Cours "Génie Logiciel II"

Chapitre 2: La méthode B AMN(Abstract Machine Notation)

Niveau: II2

Enseignante: Rim DRIRA

rim.drira@ensi-uma.tn

AU: 2016/2017

Plan

- Présentation
- Notion de machine abstraite
- Les clauses d’une machine abstraite
- Les obligations de preuve
- Définition et calcul des substitutions généralisées
- Notation de modélisation des données
- Le processus de raffinement
- Les obligations de preuve du raffinement
- La modularité

Présentation de B et de ses concepts de base

Historique

- Née dans les années 80
- **Fondateur:** Jean-Raymond Abrial (un informaticien français)
- **Livre de référence:** « The B Book » [1]
- **Le choix du nom *B*** est une sorte d'hommage aux membres du groupe BOURBAKI.
  - C’est un groupe de mathématiciens français qui a entrepris en 1939 une refonte des mathématiques en les prenant à leur point de départ logique.

Présentation

- *B* est issue de Z et VDM.
- La particularité de *B* : établir une chaîne de production qui va des spécifications au code source associé.
- B est basée sur la théorie des ensembles et la logique du premier ordre

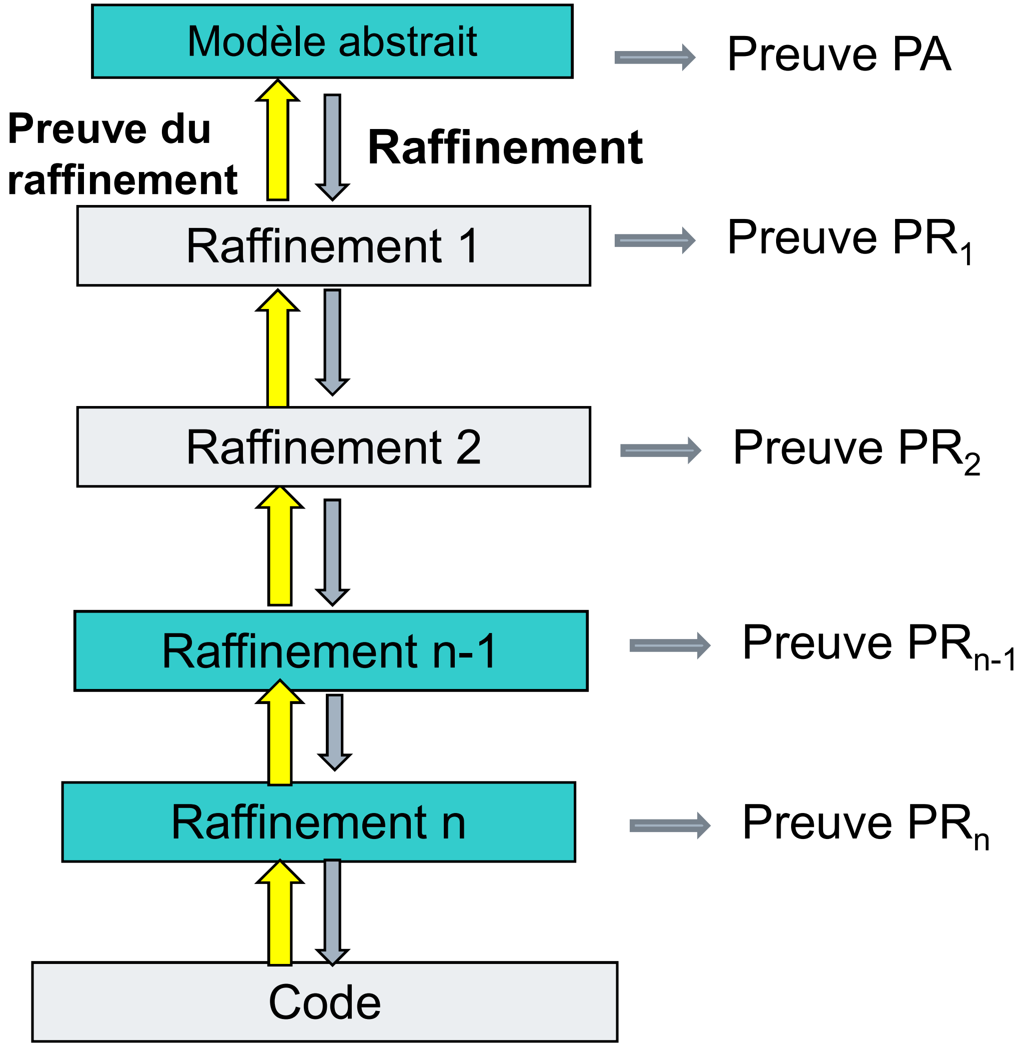
Présentation

- Les logiciels conçus avec la méthode B fonctionnent conformément à leurs spécifications
  - Les propriétés sont exprimées par des invariants et vérifiées par preuves formelles.
- Un exemple industriel de l'utilisation de la méthode B: le pilote automatique embarqué (PAE) de METEOR de la ligne 14 du métro parisien

Méthode B

PA: propriété abstraite

PR: propriété de raffinement



Documentation et outillage

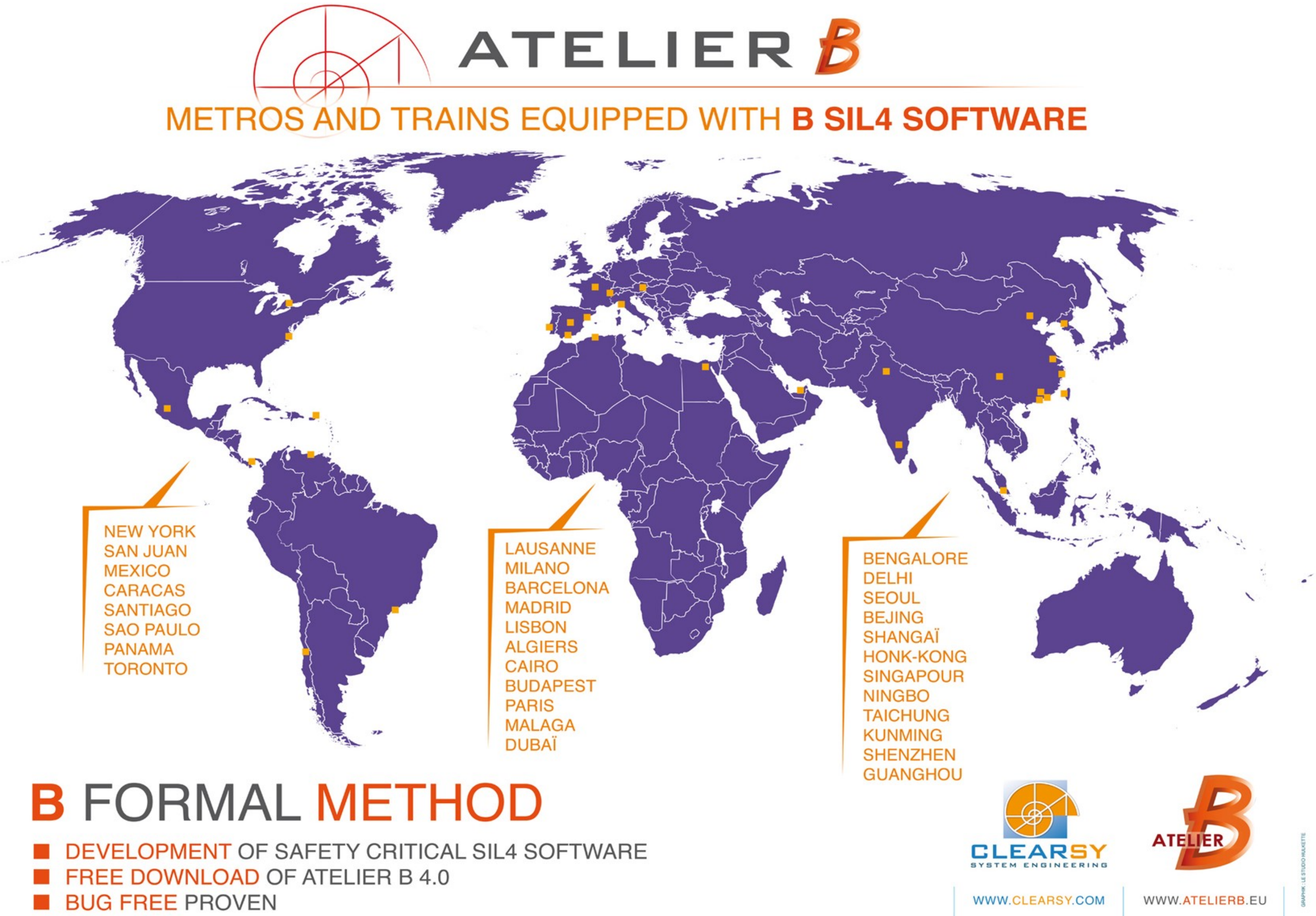
- Documentation disponible
  - http://www.dmi.usherb.ca/~frappier/igl501/ref/B/Manrefb.pdf
- Outils en licence libre
  - Exemple: Atelier B (http://www.atelierb.eu)

Atelier B

- Développé par la société ClearSy,
- Outil qui permet une utilisation opérationnelle de la méthode formelle B pour des développements de logiciels prouvés
- Disponible en deux versions:
  - une version communautaire accessible à tous sans limitation,
  - une version maintenue accessible aux possesseurs d'un contrat de maintenance.

http://www.atelierb.eu

Atelier B



Source: http://www.atelierb.eu

Atelier B

Avec l’atelier B:

1. On définit les données internes d’une machine et leurs propriétés invariantes ;
2. On définit les opérations d’accès à la machine ;
3. L’atelier vérifie la syntaxe et le typage ;
4. L’atelier génère les preuves à faire ;
5. L’atelier tente de les prouver automatiquement ;
6. Si certaines preuves n’ont pas été démontrées, on détermine si :
   - l’opération est incorrecte et on corrige
   - le prouveur n’est pas assez « fort » et l’on aide le à réaliser la démonstration

Notion de machine abstraite

- Dans B, un composant logiciel est appelé machine abstraite qui contient:
  - Des données
  - Des opérations
- Connue aussi sous l'acronyme **AMN** (Abstract Machine Notation)
- Avec B,
  - On spécifie
  - On prouve
  - On raffine
  - On code

    Une (ou plusieurs) **machine(s) abstraite(s)**
- Cette notion est proche de:
  - Type abstrait, Module, Classe, Composant

Quelques remarques

- Les lettres minuscules et majuscules sont distinguées.
- Les commentaires sont délimités par les deux caractères de début de commentaire "/*" et par les deux caractères de fin de commentaires "*/".
- Un identificateur est une séquence de lettres, de chiffres ou du caractère souligné "_". Le premier caractère doit être une lettre.
  - Tout ce qu'on lui attribue nous même des noms doit respecter les règles de définition d'un identificateur: le nom d'une machine, d'une variable, d'une constante, etc.

Notion de machine abstraite

Structure d'une machine abstraite

PARIE ENTETE

Nom de la machine

Paramètres de la machine

Contraintes sur les paramètres

PARTIE STATIQUE

Déclaration d'ensembles

Déclaration de constantes

Propriétés des constantes

Déclaration des variables (état)

Invariant (caractérisation de l'état)

PARTIE DYNAMIQUE

Initialisation des variables

Opérations

END

Principales clauses

En-tête

MACHINE
Nom de la machine abstraite (idem nom de la classe)
CONSTRAINTS
Contraintes sur les paramètres d'une machine s'il y en a

Partie Statique

SETS
Déclaration des ensembles abstraits et énumérés
CONSTANTS
Définition des constantes
PROPERTIES
Propriétés sur les constantes
VARIABLES
Déclaration des variables (idem attributs d'une classe)
DEFINITIONS
Contient une liste d'abréviations pour un prédicat, une expression ou une substitution.
INVARIANT
Typage et propriété des variables

Partie dynamique

INITIALISATION
Définition de la valeur initiale des variables (idem constructeur)
OPERATIONS
Déclaration des opérations (idem méthodes)
END

- **Une clause ne doit pas apparaître plus qu'une fois**
- **L'ordre des clauses n'est pas imposé**

Les clauses d'une machine abstraite
L'en-tête

- La partie en-tête contient:

    Nom de la machine
    Paramètres de la machine (pour la généricité)
    Contraintes sur les paramètres

- Clauses

MACHINE

Nom de la machine abstraite (paramètres formels)

CONSTRAINTS

Contraintes sur les paramètres d'une machine s'il y en a

Les clauses d'une machine abstraite **L'en-tête**

- Il est optionnel qu'une machine possède des paramètres formels
- Les paramètres d'une machine sont de deux sortes:
  - Paramètres scalaires:
    - L'identificateur doit comporter au moins un caractère minuscule
    - Type: INT, BOOL, paramètre ensemble de la machine
  - Paramètres ensembles:
    - L'identificateur ne doit pas contenir de caractères minuscules
- Exemples

MACHINE	MACHINE	MACHINE
carrefour	*réservation(Maxi)*	*pile(ELEMENT)…*
…	CONSTRAINTS	
	Maxi $\in$NAT…	

Les clauses d'une machine abstraite **L'en-tête**

La clause CONSTRAINTS

- Cette clause doit apparaître si la machine possède des paramètres scalaires formels
- Permet de typer les paramètres scalaires formels de la machine abstraite et d'exprimer des propriétés complémentaires sur ces paramètres
- Exemple:

```
MACHINE
    MA ( p1, p2, p3, ENS1, ENS2 )
CONSTRAINTS
    p1 ∈ INT   ∧
    p2 ∈ BOOL  ∧
    p3 ∈ ENS1  ∧
    card (ENS1) = 10
...
```

Les clauses d'une machine abstraite
Les données

- Les données d'une machine abstraite sont décrites au moyen des clauses suivantes:

SETS

Déclaration des ensembles abstraits et énumérés

CONSTANTS

Définition des constantes

PROPERTIES

Propriétés sur les constantes

VARIABLES

Déclaration des variables (idem attributs d'une classe)

INVARIANT

Typage et propriété des variables

Les clauses d'une machine abstraite
Les données

La clause SETS

- elle représente les ensembles introduits par la machine:
  - soit des ensembles abstraits:
    - définis par leurs noms
    - utilisés pour désigner des objets dont on ne veut pas définir la structure au niveau de la spécification.
    - Tout ensemble abstrait est implicitement fini et non vide
  - soit des ensembles énumérés définis par leurs noms et la liste de leurs éléments.

