Nantes Université

Master ALMA (M1) - XMS1IE313
Vérification et Preuves formelles
C. Attiogbé - 2025/2026

TD/TP - Introduction à la modélisation formelle (en B)

Exercice 1 : Construire l'algorithme du carré d'un entier **basique - algo**

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.

Exercice 2 : Formulation et preuve des propriétés invariantes **(propriétés Logique)**

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 vérifier :

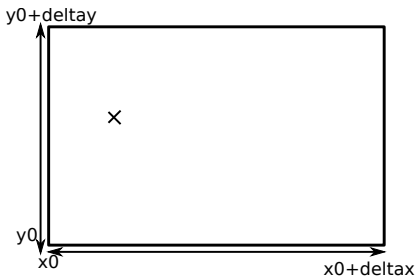
Req1	*Le véhicule ne doit rouler que si les portes sont fermées*
Req2	*On ne peut ouvrir les portes que si le véhicule est à l'arrêt*

1. Faites les abstractions nécessaires pour modéliser ce fonctionnement du véhicule.
2. Faites des hypothèses de travail et formalisez les exigences énoncées (Req1, Req2).
3. Ecrivez une machine abstraite qui structure le fonctionnement ainsi modélisé.

Exercice 3 : Bon usage des constantes - un robot dans un plan fixe **invariants**

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 de 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 des valeurs données ax et ay ; sont-elles entre les bornes de xx et yy respectivement.

Exercice 4 : Gestion de processus

relations/fonctions

Modélisation de l'évolution de processus (système d'exploitation). 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 trouve 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* 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 sagittal (Fig. 1) exprimant la relation/fonction entre les processus et leurs états.

$processus \subseteq PROCESSUS$

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

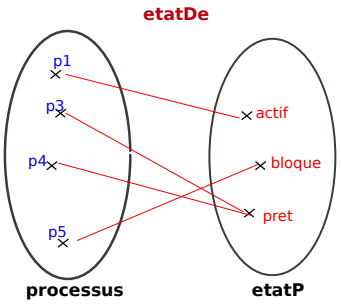


FIGURE 1 – Etat des processus

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'expression 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
$pp \in PROCESSUS \wedge pp \in prets$
$\wedge \quad prets /= \{\}$
$\wedge \quad etatDe^{-1}[\{actif\}] = \{\}$
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. recherche des processus qui sont dans l'état bloqué,
4. passage d'un processus pp de l'état bloqué à l'état prêt,
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.

squelette d'une machine B

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

petit mémo du langage de B

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

A - Surveillance automatisée d'une zone industrielle

On veut modéliser, concevoir et développer un logiciel pour la surveillance d'une zone industrielle.

La zone industrielle est structurée en secteurs (comme sur la figure Fig. 2). Un contrôleur est associé à un seul secteur ; mais plusieurs contrôleurs peuvent contrôler un même secteur.

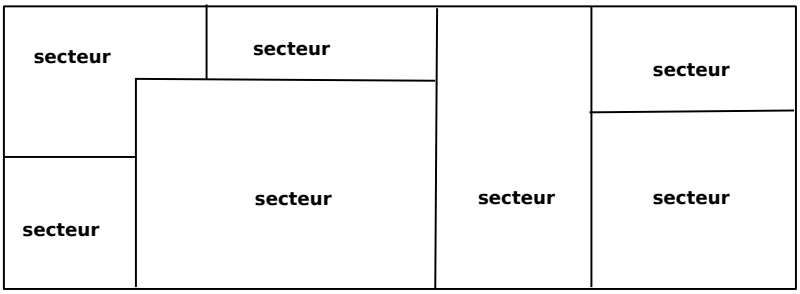


FIGURE 2 – Des secteurs dans une zone industruelle

La surveillance de la zone industrielle consiste à récuperer des informations sur les différents secteurs de la zone, les analyser, détecter d'éventuelles anomalies, puis recommander des actions ou lever des alertes. Les informations recueillies [1] sur les différents secteurs et analysées sont par exemple la température, l'hygrométrie, la luminosité, la pression, etc.

