Nantes Université

Master ALMA (M1) - XMS1IE311
Preuves et constructions formelles
C. Attiogbé - 2023/2024

TD/TP - Modélisation et preuve en B

1 Exercice : les bases

Notation symbolique	notation B
$\exists xx.(xx \in T \ \wedge \ F(xx))$	`#xx.( xx:T & F(xx))`
$\forall xx.((xx \in T \ \wedge \ P(xx)) \Rightarrow Q(xx))$	`!xx.((xx:T & P(xx)) => (Q(xx)))`

1.1 Prédicats, quantificateurs existanciel et universel, définitions, assertions

Vous allez vous inspirer de la machine B suivante, pour structurer sous forme d'une machine similaire vos réponses.

```
MACHINE
  BasePropNat
DEFINITIONS
  foo(xx,yy) == ((xx <= yy) or (yy <=xx))
; ... // d'autres définitions
ASSERTIONS
  !nn.((nn:NATURAL) => #yy.(yy:NATURAL & yy > nn)) // Tout n naturel a un superieur
& !(nn,mm).( (nn:NATURAL & mm: NATURAL) => foo(nn,mm)) // 2 nat sont tjrs comparables
& ... // d'autres prédicats
END
```

En utilisant la définition suivante,

Un entier n est divisible par un entier d, s'il existe un entier k tel que $n = k * d$.

Définissez/modélisez en B les énoncés suivants :

1. un entier nn est pair s'il est divisible par 2.
2. un entier nn impair s'il n'est pas divisible par 2.
3. Tout entier nn est soit pair soit impair.
4. Un entier nn est premier s'il n'est divisible que par 1 et par lui-même.

1.2 Construire l'algorithme du carré d'un entier

Spécifiez et construisez l'algorithme de calcul du carré d'un entier. Pour cela vous allez construire une machine abstraite qui propose :

1. une opération permettant de calculer le carré d'un entier naturel (sommes successives).
2. une opération permettant de calculer le carré d'un entier relatif.

Vous construirez dans un premier temps une machine abstraite (avec une substitution non-déterministe) ; puis dans un deuxième temps vous raffinerez la machine abstraite.

1.3 Exercices d'entraînement : PGCD, PPCM

1. Modélisez en B la construction de l'algorithme/programme de calcul du PGCD.
   Le PGCD de deux entiers naturels n et m est un entier d qui est diviseur des deux entiers n et m et qui est le plus grand de tels diviseurs.
2. Modélisez en B la construction de l'algorithme/programme de calcul du PPCM.

Vous construirez dans un premier temps une machine abstraite (avec une substitution non-déterministe) ; puis dans un deuxième temps vous raffinerez votre machine abstraite.

2 Les obligations de preuve - PO/OP

Le développeur d'une machine abstraite a deux types d'OP :
— prouver que INITIALISATION établit l'invariant : $B \wedge C \Rightarrow [U]I$
— prouver que chaque OPERATION, lorsqu'elle est appelée sous sa précondition, établit l'invariant : $I \wedge P \Rightarrow [Subst]I$

Soient I l'invariant, S, T des substitutions, nous avons les définitions logiques suivantes des substitutions

Substitution	Synt. Abs.	définition%I	équiv en logique
neutre (id.)	$skip$	$[skip]I$	I
Précondition	$P \mid S$	$[P \mid S]I$	$P \wedge [S]I$
Choix borné	$S [] T$	$[S [] T]I$	$[S]I \wedge [T]I$
Guard	$P \Longrightarrow T$	$[P \Longrightarrow T]I$	$P \Rightarrow [T]I$
Non borné	$@x.S$	$[@x.S]I$	$\forall x.[S]I$
			x lié (non libre) dans I

3 Exercices : Espace d'états et opérations basiques

Les variables définissant l'espace d'état d'un modèle sont listées dans la clause `VARIABLES` et définies dans la clause `INVARIANT`. Ces deux clauses vont toujours de paire.

Les opérations en B peuvent avoir ou non des paramètres d'entrée et sortie ; les interfaces possibles pour les opérations sont :

Aucun paramètre	`nomOperation = BEGIN ... END`
Que des paramètres d'entrée	`nomOperation(p1, p2, ... ) = PRE ... THEN ...END`
Avec des paramètres de sortie	`r1, r2, ... <-- nomOperation = BEGIN ... END`
Avec Entrée et sortie	`r1, r2, ... <-- nomOperation(p1, p2, ...) = PRE ... THEN ... END`

N.B. Dans la clause PRE on commence par typer les paramètres, quad il y en a.