Les clauses d'une machine abstraite
Les données

- Exemple:

```
MACHINE
    MA
SETS
    POSITION ;
    MARCHE = {Arret, Avant, Arriere} ;
    DIRECTION = {Nord, Sud, Est, Ouest}
...
END
```

POSITION est un ensemble abstrait

MARCHE et DIRECTION sont des ensembles énumérés

Les clauses d'une machine abstraite
Les données

- **La clause CONSTANTS**
  - On peut aussi utiliser CONCRETE_CONSTANTS
  - Déclare une liste de constantes concrètes (implémentable directement dans un langage informatique)
  - Chaque constante doit être typé dans la clause PROPERTIES
- **La clause PROPERTIES**
  - définit un prédicat qui type les constantes de la machine abstraite et qui définit leurs propriétés.

Les clauses d'une machine abstraite
Les données

- **Exemple 1**

MACHINE
MA1
CONSTANTS
PosMin,
PosMax
PROPERTIES
$PosMin \in \text{INT} \land$
$PosMax \in \text{NAT}$

...
END

- **Exemple2**

MACHINE
MA2
CONSTANTS
Cte1,
Cte2
PROPERTIES
$Cte1 \in \text{INT} \land$
$Cte2 \in \text{NAT} \land$
$(Cte1 < 0 \Rightarrow Cte2 = 0)$
...
END

Les clauses d’une machine abstraite **Les données**

La clause VARIABLES

- Elle introduit la liste des variables de la machine.
- Ces variables représentent l´état de la machine.
- Les variables sont les seules objets qui peuvent être modifiés directement dans l'initialisation et les opérations de la machine.
- Cette clause peut être absente si la machine n'a pas de variables.
- Cette clause **doit être** accompagnée des clauses INVARIANT et INITIALISATION.

Les clauses d'une machine abstraite
Les données

La clause INVARIANT

- regroupe un ensemble de prédicats qui:
  - permettent de **typer les variables**
  - définissent les **propriétés** des variables que doit vérifier l'état de la machine à tout moment
- **Il est fondamental qu'une machine avec variable(s) soit munie d'un invariant.**

Les clauses d'une machine abstraite
Les données

- **Exemple 1**

MACHINE *MA*
VARIABLES
var1, *var2, var3*
INVARIANT
var1 ∈ NAT ∧
var2 ∈ BOOL ∧
var3 ∈ INT ∧
(*var1* > *var3* ∧ *var2* = TRUE)
INITIALISATION
var1:=2||
var2:=TRUE||
var3:=-1
... END

Invariant de typage des variables:

- Var1 est un entier naturel, var2 est un booléen et var 3 est un entier relatif.

Propriété sur les variables

- Var1 doit être supérieur strictement à var3 et var2 doit être vrai

Les clauses d'une machine abstraite
Les données

❑ Exemple2

MACHINE Chaudière

SETS

ETATS = {activé, désactivé}

VARIABLES

T, Alarme

INVARIANT

$T \in NAT \wedge Alarme \in ETATS \wedge$

$(T > 130) \Leftrightarrow (Alarme = activé)$

…

END

Les clauses d'une machine abstraite
Les données

- **Exemple3: Machine sans variables**

MACHINE Calculatrice

CONSTANTS

min_ent, max_ent, ENTIER

PROPERTIES

$min\_ent = -2^{31} \wedge$

$max\_ent = 2^{31} - 1 \wedge$

$ENTIER = min\_ent .. max\_ent$

...

END

Synthèse

```
MACHINE M(X, u)
CONSTRAINTS
    C(u)
SETS
    S; T = {a, b}
CONSTANTS
    c, d
PROPERTIES
    R(c, d)
VARIABLES
    x, y
INVARIANT
    I(x, y, c, d, u)
INITIALISATION
    U(x, y)
OPERATIONS
    r ← op(p) = PRE P THEN K END ;
    ...
END
```

Les clauses d'une machine abstraite
La partie dynamique

- Composée des clauses suivantes:

INITIALISATION

Définition de la valeur initiale des variables

OPERATIONS

Déclaration des opérations

Les clauses d’une machine abstraite
La partie dynamique

La clause INITIALISATION

- Permet d’attribuer une valeur initiale à chaque variable propre à la machine.
- L’initialisation doit satisfaire l'invariant de la machine.
- Clause obligatoire si la clause VARIABLE est présente: toute variable propre à la machine doit être initialisée.

Les clauses d'une machine abstraite
La partie dynamique

Exemple

MACHINE *MA*

VARIABLES

var1, *var2*

INVARIANT

var1 ∈ NAT ∧

var2 ∈ INT ∧

(*var1* > *var2*)

INITIALISATION

var1, var2:=2,-1;

... END

Les clauses d'une machine abstraite
Les opérations

- **La clause OPERATIONS** permet de définir les opérations d'une machine
- Une opération est composée de:
  - **Une en-tête:** un nom et des paramètres (u et w) s'il y en a
  - **Une pré-condition** (Q)
  - **Une transformation** (V)
    - Définit le comportement de l'opération vis à vis de l'état de la machine
      - Une transformation des données internes et une affectation de valeurs aux paramètres de sortie

$$\underbrace{u \leftarrow O\ (w)}_{\text{en-tête}} = \underbrace{\text{PRE } Q}_{\text{pré-condition}} \text{ THEN } \underbrace{V}_{\text{transformation}} \text{ END}$$

Les clauses d'une machine abstraite
Les opérations

Exemple

MACHINE Calculatrice

CONSTANTS

min_ent, max_ent, ENTIER

PROPERTIES

min_ent = $-2^{31} \wedge$ max_ent = $2^{31} - 1 \wedge$

ENTIER = min_ent .. max_ent

OPERATIONS

ecrire_nombre (a) … ;

a ⟵ lire_nombre … ;

c ⟵ plus (a, b) … ;

c ⟵ moins (a, b) … ;

c ⟵ mult (a, b) … ;

r, o ‹ — add (a, b) … ;

END

Les clauses d’une machine abstraite
Les opérations

- Une opération peut être paramétrée en entrée (***w***) et en sortie (***u***) par une liste de paramètres.
- Les paramètres en entrée **w** sont typés explicitement dans la pré-condition ***Q*** de l’opération.
- La pré-condition Q permet aussi de définir les conditions de fonctionnement de l’opération
- Les paramètres en sortie **u** sont typés implicitement en fonction des types des expressions définissant la valeur de ces paramètres dans ***V***

$$\underbrace{u \leftarrow O\,(w)}_{\textbf{en-tête}} = \underbrace{\textbf{PRE}\ Q}_{\textbf{pré-condition}}\ \textbf{THEN}\ \underbrace{V}_{\textbf{transformation}}\ \textbf{END}$$

Les clauses d’une machine abstraite
Les opérations

- La définition des opérations s’appuie sur le **langage des substitutions généralisées**
- La transformation de base est la substitution simple *devient égal*, notée **:=**
- Les autres transformations permettent de modéliser le résultat de tous les changements d’états réalisables :
  - Choix borné
  - Substitution gardée
  - Choix non borné
  - Etc.

Les clauses d'une machine abstraite
Les opérations

Exemple1

MACHINE **Chaudière**
SETS
    ETATS = {activé, désactivé}
VARIABLES
    T, Alarme
INVARIANT
    $T \in NAT \wedge$
    $Alarme \in ETATS \wedge$
    $(T > 130) \Leftrightarrow (Alarme = activé)$
INITIALISATION
T:=0||Alarme:=désactivé
OPERATIONS
changerT (v) =
PRE
    $\mathbf{v \in NAT \wedge v \leq 130}$
THEN
    T := v
END
…

Les clauses d'une machine abstraite
Les opérations

Exemple2

```
MACHINE Calculatrice
    CONSTANTS
    min_ent, max_ent, ENTIER
    PROPERTIES
    min_ent = -2^31 ∧
    max_ent = 2^31- 1 ∧
    ENTIER = min_ent .. max_ent
    OPERATIONS
        r <— plus (a, b) =
        PRE
            a ∈ ENTIER ∧
            b ∈ ENTIER ∧
            a + b ∈ ENTIER
        THEN
            r := a + b
        END ;
```

```
r, o <— add (a, b) =
PRE
    a ∈ ENTIER ∧
    b ∈ ENTIER
THEN
    IF a + b ∈ ENTIER
    THEN
    r := a + b ||o := FALSE
    ELSE
    o := TRUE
    END
END
END
```

Les clauses d'une machine abstraite
Les opérations

Exercice

MACHINE

reservation

…

OPERATIONS

reserver(nbPlaces)=

PRE

nbPlaces $\in$ ***NAT1*** $\wedge$

nbPlaces ≤ nbPlacesLibres

THEN

nbPlacesLibres := nbPlacesLibres-nbPlaces

END

END

Exercice: Réservation

- On souhaite spécifier un système de réservation de places sur un vol limité à *maxi* places. Les opérations à écrire sont:
  - <u>Réserver:</u> permet de réserver une place si c'est possible.
  - <u>Libérer:</u> permet d'annuler la réservation d'une place.
- **Remarque:** la gestion des numéros de places n'est pas demandée.

Travail demandé:

1. Spécifier les données de la machine B associée à ce système
2. Décrire l'invariant de ce modèle
3. Donner la machine résultante

Solution

MACHINE
Reservation1
CONSTANTS
maxi
PROPERTIES
maxi ∈ NAT
VARIABLES
NbPlLib
INVARIANT
NbPlLib ∈ 0..maxi
INITIALISATION
NbPlLib:=maxi

OPERATIONS
reserver= PRE NbPlLib ≠ 0 THEN NbPlLib:=NbPlLib-1 END;
liberer= PRE NbPlLib ≠ maxi THEN NbPlLib:=NbPlLib+1 END
END

Exercice : Réservation

1. Modifier la machine précédente pour:
   - Permettre à la machine de recevoir maxi en paramètre
   - Améliorer les opérations: elles doivent retourner en résultat «échec » ou « succès ».
2. Ecrire une nouvelle machine qui permet de réserver et libérer un nombre de places donné en paramètre.
3. Ecrire une nouvelle machine permettant de gérer les places par leurs numéros.

Solution

1/

MACHINE

Reservation2(maxi)

CONSTRAINTS

maxi ∈ NAT

SETS

RESULTAT={échec, succès}

VARIABLES

NbPlLib

INVARIANT

NbPlLib ∈ 0..maxi

INITIALISATION

NbPlLib:=maxi

OPERATIONS

Res <-- reserver= IF NbPlLib ≠ 0 THEN NbPlLib:=NbPlLib-1 || Res:=succès
    ELSE Res:= échec END;

Res <-- liberer= IF NbPlLib ≠ maxi THEN NbPlLib:=NbPlLib+1 || Res :=
    succès ELSE Res:= échec END

END

Solution

2/
MACHINE
Reservation3(maxi)
CONSTRAINTS
maxi ∈ NAT
SETS
RESULTAT={échec, succès}
VARIABLES
NbPlLib
INVARIANT
NbPlLib ∈ 0..maxi
INITIALISATION
NbPlLib:=maxi
OPERATIONS
Res <-- reserver(nb) =
PRE nb ∈ NAT THEN
IF NbPlLib >= nb THEN NbPlLib:=NbPlLib-nb || Res:=succès
ELSE Res:= échec
END
END;
Res <-- liberer(nb)=
PRE nb ∈ NAT THEN
IF NbPlLib+nb <= maxi THEN NbPlLib:=NbPlLib+nb || Res := succès ELSE Res:= échec END
END
END

Solution

3/

MACHINE Reservation4
CONSTANTS
maxi, PLACES
PROPERTIES
maxi ∈ NAT ∧ PLACES = 0..maxi
SETS
RESULTAT={échec, succès}
VARIABLES
OCCUPES
INVARIANT
OCCUPES ⊆ PLACES
INITIALISATION
OCCUPES:=∅
OPERATIONS
place <-- reserver = PRE PLACES ≠ OCCUPES THEN
ANY pp WHERE pp ∈ PLACES - OCCUPES THEN
OCCUPES:=OCCUPES ∪ {pp}||place:=pp END
END;
liberer(place)= PRE place ∈ PLACES ∧ place ∈ OCCUPES THEN
OCCUPES:=OCCUPES - {place}
END
END

Exercice

- On souhaite spécifier le comportement d'un système de contrôle d'une barrière d'un passage à niveau en utilisant la méthode B.
- La barrière est initialement relevée. Si un train arrive alors elle est baissée et elle reste dans cet état tant qu'il ya des trains qui arrivent. Elle est relevée si le dernier train quitte la section.

1) Spécifier les données du modèle B associé à ce système
2) Décrire les invariants de ce modèle
3) Donner le modèle résultant

- **Remarque :** Il est possible de commencer par construire un automate associé à ce contrôleur et, par la suite, donner le modèle résultant.

Solution

```
MACHINE barriere
SETS
etat={relevée, baissée}
VARIABLES
barrier, NbT
INVARIANT
barrier∈etat ∧ NbT ∈ NAT ∧ ((NbT=0 ⇒ barrier=relevée) or (NbT>0
    ⇒ barrier=baissée))
INITIALISATION
barrier:=relevée||NbT:=0
OPERATIONS
arriveeTrain=
IF barrier= relevée THEN NbT:=NbT+1 ||barrier:=baissée
ELSE NbT:=NbT+1
END;
passageTrain= PRE barrier=baissée THEN
IF NbT>1 THEN NbT:=NbT-1 ELSE NbT:=0||barrier:=relevée END
END END
```

Autres clauses

- Clause **ASSERTIONS**: liste de prédicats appelés assertions portant sur les variables. Ce sont des résultats intermédiaires déduits de l'invariant susceptibles d'aider la preuve.

MACHINE
    MA
CONCRETE_VARIABLES
    var
INVARIANT
    $var \in$ INT $\wedge$
    $var^2 = 1$
ASSERTIONS
    $var = 1 \vee var = -1$
...
END

- Clauses **SEES**, **USES**, **INCLUDES**, **IMPORTS**, etc. (Pour la modularité)

Autres clauses
La clause DEFINITIONS

- Permet de définir une liste d'abréviations pour un prédicat, une expression ou une substitution:
  - À travers des déclarations explicites
  - À partir de fichiers de définitions à inclure
- Les définitions sont locales à la machine où elles sont définies et peuvent être utilisées même dans le texte qui précède leur déclaration.
- Les définitions peuvent dépendre d'autres définitions mais ne doivent pas conduire à des dépendances cycliques.
- Les définitions explicites peuvent être paramétrées

Autres clauses
La clause DEFINITIONS

Exemple1

...

