La Méthode B

Frédéric Gervais

Université Paris-Est, LACL

Séminaire francilien de sûreté de fonctionnement
6 février 2015

Méthode B

- Méthode de conception formelle
- Cycle complet :
  - Spécification
  - Raffinement
  - Implémentation
- Prérequis : cahier des charges, description des besoins
- Vérification et validation

Méthode B

- Raffinement :
  - Spécification : ce que fait le logiciel
  - Programme : comment le logiciel fait
  - Raffinement : passage du quoi au comment
- Outils :
  - Atelier B
  - ProB
  - Rodin (Event-B)

Jean-Raymond Abrial

- Auteur d’un des premiers SGBD réseau (Socrate 1968 à 1974 à Grenoble)
- Auteur d'un article à l’origine des modèles de données et orientés objet (1974)
- Un des auteurs du langage ADA (1978-79, rendez-vous et concurrence)
- Auteur de la notation Z (1974 à 1980 chez EDF puis au sein du PRG d’Oxford, alors dirigé par Tony Hoare)
- Créateur de la méthode B (années 90)

Applications de B

- Domaine ferroviaire :
  - SNCF KVB : système de contrôle de vitesse à balises (Alstom)

    60000 lignes de B, 10000 preuves, 22000 lignes ADA
  - RATP Météor : système de pilotage ligne 14 du métro (Siemens Transportation Systems)

    107000 lignes de B, 29000 preuves, 87000 lignes ADA
  - Roissy VAL : système de pilotage (Siemens)

    183000 lignes de B, 43000 preuves, 158000 lignes ADA
  - Autres projets : automatisation ligne 1 du métro, métro de New York, etc.

Applications de B

- Autres domaines :
  - Automobile (électronique, PSA)
  - Aéronautique (séparation fusée, EADS)
  - Banque (cas d'étude DAB)
  - Industrie (conditions travail, INRS)
  - Nucléaire (centrale électrique, EDF)

Langage B

- Composants : machine abstraite, raffinement, implémentation
- Langage basé sur :
  - Théorie des ensembles
  - Logique du premier ordre
  - Langage de substitutions généralisées

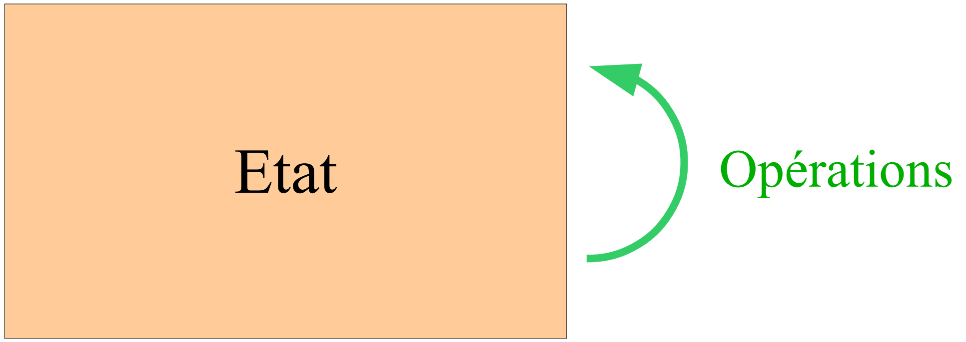
Exemple : gestion ressources

```
MACHINE            ResourceManager
SET                RES
VARIABLES          res_libre
INVARIANT          res_libre ⊆ RES
INITIALISATION     res_libre := ∅
OPERATIONS
    alloc (r) ≙
    PRE    r ∈ res_libre
    THEN   res_libre := res_libre - {r}
    END;

    free (r) ≙
    PRE    r ∈ RES - res_libre
    THEN   res_libre := res_libre ∪ {r}
    END;

    setres (set_r) ≙
    PRE    set_r ⊆ RES
    THEN   res_libre := set_r
    END;
END
```

Machine abstraite



Machine abstraite

Machine	*Nom(param)*
Sets	*S*
Variables	*V*
Invariant	*INV*
Initialisation	*Init*
Operations	
Op = $P \mid S$	
End	

Machine abstraite

- Partie statique :
  - Ensembles
  - Variables d'état
  - Propriétés d'invariance

Machine	*Nom(param)*
Sets	*S*
Variables	*V*
Invariant	*INV*
Initialisation	*Init*
Operations	
Op = $P \mid S$	
End	

Machine abstraite

- Partie statique :
  - Ensembles
  - Variables d'état
  - Propriétés d'invariance
- Partie dynamique :
  - Initialisation
  - Opérations

