

Question 3. (1.5 points) L'ingénieur sous la pression de sa direction trouve que les tâches T2 et T4 sont sous-utilisées. Il décide donc d'augmenter le rythme ou la fréquence de production de T2 et T4 et d'augmenter la priorité de T4 comme le résume le tableau ci-dessous. De nouveau calculer le pire temps de production d'une casserole, c'est à dire le pire temps de réponse de T5 et conclure.

Tâche	Priorité	Capacité	Période
T1	90	15	160
T2	100	10	20
T3	90	5	160
T4	100	20	40
T5	90	100	160

La période de $T2$ valant 20, $T2$ s'exécutent 2 fois pendant le temps de réponse de R(T1)=10+15+10=35. A la différence de précédemment, T3 est interrompue par T4 qui s'exécute 2 fois et donc R(T3)=35+1+5+20=61. On en déduit R(T5)=61+1+100=162. Donc T5 ne respecte pas son échéance. Le système ne peut pas être chargé comme prévu.

4. Exemple comparant Global-EDF et Global-LLF (2 points)

On considère des tâches synchrones (prêtes à t=0) avec échéances implicites (D = T).

Tâche	Budget	Période
T_1	2ε	T
...	...	...
T_m	2ε	T
T_{m+1}	T	$T+\varepsilon$

Question 1. (0.5 point)
Calculer le taux d'utilisation. Vers quelle valeur tend-il lorsque ε tend vers 0 ? La condition nécessaire d'ordonnancement du système est-elle vérifiée ?

> $U(\varepsilon) = \frac{2m\varepsilon}{T} + \frac{T}{T+\varepsilon}$. On en déduit $\lim_{\varepsilon \to 0} U(\varepsilon) = 1$ et la condition nécessaire $U = 1 \leq m$ est assurée.

Question 2. (0.5 point) Appliquer Global-EDF en expliquant cette politique d'ordonnancement. Que peut-on conclure ?

> Les m premières tâches démarrent car elles ont une échéance plus proche et deviennent libres simultanément après 2ε. T_{m+1} rate son échéance car elle se finira à $T + 2\varepsilon$

Question 3. (0.5 point) Appliquer Global-LLF en expliquant cette politique d'ordonnancement. Que peut-on conclure ?

> T_{m+1} a la plus petite laxité ie ε. Elle s'exécute en priorité sur 1 coeur et $m-1$ parmi les tâches identiques vont s'exécuter sur les coeurs restants. Les m tâches identiques vont s'exécuter alternativement sur $m-1$ processeurs avec succès. Le système est ordonnançable avec Global-LLF.

Question 4. (0.5 point) Quel ordonnancement en théorie vaut-il mieux utiliser ? Mais quel problème se pose en pratique ?

> Sur cet exemple et dans la théorie plus généralement, Global-LLF domine Global-EDF, mais LLF produit un grand nombre de préemptions de sorte qu'en pratique, il n'est pas utilisable.

Part II.
Solvers et Logique

On vous demande de rédiger cette partie sur une copie séparée.

5. Logique Booléenne (7 points)

On donne les formules booléennes suivantes, de trois variables $a, b, c \in \mathbb{B}$

$$f_1 = a \vee (b \wedge c)$$

$$f_2 = c \wedge ((a \wedge \bar{b}) \vee (\bar{a} \wedge b))$$
$$f_3 = (c \vee \bar{a}) \wedge (a \vee b) \wedge (\bar{b} \vee c) \wedge \bar{c}$$

Question 1. (1,5 point) Pour chaque formule, dites si la formule est SAT (on exhibera alors une solution) ou UNSAT (on expliquera le raisonnement).

> $f_1(1,0,0) = 1$ SAT (ou dès que a est vrai)
> $f_2(1,0,1) = 1$ SAT (ou (0,1,1) marche aussi)
> f_3 UNSAT, car $\bar{c}$ (clause unitaire) donc $\bar{a}$ par propagation dans $(c \vee \bar{a})$, donc b par propagation dans $(a \vee b)$, donc la clause $(\bar{b} \vee c)$ devient c ce qui est en conflit avec $\bar{c}$.
> Barème : 30 % f1, 30 % f2, 40 % f3

Question 2. (1,5 point) Pour chacune de ces formules, dessiner le (RO)BDD correspondant, avec l'ordre $a < b < c$.

> f1 : a=1 mène à terminal 1, a=0 mène à b=1,c=1.
> f2 : a=1,b=0 et a=0,b=1 mènent au même noeud c=1.
> f3 : terminal 0.
> Barème : 20 % f1, 40 % f2, 40 % f3
> f1 n'a pas de sous arbre partagé, donc est plus facile. On donne la moitié dès que c'est un diagramme de décision correct, même s'il n'est pas réduit.