DEFINITIONS

Def1 == A = jaune $\wedge$ B = jaune;

Def2(aa, bb) == (aa = rouge $\wedge$ bb $\neq$ rouge) $\vee$ (aa $\neq$ rouge $\wedge$ bb=rouge);

Def3 == Def2(A, B)

CONSTANTS

...

- Le corps de la définition *def1* est " A = jaune $\wedge$ B = jaune ".
- Def2 est une définition paramétrée
- Def3 est définie par appel à Def2

Les obligations de preuve

Les obligations de preuve (Preuve de cohérence)

- Une machine abstraite est dite cohérente lorsque l'initialisation et chaque opération préservent les propriétés invariantes.
- La dernière étape de la spécification consiste à prouver la cohérence de chaque machine abstraite
- Une obligation de preuve est une formule mathématique à démontrer afin d'assurer qu'un composant B est correct.
- Un point fort de B est la génération automatique des obligations de preuve

Les obligations de preuve

- Toutes les obligations de preuve ont la structure suivante :

H $\Rightarrow$ **P** où P et H sont des prédicats.

- Cette formule signifie qu'il faut démontrer le but P sous l'hypothèse H, H étant généralement une conjonction de prédicats.

Les obligations de preuve

- Les obligations de preuve relatives à une machine abstraite concernent principalement :
  - **la correction de l'initialisation :** l'initialisation doit établir l'invariant de la machine;
  - **la correction des opérations :** les opérations doivent préserver l'invariant;
- Concerne aussi:
  - **la correction des assertions** (clause ASSERTIONS): chaque assertion doit être satisfaite sous l'hypothèse que l'invariant l'est;
  - **la correction de l'instanciation lors de l'inclusion de machines** (clause INCLUDES) : les paramètres effectifs d'instanciation doivent vérifier les contraintes des paramètres de la machine incluse;

Les obligations de preuve

- Les obligations de preuve relatives à une machine abstraite concernent principalement :
  - **la correction de l'initialisation :** l'initialisation doit établir l'invariant de la machine;
  - **la correction des opérations :** les opérations doivent préserver l'invariant;

Les obligations de preuve

Correction de l’initialisation

Les hypothèses de la correction de l'initialisation sont :

- Contraintes des paramètres de la machine abstraite
- Propriétés des constantes de la machine abstraite,

Sous ces hypothèses, le but à démontrer est :

- L'invariant de la machine abstraite après application de l’initialisation

Les obligations de preuve

Correction des opérations

Les hypothèses sont:

- Contraintes des paramètres de la machine abstraite
- Propriétés des constantes de la machine abstraite,
- Invariants et assertions de la machine abstraite,
- La pré-condition de l'opération

Sous ces hypothèses, le but à démontrer est :

- L'invariant de la machine abstraite, après application de la substitution définissant l'opération.

Les obligations de preuve

- Soient **S** une substitution et **P** un prédicat:
  - La notation **[S] P** se lit « la substitution S établit le prédicat P »
  - **[S] P** représente le prédicat obtenu après transformation de P par la substitution S.
- Chaque substitution généralisée se définit en précisant quel est le prédicat obtenu après application de la substitution à un prédicat quelconque.
- **Exemple pour la substitution simple:**
  - [x:=E]I représente I dans laquelle toutes les occurrences libres de x sont remplacées par E.

Les obligations de preuve

- Le prédicat P c'est l'invariant I et la substitution S est V, on obtient les obligations de preuve suivantes:
  - Obligation de preuve de l'initialisation: démontrer que le prédicat **H $\Rightarrow$[Init]I** est valide
  - Obligation de preuve pour une opération: démontrer que le prédicat **H $\Rightarrow$ [V]I** est valide

Les obligations de preuve

Exemple

MACHINE
    M
VARIABLES
    V
INVARIANT
    I
INITIALISATION
Init
OPERATIONS
    $u \leftarrow O(W) =$
    PRE
        Q
    THEN
        V
    END
END

Obligation de preuve pour l'initialisation:

Aucun hypothèse

But à prouver**: I après Init** (est ce que l'initialisation établit l'invariant?) ➡ **[Init]I**

Obligation de preuve pour l'opération O:

On prouve que

- lorsque la machine est dans un état correct: les propriétés invariantes **I** sont supposées vérifiées
- lorsque l'opération est appelée avec la pré-condition **Q:** Q est supposée vérifiée
- alors sa transformation **V** amène la machine dans un état correct: → un état qui satisfait l'invariant **I**

$I \wedge Q \Rightarrow$ I après V ➡ **$I \wedge Q \Rightarrow [V]\ I$**

Les obligations de preuve

Exemple1

MACHINE
    M
VARIABLES
    V
INVARIANT
    I
INITIALISATION
Init
OPERATIONS
    u ← O(W) =
    PRE
        Q
    THEN
        V
    END
END

Les obligations de preuve pour la machine M:

[Init]I

$\mathbf{I} \wedge \mathbf{Q} \Rightarrow \mathbf{[V]\ I}$

Les obligations de preuve

Exemple2

MACHINE
M(u)
CONSTRAINTS
C
CONSTANTS
c
PROPERTIES
R
VARIABLES
x
INVARIANT
I
INITIALISATION
U
OPERATIONS
r   op(p) = PRE P THEN K END;

Obligation de preuve pour l'initialisation:
$C \wedge R \Rightarrow [U]I$

Obligation de preuve pour l'opération O: $C \wedge R \wedge I \wedge P \Rightarrow [K]\ I$

Exercice

- Écrire les obligations de preuve de la machine réservation

Éléments de modélisation en B

- **Langage ensembliste:** pour décrire le modèle de données
- **Logique des prédicats du premier ordre** pour décrire les propriétés
- **Langage des substitutions généralisées** pour décrire les actions

Les substitutions généralisées

Introduction

- Les substitutions généralisées sont utilisées pour décrire le corps de l'initialisation et des opérations d'un composant (clauses INITIALISATION et OPERATIONS)
- La sémantique du langage des substitutions est définie par un calcul des substitutions qui permet d'effectuer les évaluations des obligations de preuve

Les substitutions généralisées

Non déterminisme

- Une substitution possède un comportement non déterministe si elle décrit plusieurs comportements possibles sans préciser lequel sera effectivement choisi.
- En B, les substitutions des machines et des raffinements peuvent être non déterministes.
- Le non déterminisme décroît lors du raffinement.

Les substitutions généralisées

Opérateur ou mot réservé	Nom de la production grammaticale
BEGIN	Substitution bloc
skip	Substitution identité
:=	Substitution devient égal
:( )	Substitution devient tel que
:∈	Substitution devient élément de
PRE	Substitution précondition
ASSERT	Substitution assertion
CHOICE	Substitution choix borné
IF	Substitution conditionnelle
SELECT	Substitution sélection
CASE	Substitution cas
ANY	Substitution choix non borné
LET	Substitution définition locale
VAR	Substitution variable locale
;	Substitution séquence
WHILE	Substitution tant que
←	Substitution appel d'opération
||	Substitution simultanée

Substitution devient égal

- Soit x une variable, e une expression et P un prédicat, alors :

$[\ x := e\ ]\ P$ est le prédicat obtenu en remplaçant toutes les **occurrences libres** de x dans P par e.

Règle de typage:

- Dans la substitution $x := e$, x et e doivent être du même type T.
- Dans la substitution $x1,.., xn := e1, ..., en$, chaque xi doit être du même type que ei.

Substitution devient égal

- Une **variable** est dite **libre** dans un prédicat ou dans une expression si:
  - Elle est présente dans la formule

  et
  - Elle est présente dans des sous formules qui ne sont pas sous la portée de certains quantificateurs introduisant une variable quantifiée de même nom.
- **Exemple:** Soit le prédicat: $\forall n \cdot (n \in \mathbb{N} \Rightarrow n > x)$

y est-elle libre? n est-elle libre? x est-elle libre?

Exemple de calcul: OP de reserver

MACHINE
  reservation
VARIABLES
  nbPlacesLibres
INVARIANT
  $nbPlacesLibres \in 0..100$
...
OPERATIONS
  reserver(nbPlaces)=
    PRE
      $nbPlaces \in NAT1 \wedge$
      $nbPlaces \leq nbPlacesLibres$
    THEN
      $nbPlacesLibres := nbPlacesLibres\text{-}nbPlaces$
    END
END ` ` `

$I \wedge Q \Rightarrow [V]I$ devient:

1. $(nbPlacesLibres \in 0..100 \wedge nbPlaces \in NAT1 \wedge nbPlaces \leq nbPlacesLibres)$
$\Rightarrow$
$[nbPlacesLibres := nbPlacesLibres\text{-}nbPlaces]$
  $(nbPlacesLibres \in 0..100)$

2. $(nbPlacesLibres \in 0..100 \wedge nbPlaces \in NAT1 \wedge nbPlaces \leq nbPlacesLibres)$
$\Rightarrow (nbPlacesLibres\text{-}nbPlaces \in 0..100)$

Substitution simultanée

- Correspond à l'exécution simultanée de deux substitutions

Deux syntaxes:

- x := E | | y := F
- x, y := E, F

Sémantique

[x, y := E, F] I ⇔

[z := F][x := E][y := z]I z étant une variable distincte de x et y et non libre dans I

Substitution simultanée

- **Exemple:** permutation de 2 variables

$[x, y := y, x]\ (x < y)$

$[z := x][x := y][y := z]\ (x < y)$

$[z := x][x := y]\ (x < z)$

$[z := x]\ (y < z)$

$(y < x)$

Substitution choix borné

- Permet de définir un nombre fini de comportements possibles sans préciser lequel sera effectivement choisi (non déterministe).
- n'est pas une substitution d'implémentation
- ***Syntaxe***
  - **CHOICE *S1* OR … OR *Sn* END**
  - *S1*, …, *Sn* des substitutions (avec $n \geq 2$)
- ***Sémantique***

[CHOICE *S1* OR … OR *Sn* END] *I* ⇔ [*S1*] *I* ∧ … ∧ [*Sn*] *I*

Substitution choix borné

- **Exemple** : Définition des résultats de l'examen du permis de conduire

CHOICE résultat := réussite ||

permis := permis U {candidat}

OR résultat := échec

END

- **Calculer :**

[CHOICE résultat := réussite || permis := permis U {candidat} **OR** résultat := échec **END]** (permis ⊆ PERS-MAJEUR)

Select (Garde et Choix Borné)

- La substitution SELECT permet d'exprimer des gardes qui définissent les conditions d'exécution de l'instruction.
- **Syntaxe** **Sémantique**

Syntaxe	Sémantique
[**SELECT Q1 THEN S1**	**(Q1 ⇒ [S1]I) ∧**
WHEN Q2 THEN S2	**(Q2 ⇒ [S2]I) ∧**
...	**...**
WHEN Qn THEN Sn	**(Qn ⇒ [Sn]I) ∧**
[**ELSE SE**]	**((¬Q1∧¬Q2∧...∧¬Qn) ⇒ [SE]I)**
END]	

- Les gardes peuvent ne pas être disjointes, dans ce cas le comportement est non-déterministe.

Select (Garde et Choix Borné)

- **Exemple**

  SELECT $x \geq 0$ THEN

  $y := x^2$

  WHEN $x \leq 0$ THEN

  $y := - x^2$

  END

Substitution identité

- La substitution identité ***Skip*** ne modifie pas le prédicat sur lequel elle est appliquée.
- Elle est notamment utilisée pour décrire que certaines branches d'une substitution IF, CASE ou SELECT ne modifient pas les variables.

Sémantique:

$$[skip]\ I \Leftrightarrow I$$

Condition par Cas

- La substitution CASE permet de définir différents comportements possibles en fonction de la valeur d'une expression.
- **Syntaxe:**

```
CASE E OF
    EITHER L1 THEN S1
    [OR L2 THEN S2
    ...
    OR Ln THEN Sn]
     [ELSE SE]
END
```

- Chaque branche EITHER et OR est constituée d'une liste non vide de constantes littérales (entier, énuméré, booléen).
- Les L1, .. , Ln doivent être disjoints (déterminisme) et de même type que E.
- Si le ELSE est absent, l'échec des Li réalise la substitution *skip*.

Condition par Cas

Sémantique:

CASE E OF	
EITHER L1 THEN S1	**$E \in \{L1\} \Rightarrow [S1]I \wedge$**
[OR L2 THEN S2	**$E \in \{L2\} \Rightarrow [S2]I \wedge$**
…	**…**
OR Ln THEN Sn]	**$E \in \{Ln\} \Rightarrow [Sn]I \wedge$**
[ELSE SE]	**$E \notin \{L1,L2…Ln\} \Rightarrow [SE]I$**
	(en l'absence de ELSE
END	**SKIP**)

Condition par Cas

- **Exemple:**

```
MACHINE
triangle
SETS
TRIANG_TYPE = {scalene,
    isosceles, equilateral,
    no_triangle}
OPERATIONS
tr_type <-- classify(s1, s2,
    s3) =
PRE
s1 ∈ INT ∧ s1 ≠0 ∧
s2 ∈ INT ∧ s2 ≠0 ∧
s3 ∈ INT ∧ s3 ≠0
THEN
IF s1+s2 ≤ s3 or s2+s3
     ≤ s1 or s1+s3 ≤ s2
THEN tr_type :=
     no_triangle
ELSE
CASE card({s1, s2, s3})
     OF
EITHER 1 THEN
     tr_type:=equilateral
OR 2 THEN
     tr_type:=isosceles
OR 3 THEN
     tr_type:=scalene
END END
END
END
END
```

Substitution conditionnelle IF

- La substitution conditionnelle IF est définie selon plusieurs formes :

1. IF P THEN S END

Sémantique

$$[P \Rightarrow S]I \quad P \Rightarrow [S]I$$

2. IF P1 THEN S1 ELSE T END

Sémantique

$$[\text{IF } P1 \text{ THEN } S1 \text{ ELSE } T \text{ END}]I \Leftrightarrow$$

$$(P1 \Rightarrow [S1]I) \wedge (\neg P1 \Rightarrow [T]I)$$

Substitution conditionnelle IF

3. Forme générale du IF

Syntaxe:

IF P1 THEN S1
[ELSIF P2 THEN S2
...
ELSIF Pn THEN Sn]
[ELSE SE]
END

$\Leftrightarrow$

Sémantique:

$P1 \Rightarrow [S1]I \wedge$
$\neg P1 \wedge P2 \Rightarrow [S2]I \wedge$
...
$\neg P1 \wedge \neg P2 \wedge \ldots \neg Pn\text{-}1 \wedge Pn \Rightarrow [Sn]I \wedge$
$\neg P1 \wedge \neg P2 \wedge \ldots \neg Pn\text{-}1 \wedge \neg Pn \Rightarrow [SE]I$

Substitution conditionnelle IF

- ***Exemple1***

IF $x \in \{ 2, 4, 8 \}$ THEN
    $x := x / 2$
END

- ***Exemple2***

    IF v > 130 THEN
        T := v || Alarm := désactivé
    ELSE
        T := v
    END

- ***Exemple3***

IF $v = 0$ THEN
    signe := 0
ELSIF $v > 0$ THEN
    signe := 1
ELSE
    signe := -1
END

Substitution conditionnelle IF

A ne pas confondre avec la pré-condition

- La substitution pré-condition n'est utilisable que si le prédicat est valide <u>alors que</u> la substitution gardée est toujours réalisée mais son résultat dépend de la validité d'un prédicat.