3.1 Calculatrice rudimentaire à 8 bits

Spécification en B d'une machine à calculer rudimentaire qui manipule des valeurs entières stockables sur 8 bits. La machine à calculer dispose de deux registres : un registre principal nommé RP et un registre secondaire nommé RS. La machine propose les opérations suivantes.

— storeRP(val) : stockage d'une valeur val donnée dans le registre RP.
— storeRS(val) : stockage d'une valeur val donnée dans le registre RS.
— incRP1 : incrémentation de son registre principal de 1.
— decRP1 : décrémentation de son registre principal de 1.
— cmp : comparaison de deux valeurs, l'une dans RP l'autre dans RS ; l'opération cmp renvoie un résultat qui est 1 ou 0 (valeurs égales, ou valeurs différentes).
— getRP : récupération de la valeur stockée dans RP (prévoir une variable de sortie).

Après analyse de l'énoncé,

1. écrivez la spécification en B de la calculatrice ainsi décrite.
2. écrivez les obligations de preuve relatives à votre machine ; tentez de les prouver.

N.B. La substitution `vv := bool(predicate)` donne la valeur de vérité du prédicat `predicate` à la variable vv. Par exemple vv := bool(xx < 9) donnera la valeur `true` ou `false` à vv.

3.2 Formulation et preuve des propriétés invariantes

On considère le système logiciel d'assistance d'un véhicule. On peut faire rouler ou arrêter le véhicule ; le véhicule est donc soit en train de rouler soit à l'arrêt. Le véhicule a des portes qu'on peut fermer ou ouvrir ; les portes sont donc soit fermées soit ouvertes.

On exprime les exigences (propriétés) suivantes, que le système d'assistance doit assurer :

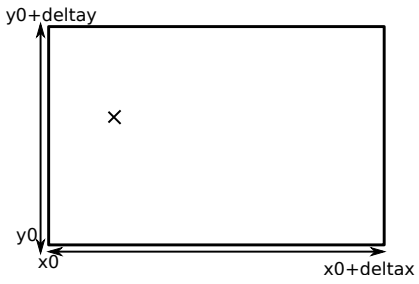
Req1	*Le véhicule ne doit rouler que si les portes sont fermées*
Req2	*On ne peut ouvrir les portes que si le véhicule est à l'arrêt*

1. Faites les abstractions nécessaires pour modéliser le véhicule.
2. Formalisez les exigences énoncées (Req1, Req2).
3. Ecrivez une machine abstraite qui structure le véhicule ainsi modélisé.

3.3 Bon usage des constantes - un robot dans un plan fixe

Soit à contrôler l'évolution des valeurs de deux variables xx, yy. Les valeurs sont entières. Elles peuvent représenter les coordonnées (x,y) d'une pointe traçante ou d'un robot.

Chacune des deux variables évolue sur une plage δ à partir de sa valeur initiale ; par exemple les valeurs possibles de xx sont entre x_0 et $x_0+\delta_x$; les valeurs possibles de yy sont entre y_0 et $y_0 + \delta_y$. Il est judicieux que x_0, y_0 $delta_x$ $delta_y$ soient des valeurs fixes (constantes), une fois définies.



On veut écrire en B une machine abstraite GestPosition qui proposera des opérations à un programme de contrôle des mouvements de la pointe traçante/ou du robot.

1. Explicitez d'une part les contraintes (ou propriétés) qui relient x, x_0, $dela_x$, et d'autre part celles qui relient y, y_0, $dela_y$.

2. Construisez en conséquence une machine abstraite B (sans les opérations) pour décrire l'espace d'états correspondant à l'énoncé. Utilisez les clauses `CONSTANTS` et `PROPERTIES` qui vont toujours de paire.

Vous compléterez en TP, la machine abstraite par la spécification des opérations. Et vous ferez le raffinement (+ génération du code en C).