Question 3. (1 point) On souhaite vérifier si deux formules booléennes f et f' sont équivalentes. Comment faire avec un solver SAT ? Et avec un solver utilisant des ROBDD ?

> En SAT, on teste si : $f \neq f'$ (i.e. le problème $(f \wedge \bar{f'}) \vee (\bar{f} \wedge f')$. Si SAT, on a un contre-exemple, si UNSAT on a prouvé l'équivalence.
> En BDD, on compare simplement si $BDD(f) = BDD(f')$. Cette comparaison est $O(1)$ si l'on a les BDD construits.
> 50 % chacun, dont 25 % sur l'explication claire.

Question 4. (1 point) Sous quelle forme est soumise la formule booléenne à analyser à un solver SAT ? Expliquez les termes "clause unitaire", "clause vide".

> Donc c'est une CNF, i.e. un produit de sommes. On appelle chaque somme (OR logique) une *clause* composée de *littéraux* qui sont des variables en forme positive ou niée.
> La clause unitaire ne contient qu'un seul littéral, et impose donc que la variable mentionnée prenne la valeur de son littéral. Ceci mène à la propagation unitaire, une règle de déduction importante en SAT.
> Une clause vide apparaît quand il y a une contradiction, cela signifie que le problème c'est maintenant "clause1 and clause2 ... and FALSE" donc on est UNSAT (avec les affectations choisies).
> 35 % chaque point, maxé à 100.

Question 5. (1 point) On parles de *plus petit point fixe* (least fixed point, *lfp*) quand on part d'un petit ensemble et qu'on y accumule des éléments petit à petit jusqu'à convergence. Au contraire on parles de *plus grand point fixe (gfp)* quand on part d'un gros ensemble et qu'on écrème petit à petit des éléments jusqu'à convergence. Expliquez dans un model-checker CTL symbolique (basé sur des DD) quand et comment on utilise ces opérations ?

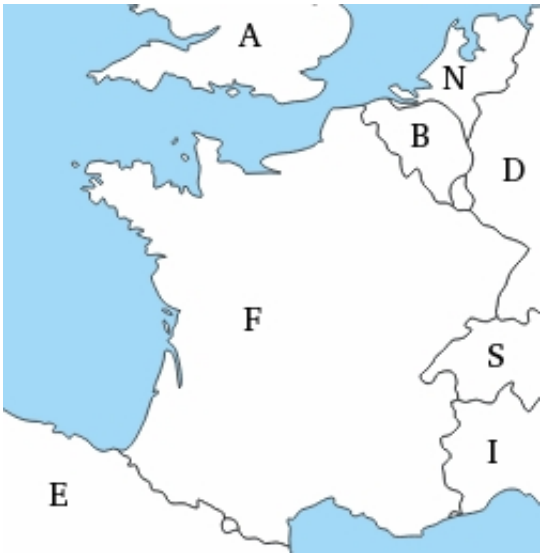
> E a U b nécessite un lfp, on part des états satisaisant "a", on ajoute les predecesseurs satisfaisant "b" jusque convergence.
>
> EG a nécéssite un gfp, on part des états satisfaisant "a", on retire les éléments qui n'ont pas de predecesseur dans l'ensemble.
>
> 50 % chaque, on attend une réponse précise (graine, critère pour ajouter/écrémer).

Question 6. (1 point) On souhaite réaliser le OU logique (ou union) de deux BDD a et b, $c = a \vee b$. L'algorithme consiste à poser les cas terminaux : $(a = 1 \vee b = 1) \rightarrow c = 1; a = 0 \rightarrow c = b; b = 0 \rightarrow c = a$. Pour la récursion si on note $a[0]$ le fils "false" de a et $a[1]$ son fils vrai, on doit calculer récursivement un noeud de fils false $c[0] = a[0] \vee b[0]$ et de fils vrai $c[1] = a[1] \vee b[1]$. Expliquez comment réaliser cette opération en complexité proportionelle au nombre de *noeuds* des BDD a et b plutôt qu'en complexité proportionelle au nombre de chemins dans ces deux structures ? En déduire une borne approximative sur la complexité de cette opération.