Exemple de calcul de l'opération changerT

```
MACHINE Chaudiere
SETS
        ETATS = {activé, désactivé}
VARIABLES
        T, Alarme
INVARIANT
        T ∈ NAT ∧
        Alarme ∈ ETATS ∧
        (T > 130) ⇔ (Alarme = activ
….
OPERATIONS
changerT (v) ==
PRE
        v ∈ NAT ∧ v ≤ 130
THEN
        T := v
END
…
```

I ∧ Q ⇒ [V]I devient

1. $T \in NAT \wedge Alarme \in ETATS \wedge$
$((T > 130) \Leftrightarrow (Alarme = activé))$
$\wedge\ v \in NAT \wedge v \leq 130$
$\Rightarrow$
$[T := v]\ (T \in NAT \wedge Alarme \in ETATS \wedge$
$((T > 130) \Leftrightarrow (Alarme = activé)))$

Après application de la substitution, on obtient:

2. $T \in NAT \wedge Alarme \in ETATS \wedge$
$((T > 130) \Leftrightarrow (Alarme = activé))$
$\wedge\ v \in NAT \wedge v \leq 130$
$\Rightarrow$
$(v \in NAT \wedge$
$Alarme \in ETATS \wedge$
$((v > 130) \Leftrightarrow (Alarme = activé))).$

Exemple de calcul: opération changerT

MACHINE **Chaudiere**
SETS
    ETATS = {activé, désactivé}
VARIABLES
    T, Alarme
INVARIANT
    $T \in NAT \wedge$
    $Alarme \in ETATS \wedge$
    $(T > 130)\ ) \Leftrightarrow (Alarme = activé)$
INITIALISATION …
OPERATIONS
changerT (v) ==
PRE
    $v \in NAT \wedge alarme = désactivé$
THEN
    IF ($v \leq 130$) THEN T := v
    ELSE T:= V || alarme := activé
END
…

$I \wedge Q \Rightarrow [V]I$ devient

1. $T \in NAT \wedge Alarme \in ETATS \wedge$
$((T > 130) \Leftrightarrow (Alarme = activé))$
$\wedge\ v \in NAT \wedge alarme = désactivé$
$\Rightarrow$
$(v \leq 130 \Rightarrow [T := v]$
$(T \in NAT \wedge$
$Alarme \in ETATS \wedge$
$((T > 130)\ ) \Leftrightarrow (Alarme = activé)))$
$\wedge (v > 130 \Rightarrow$
$[T := v \parallel alarme := activé]$
$(T \in NAT \wedge$
$Alarme \in ETATS \wedge$
$((T > 130)\ ) \Leftrightarrow (Alarme = activé)))$
…

Substitution Choix non borné

- **Syntaxe:**

  ANY x WHERE Q THEN S END

- L'instruction ANY correspond à un choix arbitraire d'une valeur de x suivant le prédicat Q
- La variable x doit être typée dans Q
- Non déterministe et n'est pas une substitution d'implémentation
- ***Sémantique***
  - $\forall x.(Q \Rightarrow [S]I)$
- Avec plusieurs variables (qui doivent être deux à deux distinctes):

  ANY x1,x2,…,xn **WHERE** Q **THEN** T **END**

Substitution Choix non borné

- **Exemple :** Donner une expression qui choisit un triangle rectangle arbitraire et affecte la longueur de ses côtés aux variables a,b,c:

  ANY x,y,z

  WHERE $x \in NAT1 \wedge y \in NAT1 \wedge z \in NAT1 \wedge x^2 + y^2 = z^2$

  THEN a, b, c := x, y, z

  END

Dérivées du choix non borné

- **Substitution « devient élément de »**
  - **Syntaxe: $X :\in E$**

Avec E un ensemble, *X* une liste de variables modifiables non vide.

  - **Sémantique:**
    - $X :\in E \Leftrightarrow$ ANY Y WHERE $Y \in E$ THEN $X := Y$ END
    - $\forall Y.(Y \in E \Rightarrow [X:=Y]I)$
  - Exemples :
    - *i1* : ∈ INT ;
    - *b1* : ∈ BOOL ;
    - *x1* : ∈ -10 .. 10 ;
    - *y1*, *y2* : ∈ {1, 3, 5} * NAT

Dérivées du choix non borné

- **Substitution « devient tel que »**
  - **Syntaxe: x:(Q)**
    - *Avec Q* un prédicat, *X* une liste de variables modifiables deux à deux distinctes.
    - Les variables de la liste X, doivent être typées dans le prédicat Q
  - **Sémantique:**
    - X :(Q) $\Leftrightarrow$ ANY Y WHERE [X:=Y]Q THEN X:=Y END
  - Exemples :
    - $x : (x \in \mathrm{INT} \wedge x > -4 \wedge x < 4)$;
    - $a, b : (a \in \mathrm{INT} \wedge b \in \mathrm{INT} \wedge a^2 + b^2 = 25)$;

Résumé

Le langage des substitutions :

- comprend des actions de choix non déterministes (CHOICE, ANY, SELECT)
  - En modélisation, il n'est pas toujours utile de préciser tous les aspects de la machine
- comprend la construction PRE pour imposer une pré-condition pour l'entrée dans une opération
- comprend l'action *skip* qui n'a pas d'effet sur l'état modélisé du système
- comprend des actions standards des langages de programmation :
  - Affectation,
  - Si
  - case
- ne comprend pas d'itération

EXERCICE: Contrôleur de feux de circulation d'un carrefour

- On souhaite spécifier le fonctionnement d'un contrôleur de feux tricolores d'un carrefour.
- Les feux peuvent être :
  - Hors service : tous les feux sont au jaune
  - En service : ils évoluent selon: rouge $\leadsto$ vert $\leadsto$ jaune $\leadsto$ rouge
- Propriétés souhaitées:
  - Les véhicules ne peuvent s'engager dans les deux voies simultanément (sûreté).
  - les véhicules ne sont pas bloqués infiniment sur l'une des (ou les deux) voies (vivacité).
- Les opérations à spécifier sur les feux de circulation sont: Mise_ en_service, Mise_hors_service et Changer_feu

Travail demandé:

1. Spécifier les données de la machine B associée à ce système
2. Décrire les invariants de ce modèle
3. Donner la machine résultante

Solution

- Définition des couleurs des feux :

  Un ensemble énuméré: COULEUR = {rouge , jaune , vert}
- L'évolution des feux peut être décrite par une fonction constante:

  suiv $\in$ Couleur $\rightarrow$ Couleur

  suiv (rouge) = vert, suiv (vert) = jaune , suiv (jaune) = rouge
- Variables: feuA, feuB
- Les propriétés souhaitées peuvent être décrites par:

  $(feuA=rouge \wedge feuB\neq rouge) \vee (feuA\neq rouge \wedge feuB=rouge)$

Solution

MACHINE
carrefour
SETS
COULEUR = {rouge, jaune, vert}
CONSTANTS
Suiv
PROPERTIES
Suiv ∈ COULEUR → COULEUR ∧
Suiv(rouge) = vert ∧
Suiv(vert) = jaune ∧
Suiv(jaune) = rouge
VARIABLES
feuA, feuB
DEFINITIONS
hs == feuA = jaune ∧ feuB = jaune;
service(aa,bb) == (aa = rouge ∧ bb ≠ rouge) or (aa ≠ rouge ∧ bb = rouge);
es == service(feuA, feuB)
INVARIANT
feuA, feuB ∈ COULEUR * COULEUR ∧ (hs or es)
INITIALISATION
feuA, feuB := jaune, jaune

Solution

```
OPERATIONS
Mise_en_service = PRE hs
THEN ANY fa, fb WHERE
fa ∈ COULEUR ∧ fb ∈ COULEUR ∧ (service(fa,fb))
THEN
feuA := fa||feuB := fb
END
END;
Mise_hors_service = PRE es
THEN
feuA, feuB := jaune, jaune
END;
Changer_feu = PRE es
THEN
ANY fa, fb WHERE fa ∈ COULEUR ∧ fb ∈ COULEUR ∧
((fa = feuA ∧ fb = Suiv(feuB)) or
(fa = Suiv(feuA) ∧ fb = feuB) or
(fa = Suiv(feuA) ∧ fb = Suiv(feuB)) ∧
(service(fa,fb)))
THEN
feuA, feuB := fa, fb
END
END
END
```

Notation de modélisation des données

- En plus du langage des substitutions généralisés, la notation B est fondée sur la logique des prédicats du 1er ordre étendue à un fragment de la théorie des ensembles.
  - Logique du 1^{er} ordre
  - Expressions ensemblistes
  - Relations et fonctions
  - Arithmétique

Notation B

Prédicats

Expression	Math	ASCII
Conjonction	$P \wedge Q$	P & Q
Disjonction	$P \vee Q$	P or Q
Implication	$P \Rightarrow Q$	P => Q
Equivalence	$P \Leftrightarrow Q$	P <=> Q
Négation	$\neg P$	not P
Quantification Universelle	$\forall z . (P \Rightarrow Q)$	!z . (P => Q)
Quantification Existencielle	$\exists z . (P \wedge Q)$	# z . (P & Q)
Egalité	$E = F$	E = F
Inégalité	$E \neq F$	E /= F

- **P** ⇒ **Q** est vrai si et seulement si Q est vrai ou P n'est pas vrai
- **P** ⇔ **Q** est vrai ssi P ⇒ Q et Q ⇒ P sont vrais

Ensembles : les prédéfinis

- $\mathbb{N}$ les entiers naturels (NATURAL en ASCII)
- $\mathbb{N}1$ les entiers naturels non nuls (NATURAL1 en ASCII)
- $\mathbb{Z}$ les entiers relatifs (INTEGER en ASCII)
- I..J les intervalles d'entier, l'ensemble des valeurs comprises entre I et J (bornes incluses)
- INT les entiers relatifs implantables : MININT..MAXINT
- NAT les entiers naturels implantables : 0..MAXINT
- NAT1 les entiers naturels non nuls implantables : 1..MAXINT
- BOOL les booléens = {FALSE,TRUE}
  - *bool(P) retourne le booléen résultat d'une formule P*
- STRING l'ensemble des chaînes de caractères.

Expressions arithmétiques

Expression	B	ASCII
Plus grand	m > n	m > n
Plus grand ou égal	$m \geq n$	m >= n
Plus petit	m < n	m < n
Plus petit ou égal	$m \leq n$	m <= n
Maximum des éléments d'un ens.	max(S)	max(S)
Minimum des éléments d'un ens.	min(S)	min(S)
Division entière Modulo	m div n m mod n	m / n m mod n
Puissance	x^y	x**y
Successeur Prédécesseur	succ pred	succ pred

Notation ensembliste

Expression	Math	ASCII
Ensemble singleton	$\{E\}$	{E}
Ensemble énuméré	$\{E, \ldots, F\}$	{E,...,F}
Ensemble vide	$\varnothing$	{}
Ensemble en compréhension	$\{ z \mid P\}$	{ z | P}
Union Intersection	$S \cup T$ $S \cap T$	S \/ T S /\ T
Différence	$S - T$	S - T

- Exemple d'ensemble énuméré
  - E={1,3,5,7,9}
- Exemple d'ensemble définit par Compréhension
  - $E=\{x \mid x \in Z \wedge x>0 \wedge x<10 \wedge x \bmod 2 =1\}$

Notation ensembliste

Expression	Math	ASCII
Appartenance	$E \in S$	E : S
Non appartenance	$E \notin S$	E / : S
Inclusion	$S \subseteq T$	S < : T
Inclusion stricte	$S \subset T$	S <<: T
Non inclusion stricte	$S \not\subset T$	S /<<: T

Notation ensembliste

Expression	Math	ASCII	Définition
Paire ordonnée	$x \mapsto y$	x |-> y	(x, y)
Produit Cartésien	$S \times T$	S * T	$\{x, y \mid x \in S \wedge y \in T\}$
Ensemble des parties	$\mathbb{P}(S)$	POW(S)	$\{s \mid s \subseteq S\}$
Ensemble des parties non vide	$\mathbb{P}_1(S)$	POW1(S)	$\mathbb{P}(S) - \{\}$
Cardinalité	card(S)	card(S)	

choice(E): un élément indéterminé de l'ensemble E

Relations

- *r* est un sous-ensemble du produit cartésien *D x A* de deux ensembles
  - *D* : **ensemble de départ**
  - *A* : **ensemble d'arrivée**
- $r \subseteq D \times A$ ou $r \in P(D \times A)$ ou $\mathbf{r} \in \mathbf{D \leftrightarrow A}$
  - *r est une relation de D dans A*
  - *D*↔A désigne l'ensemble de toutes les relations de D vers A
- Soit un couple *d*|->*a* d'une relation *r*
  - *a* est une **image** de *d* par *r*
  - *d* est un **antécédent** de *a* par *r*

Relations

Expression	Math	ASCII	Définition
Relation	$S \leftrightarrow T$	S <-> T	$\mathbb{P}(S \times T)$
Domaine	dom(R)	dom(R)	$\mathrm{dom}(R) = \{x \mid x \in S \wedge \exists y.(y \in T \wedge x \mapsto y \in R)\}$ $R \in S \leftrightarrow T$
Image	ran(R)	ran(R)	$\mathrm{ran}(R) = \{y \mid y \in T \wedge \exists x.(x \in S \wedge x \mapsto y \in R)\}$ $R \in S \leftrightarrow T$

