Durée totale : **3h**. Tous documents autorisés. Nombre de pages : **3**.
Barème indicatif : moitié test, moitié model-checking.

Exercice 1 : Test

Tous les tests demandés sont des **tests unitaires écrits en JUnit4**, et pourront utiliser Mockito ou des mocks manuels quand c'est nécessaire. **La présentation devra être soignée**.

On s'intéresse à un *simulateur* de *système chimique*. Le système chimique à simuler est composé d'un ensemble de réactions chimiques et d'une solution courante. Une *solution* est un ensemble de *molécules*, par exemple $\{A, B, E\}$. Une *réaction chimique* est une équation de transformation du style :
- $A, B \rightarrow C_1, D$ qui transforme une molécule A et une molécule B en une molécule C_1 et une molécule D
- $A \rightarrow \emptyset$ qui fait disparaître une molécule A
- $C_1 \rightarrow C_2$ qui transforme une molécule C_1 en une molécule C_2.

Une réaction chimique s'applique à une solution courante quand cette solution contient la partie gauche de la règle. Par exemple, étant donnée la solution courante $\{A, B, E\}$, on peut appliquer les réactions
- $A, B \rightarrow C_1, D$: la solution résultante est $\{E, C_1, D\}$
- $A \rightarrow \emptyset$: la solution résultante est $\{E, B\}$

mais pas la réaction $C_1 \rightarrow C_2$, qui par contre peut s'appliquer à la solution $\{E, C_1, D\}$.

L'algorithme de simulation est une suite de pas de simulation (*simulation step*)[1]. À chaque pas, le simulateur regarde dans le système chimique à simuler quelles réactions sont applicables à la solution courante, en choisit une aléatoirement, et l'applique à la solution courante dans le système chimique. Dans l'exemple donné plus haut, il choisira de manière équiprobable une réaction parmi les réactions $A, B \rightarrow C_1, D$ et $A \rightarrow \emptyset$, et l'appliquera à la solution $\{A, B, E\}$.

On souhaite tester le pas de simulation d'un simulateur qui implante l'algorithme donné ci-dessus en y ajoutant l'arrêt potentiel du simulateur :
- si aucune réaction ne s'applique à la solution courante, ce pas de simulation arrête le simulateur ;
- si au moins une réaction s'applique, le pas de simulation n'arrête pas le simulateur.

À vous d'écrire les tests les plus simples en réfléchissant notamment à la répartition des responsabilités et à l'architecture de votre application. Tout ce qui n'est pas nécessaire au test du pas de simulation n'est pas demandé (optique TDD). Les tests devront être les plus lisibles possibles.

Q 1.1 : Donner, à l'exclusion de toute autre chose :
- une classe de test pour le simulateur (pas de simulation uniquement) ;
- un diagramme UML de votre application listant les signatures des méthodes pour chaque classe ou interface Java utilisée.

□

On souhaite maintenant tester une classe représentant une solution chimique, qui peut contenir plusieurs molécules de chaque espèce. Par exemple la solution S contient 2 molécules de l'espèce A et 3 de l'espèce B, donc au total 5 molécules et 2 espèces. On représentera une molécule par une `String` non vide. Les fonctionnalités à tester sont réduites aux suivantes :
- création d'une solution vide ;
- ajout d'une molécule à une solution (ex : ajouter un A) ;
- accès au nombre de molécules d'une espèce donnée de la solution (ex : combien y-a-t-il de A ?) ;
- accès au nombre d'espèces de la solution.

Q 1.2 : Donner une classe de test pour cette classe, contenant les tests des fonctionnalités demandées. Les tests devront être les plus lisibles possible. □

[1] C'est très simplifié pour les besoins de l'examen.

Exercice 2 : Model-checking

On s'intéresse à un protocole de transmission de données, dans 3 versions différentes ($protocole_1$, $protocole_2$, $protocole_3$). Le but du protocole est, en boucle, d'initialiser une donnée courante, de tenter de la transmettre, et de tester si cette donnée a bien été transmise.

On utilisera les variables propositionnelles de l'ensemble $\mathcal{P} = \{$ `init`, `try`, `delivered` $\}$. `init` est vrai uniquement dans l'état d'initialisation de la donnée courante, `try` est vrai uniquement dans l'état dans lequel le protocole tente de transmettre la donnée courante, et `delivered` est faux si le test de transmission indique que la transmission de la donnée courante a échoué, ou si la transmission n'a pas encore été testée.