. `setX(nx)` : changement de la valeur de *xx* par la valeur *nx* donnée en paramètre ;
. `setY(ny)` : changement de la valeur de *yy* par la valeur *ny* donnée en paramètre ;
. `moveRightX(dx)` : déplacement de *xx* de `dx` vers la droite (dans le plan) ;
. `moveLeftX(dx)` : déplacement de *xx* de `dx` vers la gauche ;
. `moveUpY(dy)` et `moveDownY(dy)` ;
. `cx <-- getX` : récupération de la valeur de *xx* ;
. `cy <-- getY` : récupération de la valeur de *yy* ;
. `res <-- checkXY(ax, ay)` : test si les valeurs données *ax* et *ay* sont entre les bornes de *xx* et *yy* respectivement.

4 Exercices : Théorie des ensembles - modélisation

```
MACHINE
  CtrlProcessus
SETS
    ...
CONSTANTS
    ...
PROPERTIES
    ...
VARIABLES
    ...
INVARIANT
    ...
INITIALISATION
    ...
END
```

`#  et  !`	il existe et quel que soit
`x : E`	x appartient à E
`E <: F`	E inclus dans F
`x /: E  E /<: F`	négations de : et de < :
`E \/ F`	union
`E /\ F`	intersection
`a |-> b`	couple (a,b)
`r: E <-> F`	relation
`f: E +-> F`	fonction partielle
`f: E --> F`	fonction totale
`f: E >--> F`	injection
`f: >->> F`	surjection totale
`E <| r`	restriction de domaine
`E <<| r`	antirestriction de domaine

petit mémo du langage de B

4.1 Dépendances entre variables d'un modèle **ensembles, fonctions**

1. Soit une machine abstraite M_a caractérisée par l'espace d'états défini par les variables *ff* et *xs* telles que, avec $\mathbb{N}$ l'ensemble des entiers naturels, l'invariant est :

	$ff \in \mathbb{N} \nrightarrow \mathbb{N}$	// ff une fonction partielle
$\wedge$	$xs \subseteq \mathbb{N}$	// xs un sous-ensemble de NAT
$\wedge$	$xs = \text{dom}(ff)$	// xs a les mêmes éléments que le domaine de ff
$\wedge$	$card(xs) < 224$	

   (a) Proposez une initialisation de la machine M_a.

   (b) Ecrivez l'obligation de preuve (PO) de cohérence de la machine M_a, telle qu'elle est définie jusque là.

2. Soit une opération *addElement*(xx) de M_a qui ajoute soit $xx \mapsto 1$ soit $xx \mapsto 0$, à ff .

   (a) Donnez la précondition de l'opération *addElement*(xx).

   (b) Proposez une modélisation (non-déterministe) de l'opération *addElement*(xx).

   (c) Montrez, en vous basant sur la PO adéquate, que *addElement*(xx) préserve l'invariant.

3. On veut maintenant écrire une opération de M_a qui supprime un élément mm de xs. Proposez une spécification en B de cette opération *removeElement*(mm).

4.2 Gestion de processus **relations/fonctions**

Modélisation de l'évolution de processus. Considérons le contexte d'un système d'exploitation. On a des processus qui sont créés, qui vivent et qui se terminent.

Soit PROCESSUS l'ensemble de base de tous les processus. On considérera un sous-ensemble *processus* correspondant au processus effectivement manipulés dans le système.

Un processus est caractérisé par un numéro unique et un numéro caractérise un seul processus ; tous les processus manipulés ont un état : prêt, actif, bloqué.

A tout moment tout processus est dans un seul état ; plusieurs processus peuvent être dans le même état.

Tout nouveau processus créé se trouvera dans l'état prêt et a un numéro unique (différent de ceux des processus existants déjà). On ne pourra pas créer plus de maxP processus.

On suppose qu'il y a un ordonnanceur qui fait le changement d'état des processus ; par exemple l'ordonnanceur peut, lorsque c'est possible (on ne détaille pas ici comment), faire passer un quelconque des processus de l'état prêt vers l'état actif ; un quelconque des processus de l'état bloqué vers l'état prêt.
On veut s'assurer aussi que

Propriété	*il n'y a jamais plus d'un processus à l'état* `actif`.