Exemple : Soit $R = \{1 \mapsto a, 2 \mapsto b, 2 \mapsto c, 3 \mapsto a, 3 \mapsto c\}$

$\mathrm{dom}(R) = \{1, 2, 3\}$

$\mathrm{ran}(R) = \{a, b, c\}$

Relations

Expression	Math	ASCII	Définition
Restriction de domaine	$U \lhd R$	U <| R	$\{x \mapsto y \mid x \mapsto y \in R \wedge x \in U\}$ $R \in S \leftrightarrow T$
Anti-restriction de domaine	$U \mathbin{⩤} R$	U <<| R	$\{x \mapsto y \mid x \mapsto y \in R \wedge x \notin U\}$ $R \in S \leftrightarrow T$
Restriction d'image	$R \rhd V$	R |> V	$\{x \mapsto y \mid x \mapsto y \in R \wedge y \in V\}$ $R \in S \leftrightarrow T$
Anti-restriction d'image	$R \mathbin{⩥} V$	R |>> V	$\{x \mapsto y \mid x \mapsto y \in R \wedge y \notin V\}$ $R \in S \leftrightarrow T$

Exemple : Soit $R = \{1 \mapsto a, 2 \mapsto b, 2 \mapsto c, 3 \mapsto a, 3 \mapsto c\}$

$\{1,3\} \lhd R = \{1 \mapsto a, 3 \mapsto a, 3 \mapsto c\}$

$R \mathbin{⩥} \{c\} = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto a\}$

Relations

Expression	Math	ASCII	Définition
Image relationnelle	R[U]	R[U]	$\{ y \mid x \mapsto y \in R \wedge x \in U \}$

Exemple : Soit $R = \{1 \mapsto a, 2 \mapsto b, 2 \mapsto c, 3 \mapsto a, 3 \mapsto c\}$

$R[\{1,3\}] = \{a,c\}$

$R[\{1,2,3,4\}] = \{a,b,c\}$

$R[\{4,5\}] = \{\}$

$R[U] = ran(U \lhd R)$

Relations

Expression	Math	ASCII	Définition
Inverse	R^{-1}	$R^{\sim}$	$\{y \mapsto x \mid x \mapsto y \in R)$
Composition relationnelle	$R_0 ; R_1$	$R_0 ; R_1$	$\{x \mapsto y \mid x \in S \wedge y \in T \wedge \exists z . (z \in U \wedge x \mapsto z \in R_0 \wedge z \mapsto y \in R_1)\}$ $R_0 \in S \leftrightarrow U \wedge R_1 \in U \leftrightarrow T$
Identité	id(S)	id(S)	$\{x \mapsto x \mid x \in S \}$

Règles :

$$(S \lhd R)^{-1} = (R^{-1}) \rhd S$$

$$(R \rhd T)^{-1} = T \lhd (R^{-1})$$

Relations

Expression	Math	ASCII	Définition
Surcharge relationnelle	$R0 \Leftarrow\!\!+ R1$	R0 <+ R1	$(\mathrm{dom}(R1) \lhd R0) \cup R1$

Exemple :

$R0 = \{2 \mapsto a, 2 \mapsto d, 3 \mapsto b, 4 \mapsto c, 4 \mapsto d\}$

$R1 = \{0 \mapsto a, 1 \mapsto b, 2 \mapsto c\}$

$R0 <+ R1 = \{0 \mapsto a, 1 \mapsto b, 2 \mapsto c, 3 \mapsto b, 4 \mapsto c, 4 \mapsto d\}$

Relations-Exemple

- Soit PERSONNE, l'ensemble de toutes les personnes et les relations père, mère

  père $\in$ PERSONNE $\leftrightarrow$ PERSONNE

  mère $\in$ PERSONNE $\leftrightarrow$ PERSONNE

- On peut définir :

  pèreOuMère = père $\cup$ mère

  frèreEtSoeur = (pèreOuMère^{-1} ; pèreOuMère) – id(PERSONNE)

- Exemple :

père = {jean $\mapsto$ julie, jean $\mapsto$ anne, claude $\mapsto$ jean, claude $\mapsto$ alice}

mère = {édith $\mapsto$ julie, édith $\mapsto$ anne, isabelle $\mapsto$ jean, sylvie $\mapsto$ alice}

Relations-Exercice

- Soit **PAYS**, l'ensemble de tous les pays

  LANGUE, l'ensemble de toutes les langues

  parle, la relation associant un pays à ses langues officielles

$parle \in PAYS \leftrightarrow LANGUE$

$parle = \{F \mapsto français, CH \mapsto français, CH \mapsto allemand, CH \mapsto italien, CH \mapsto romanche, B \mapsto français, B \mapsto néerlandais, D \mapsto allemand, ...\}$

- **UE**, l'ensemble des pays de l'Union Européenne: $UE \subseteq PAYS$
- Définir :
  - l'ensemble des pays anglophones
  - l'ensemble des pays francophones hors de l'Union Européenne
  - l'ensemble des pays possédant plus d'une langue officielle
  - La relation même_langue des pays possédant une même langue

Relations-Solution de l'exercice

- l'ensemble des pays anglophones

  $parle^{-1}[\{anglais\}]$

  $dom(parle \rhd \{anglais\})$

  $\{x \mid x \in PAYS \wedge x \mapsto anglais \in parle\}$

- l'ensemble des pays francophones hors de l'Union Européenne

  $parle^{-1}[\{français\}] - UE$

  $dom(UE \lhd (parle \rhd \{français\}))$

  $\{x \mid x \in PAYS \wedge x \notin UE \wedge x \mapsto français \in parle\}$

- l'ensemble des pays possédant plus d'une langue officielle $\{x \mid x \in PAYS \wedge card(parle[\{x\}]) > 1\}$

- La relation même_langue des pays possédant une même langue

  $(parle\ ;\ parle^{-1}) - id(PAYS)$

Fonctions

- Les fonctions **sont des relations** pour lesquelles **chaque élément du domaine possède au plus une image** :
  - $x1 = x2 \Rightarrow f(x1) = f(x2)$
- Exemple:

  $exemplaire \in livre \leftrightarrow \mathbb{N}1$
  - exemplaire = {B-Book↦1, B-Book↦2, UML↦1, UML↦2, GL↦1}

  $emprunt \in exemplaire \nrightarrow abonne$
  - emprunt = {(B-Book↦1) ↦ab1, (B-Book↦2) ↦ab2, (UML↦1) ↦ab2, (UML↦2) ↦ab3, (GL↦1) ↦ab4}
- Types de fonctions :
  - partielle ou totale,
  - injective, surjective ou bijective.

Fonctions

- la fonction totale
  - tout élément de l'ensemble de départ a une et une seule image
- La fonction partielle
  - tout élément de l'ensemble de départ a au plus une image
- l'injection
  - tout élément de l'ensemble d'arrivée a au plus un antécédent
- la surjection
  - tout élément de l'ensemble d'arrivée a au moins un antécédent

Fonctions

Expression	Math	ASCII	Définition
Fonction partielle	$S ⇸ T$	S +-> T	$\{f \mid f \in S \leftrightarrow T \wedge f^{-1};f \subseteq id(T)\}$
Fonction totale	$S \rightarrow T$	S --> T	$\{f \mid f \in S ⇸ T \wedge dom(f) = S\}$
Injection partielle	$S ⤔ T$	S >+> T	$\{f \mid f \in S ⇸ T \wedge f^{-1} \in T ⇸ S\}$
Injection totale	$S \rightarrowtail T$	S >-> T	$S ⤔ T \cap S \rightarrow T$
Surjection partielle	$S ⤀ T$	S +->> T	$\{f \mid f \in S ⇸ T \wedge ran(f) = T\}$
Surjection totale	$S \twoheadrightarrow T$	S -->> T	$S ⤀ T \cap S \rightarrow T$
Bijection partielle	$S ⤗ T$	S >+>> T	$S ⤔ T \cap S ⤀ T$
Bijection totale	$S ⤖ T$	S >->> T	$S \rightarrowtail T \cap S \twoheadrightarrow T$

Fonctions

- L'image fonctionnelle d'un élément : **f(x)**
  - f doit être une fonction
  - x doit appartenir à son domaine
- fonctions abstraites (lambda expressions) :

  $\boldsymbol{\lambda x.(x \in D \mid e)}$
  - D est le domaine de définition de la fonction
  - e une expression paramétrée par x définissant l'image pour tout x de D
  - (λ noté % en ASCII)
- Exemple:

$$\lambda x.(x \in 2 .. 6 \wedge x \neq 4 \mid x^2) = \{2 \mapsto 4, 3 \mapsto 9, 5 \mapsto 25, 6 \mapsto 36\}$$

Fonctions

Lamdba expression-Exercice

- En utilisant les lambda expressions, définir:
  1. Une fonction qui associe n avec $1 + 2^n$
  2. Une fonction qui associe un ensemble non vide d'entiers naturels avec son plus petit membre
- **Réponse:**
  1. $\lambda x.(x \in \mathbb{N} \mid 1 + 2^x)$
  2. $\lambda s.(s \in \mathbb{P}(\mathbb{N}) \wedge s \neq \{\} \mid \min(s))$

Exercice- Les relations familiales

- Soit les ensembles et relations suivants :
  - PERSONNE, ensemble des personnes
  - GENRE = {masculin, féminin}
  - père_de, relation père / enfant
  - mère_de, relation mère / enfant
- Définir les relations ou fonctions suivantes :
  - genre,
  - frère_de, soeur_de,
  - mari_de, femme_de
  - oncle_de, tante_de,
  - cousin_de,

Solution

Définitions:

- $genre \in PERSONNE \rightarrow GENRE$
- $frère\_de \in genre^{-1}[\{masculin\}] \leftrightarrow PERSONNE$
- $sœur\_de \in genre^{-1}[\{féminin\}] \leftrightarrow PERSONNE$
- $mari\_de \in genre^{-1}[\{masculin\}] \rightarrowtail\!\!\!\!+\; genre^{-1}[\{féminin\}]$
- $femme\_de \in genre^{-1}[\{féminin\}] \rightarrowtail\!\!\!\!+\; genre^{-1}[\{masculin\}]$
- $oncle\_de = frère\_de \,;\, (père\_de \cup mère\_de)$
- $tante\_de = sœur\_de \,;\, (père\_de \cup mère\_de)$
- $cousin\_de = \{x \mapsto y \mid x \in genre^{-1}[\{masculin\}] \wedge y \in PERSONNE \wedge \exists z1,z2 \,.\, (z1 \mapsto x \in (père\_de \cup mère\_de) \wedge z2 \mapsto y \in (père\_de \cup mère\_de) \wedge z1 \mapsto z2 \in (frère\_de \cup sœur\_de))\}$

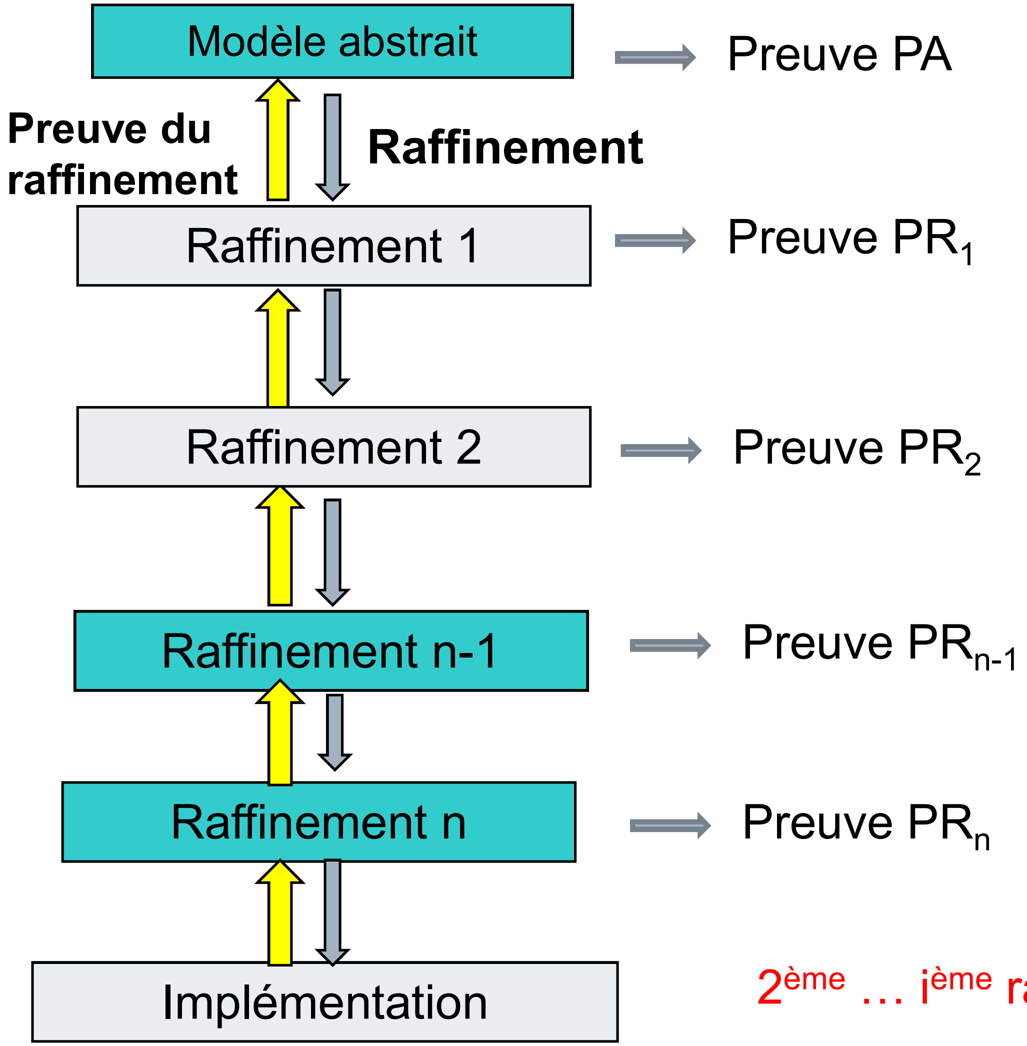
Le processus de raffinement et les obligations de preuve du raffinement

Introduction

- La finalité du raffinement est l'obtention du code exécutable
- **Raffinement :** Passage progressif d'un modèle abstrait (utilisant des objets mathématiques) vers un modèle concret (utilisant des objets informatiques):
  - Changement de niveau d'abstraction (ajout de détails)
  - Choix (conception)
- Phases de conception et implémentation dans le cycle de vie du logiciel.

