

TP1 – Des Cartons et des Hommes

La société *Ad Hoc Logistics* (pour Automatic Distribution of Higher Order Carton) est spécialisée dans la fabrication et l'acheminement de cartons. Dans ce procédé, elle emploie un grand nombre d'employés pour déplacer les cartons de la sortie de la chaîne de fabrication aux camions qui se chargeront du transport.

Seulement voilà, ce déplacement manuel a du mal à tenir la demande croissante en cartons et cause parfois des erreurs, l'être humain étant faillible.

Aussi, *Ad Hoc* a décidé d'investir dans leur centre de R&D dans le but de réaliser un automate logistique qui se chargera de réaliser le travail qui incombait alors aux humains et dont le fonctionnement serait **partiellement prouvé** à l'aide d'Event-B.

En tant que spécialiste des cartons et de la logistique (mais aussi d'Event-B), votre rôle va donc être de concevoir et de prouver cet automate.

1 Abstraction

Le principe à la base d'un automate logistique est de conduire des objets d'un point à un autre ; mais la façon de déplacer ces objets pourrait très bien changer sans que le fonctionnement global du système ne soit remis en cause. Pour cette raison, il apparaît intéressant de se pencher, dans un premier temps, sur une *abstraction* du système ; autrement dit, une représentation très générale et très générique de ce dernier.

Question 1 : Devra-t-on modéliser chaque type d'automate que l'on voudra implémenter ? Est-il pertinent de modéliser (à ce niveau) le mode de transport des cartons ? Le contenu des cartons ? Les cartons ? Dressez alors une liste exhaustive et minimale de ce dont on a besoin pour écrire ce modèle abstrait. Donnez ensuite les opérations les plus basiques dont devrait être capable le système.

Le but du modèle abstrait est en fait de définir les "bonnes" propriétés générales que devraient avoir un automate quel qu'il soit. En particulier, on peut s'intéresser à deux propriétés d'un tel automate :

1. L'automate ne perd pas de carton
2. L'automate finit par traiter tous les cartons

Dans un premier temps, on s'intéresse à la première propriété.

Question 2 : Exprimez la propriété (1) avec la logique d'Event-B (premier ordre et théorie des ensembles). De quel genre de propriété s'agit-il ?

2 Modèle Event-B

(Note : les notations proposées le sont à titre indicatif. Libre à vous de concevoir votre modèle comme bon vous semble.)


Maintenant que nous avons mené notre réflexion préliminaire sur la conception de ce système, nous allons pouvoir passer à l'écriture de son modèle Event-B.

Ouvrez le logiciel *Rodin*¹ et créez un nouveau projet (*AutomateLogistique* par exemple).

Notez que les éditeurs de *Rodin* sont des *sémantiques* : on ajoute des champs et des parties à un fichier via un menu contextuel (click-droit sur un élément pour afficher ce menu) ; c'est seulement une fois créés que vous pourrez éditer les champs du modèle.

1. Sur les machines de l'école, Rodin se trouve à `/mnt/n7fs/ens/rodin/rodin`

2.1 Contexte

Créez un nouveau composant Event-B de type *contexte* en cliquant sur  dans l'explorateur. *Rodin* vous demande un nom (`Automate_ctx_0` par exemple).

Ce contexte va principalement contenir un *ensemble abstrait* (*carrier set*) : l'ensemble des cartons qui seront traités par la machine. On écrit alors :


Listing 1 – Première Partie du Contexte

```

1  CONTEXT
2      Automate_ctx_0
3  SETS
4      CARTONS                — Ensemble des cartons
5  END
```

Complétez le contexte dans *Rodin* à l'aide du menu contextuel : click-droit dans le contexte puis *Add Carrier Set*.

2.2 Début de la Machine

Le contexte étant maintenant terminé, on peut créer une machine qui verra ce contexte et modélisera la partie dynamique de notre automate. Toujours dans le même projet, créez un nouveau composant Event-B avec , cette fois-ci de type *machine* (que l'on pourra appeler `Automate_0`).

L'automate abstrait possède trois variables : son entrée, sa sortie, et les cartons qu'il est en train de déplacer. Ces trois variables sont des ensembles de cartons. On écrit donc :

Listing 2 – Début de la Machine

```

1  MACHINE
2      Automate_0
3  SEES
4      Automate_ctx_0
5  VARIABLES
6      Entrée                — Variables du système
7      Sortie
8  INVARIANTS
9      inv1: Entrée ⊆ CARTONS — "Type" des variables
10     inv2: Sortie ⊆ CARTONS
```

Notez que l'on pourrait écrire également $\text{Entrée} \in \mathbb{P}(\text{CARTONS})$, ce qui est tout à fait équivalent (en tout cas d'un point de vue modélisation).