Une *propriété* attendue du logiciel est que tous les secteurs de la zone soient contrôlés (par au moins un contrôleur).

Les contrôleurs sont regroupés par catégorie ; il n'y a que trois catégories (Ca_1, Ca_2, Ca_3) ; chaque contrôleur est dans une seule catégorie.

Pour simplifier, les informations recueillies sur les secteurs sont d'une seule nature (la température) et sont des entiers naturels entre un seuil min (`SeuilMin`) et un seuil max (`SeuilMax`).

1. par des capteurs, dont on ne tiendra pas compte ici.

Ainsi la température d'un secteur est normale, si elle est située entre `SeuilMin` et `SeuilMax`. On peut avoir des contrôleurs sans secteurs associés.

Pour vous aider dans la modélisation, nous considérons les ensembles de base suivants, que vous adapterez et compléterez au besoin :

— Un ensemble de secteurs SECTEUR ;
— Un ensemble de contrôleurs CTRL ;
— Un ensemble de catégories CATEGCONTROLEUR = $\{Ca_1,\ Ca_2,\ Ca_3\}$.

1. Construisez, sous forme d'un diagramme sagittal, une relation/fonction, entre les contrôleurs CTRL et les secteurs qu'ils contrôlent. Choisissez qui contrôle quoi, et nommez convenablement votre relation/fonction.
2. Donnez en extension les éléments de votre relation (vous l'utiliserez plus tard pour l'initialisation).
3. Donnez un diagramme sagittal, qui explicite le fait qu'un contrôleur n'appartient qu'à une seule catégorie.
4. Donnez en extension ce diagramme sagittal (vous l'utiliserez plus tard pour l'initialisation).
5. Ecrivez progressivement une machine B comme abstraction d'une partie du logiciel à développer. Ecrivez dans un premier temps une première machine abstraite sans ses opérations.

Spécifiez en B, en complétant la machine commencée précédemment, les opérations suivantes :

6. Une opération qui ajoute au système, un contrôleur Co_i avec la catégorie Ca_k.
7. Une opération qui enregistre qu'un secteur Se_i donné est contrôlé par un contrôleur Co_j (déjà connu dans le système).
8. Une opération qui enregistre pour un secteur Se_i donné, la valeur vv, comme température prélevée.
9. Une opération qui indique si la température d'un secteur Se_i est normale ou pas.
10. Une opération qui indique quels contrôleurs sont dans la catégorie Ca_i.
11. Une opération qui permet de retrouver les secteurs contrôlés par des contrôleurs de la catégorie Ca_j.

Raffinement

12. Ecrivez une machine de raffinement pour :
    (a) raffiner les ensembles abstraits de votre machine.
    (b) raffiner les constantes utilisées.
13. Génération du code en langage C

B - Editeur graphique

On veut modéliser un éditeur graphique (composant logiciel). Cet éditeur est rudimentaire, il manipule des objets graphiques ayant des formes, des couleurs définies, des dimensions, $\cdots$.

L'éditeur graphique permettra de créer des objets graphiques, de leur donner une couleur, une forme, $\cdots$

L'éditeur en vue, va gérer un ensemble d'objets graphiques avec leurs formes et couleurs (on ignore les autres caractéristiques) ; on pourra créer un objet, supprimer un objet, changer la couleur, sélectionner des objets, etc

Dans le cadre de la modélisation de cet éditeur graphique, on considère **OBJGRAPH** comme l'ensemble de tous les objets graphiques imaginables. On considère que :