Rappel: Méthode B

PA: propriété abstraite
PR: propriété de raffinement



2ème … ième raffinement

Le processus de raffinement

- **MACHINE**: décrit un comportement de manière abstraite par son état et ses opérations;
- **REFINEMENT** (raffinement) est une étape du développement d'une machine; détaille la structure et les opérations de la machine.
- **IMPLEMENTATION**: le dernier maillon du développement d'une machine, le plus proche du codage

Le processus de raffinement

Syntaxe

- Utiliser la clause REFINES pour faire le lien entre la machine abstraite et son raffinement

```
REFINEMENT
MM_R1
REFINES
MM
...
END
```

- Le dernier raffinement est une implémentation

```
IMPLEMENTATION
MM_I1
REFINES
MM_R1
...
END
```

Le processus de raffinement

Propriétés du raffinement

- **Le raffinement B est correct :**

tout ce qui est vrai pour le composant raffiné est aussi vrai pour le raffinement

- **Le raffinement B est transitif :**

($M1 \subseteq M2$ et $M2 \subseteq M3$) alors $M1 \subseteq M3$

M2 raffine M1 est notée $M1 \subseteq M2$

Processus de raffinement

QUOI RAFFINER?

1. Les **variables et l'invariant**

2. Les **opérations**

Processus de raffinement: **comment raffiner?**

1. Raffinement des variables et de l'invariant

- Introduire de nouvelles variables (plus concrètes),
- Choix de structures de données moins abstraites,
- Liaisons entre les variables concrètes et abstraites par un **invariant de liaison (appelé aussi invariant de collage)**

Processus de raffinement: comment raffiner?

Exemple

MACHINE Ensembles
SETS
ENS
VARIABLES
ens
INVARIANT
ens $\subseteq$ **ENS**
INITIALISATION
ens:= $\varnothing$

REFINEMENT Ensembles1

REFINES Ensembles

VARIABLES

tab

INVARIANT

tab ϵ ENS $\rightarrow$ **BOOL ^**

$\left.\text{Tab}^{-1}\ [\{\textbf{TRUE}\}] = \textbf{ens}\right\}$ **Invariant de liaison**

INITIALISATION

tab := ENS x {**FALSE}**

Processus de raffinement: comment raffiner?

2. Raffinement des opérations

- Toute opération doit être raffinée avec la même signature
- Réécrire les opérations abstraites avec les nouvelles variables et les substitutions appropriées →Introduire des **substitutions de raffinements**
  - <u>Substitution séquencement</u>
  - <u>Substitution Variable locale</u>
  - <u>Substitution boucle tant que</u>(seulement pour l'implémentation)
- Expliciter les algorithmes
- Lever le non-déterminisme
- Affaiblir les pré-conditions jusqu'à les faire disparaître.

Processus de raffinement: comment raffiner?

- Exemple

```
d ‹ — empiler(w) =
PRE w ∈ VALEUR
THEN
CHOICE
      d := ok ||
      sui := sui ← w
      OR
      d := ko
END ;
```

```
d ‹ — empiler(w) =
PRE w ∈ VALEUR
THEN
      IF nb < taille_max THEN
      d := ok ||
      nb := nb + 1 ||
      tab (nb + 1) := w
ELSE
      d := ko
END ;
```

Substitutions de raffinement

Substitution séquencement

- Soient S et T deux substitutions, la substitution séquencement est notée: $S \; ; \; T$
- Sémantique :

$$[S; T]R = [S][T]R$$
$$= [S]([T]R)$$

- Exemple :

Xx:=yy||yy:=xx devient zz := yy ; yy := xx ; xx := zz

Substitutions de raffinement

Substitution variable locale

- La notation est : **VAR *X* IN *S* END**
- La sémantique:
  - Soient *X* une liste de variables deux à deux distinctes, *S* une substitution et *P* un prédicat, alors :

$$[\text{VAR } X \text{ IN } S \text{ END}]\, P \;\Leftrightarrow\; \forall X\,.\,[S]\, P$$

- Les variables locales sont accessibles en lecture et en écriture.

Substitutions de raffinement

Substitution variable locale

- **Exemple:**

MACHINE Permutation
OPERATION

Permut(xx,yy)=PRE xx:NAT ∧yy:NAT
BEGIN
xx:=yy||yy:=xx
END
END

IMPLEMENTATION
Permutation1
Refines Permutation
Permut(xx,yy)=
BEGIN
VAR zz IN
zz **:= yy ; yy := xx ; xx := zz**
END
END
END

Substitutions de raffinement

Substitution boucle tant que:

WHILE *P* DO *S* INVARIANT *I* VARIANT *V* END

- La substitution S est exécutée tant que le prédicat P est vraie.
- Le variant V est une expression entière qui permet de démontrer que la boucle se termine au bout d'un nombre fini d'itérations.
  - Pour cela, il faudra prouver que *V* est une expression entière, positive qui décroît strictement à chaque itération.
- I est l'invariant de la boucle: donne des propriétés sur les variables utilisées dans la boucle.
  - L'invariant permet de prouver qu'à chaque pas la boucle est possible et qu'elle donne bien le résultat produit à la sortie.
- C'est une substitution d'implémentation

Substitutions de raffinement

Exemple

```
VAR varLoc, cpt IN
    varLoc:=var1; cpt:=0
    WHILE cpt<5 DO
        varLoc:= varLoc+1;
        cpt:=cpt+1
    INVARIANT
        cpt ∈ NAT ∧
        cpt ≤ 5 ∧
        varLoc ∈ NAT ∧
        varLoc = var1 + cpt
    VARIANT 5-cpt
    END
END
```

Substitutions de raffinement

- Sémantique

$$[\text{WHILE } P \text{ DO } S \text{ INVARIANT } I \text{ VARIANT } V \text{ END}]\, R \;\Leftrightarrow$$

$$I \;\wedge$$

$$\forall X . (I \wedge P \Rightarrow [S]\, I) \;\wedge$$ ***Prouver la préservation de l'invariant***

$$\forall X . (I \Rightarrow V \in \mathbb{N}) \;\wedge$$ ***Prouver que V est une expression entière positive***

$$\forall X . (I \wedge P \;\Rightarrow\; [n := V ; S](V < n)) \;\wedge$$ ***Prouver que V décroît strictement à chaque itération***

$$\forall X . (I \wedge \neg P \;\Rightarrow\; R))$$ ***Prouver la sortie de la boucle***

- *X* représente la liste des variables libres apparaissant dans *S* et *I*
- *n* une variable non libre dans *V*, *I*, *P* et *S*

La clause VALUES

- La clause VALUES permet de donner une valeur aux constantes concrètes et aux ensembles abstraits de l'implémentation.
- Le nom de la clause VALUES est suivi d'une liste de valuations
- Chaque valuation permet de donner explicitement une valeur à une constante concrète ou à un ensemble abstrait.
- Syntaxe: NomDonnéeAValuer = valeur

La clause VALUES

- Exemple1

MACHINE *MA*	IMPLEMENTATION *MA_i*
SETS	REFINES *MA*
AbsSet	VALUES
	$AbsSet = 0 \,..\, 100$

La clause VALUES

- Exemple2

MACHINE Reservation
CONSTANTS
maxi
PROPERTIES
maxi $\in$ **NAT**
VARIABLES

IMPLEMENTATION
Reservation1_i
REFINES
Reservation1_r
VALUES maxi=100

Exercice: Réservation

Soit la machine abstraite suivante:

```
MACHINE Reservation
CONSTANTS
maxi
PROPERTIES
maxi ∈ NAT
VARIABLES
NbPlLib
INVARIANT
NbPlLib ∈ 0..maxi
INITIALISATION
NbPlLib:=maxi
OPERATIONS
reserver= PRE NbPlLib ≠ 0 THEN NbPlLib:=NbPlLib-1 END;
liberer= PRE NbPlLib ≠ maxi THEN NbPlLib:=NbPlLib+1 END
END
```

1. Raffiner cette machine (penser à utiliser les variables **occupes** et **libres** contenant respectivement les places occupés et les places libres)
2. Donner une implémentation basée sur le raffinement de la question précédente

Solution 1/3

```
REFINEMENT
   Reservation_r
REFINES
   Reservation
VARIABLES
    libres, occupes
DEFINITIONS
    SIEGES== 0..maxi-1
INVARIANT
    libres ⊆ SIEGES ∧ occupes ⊆ SIEGES ∧ occupes ∩ libres = {}∧ occupes ∪ libres = SIEGES ∧
NbPlLib=card(libres)
INITIALISATION
    libres:=SIEGES || occupes:={}
OPERATIONS
   reserver = PRE libres ≠ {} THEN
      ANY ss WHERE ss ∈ libres THEN  libres:=libres -{ss}||occupes:=occupes∪{ss}
    END
   END;

   liberer = PRE occupes ≠ {} THEN
       ANY ss WHERE ss ∈ occupes THEN  libres:=libres ∪ {ss}||occupes:=occupes- {ss}
   END
END
END
```

Solution 2/3

```
IMPLEMENTATION
   Reservation1_i
REFINES
   Reservation1_r
CONCRETE_VARIABLES
   occupation
VALUES maxi=100
INVARIANT
    occupation ∈ (0..99) →BOOL ∧ dom(occupation)=libres ∪ occupes
INITIALISATION
   occupation := (0..99)*{FALSE}
OPERATIONS
   reserver = VAR cpt, vv IN
       cpt:=0;vv:=occupation(cpt);
       WHILE cpt < 99 ∧ vv=TRUE DO
           cpt:=cpt+1;
           vv:=occupation(cpt)
       INVARIANT
           cpt ∈ NAT ∧ vv ∈ BOOL  ∧ occupation[0..cpt-1] ⊆{TRUE}
       VARIANT
           maxi-cpt
       END;
       IF vv=FALSE THEN
         occupation (cpt):= TRUE
       END
   END;
```

Solution 3/3

```
   liberer =
       VAR cpt, vv IN
       cpt:=0;vv:=occupation(cpt);
       WHILE cpt <99 ∧ vv=FALSE DO
           cpt:=cpt+1;
           vv:=occupation(cpt)
       INVARIANT
           cpt ∈ NAT ∧ vv ∈ BOOL ∧ occupation[0..cpt-1] ⊆{FALSE}
       VARIANT
           maxi-cpt
       END;
       IF vv=TRUE THEN
          occupation (cpt):= FALSE
       END   END
END
```

Réservation: code généré avec l'AtelierB

```
#ifndef _Reservation1_h
#define _Reservation1_h

#include <stdint.h>
#include <stdbool.h>
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */


/* Clause SETS */

/* Clause CONCRETE_VARIABLES */


/* Clause CONCRETE_CONSTANTS */
/* Basic constants */
#define Reservation1__maxi 100
/* Array and record constants */

extern void Reservation1__INITIALISATION(void);

/* Clause OPERATIONS */

extern void Reservation1__reserver(void);
extern void Reservation1__liberer(void);

#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* _Reservation1_h */
```

Réservation: code généré avec l'AtelierB

```
#include "Reservation1.h"

/* Clause CONCRETE_CONSTANTS */
/* Basic constants */
#define Reservation1__maxi 100
/* Array and record constants */
/* Clause CONCRETE_VARIABLES */

static bool Reservation1__occupation[100];
/* Clause INITIALISATION */
void Reservation1__INITIALISATION(void)
{

    unsigned int i = 0;
    for(i = 0; i < 99;i++)
    {
        Reservation1__occupation[i] = false;
    }
}
/* Clause OPERATIONS */
void Reservation1__reserver(void)
{
    int32_t cpt;
    bool vv;

    cpt = 0;
    vv = Reservation1__occupation[cpt];
    while(((cpt) < (99)) &&
    (vv == true))
    {
        cpt = cpt+1;
        vv = Reservation1__occupation[cpt];
    }
    if(vv == false)
    {
        Reservation1__occupation[cpt] = true;
```

```
void Reservation1__liberer(void)
{
    int32_t cpt;
    bool vv;

    cpt = 0;
    vv = Reservation1__occupation[cpt];
    while(((cpt) < (99)) &&
    (vv == false))
    {
        cpt = cpt+1;
        vv = Reservation1__occupation[cpt];
    }
    if(vv == true)
    {
        Reservation1__occupation[cpt] = false;
    }}
```

Exercice: Maximum

```
MACHINE
    Maximum
VARIABLES
yy
INVARIANT
yy ∈ ℙ(NAT1)
INITIALISATION
yy := {}
OPERATIONS
enter(nn) =
PRE nn ∈ NAT1 THEN
yy := yy ∪ {nn}
END ;
mm <-- maximum =
PRE yy ≠∅ THEN
mm := max(yy)
END
END
```

```
REFINEMENT
   Maximum_r
REFINES
   Maximum
VARIABLES
   zz
INVARIANT
   zz ∈ NAT ∧ zz = max(yy ∪ {0})
INITIALISATION
zz := 0
OPERATIONS
enter(nn) =
PRE nn ∈ NAT1 THEN
zz := max({zz,nn})
END ;
mm <-- maximum = PRE zz ≠ 0
     THEN mm := zz END
END
```

```
IMPLEMENTATION
   Maximum_i
REFINES
   Maximum_r

CONCRETE_VARIABLES
   zz
INITIALISATION
   zz := 0
OPERATIONS
enter(nn) =
IF nn > zz THEN
zz := nn
END ;
mm <--   maximum =
BEGIN mm := zz END
END
```

Remarque: max (E) donne le maximum dans un ensemble E qui doit être non vide

Maximum: code C généré par l'AtelierB

```
#ifndef _Maximum_h
#define _Maximum_h

#include <stdint.h>
#include <stdbool.h>
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */


/* Clause SETS */

/* Clause CONCRETE_VARIABLES */


/* Clause CONCRETE_CONSTANTS */
/* Basic constants */
/* Array and record constants */
extern void Maximum__INITIALISATION(void);

/* Clause OPERATIONS */

extern void Maximum__enter(int32_t nn);
extern void Maximum__maximum(int32_t *mm);

#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* _Maximum_h */
```