Complétez la machine dans *Rodin* (toujours à l'aide du click-droit dans la machine).

Histoire de préparer le terrain pour de futures preuves, il est nécessaire de remarquer une propriété intéressante du système : un carton ne peut pas être à la fois en entrée et en sortie.

Question 3 : Écrivez l'invariant auquel cela correspond (`inv3`).

2.3 Événements

Une machine Event-B commence toujours par un événement appelé **INITIALISATION**, dans la machine par défaut. Cet événement sert à initialiser les différentes variables, autrement dit **Entrée** et **Sortie** dans notre cas.

Question 4 : Que contiennent les ensembles **Entrée** et **Sortie** au tout début du fonctionnement de la machine ? Déduisez-en les actions à écrire dans l'événement **INITIALISATION**.

Complétez l'événement **INITIALISATION** dans *Rodin* (click-droit sur l'événement puis *Add Action*).

Comme déterminé plus haut, l'automate logistique fait fondamentalement deux choses : prendre des cartons de l'entrée et poser des cartons à la sortie, ce que l'on va modéliser par deux événements dans la machine (pourquoi pas **Prendre** et **Poser**).

Il est à noter que ce n'est pas l'automate qui "choisit" quel carton il prend ou pose. Généralement, le carton arrive ou est là, et l'automate fait avec. Pour modéliser cette subtilité, Event-B met à disposition un concept qui peut justement être interprété de cette façon : les paramètres d'événements ou *event parameters* (champs **ANY**).

Le début des événements s'écrit donc comme ceci :

Listing 3 – Débuts de **Prendre** et **Poser**

```

1  Prendre
2  ANY
3      c
4  WHERE
5      grd1: c ∈ CARTONS      — "Type" de c
6      grd2: ...              — Contrainte sur c (cf q. 5)
7  THEN
8      act1: Entrée := ...     — Retirer c à l'entrée (cf q. 7)
9  END
10
11 Poser
12 ANY
13     c
14 WHERE
15     grd1: c ∈ CARTONS      — "Type" de c
16     grd2: ...              — Contraintes sur c (cf q. 6)
17 THEN
18     act1: Sortie := ...     — Ajouter c à la sortie
19 END

```

Ajoutez et complétez ces événements dans *Rodin* (à l'aide de click-droit + *Add Event*).

Question 5 : Où l'automate peut-il prendre un carton ? Déduisez-en la garde **grd2** de l'événement **Prendre**.

Question 6 : À quelle(s) condition(s) un carton peut-il être déposé sur la sortie ? Déduisez-en la garde **grd2** de l'événement **Poser**. (*On prendra soin de ne pas déposer un carton qui est déjà sur la sortie, ni un carton qui n'a pas encore été pris par l'automate.*)

Pour écrire les actions de ces événements, il ne faut pas oublier que les variables que l'on traite sont des ensembles.

Question 7 : Comment écrire le fait de retirer ou d'ajouter un carton à l'entrée et à la sortie ? Déduisez-en les actions à écrire pour **Prendre** et **Poser**. (*Attention à bien identifier ce qui est un ensemble et ce qui est un élément de cet ensemble.*)

Questions subsidiaires

Nous nous attacherons plus tard aux obligations de preuve générées par ce modèle. Néanmoins, il est des questions que l'on peut déjà se poser :

Question 8 : Les affectations faites dans l'événement **INITIALISATION** respectent-elles bien tous les invariants ?

Question 9 : Qu'est-ce qui garantit que des opérations telles que \setminus (différence ensembliste) et \cup (union ensembliste) ne changent pas la nature des ensembles utilisés dans le modèle ? (Autrement dit que les ensembles restent des ensembles de cartons.)

Question 10 : Donnez un argument (informel) au fait que des cartons ne se dupliquent pas. Est-il possible que des cartons disparaissent ?

3 Animation de Modèle

Avant de passer à la suite, parlons un peu d'animation de modèle et de *model checking*.

Faites un click droit sur la machine et cliquez ensuite sur *Start Animation / Model Checking*. Rodin va ouvrir la perspective *ProB*.

ProB est un model-checker basé sur Prolog et qui permet d'une part d'animer des modèles Event-B en simulant les événements de la machine, et d'autre part de vérifier certaines propriétés à base de model-checking (typiquement, des propriétés LTL sur la machine).