Soit PROCESSUS l'ensemble (type) de processus, *processus* le sous-ensemble de processus que nous manipulons dans le système et *etatP* les états des processus dans le système. Pour commencer le travail, on nous propose un modèle (incomplet) ainsi que le diagramme de Euler-Venn (Fig. 1) exprimant la relation/fonction entre les processus et leurs états.

$$processus \subseteq PROCESSUS$$

$$etatP = \{actif, pret, bloque\}$$

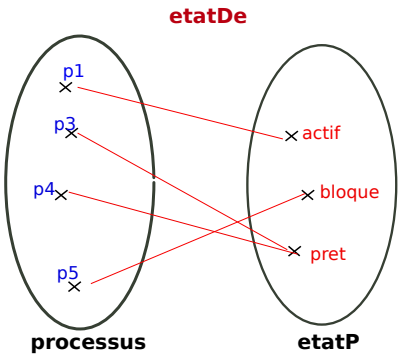


Figure 1 – Etat des processus

Ainsi, à tout moment l'état des processus de notre système peut être représenté par la fonction totale $etatDe : processus \rightarrow etatP$

1. Ecrivez en extension la fonction totale telle que elle est donnée dans la figure Fig.1
2. Que donne l'extression suivante : $etatDe^{-1}[\{actif\}] = ???$
3. Soit *prets* une variable, écrivez la substitution qui récupère dans *prets* l'ensemble des processus qui sont dans l'état `pret`.
4. Que fait l'opération suivante ?

```
ANY pp
WHERE
pp ∈ PROCESSUS ∧ pp ∈ prets
∧   prets /= {}
∧   etatDe⁻¹[{actif}] = {}
THEN
etatDe(pp) := actif
END
```

Ecrivez maintenant les opérations suivantes pour compléter le modèle :

1. création d'un nouveau processus, avec un numéro qui n'est pas utilisé.
2. trouver un quelconque des processus qui est dans l'état bloqué.
3. passage d'un processus pp de l'état bloqué à l'état prêt,
4. recherche les processus qui sont dans l'état bloqué,
5. passage d'un processus pp de l'état prêt à l'état actif,
6. arrêt d'un processus, il n'existera alors plus dans la liste des processus gérés.

Complétez maintenant le modèle en permettant d'attribuer une valeur de priorité à chaque processus. La priorité est un entier naturel. On pourra ainsi comparer la priorité des processus.

7. Raffiner le modèle précédent en intégrant la priorité. On réécrira par exemple l'opération qui passe un processus de l'état prêt à l'état actif, en prenant le plus prioritaire des prêts.

5 Etudes de cas

5.1 Bicloo, Parc de Bicloo, Parcs de Bicloo **ensembles, relations**

On a un ensemble de Bicloo. Ce ne sont pas ici des vélos mais des objets abstraits. Pour ranger nos Bicloo, on a **un parc** qui est consititué d'un **ensemble de places**, une place ne peut contenir qu'un seul Bicloo et un Bicloo ne prend qu'une place. Cependant, les Bicloo sont hyperactifs et ne restent pas sur place.

Afin de les gérer, on a recours à d'**autres parcs**, de façon à ce que les Bicloo puissent se reposer de leurs intenses activités. Un Bicloo peut se trouver donc dans n'importe quel parc, où il y a de la place.

Pour que cela soit amusant, on définit quelques propriétés

Propriété 1	*on ne peut avoir dans l'ensemble des parcs plus de Bicloo qu'on en possède initialement.*
Propriété 2	*on ne peut mettre plus de Bicloo que de places dans un parc.*
Propriété 3	*Un Bicloo peut à un moment donné ne pas être dans un des parcs.*

1. Proposez une modélisation en B, qui respecte les exigences énoncées.
2. Proposez quelques opérations pour éprouver votre modèle de gestion du parc :
   (a) sortir un Bicloo d'un parc ;
   (b) ranger un Bicloo (issu d'un parc) dans un parc.
   ...

On peut faire l'hypothèse qu'on acquiert aussi de nouveaux Bicloo.

5.2 Diffusion d'images captées par des drones

On veut modéliser un logiciel qui collecte, stocke et diffuse des images captées par des drones. On a un ensemble de drones qui sont tous de type *civil* ou *militaire*. Chaque drone a un identifiant unique et d'autres caractéristiques ignorées ici. Les drones prenent des images.