Maximum : code C généré par l'atelier B

```
#include "Maximum.h"

/* Clause CONCRETE_CONSTANTS */
/* Basic constants */

/* Array and record constants */
/* Clause CONCRETE_VARIABLES */

static int32_t Maximum__zz;
/* Clause INITIALISATION */
void Maximum__INITIALISATION(void)
{

    Maximum__zz = 0;
}

/* Clause OPERATIONS */

void Maximum__enter(int32_t nn)
{
    if((nn) > (Maximum__zz))
    {
        Maximum__zz = nn;
    }
}

void Maximum__maximum(int32_t *mm)
{
    (*mm) = Maximum__zz;
}
```

Exercice: Carrefour

```
MACHINE carrefour
SETS
COULEUR = {rouge, jaune, vert}
CONSTANTS Suiv
PROPERTIES
Suiv ∈ COULEUR → COULEUR ∧
Suiv(rouge) = vert ∧ Suiv(vert) = jaune ∧ Suiv(jaune) = rouge
VARIABLES
feuA, feuB
DEFINITIONS
hs == feuA = jaune ∧ feuB = jaune;
service(aa,bb) == (aa = rouge ∧ bb ≠ rouge) or (aa ≠ rouge ∧ bb = rouge);
es == service(feuA, feuB)
INVARIANT
feuA, feuB ∈ COULEUR * COULEUR ∧ (hs or es)
INITIALISATION
feuA, feuB := jaune, jaune
OPERATIONS
Mise_en_service = PRE hs
THEN ANY fa, fb WHERE
fa ∈ COULEUR ∧ fb ∈ COULEUR ∧ (service(fa,fb))
THEN
feuA := fa||feuB := fb
END
END;
Mise_hors_service = PRE es
THEN
feuA, feuB := jaune, jaune
END;
Changer_feu = PRE es THEN
ANY fa, fb WHERE fa ∈ COULEUR ∧ fb ∈ COULEUR ∧
((fa = feuA ∧ fb = Suiv(feuB)) or
(fa = Suiv(feuA) ∧ fb = feuB) or
(fa = Suiv(feuA) ∧ fb = Suiv(feuB)) ∧
(service(fa,fb)))
THEN feuA, feuB := fa, fb
END END END
```

Solution 1/2

```
MACHINE
    carrefour
SETS
COULEUR = {rouge, jaune, vert}
ABSTRACT_CONSTANTS
Suiv
PROPERTIES
Suiv ∈ COULEUR → COULEUR ∧
Suiv(rouge) = vert ∧
Suiv(vert) = jaune ∧
Suiv(jaune) = rouge
VARIABLES
feuA, feuB
DEFINITIONS
hs == feuA = jaune ∧ feuB = jaune;
service(aa,bb) == (aa = rouge ∧ bb ≠ rouge) or (aa ≠ rouge ∧ bb = rouge);
es == service(feuA, feuB)
INVARIANT
feuA, feuB ∈ COULEUR * COULEUR ∧ (hs or es)
INITIALISATION
feuA, feuB := jaune, jaune
OPERATIONS
Mise_en_service = PRE hs
THEN ANY fa, fb WHERE
fa ∈ COULEUR ∧ fb ∈ COULEUR ∧ (service(fa,fb))
THEN
```

Solution 2/2

```
IMPLEMENTATION
   carrefour_i
REFINES
   carrefour
CONCRETE_VARIABLES
   feuA,feuB

INITIALISATION
   feuA := jaune; feuB  := jaune
OPERATIONS
   Mise_en_service =
   IF feuA=jaune ∧ feuB=jaune THEN
   feuA := rouge ; feuB := vert
   END;
   Mise_hors_service =
   BEGIN
   feuA := jaune; feuB  := jaune
   END;
   Changer_feu =
   IF feuA ≠ jaune ∧ feuB ≠ jaune THEN
         IF feuA = vert ∧ feuB= rouge THEN feuA := jaune
          ELSE IF feuA = jaune ∧ feuB = rouge
            THEN
            feuA := rouge ;
            feuB := vert
       ELSE IF feuA = rouge ∧ feuB = vert
       THEN feuB := jaune
       ELSE IF feuA = rouge ∧ feuB = jaune
       THEN feuA := vert ;
           feuB := rouge
       END
   END END
END
END

END
```

Carrefour: code C

```
#ifndef _carrefour_h
#define _carrefour_h

#include <stdint.h>
#include <stdbool.h>
#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */


/* Clause SETS */
typedef enum
{
    carrefour__rouge,
    carrefour__jaune,
    carrefour__vert

} carrefour__COULEUR;

/* Clause CONCRETE_VARIABLES */


/* Clause CONCRETE_CONSTANTS */
/* Basic constants */
/* Array and record constants */
extern void carrefour__INITIALISATION(void);

/* Clause OPERATIONS */

extern void carrefour__Mise_en_service(void);
extern void carrefour__Mise_hors_service(void);
extern void carrefour__Changer_feu(void);

#ifdef __cplusplus
}
#endif /* __cplusplus */


#endif /* _carrefour_h */
```

Carrefour: code C

```c
#include "carrefour.h"

/* Clause CONCRETE_CONSTANTS */
/* Basic constants */

/* Array and record constants */
/* Clause CONCRETE_VARIABLES */

static carrefour__COULEUR carrefour__feuA;
static carrefour__COULEUR carrefour__feuB;
/* Clause INITIALISATION */
void carrefour__INITIALISATION(void)
{

    carrefour__feuA = carrefour__jaune;
    carrefour__feuB = carrefour__jaune;
}

/* Clause OPERATIONS */

void carrefour__Mise_en_service(void)
{
    if((carrefour__feuA == carrefour__jaune) &&
    (carrefour__feuB == carrefour__jaune))
    {
        carrefour__feuA = carrefour__rouge;
        carrefour__feuB = carrefour__vert;
    }
}

void carrefour__Mise_hors_service(void)
{
    carrefour__feuA = carrefour__jaune;
    carrefour__feuB = carrefour__jaune;
}
```

```c
void carrefour__Changer_feu(void)
{
    if(((carrefour__feuA) != (carrefour__jaune)) &&
    ((carrefour__feuB) != (carrefour__jaune)))
    {
        if((carrefour__feuA == carrefour__vert) &&
        (carrefour__feuB == carrefour__rouge))
        {
            carrefour__feuA = carrefour__jaune;
        }
        else
        {
            if((carrefour__feuA == carrefour__jaune) &&
            (carrefour__feuB == carrefour__rouge))
            {
                carrefour__feuA = carrefour__rouge;
                carrefour__feuB = carrefour__vert;
            }
            else
            {
                if((carrefour__feuA == carrefour__rouge) &&
                (carrefour__feuB == carrefour__vert))
                {
                    carrefour__feuB = carrefour__jaune;
                }
                else
                {
                    if((carrefour__feuA == carrefour__rouge) &&
                    (carrefour__feuB == carrefour__jaune))
                    {
                        carrefour__feuA = carrefour__vert;
                        carrefour__feuB = carrefour__rouge;
                    }
                }
            }
        }
    }
}
```

Les obligations de preuve du raffinement

- Les obligations de preuve principales relatives à un REFINEMENT concernent :
  1. **la correction de l'initialisation**
  2. **la correction des opérations**
- **Pour les IMPLEMENTATION,** s'ajoutent les obligations de preuve des valuations (clause VALUES)

Les obligations de preuve du raffinement

Soit la machine Ma et son raffinement Mr :

MACHINE Ma

...

INVARIANT

I

INITIALISATION

INIT

OPERATIONS

P|S

END

REFINEMENT Mr

REFINES Ma

...

INVARIANT

IR

INITIALISATION

INITR

OPERATIONS

PR|SR

END

Les obligations de preuve du raffinement de l'initialisation

Les hypothèse sont:

- Contraintes de la machine abstraite,
- Propriétés des constantes du développement vertical,

Sous ces hypothèses, le but à démontrer est :

- L'initialisation de **Mr** doit établir qu'il est impossible que l'initialisation de **Ma** établisse la négation du changement de variable:

  [INITR] ¬ [INIT] ¬ IR

- **Exemple: Maximum_r**

1. [zz := 0] ¬ [yy := Φ] ¬ (zz ∈ NAT ∧zz = max (yy ∪ {0}))
2. [zz := 0] (zz ∈ NAT ∧ zz = max (Φ ∪ {0}))
3. 0 ∈ NAT ∧ 0 = 0

Exercice d'application

```
MACHINE
    OP01
VARIABLES
    v1
INVARIANT
    v1 ∈ 0..10
INITIALISATION
    ANY valeur WHERE
        valeur ∈ 1..5
    THEN
        v1 := valeur
    END
END
```

```
REFINEMENT
    OP01_1
REFINES
    OP01
VARIABLES
    v2
INVARIANT
    v2 = 2 * v1
INITIALISATION
    v2 := 2
END
```

Calculer l'obligation de preuve de l'initialisation du raffinement?

- La contraposée de l'invariant est :

$$v2 \neq 2 * v1$$

- L'initialisation du composant raffiné, appliquée a ce prédicat:

$$[\text{ANY valeur WHERE valeur} \in 1..5 \text{ THEN v1} := \text{valeur END}](v2 \neq 2 * v1)$$

- Ce qui devient, par définition de la substitution ANY :

$$\forall \text{valeur}.(\text{valeur} \in 1..5 \Rightarrow v2 \neq 2 * \text{valeur})$$

- La contraposée de ce dernier prédicat est:

$$\exists \text{valeur} . (\text{valeur} \in 1..5 \wedge v2 = 2 * \text{valeur})$$

- L'application de l'initialisation du raffinement nous permet d'instancier v2 par 2, nous obtenons alors:

$$\exists \text{valeur} . (\text{valeur} \in 1..5 \wedge 2 = 2 * \text{valeur})$$

Obligations de preuve de raffinement d'une opération

Contraintes des paramètres de la machine $\wedge$

Propriétés des constantes du développement vertical $\wedge$

Assertions $\wedge$

Invariant de la spécification $\wedge$

Invariant du raffinement $\wedge$

Pré-condition de la spécification $\wedge$

$\Rightarrow$

Pré-condition du raffinement $\wedge$

[Action du raffinement]

$\neg$ ***[Action de la spécification]***

$\neg$ ***Changement de variable du raffinement***

$$\mathbf{I \wedge IR \wedge P => PR \wedge [SR] \neg [S] \neg IR}$$

Obligations de preuve de raffinement d'une opération

- ***Si** : les valeurs des variables des deux composants (abstrait et raffiné) avant l'opération respectent les invariants **I** et **IR***
  - ***Et si** : on est dans les conditions **P** d'exécution de l'opération abstraite*
  - ***Alors :***
    - *On doit être dans les conditions **PR** d'exécution de l'opération raffinée*
    - *Quelque soit les nouvelles valeurs prises par les variables parmi celles définies par la substitution **SR** de la machine raffinée, elles doivent correspondre par la transformation **IR** à l'une des valeurs définies par la substitution **S** de la machine abstraite*

$$\mathbf{I \wedge IR \wedge P \Rightarrow PR \wedge [SR]\neg[S]\neg IR}$$

Exemple1 : Obligation de preuve de raffinement de entrer(nn) - *Exercice Maximum*

1. $yy \in P(NAT1) \wedge zz \in NAT \wedge zz = max\ (yy \cup \{0\}) \wedge nn \in NAT1$

   $\Rightarrow$

   $nn \in NAT1 \wedge [zz := max\ (\{zz, nn\})] \neg [yy := yy \cup \{nn\}]$

   $\neg\ (zz \in NAT \wedge zz = max\ (yy \cup \{0\}))$

2. $yy \in P(NAT1) \wedge zz \in NAT \wedge zz = max\ (yy \cup \{0\}) \wedge nn \in NAT1$

   $\Rightarrow$

   $nn \in NAT1 \wedge [zz := max\ (\{zz, nn\})]$

   $(zz \in NAT \wedge zz = max\ (yy \cup \{nn\} \cup \{0\})$

3. $yy \in P(NAT1) \wedge zz \in NAT \wedge zz = max\ (yy \cup \{0\}) \wedge nn \in NAT1$

   $\Rightarrow$

   $nn \in NAT1 \wedge max\ (\{zz, nn\}) \in NAT \wedge (max\ (\{zz, nn\}) = max\ (yy \cup \{nn\} \cup \{0\}))$

Exemple2 : Obligation de preuve de raffinement de mm←maximum - *Exercice Maximum*

Cas d'une opération avec paramètre de sortie: Effectuer un changement de variable, l'obligation de preuve devient:
$I \wedge IR \wedge P \Rightarrow PR \wedge [[mm := mm']SR] \neg [S] \neg (IR \wedge mm = mm')$

Application à l'opération maximum

1. $yy \in P(NAT1) \wedge zz = \max (yy \cup \{0\}) \wedge yy \neq \Phi$

   $\Rightarrow$

   $zz \neq 0 \wedge [mm' := zz] \neg [mm := \max (y)] \neg (zz = \max (yy \cup \{0\}) \wedge mm = mm')$
2. $yy \in P(NAT1) \wedge zz = \max (yy \cup \{0\}) \wedge y \neq \Phi$

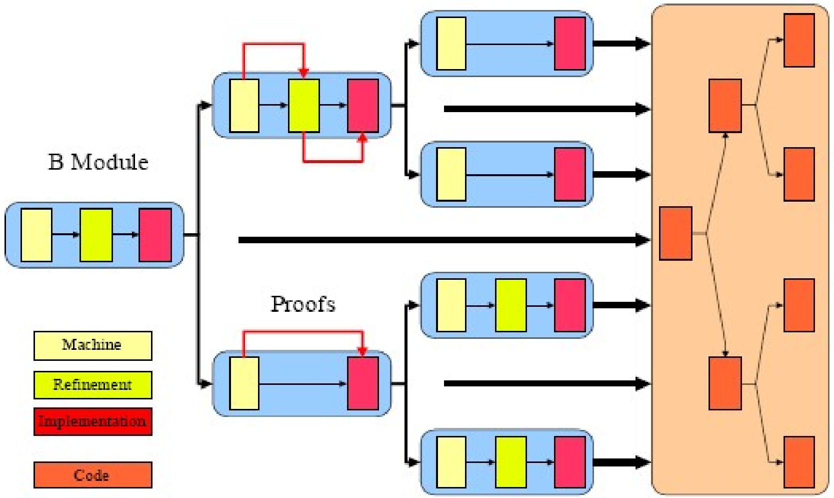
   $\Rightarrow$

   $zz \neq 0 \wedge [mm' := zz] (zz = \max (yy \cup \{0\}) \wedge \max (yy) = mm')$
3. $yy \in P(NAT1) \wedge zz = \max (yy \cup \{0\}) \wedge yy \neq \Phi$

   $\Rightarrow$

   $zz \neq 0 \wedge (zz = \max (yy \cup \{0\}) \wedge \max (yy) = zz)$

La modularité dans la méthode B



La modularité

- Dans la conception de logiciels de taille importante, la décomposition en plusieurs machines s'impose.
- Un développement complet en B se déroule dans le cadre d'un **projet B**.
- La construction d'un projet se fait à l'aide du développement de **modules** reliés par des **liens** qui doivent respecter certaines règles.
- Un module B permet de modéliser un sous système:
  - Possède toujours une machine abstraite
  - Peut posséder un ou plusieurs raffinements
  - Peut posséder une implémentation

→ Ce qui permet un développement progressif de la spécification et la séparation des preuves pour chaque machine.