- on a l'ensemble (objetsCourants) des objets graphiques créés et non supprimés (on simule ainsi l'espace de travail de l'éditeur graphique) ;
- on aura un nombre limité (mais suffisamment grand) d'objets dans l'espace de travail ; ce nombre est par exemple fixé à la configuration du logiciel ;
- chaque objet graphique créé a une forme et une couleur ; on ne prendra pas en compte d'autres caractéristiques comme les dimensions, etc.
- on n'a que quatre formes : *carré*, *rectangle*, *triangle*, *cercle* ;
- on n'a que les couleurs *rouge*, *vert*, *bleu* ;
- la couleur par défaut à la création de tout objet est la couleur *bleu* ; on peut changer ensuite les couleurs avec une opération.
- on a deux types de couleur (contraste) : *vive*, *douce* ; on pourra attribuer un de ces deux types à chaque couleur, par exemple au moment de la configuration du logiciel ; mais une fois configuré cela ne va pas changer ; par exemple on pourra indiquer (rouge, vive), (vert, douce), ... et cela restera inchangé ;
- on peut sélectionner/désélectionner un des objets courants ;
- à tout moment on dispose des objets sélectionnés (objetSelectionnés), par exemple pour leur appliquer une opération ; bien sûr il peut ne pas y a voir d'objet sélectionné.
- on a une corbeille, qui contiendra les objets graphiques supprimés (pensez au couper/-coller).

On veut garantir les propriétés (invariantes) suivantes :

`P1`	*Aucun objet graphique ne peut se trouver à la fois dans l'espace de travail et dans la corbeille*.
`P2`	*Les objets sélectionnés sont parmi les objets graphiques courants*.

1. Analysez bien l'énoncé et faites des diagrammes sagittaux, pour représenter les données de l'énoncé (nommez convenablement vos relations et fonctions).
2. Modéliser en B, à l'aide de machine/s abstraite/s, l'espace d'état du modèle du logiciel (éditeur graphique). Veillez à indiquer comment les propriétés sont prises en compte (par exemple, utiliser les étiquettes des propriétés dans les commentaires de la machine).
3. Ecrivez une opération (`creerObjet(ff)`) qui permet de créer un nouvel objet graphique dont on donne la forme (ff), et de l'ajouter aux objets courants.
4. Ecrivez une opération (`formeDeObjet`) qui, ayant en paramètre un objet (parmi les objets courants), retourne sa forme.

5. Ecrivez une opération (`objetsDeForme`) qui donne tous les objets ayant la forme donnée en paramètre ;
6. Ecrivez une opération (`objetsCouleurType`), qui renvoie tous les objets dont la couleur a le type (vive, douce) donné en paramètre.
7. Ecrivez une opération (`supprObjet`) qui supprime l'objet donné en paramètre, des objets courants. L'objet supprimé se retrouve dans la corbeille.
8. Ecrivez une opération (`supprSelection`) qui supprime l'ensemble des objets actuellement sélectionnés de l'espace de travail. Les objets supprimés se retrouvent dans la corbeille.
9. Ecrivez une opération (`changeCouleurObjet`) pour changer la couleur d'un objet courant donné en paramètre, par la couleur donnée en paramètre ;
10. Ecrivez une opération (`changeCouleurSelection`) change la couleur de tous les objets actuellement sélectionnés, par la couleur passée en paramètre.

Raffinement

12. Ecrivez une machine de raffinement pour :
    (a) raffiner les constantes utilisées, raffiner les ensembles abstraits de votre machine.
    (b) raffiner les relations et fonctions, opérations.
13. Génération du code en langage C

C - Contrôle de systèmes de production

On considère la construction formelle (de composants critiques) d'un logiciel de pilotage d'un site industriel.

Le site industriel est constitué de plusieurs procédés (ou postes de fabrication par exemple). Ces procédés consomment des ressources pour produire des pièces de différentes natures. Les procédés sont pilotés par un ensemble de contrôleurs ; les contrôleurs exploitent pour cela des données venant de capteurs reliés aux procédés.

Pour les besoins de cet exercice, nous considérons les hypothèses de travail suivantes pour la modélisation d'une partie de l'architecture du logiciel de pilotage.