> Il est donc ESSENTIEL de mettre un cache. Imaginons un DD tout simple, comme le résultat de Hanoi, si on récurse gentiment sur les deux fils sans cache, on va traverser tous les chemins des deux DD, ce qui est un nombre exponentiel si le DD est très compact. Le cache à une structure simple, il associe à des couples de noeuds le résultat de l'union. Il contient donc au *pire* (en exagérant même un peu) tous les noeuds de a en produit cartésien avec tous les noeuds de b, on dit que l'opération est quadratique sur la taille des deux BDD en nombres de noeuds, $O(|a| * |b|)$ en gros.
>
> 40 % il faut un cache 30 % explication 30 % on dit que c'est quadratique

6. SMT (3 points)

On considère la carte suivante, que l'on souhaite colorier à l'aide de seulement trois couleurs (Rouge, Vert, ou Bleu), mais de manière à ce que deux pays qui partagent une frontière terrestre n'aient jamais la même couleur.



Question 1. (2 points) Expliquez comment résoudre ce problème (ou décider qu'il n'a pas de solution !) à l'aide d'un solver SMT. On sera aussi précis que possible, eg. on expliquera les variables et les assert à poser (en syntaxe logique du premier ordre $v \in \mathbb{B}, \vee, \wedge, \ldots$... ou avec celle du solver en notation préfixe sont OK) et aussi comment interpréter le résultat rendu par le solver dans les deux cas SAT et UNSAT.

Donc une variable par pays, valeur limitée au domaine R,V,B (ou un entier entre 0 et 2 inclus).

Ensuite une contrainte par frontière, qui dit que les variables de deux pays prennent une valeur distincte l'une de l'autre.

$$F \neq E, F \neq I, F \neq S, F \neq D, F \neq B, S \neq I, S \neq D, D \neq B, D \neq N, B \neq N.$$

SAT rend la solution (via get-model), UNSAT signifie qu'il n'en existe pas.

80 % sur le codage, 20 % variables et leur domaine, 30 % on ajoute une contrainte/assert par frontière, 30 % on donne les contrainte de l'exemple explicitement

Ici un début de codage en syntaxe du solver :

```
(declare-datatypes () ((Color Red Green Blue)))
(declare-const D Color)
(declare-const F Color)
(declare-const B Color)

; etc...

(assert (not (= D F)))
(assert (not (= D B)))
(assert (not (= F B)))

; etc...
(check-sat)
(get-model)
```

Et une trace SAT :

```
sat
(model
  (define-fun D () Color
    Blue)
  (define-fun B () Color
    Red)
  (define-fun F () Color
    Green)
)
```

Avec le type énuméré ça se lit plutôt facilement.

20 % sur le résultat, et comment l'interpréter (10 pour SAT, 10 pour UNSAT).

Question 2. (1 point)Si l'on ajoute le Luxembourg, situé entre **F**rance, **D**eutschland, et **B**elgique, le problème n'a plus de solution. Expliquez pourquoi.

donc c'est pour voir si on a compris les contraintes qu'on a posé en principe en question 1 avec un exemple pratique.

Ici, on a que deux à deux A, D, et B doivent être différents, mais on ajoute que L doit aussi être différent deux à deux de tous les autres, ce qui n'admet pas de solutions avec trois couleurs.

50 % on explique qu'on a un all-diff/distinct sur un ensemble qui contient 4 éléments, alors que les variables n'ont que 3 valeurs possibles.

50 % l'explication utilise/raisonne bien sur les concepts et contraintes identifiées à la question précédente. C'est le comportement du solver qu'on cherche à expliquer pas juste l'intuition, qu'on pouvait déjà avoir sans avoir suivi l'UE.

La réponse sur : ce pays L aura 3 voisins donc on ne peut pas, mais où l'on ne précise pas que ces voisins sont deux à deux distincts ne donne pas de points. Après tout la France à plein de voisins sans

> que ça pose problème au début de l'énoncé.

Question 3. (Bonus +0,5 point) Avec quatre couleurs au lieu de trois, dessinez une carte où le solver SMT déciderait qu'il n'y a pas de solution (se toucher par un coin ne compte pas comme une frontière terrestre). On recommande de n'essayer de répondre à cette question que si l'on a fini les autres exercices.

> Bon s'ils me citent le théorème des quatre couleurs ils ont 0,5 point bonus. S'ils disent qu'ils ne pensent pas que ce soit possible je donnes déjà 0,25. Sinon s'ils trouvent une carte qui marche je donnes un million d'euros :D
>
> Ce théorème prouve effectivement il n'existe pas de telle carte.
>
> La preuve du théorème elle même est difficile car elle présente une forte combinatoire de cas possibles, elle n'a été réalisée qu'à l'aide d'outils informatiques à ce jour. La preuve de 2005 est considérée comme un grand succès car elle utilise Coq, un assistant de preuve mathématique généraliste.