Les protocoles à étudier sont les suivants :
- $protocole_1$: ce protocole fonctionne dans un environnement idéal dans lequel toute tentative de transmission réussit. Le protocole enchaîne donc en boucle la phase d'initialisation, la tentative de transmission, et le test (positif) de fin de la transmission.
- $protocole_2$: même fonctionnement que $protocole_1$ mais la transmission peut échouer. Ce protocole fonctionne aussi en boucle, mais après la phase d'initialisation et la tentive de transmission, le test indique soit que la donnée est bien transmise, soit qu'elle ne l'est pas et alors on l'abandonne ; dans les 2 cas le protocole retourne en phase d'initialisation.
- $protocole_3$: même fonctionnement que $protocole_2$ mais, si la donnée n'a pas été transmise avec succès, le protocole essaie à nouveau de la transmettre au lieu de d'abandonner la transmission.

Dans les modèles de protocole, vous nommerez chaque état, et l'annoterez avec l'ensemble des variables propositionnelles de $\mathcal{P}$ vraies en cet état, les autres étant par convention fausses dans cet état. Ainsi, si par exemple dans un état *fail* les variables `init`, `try` et `delivered` sont fausses, l'état *fail* n'a aucune étiquette. Si dans un état *success* seule la variable `delivered` est vraie, alors *success* est étiqueté avec `delivered` uniquement.

Les questions 2.2, 2.3 et 2.5 dépendent de la question 2.1.

Q 2.1 : Donner trois automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$ qui modélisent respectivement les comportements des protocoles $protocole_1$, $protocole_2$ et $protocole_3$. □

Q 2.2 : Donner un processus Spin qui a le même comportement que $\mathcal{A}_1$. □

Q 2.3 : Donner trois ω-expressions régulières qui décrivent l'ensemble des exécutions infinies respectivement des automates $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$ (utiliser les noms d'états que vous avez choisis). □

Q 2.4 : Exprimer en LTL les propriétés suivantes :
- ϕ_1 : le protocole démarre en phase d'initialisation ;
- ϕ_2 : à chaque fois que la transmission a réussi, le protocole sera en phase d'initialisation l'instant d'après ;
- ϕ_3 : la transmission réussira un jour ;
- ϕ_4 : ϕ_3 est fausse ;
- ϕ_5 : la phase d'initialisation a lieu infiniment souvent.
- ϕ_6 : la phase d'initialisation et de tentative de transmission n'ont jamais lieu en même temps.

□

Q 2. 5 : Pour chaque formule donnée à la question 2.4 et chaque modèle de protocole, dites si l'automate satisfait la formule et justifier **brièvement** votre réponse[2]. Même si vous n'êtes pas sûr de votre automate, vous pouvez intuitivement deviner la réponse à partir de la description informelle des protocoles. □

[2]Si votre réponse est positive, donner juste assez d'explications **rigoureuses** pour me faire comprendre que vous n'avez pas répondu au hasard. Attention à ne pas perdre trop de temps en explications inutiles.

Q 2. 6 : Donner une formule CTL* ϕ_7 qui exprime : "il existe une exécution dans laquelle jamais aucune transmission ne réussit". Dire pour chaque modèle de protocole si cette formule est vérifiée, en justifiant votre réponse. □

Q 2.7 : Est-il possible d'écrire ϕ_7 en LTL ? Pourquoi ? Est-il possible de vérifier tout de même cette propriété avec Spin ? Si oui expliquer comment. □

Q 2.8 : Exprimer en LTL les propriétés suivantes :
- ϕ_8 : la transmission réussira un jour, et jusqu'à ce qu'elle réussisse le protocole sera en phase d'initialisation ou de transmission ;
- ϕ_9 : si la transmission réussit un jour, alors jusqu'à ce qu'elle réussisse le protocole sera en phase d'initialisation ou de transmission.

□

Q 2.9 : Parmi les formules ϕ_1 à ϕ_9, donner :
- une formule de sûreté ;
- une formule de vivacité ;
- une formule d'équité.

□