On a un ensemble de procédés, un ensemble de contrôleurs, un ensemble de capteurs. On a aussi les exigences suivantes. D'une part, chaque procédé est relié à un ou plusieurs capteurs ; chaque capteur est relié à un seul contrôleur, mais tous les capteurs d'un procédé doivent être reliés au même contrôleur. D'autre part, un contrôleur pilote un ensemble de procédés, mais un procédé est piloté par un seul contrôleur.

On proposera différentes opérations pour relier les entités : procédés, capteurs, contrôleurs en respectant les exigences énoncées.

On veut s'assurer dans l'architecture du logiciel (avec les exigences précédentes) que :

ControleCoherence	*tout procédé est piloté par un seul contrôleur, et tous les capteurs reliés à un procédé sont bien connectés au même contrôleur qui pilote le procédé.*

Au préalable à l'écriture du modèle formel en B, proposer des diagrammes sagittaux, Euler-Venn des données mises en jeu.

Proposer une machine abstraite qui donne le modèle formel de l'architecture ainsi décrite (sans les opérations pour le moment ; elles seront écrites dans les questions suivantes). En plus des commentaires, vous nommerez convenablement et lisiblement vos variables de façon à faciliter la lecture et la compréhension de votre modèle.

Pour vous aider dans l'analyse, on suggère quelques fonctionnalités (opérations) qui suivent. Il faudra spécifier en précisant vos hypothèses et conditions, chacune des opérations.

1. Une opération ajProcede(pr) qui ajoute un nouveau procédé à l'ensemble des procédés existants dans le logiciel.
2. Une lierControleurCapteur(Kj, Ci) qui ajoute d'un lien entre un capteur (K_j) et un contrôleur (C_i).
3. Une opération lierCapteurProcede(Kj, pr) pour l'ajout d'un lien entre un capteur (K_j) et un procédé (pr_i).
4. Une opération piloterProcede(Ci, pri) qui enregistre dans le modèle qu'un contrôleur (C_i) pilote un procédé (pr_i).
5. Une opération capteursDeCtrl(Ci) qui donne tous les capteurs reliés à un contrôleur Ci donné.
6. Une opération controleurDePcd(pri) qui donne le contrôleur d'un procédé pri donné.

On considère maintenant la fabrication de pièces. On considère un ensemble de pièces (PIECE) ; chaque pièce qui sera fabriquée sera d'un type $T1$, $T2$ ou $T3$. On considère aussi

un ensemble de ressources (RESSOURCE) ; des ressources seront utilisées pour fabriquées des pièces. On suppose ici qu'on ne fabrique que des pièces d'un des trois types $T1$, $T2$ et $T3$.

Les règles de fabrication sont données par des experts et sont prises en compte dans la fabrication par des contraintes simples, par exemple : une pièce p_i nécessite plusieurs (mais un nombre limité) ressources différentes r_j en quantité q_k associée. Par exemple, pour fabriquer p_1 on a besoin des ressources suivantes et les quantités associées $(r_1, 5), (r_3, 8), (r_6, 2)$.

Vous allez proposer progressivement un complément (raffinement) à la machine abstraite précédente, pour traiter la fabrication de pièces.

7. Proposer un complément au modèle, afin d'y définir un ensemble de ressources utilisées, et pour chacune des ressources les quantités disponibles.

On se propose de définir dans le modèle formel une relation contrFab qui contient pour chaque type de pièce à fabriquer ces contraintes de fabrication.

8. Proposer la modélisation de la relation contrFab comme un complément au modèle, afin de définir pour chaque type de pièce à fabriquer, les ressources nécessaires et les quantités associées.
9. Proposer une opération pi = fabriquerPieceT1 qui modélise la fabrique une pièce de type T1. L'opération sera abstraite (vide ; elle ne fait pas de traitement spécifique), elle renvoie juste une pièce quelconque de type T1, mais on écrira sa précondition en exploitant par exemple la relation précédente contrFab.

Raffinement

12. Ecrivez une machine de raffinement pour :
    (a) raffiner les constantes utilisées, raffiner les ensembles abstraits de votre machine.
    (b) raffiner les relations et fonctions, opérations.
13. Génération du code en langage C