Machine	*Nom(param)*
Sets	*S*
Variables	*V*
Invariant	*INV*
Initialisation	*Init*
Operations	
Op = $P \mid S$	
End	

Clauses d'une machine abstraite

- **MACHINE** NomMachine(liste de paramètres)
- **CONSTRAINTS** Propriétés des paramètres
- **SETS** Liste des ensembles abstraits et énumérés
- **CONSTANTS** Liste des constantes
- **PROPERTIES** Propriétés des constantes et ensembles
- **VARIABLES** Liste des variables d'état
- **INVARIANT** Typage et propriétés des variables
- **DEFINITIONS** Liste des définitions
- **INITIALISATION** Initialisation des variables
- **OPERATIONS** Liste des opérations

Langage pour la statique

- Langage de description des données et de leurs propriétés
- Théorie des ensembles : ensembles, relations, fonctions, séquences, etc ...
- Logique des prédicats du 1er ordre avec égalité

Notations logiques de base et types de base

- Opérateurs booléens : $\vee \; \neg \; \wedge \; \Rightarrow \; \Leftrightarrow$
- Quantificateurs : $\exists \;\; x \bullet P$ et $\forall \;\; x \bullet P$
- Paires ordonnées : $x \mapsto y$ (ou x,y) *avec* x *et* y *deux expressions quelconques*
- Prédicat d'égalité : =
- Types de base : $\mathbb{N}$, $\mathbb{Z}$, BOOL, STRING

  avec les opérateurs usuels

Ensembles

- Permet de définir toute **collection d'objets non ordonnés et sans double**
- Prédicat fondamental : $\mathbf{x \in S}$
- Définition :

  - en extension : $\{a,e,i,o,u,y\}$

  - en compréhension : $\{x \mid P\}$ ou $\{x \in S \mid P\}$

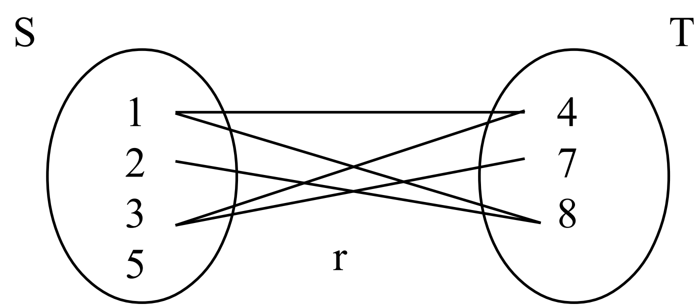
  - ensemble vide : $\{\}$ ou $\emptyset$

Opérateurs ensemblistes

appartenance :	$x \in S$
union :	$S \cup T$
intersection :	$S \cap T$
différence :	$S - T$
inclusion :	$S \subseteq T$
inclusion stricte :	$S \subset T$
égalité :	$S = T$
cardinal :	**card(S)** : nbre d'éléments de S si S est fini
produit cartésien :	$\mathbf{S \times T} = \{x \mapsto y \mid x \in S \wedge y \in T\}$
ensemble des parties :	$\mathbb{P}\mathbf{(S)} = \{x \mid x \subseteq S\}$
choix :	**Choix S** : renvoie un élément arbitraire de S, si S non vide
union distribuée :	$\bigcup S$ — $\bigcup S = \{x \mid \exists y \in S \,.\, x \in y\}$ avec S un ensemble d'ensembles,
intersection distribuée :	$\bigcap S$ — $\bigcap S = \{x \mid \forall y \in S \,.\, x \in y\}$ avec S un ensemble *non vide* d'ensembles

Relations

- Permet de modéliser toutes les notions d'associations, correspondances, … très fréquentes en informatique.
- 1 relation = 1 ensemble de paires
- Notation : $\mathbf{S \leftrightarrow T = \mathbb{P}\ (S \times T)}$ *ensemble des relations entre S et T*



$$r \in S \leftrightarrow T\ ,\quad r = \{1 \mapsto 4,\ 1 \mapsto 8,\ 2 \mapsto 8,\ 3 \mapsto 4,\ 3 \mapsto 7\}$$

- Comme une relation est un ensemble, on peut lui appliquer tous les opérateurs ensemblistes

Opérateurs relationnels

- **Inverse** : $\mathbf{r^{-1}} = \{x \mapsto y \mid y \mapsto x \in r\}$

  $r \in S \leftrightarrow T \Leftrightarrow r^{-1} \in T \leftrightarrow S$
- **Composition :** $p \in S \leftrightarrow T \wedge q \in T \leftrightarrow U \Rightarrow \mathbf{p\,;q \in S \leftrightarrow U}$