3.1 Simulation Manuelle

En haut à gauche de la fenêtre se trouve la liste des événements, incluant un événement spécial *SETUP_CONTEXT*, qui permet d'initialiser ce qui doit l'être (donc ici : **CARTONS**).

Double-cliquez sur cet événement : ProB va initialiser **CARTONS** avec des objets. Double-cliquez sur *INITIALISATION* pour réaliser l'initialisation de la machine. Vous aurez alors accès à un ensemble **Entrée** contenant un carton. Vous pouvez alors cliquer sur **Prendre** puis sur **Poser...** et c'est à peu près tout !

Cliquez sur l'icône 🔄 pour recommencer la simulation. Cette fois-ci, faite un click droit sur *SETUP_CONTEXT* et sélectionnez *Non-deterministic choice #2*. Cela va initialiser **CARTONS** avec deux objets.

Double-cliquez sur les différents événements pour les réaliser. Lorsque plusieurs façons de réaliser un événement sont possibles, vous pouvez faire un click-droit et sélectionner un choix (malheureusement les choix ne sont pas très explicites).

3.2 Violation d'Invariant

La fonctionnalité d'animation de modèle de ProB est utile lorsqu'il s'agit de se convaincre sommairement que le système n'a pas de comportement aberrant ; mais la vraie force de ce outil réside dans ses capacités de vérification et de recherche de contre-exemples.

Par exemple, ProB permet de chercher les violations d'invariants par la machine : cliquez sur la flèche à côté de *Checks* et choisissez *Model Checking*. Dans la boîte de dialogue, ne cochez que la case *Find Invariant Violations* puis cliquez sur *Start Model Checking*. Si vous avez bien écrit la machine, ProB devrait terminer sans message d'erreur.

Pour tester cette fonctionnalité plus avant, vous pouvez volontairement écrire un invariant faux et voir, avec la même technique, si ProB le détecte.

3.3 Propriétés LTL

En plus de ce que nous avons vu, ProB est capable de vérifier des propriétés LTL (Logique Temporelle Linéaire) à l'aide de *model checking*. Pour cela, vous pouvez cliquer à nouveau sur la flèche à côté de *Checks* et sélectionner *LTL Model Checking*. ProB vous demande alors de rentrer une formule.

La syntaxe de ProB pour la LTL est partiellement décrite dans la table 1. Vous pouvez également la retrouver à cette adresse https://www3.hhu.de/stups/prob/index.php/LTL_Model_Checking.

Symbole	Description
G	<i>Globally</i> , toujours, \square
F	<i>Finally</i> , éventuellement, \diamond
{ ... }	Tester un prédicat écrit en B
&	Et logique
or	Ou logique
=>	Implication logique
[Op]	Teste si le prochain événement à être exécuté est Op

TABLE 1 – Éléments de Syntaxe LTL ProB

Par exemple, on peut exprimer en LTL que l'entrée de l'automate n'est jamais vide avec

$$G \{ \text{Entrée} \neq \{\} \} \quad \text{ou encore} \quad G \{ \text{card}(\text{Entrée}) > 0 \}$$

Si vous demandez à ProB de vérifier cette formule, il vous répondra qu'il a trouvé un contre-exemple à la propriété écrite et vous propose de vous la montrer. Vous obtiendrez alors quelque chose de similaire à la figure 1.

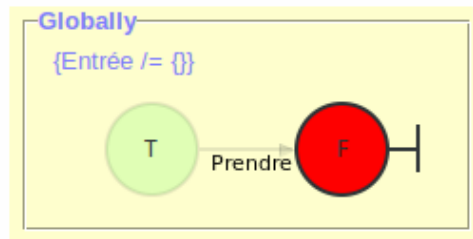


FIGURE 1 – Contre-exemple de ProB

Concrètement, cela vous indique qu'une façon de violer la formule est de partir d'un état initial avec une entrée qui contient un carton puis de faire une transition **Prendre** (ce qui vide bien la variable **Entrée**).

Question 11 : Exprimez en LTL puis vérifiez la propriété (2) du début du TP. (*Deux solutions équivalentes pour cette question.*)

Question 12 : Comment exprimer le fait qu'un carton ait été pris de l'entrée mais pas encore acheminé vers la sortie? Exprimez alors en LTL puis vérifiez la propriété suivante : « À tout moment, s'il y a des cartons pris mais pas encore acheminés, l'automate finira par poser un carton. ».

Question 13 : Exprimez en LTL puis vérifiez la propriété de non-duplication énoncée dans la question 10.

Question 14 : Exprimez en LTL puis vérifiez la propriété de conservation énoncée dans la question 10.