Le logiciel va gérer un ensemble d'images (ou base d'images) ; on ne considère que les identifiants des images ; pour chaque image, on a l'identifiant du drone qui l'a captée.

Le logiciel en vue va permettre de diffuser certaines images ; on dispose à tout moment des images diffusées. On veut garantir les propriétés suivantes :

P1	*Toutes les images diffusées sont issues de drones civils.*
P2	*Aucune image ne peut se trouver à la fois dans les images prises par les drones militaires et dans les images prises par les drones civils.*
P3	*il n'y a aucune image qui ne soit prise par un des drones connectés au logiciel.*

1. Formaliser en B, à l'aide de machine/s abstraite/s, l'espace d'état du modèle du logiciel. Veillez à indiquer comment les propriétés sont prises en compte.
2. Ecrivez une opération qui permet d'ajouter dans le logiciel un drone, dont on donne l'identifiant et le type.
3. Ecrivez une opération qui, étant donné une image, indique le type (*civil* ou *militaire*) du drone qui l'a prise.
4. Ecrivez une opération qui, trouve toutes les images captées par un drone donné ;
5. Ecrivez en précisant les paramètres, une opération qui permet d'ajouter une nouvelle image ;
6. Ecrivez une opération, qui permet de trouver toutes les images issues de drones civils.
7. Ecrivez une opération qui supprime de la base d'images toutes les images captées par un drone donné.

5.3 Supervision de circulation de trains

On veut modéliser une partie d'un système de régulation de trains sur des voies ferrées (rails). Dans le système considéré, on a des voies ferrées (des rails) sur lesquelles circulent un ensemble de trains. Un système de supervision et de contrôle doit disposer et enregistrer le maximum de données pour assurer la cohérence et la sécurité de la supervision et du contrôle.

Par exemple, les modules d'aiguillage et de signalisation récupèrent des données du système pour décider des autorisations d'accès aux voies demandées par les trains ou leurs contrôleurs.

Pour les besoins pratiques de supervision, les voies sont découpées en sections (appelées cantons). Une voie ferrée est alors une suite de cantons, dans une direction. Nous utilserons aussi les cantons pour la modélisation.

Comportement d'un train : un train roule en passant de cantons en cantons, depuis son point de départ jusqu'à son point d'arrivée. L'accès à un canton est soumis à une demande d'accès ; l'accès est autorisé ou non. De même, la sortie du canton est notifiée par le train. Un train peut s'arrêter dans un canton.

Exigences du système. Un train ne peut se trouver que dans un seul canton à la fois. Pour éviter les collisions, deux trains ne peuvent pas être en même temps dans un canton.

Quelques caractéristiques du système : Dans le système certains trains sont roulants (peuvent circuler) et d'autres non-roulants (ne peuvent pas circuler pour diverses raisons). On enregistre à tout moment la position des trains (roulants) par rapport aux cantons dans lesquels ils se trouvent.

Modélisation de l'espace d'états. En considérant les ensembles de base suivants : TRAIN, VOIE, CANTON, IDCANTON (séquence d'entiers), ...

1. modélisez l'ensemble des voies dans le système, l'ensemble des trains dans le système, en distinguant les roulants et les non roulants.
2. soit une voie avec des extrémités imaginées, et constituée de plusieurs cantons consécutifs ; proposez une modélisation d'une voie, respectant les données et les contraintes du système.
3. des voies peuvent partager des cantons ; Proposez la modélisation de l'ensemble des voies, si elle n'est pas couverte actuellement par vos éléments de modèle.

Modéliser des propriétés attendues du système.

req1	un train ne peut occuper simultanément plus d'un canton
req2	plusieurs trains ne peuvent occuper un même canton
req3	...

Structuration en machine abstraite. Structurer sous forme d'une ou plusieurs machines abstraites, les éléments de modélisation proposés jusqu'ici.

Modélisation des opérations de supervision.

1. Modélisez l'opération d'accès à un canton c_i par un train et son effet sur le système ; on rappelle qu'un train ne peut accéder qu'à un canton qu'il a demandé et qui est non occupé par un autre train.
2. Modélisez l'opération de sortie d'un canton c_i
3. Modéliser une opération qui permet à un train de demander l'accès à un canton c_i donné. L'opération retourne un booléen.
4. $\cdots$