La modularité

- Les clauses qu'on pourra utiliser pour établir différents types de liens:
  - SEES,
  - USES,
  - INCLUDES,
  - EXTENDS,
  - IMPORTS,
  - PROMOTES.

Résumé

- Hiérarchisation
  - Les clauses INCLUDES (machine et raffinement) et IMPORTS (implémentation) servent à partager les fonctionnalités (les opérations) et les données définies dans un autre module B.
  - PROMOTES: promouvoir des opérations d'instances de machines incluses (machine et raffinement) ou importées (implémentation).
  - EXTENDS: IMPORTS+PROMOTES (implémentation) ou INCLUDES+PROMOTES (machine et raffinement)
- Partage
  - Les liens SEES et USES sont des liens transversaux dans l'arborescence des dépendances d'un projet B pour le partage.

Références

[1] J.-R. Abrial, “The B book”, Cambridge University Press, 1996.

[2] "MANUEL DE REFERENCE DU LANGAGE B", Version 1.8.8, 13 Mai 2009

[3] « AtelierB-MANUEL DE REFERENCE-Obligations de preuve", version 3.7.

[4] Jacques Julliand, cours «Introduction à la Méthode B », Université de Franche-Comté, 2007.

[5] J. Christian Attiogbé, cours « Construction formelle de logiciels, La méthode B », Université de Nantes, 2010.

Fin chapitre2

Type concret / Nature	Ens. abstrait ou énuméré	Entier	Booléen	Élément d'ens. abstrait ou énuméré	Intervalle d'entiers ou sous ensemble d'ens. abstrait	Tableau	Record	Chaîne de caractères
Paramètre de machine (ensemble)	X				X			
Ensemble abstrait ou énuméré	X							
Paramètre de machine (scalaire)		X	X	X				
Énuméré littéral				X				
Constante concrète		X	X	X	X	X	X	
Variable concrète		X	X	X		X	X	
Paramètre d'entrée d'opération (non locale ou locale)		X	X	X		X	X	X
Paramètre de sortie d'opération (non locale ou locale)		X	X	X		X	X	
Variable locale		X	X	X		X	X	

Substitution gardée

A ne pas confondre avec la pré-condition

- Substitution pré-condition: PRE P THEN S END

  Sémantique

  $P \wedge [S]I$

- Substitution gardée: IF P THEN S ELSE T END

  Sémantique

  $(P \Rightarrow [S]I) \wedge (\neg P \Rightarrow [T]I)$

- La substitution pré-condition n'est utilisable que si le prédicat est valide alors que la substitution gardée est toujours réalisée mais son résultat dépend de la validité d'un prédicat.

Exemple: Distributeur de boissons

On souhaite spécifier le fonctionnement d'une machine qui délivre des boissons.
Le fonctionnement est décrit comme suit :
La machine peut être en arrêt ou en marche,
Il y a un bouton pour mettre en marche et arrêter la machine

Les actions suivantes n'ont d'effet que si la machine est en marche

- L'usager peut sélectionner une certaine boisson. Il y en a 3 : café, thé ou chocolat.
- Après la sélection, l'appareil affiche la somme demandée.
- L'usager peut alors payer avec les différentes pièces de monnaie.
- L'appareil affiche la somme restante.
- Dès qu'une somme suffisante est versée, l'appareil délivre la boisson à l'usager.
- L'appareil rend éventuellement la monnaie.
- On ne peut commander une nouvelle boisson que si le Goblet a été retiré de son emplacement.
- A tout moment après la sélection de la boisson et avant que le distributeur ne délivre la boisson, l'usager peut annuler la commande. L'appareil doit alors rendre l'argent déjà versé s'il y a lieu

Exemple : Application à la modélisation d'un ascenseur

- On souhaite spécifier le fonctionnement simplifié d'un ascenseur.
- une porte à chaque étage
- l'appel intérieur et l'appel extérieur ne sont pas distingués
- il n'y a pas de panne
- une constante donne le nombre d'étages : max_etage (> 0)
- Les opérations sont :
  - ouvrir, fermer une porte,
  - appeler l'ascenseur,
  - déplacement de l'ascenseur
- l'ascenseur reste dans la limite des étages
- si une porte est ouverte l'ascenseur est arrêté à l'étage correspondant
- chaque appel est traité en un temps raisonnable
- si l'ascenseur est arrêté à un étage, l'appel à cet étage est considéré comme traité
- . . .

Exemple : Application à la modélisation d’un ascenseur

MACHINE $ASCENSEUR$

SETS $MODE = \{arret, mouv\}$

CONSTANTS $max\_etage, ETAGES$

PROPERTIES $max\_etage \in \mathsf{NAT}_1 \wedge ETAGES = 0..max\_etage$

VARIABLES $appels, ouvertes, pos, mode$

INVARIANT

$$ouvertes \subseteq ETAGES \ \wedge \ appels \subseteq ETAGES$$
$$\wedge \ pos \in ETAGES \ \wedge \ mode \in MODE$$
$$\wedge \ (ouvertes \neq \emptyset \Rightarrow ouvertes = \{pos\} \wedge mode = arret)$$
$$\wedge \ (mode = arret \Rightarrow pos \notin appels)$$

Exemple : Distributeur de carnets de timbres

MACHINE
  DISTRIBUTEUR
CONSTANTS
  Prix ,
  Valeur_piece
PROPERTIES
  $Prix \in NAT \wedge$
  $Valeur\_piece \in NAT$
ABSTRACT_VARIABLES
  Somme_versee,
  Nbs_demandes,
  Nbs_disponibles

INVARIANT
  $Somme\_versee \in NAT \wedge$
  $Nbs\_demandes \in NAT \wedge$
  $Nbs\_disponibles \in NAT \wedge$
  $Nbs\_demandes \leq$
   $Nbs\_disponibles \wedge$
  $Somme\_versee \leq$
   $(Nbs\_demandes \times Prix)$
INITIALISATION
  BEGIN
  $Somme\_versee := 0 \;||$
  $Nbs\_demandes := 0 \;||$
  $Nbs\_disponibles :\in NAT$
  END

Exemple : Distributeur de carnets de timbres

OPERATIONS

Indiquer_nbres $(nbc) =$

PRE $nbc \in \textbf{NAT1} \wedge nbc < Nbs\_disponibles \wedge$
$Somme\_versee = 0 \wedge Nbs\_demandes = 0$

THEN

$Nbs\_demandes := nbc$

END ;

Introduire_piece $=$

IF $Nbs\_demandes > 0 \wedge$
$Somme\_versee + Valeur\_piece \leq Nbs\_demandes \times Prix$

THEN

$Somme\_versee := Somme\_versee + Valeur\_piece$

END ;

Exemple : Distributeur de carnets de timbres

```
Delivrers =
PRE Somme_versee = Nbs_demandes × Prix
THEN
   Somme_versee := 0 ||
   Nbs_disponibles := Nbs_disponibles - Nbs_demandes ||
   Nbs_demandes := 0
END ;

Annuler =
BEGIN
   Somme_versee := 0 ||
   Nbs_demandes := 0
END
END
```

Exemple : Le contrôle d'accès aux bâtiments

- **P1.** Le modèle comprend des personnes et des bâtiments
- **P2.** Chaque personne est autorisée à pénétrer dans certains bâtiments (et pas dans d'autres). Les bâtiments non consignés dans cette autorisation sont implicitement interdits. Il s'agit d'une affectation permanente.
- **P3.** A un instant donné, une personne se trouve dans un bâtiment au plus.
- **D1.** Le système gère le passage des personnes d'un bâtiment à l'autre.
- **P4.** A un instant donné, une personne se trouve dans un bâtiment au moins.
- **P5.** Toute personne se trouvant dans un bâtiment est bien autorisée à y être.

Exemple : Le contrôle d'accès aux bâtiments

- (les personnes et les bâtiments) sont représentés par des ensembles abstraits,
- les autorisations *aut* d'accès des personnes aux bâtiments ainsi que les communications *com* entre les bâtiments sont représentées par des relations constantes,
  - c'est à dire par des ensembles de couples, dont les types sont contraints dans la clause INVARIANT.
- La situation dynamique *sit* des personnes dans les bâtiments est représentée par une fonction totale.

Exemple: Le contrôle d'accès aux bâtiments

Outre l’information sur les types,

- l’invariant affirme qu’à chaque état,
  - les personnes ne peuvent être que dans les bâtiments où elles sont autorisées d’entrer
  - *com* est irreflexive (aucun bâtiment ne communique avec lui-même).

- Le modèle présente une seule opération notée *pass* qui représente l’entrée d’une personne

  p dans un bâtiment *b* à condition que *p* soit autorisée d’entrer dans *b* et que *b* communique avec le bâtiment où se trouve *p*.

Exemple : Le contrôle d'accès aux bâtiments (specification initiale, abstraite)

MACHINE Batiment

SETS

$BAT; PERS$

CONSTANTS

aut, com

VARIABLES

sit

INVARIANT

$aut \in PERS \leftrightarrow BAT \wedge com \in BAT \leftrightarrow BAT \wedge$

$sit \in PERS \rightarrow BAT \wedge sit \subseteq aut \wedge com \cap \mathrm{id}(BAT) = \{\}$

OPERATIONS

$pass \mathrel{\widehat{=}}$ ANY p, b

WHERE $(p, b) \in aut \wedge (sit(p), b) \in com$

THEN $sit(p) := b$

END

END

II.6.7. Ensembles, fonctions, relations

MACHINE ASCENSEUR
SETS
    ASC ; DIR = {mo, de}
CONSTANT
    rdc, haut
PROPERTIES
    bas $\in$ INT $\wedge$ haut $\in$ INT $\wedge$ bas < haut
DEFINITIONS
    ETG == bas .. haut ; inactif == ASC - actif ;
VARIABLES
    actif, etage, direction, entrees, sorties
INVARIANT
    actif $\subseteq$ ASC $\wedge$
    etage $\in$ ASC $\longrightarrow$ ETG $\wedge$ dir $\in$ ASC $\longrightarrow$ DIR $\wedge$
    entrees $\in$ ETG $\leftrightarrow$ DIR $\wedge$ sortie $\in$ ASC $\leftrightarrow$ETG
…
INITIALISATION
actif, entrees, sorties := Φ, Φ, Φ | |
etage, dir := ASC x {bas}, ASC x {mo}

II.6.8. Tableau

```
MACHINE  TABLEAU (INDEX, VALEUR)
VARIABLES
   tab
INVARIANT
   tab ∈ INDEX —› VALEUR
OPERATIONS
   changer (v, i) =
       PRE
         v ∈ VALEUR ∧ i ∈ INDEX
       THEN
         tab (i) := v
       END ;
v ‹ — valeur (i) = … ;
i ‹ — chercher (v)  =
PRE v ∈ VALEUR   ∧
tab -1 [{v}] ≠ Φ
THEN  i :∈ tab -1 [{v}]
END ;
b, i ‹ — rechercher (v)  =
PRE  v ∈ VALEUR
THEN   IF tab -1 [{v}] ≠ Φ THEN
          b := true || i :∈ tab -1 [{v}]
ELSE  b := false || i :∈ INDEX
END
END ;
END
```

II.6.8. Tableau

```
b ← est_present (v) =
PRE
  v ∈ VALEUR
THEN
  b := bool (tab -1 [{v}] ≠ Φ)
END ;
i ← chercher (v) =
PRE
  v ∈ VALEUR ∧ tab -1 [{v}] ≠ Φ
THEN
  i :∈ tab -1 [{v}]
END ;
```

```
b, i ← rechercher (v) =
PRE
  v ∈ VALEUR
THEN
  IF tab -1 [{v}] ≠ Φ
  THEN
    b := true
    i :∈ tab -1 [{v}]
  ELSE
    b := false
    i :∈ INDEX
  END
END ;
END
```

Les clauses d’une machine abstraite
La clause DEFINITIONS

Exemple2

```
MACHINE      MA
DEFINITIONS
"commun1.def"  ;
<commun2.def>  ;
debut  ==  -2 × UNIT  ;
fin  ==  10 × UNIT SEES ... END
```

commun1.def

```
DEFINITIONS
UNIT == 16
```

commun2.def

```
DEFINITIONS
T == TRUE  ;
F == FALSE
```

- La liste des définitions du composant *MA* comprend les définitions explicites *debut* et *fin* ainsi que les définitions des fichiers *commun1.def* et *commun2.def*.
- Le fichier *commun1.def* (entre guillemet) est cherché à partir du répertoire local,
- Le fichier commun2.def (entre chevron) est cherché à partir de l'un des répertoires de fichiers inclus.

EXERCICE: Contrôleur d'une barrière d'un passage à niveau

- On souhaite spécifier le comportement d'un système de contrôle d'une barrière d'un passage à niveau en utilisant la méthode B.
- La barrière est initialement relevée. Si un train arrive alors elle est baissée et elle reste dans cet état tant qu'il ya des trains qui arrivent. Elle est relevée si le dernier train quitte la section.

1. Spécifier les données du modèle B associé à ce système
2. Décrire les invariants de ce modèle
3. Donner le modèle résultant
4. Calculer les obligations de preuve

- **Remarque :** Il est possible de commencer par construire un automate associé à ce contrôleur et, par la suite, donner le modèle résultant.