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 :

- $A, B \to C_1, D$ qui transforme une molécule A et une molécule B en une molécule C_1 et une molécule D
- $A \to \emptyset$ qui fait disparaître une molécule A
- $C_1 \to C_2$ 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

- $A, B \to C_1, D$: la solution résultante est $\{E, C_1, D\}$
- $A \to \emptyset$: la solution résultante est $\{E, B\}$

mais pas la réaction $C_1 \to C_2$, 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 $A, B \to C_1, D$ et $A \to \emptyset$, et l'appliquera à la solution $\{A, B, E\}$.

On souhaite tester le pas de simulation d'un simulateur qui utilise l'algorithme donné ci-dessus en procédant ainsi :

- si aucune réaction ne s'applique à la solution courante, ce pas de simulation stoppe le simulateur ;
- si au moins une réaction s'applique, le pas de simulation ne stoppe 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).

Q 1.1 : Donner, à l'exclusion de toute autre chose :

- une classe de test pour le simulateur ;
- 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 qui ne pourrait contenir qu'au plus une molécule de chaque espèce : par exemple $\{A, B\}$ est une solution, $\{\}$ est une solution, mais pas $\{A, A\}$. On représentera une molécule par une `String` non vide. Les fonctionnalités à tester sont réduites aux suivantes :

- création d'une solution vide ;
- création d'une solution à partir d'une liste de molécules ;
- ajout d'une molécule à une solution ;
- test d'inclusion d'une molécule ;
- test d'inclusion d'une solution ;
- accès à la taille de la solution.

Q 1.2 : Donner une classe de test pour cette classe, contenant les tests des fonctionnalités demandées. □

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

Exercice 2 : Model-checking

On s'intéresse au fonctionnement d'une porte de garage automatique dont le comportement nominal est de s'ouvrir à la demande, puis de se fermer au bout d'un certain temps. On considère une porte basique ($porte_1$) et deux variantes ($porte_2$ et $porte_3$).

On utilisera les variables propositionnelles de l'ensemble $\mathcal{P} = \{$ `closed`, `opening`, `opened`, `closing` $\}$, qui représentent respectivement l'état d'une porte fermée, en train de s'ouvrir, ouverte, et en train de se fermer. On modélisera uniquement la porte et ses états, et non par exemple les actions d'un utilisateur sur un bouton ou un capteur.

Les portes à étudier sont les suivantes :

$porte_1$ Cette porte de base possède un unique comportement : initialement fermée (`closed`), elle commence à s'ouvrir (`opening`) jusqu'à être complètement ouverte (`opened`), puis commence à se fermer (`closing`) avant de revenir dans l'état complètement fermée, prête à se rouvrir à nouveau.

$porte_2$ **Même fonctionnement que** $porte_1$, mais cette porte peut aussi rester dans son état courant, grâce à un bouton d'arrêt d'urgence qui permet de la bloquer.

$porte_3$ **Même fonctionnement que** $porte_1$, mais cette porte peut aussi recommencer à s'ouvrir quand elle est en train de se fermer, grâce à un capteur de passage.

Dans les modèles de porte, vous annoterez chaque état avec l'ensemble des variables propositionnelles de $\mathcal{P}$ vraies en cet état, les autres étant par convention fausses dans cet état. Comme pour les trois portes chaque variable est vraie dans un unique état, vous n'avez pas besoin de donner un nom aux états. Par exemple, l'état dans lequel `closed` est vrai peut s'appeler `closed`.

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 portes $porte_1$, $porte_2$ et $porte_3$. □

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

Q 2.3 : Donner :
- l'unique exécution de $\mathcal{A}_1$;
- une exécution de $\mathcal{A}_2$ qui n'appartient pas à $\mathcal{A}_1$;
- une exécution de $\mathcal{A}_3$ qui n'appartient pas à $\mathcal{A}_1$.

□

Q 2.4 : Exprimer en LTL les propriétés suivantes :
- ϕ_1 : la porte est initialement fermée ;
- ϕ_2 : si la porte est en train de se fermer, elle sera fermée l'instant d'après ;
- ϕ_3 : si la porte est en train de s'ouvrir, elle sera ouverte plus tard ;
- ϕ_4 : ϕ_3 est fausse ;
- ϕ_5 : la porte sera fermée un jour ;
- ϕ_6 : la porte est fermée infiniment souvent ;
- ϕ_7 : la porte ne peut pas être à la fois ouverte et fermée.

□

Q 2.5 : Pour chaque formule donnée à la question 2.4 et chaque modèle de porte, 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 portes. □

[2]Si votre réponse est positive, donner juste assez d'explications 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_8 qui exprime : "il est possible d'atteindre un point à partir duquel la porte ne sera plus jamais fermée". □

Q 2. 7 : Est-il possible d'exprimer ϕ_8 en LTL ? Pourquoi ? Est-il possible de vérifier tout de même cette propriété avec Spin ? Si oui expliquer comment. □

On suppose données les variables propositionnelles supplémentaires :
- `openingOrder` : une demande d'ouverture de porte est effectuée ;
- `lightOn` : la lumière clignote.

Q 2. 8 : Exprimer en LTL les propriétés suivantes :
- ϕ_9 : après toute demande d'ouverture, la porte finira par être ouverte, et jusqu'à ce qu'elle le soit la lumière clignotera.
- ϕ_{10} : après toute demande d'ouverture la porte finira par être ouverte avant qu'une autre demande d'ouverture lui parvienne.

□

Q 2. 9 : Parmi les formules ϕ_1 à ϕ_{10}, donner :
- une formule de sûreté ;
- une formule de vivacité ;
- une formule d'atteignabilité ;
- une formule d'équité.

□