  $p\,;q = \{x \mapsto z \mid \exists\, y \in T \,.\, x \mapsto y \in p \wedge y \mapsto z \in q\}$
- **Domaine :** $\mathbf{dom(r)} = \{x \mid x \in S \wedge \exists\, y \in T \,.\, x \mapsto y \in r\}$ $(r \in S \leftrightarrow T)$
- **Codomaine :** $\mathbf{ran(r)} = \{y \mid y \in T \wedge \exists\, x \in S \,.\, x \mapsto y \in r\}$ $(r \in S \leftrightarrow T)$
- **Identité :** $\mathbf{id(S)} = \{x \mapsto x \mid x \in S\}$
- **Fermeture réflexive et transitive : r^*** $(r \in S \leftrightarrow S)$

  $r^* = id(S) \cup r \cup r^2 \cup \ldots \cup r^i \ \ldots = \bigcup n \,.\, (n \in \mathbb{N} \mid r^n)$
- **Fermeture transitive : r^+** $(r \in S \leftrightarrow S)$

  $r^+ = r \cup r^2 \cup \ldots \cup r^i \ \ldots = \bigcup n \,.\, (n \in \mathbb{N}_1 \mid r^n)$

- **Restrictions :** $r \in T$, $U \subseteq S$, $V \subseteq T$

  restriction : $U \lhd r = \{x \mapsto y \mid (x \mapsto y) \in r \wedge x \in U\}$

  corestriction : $r \rhd V = \{x \mapsto y \mid (x \mapsto y) \in r \wedge y \in V\}$

  anti-restriction : $U ⩤ r = \{x \mapsto y \mid (x \mapsto y) \in r \wedge x \notin U\}$

  anti-corestriction : $r ⩥ V = \{x \mapsto y \mid (x \mapsto y) \in r \wedge y \notin V\}$

- **Image :** $r \in S \leftrightarrow T$, $U \subseteq S$

  $\mathbf{r[U]} = \{y \mid y \in T \wedge \exists x \in U \,.\, x \mapsto y \in r)\}$

- **Superposition :** $f \in S \leftrightarrow T$, $g \in S \leftrightarrow T$

  $\mathbf{f} \Lsh\!\!+ \mathbf{g} = (\,\mathrm{dom}(g) ⩤ f) \cup g$

  $= \{x,y \mid (x,y) \in S \times T \wedge (((x \mapsto y) \in f \wedge x \notin \mathrm{dom}(g)) \vee (x \mapsto y) \in g)\}$

- **Produit direct** : $f \in S \leftrightarrow U$, $g \in S \leftrightarrow V$

  $\mathbf{f \otimes g} = \{x,(y,z) \mid (x,(y,z)) \in S \times (U \times V) \wedge (x \mapsto y) \in f \wedge (x \mapsto z) \in g\}$

  $f \otimes g \in S \leftrightarrow U \times V$

Fonctions

- Une **fonction** est une relation telle que **chaque élément a une image au maximum**
- Notation : $\mathbf{S \mathrel{+\!\!\!\rightarrow} T}$ *l'ensemble des fonctions partielles de S dans T*

  $$S \mathrel{+\!\!\!\rightarrow} T = \{f \mid f \in S \leftrightarrow T \wedge \forall\ x \in S, (y,z) \in T \times T\ .\ ((x \mapsto y) \in f \wedge (x \mapsto z) \in f \Rightarrow y = z)\}$$

- Une fonction est une relation donc un ensemble ; tous les opérateurs ensemblistes et relationnels s'appliquent sur une fonction.
- Opérateur particulier : **application**

  $$\mathbf{f(x) = choix\ f[\{x\}]}$$

  Cela signifie que :

  - *si* $f[\{x\}] = \emptyset$ *alors* $f(x)$ *n'est pas définie*
  - $f[\{x\}]$ *est un ensemble,* $\quad f[\{x\}] \subseteq ran(f)$

    $f(x)$ *est un élément de cet ensemble,* $\quad f(x) \in ran(f)$

Soit $f \in S \mathrel{+\!\!\!\to} T$

- **fonction totale** : si chaque élément de S a exactement une image

  $\mathbf{S \longrightarrow T}$ = ensemble des fonctions totales de S dans T

  $S \longrightarrow T = \{f \mid f \in S \mathrel{+\!\!\!\to} T \wedge \mathrm{dom}(f)= S\}$

- **fonction surjective** : si chaque élément de T est image d'au moins un élément de S

  $\mathbf{S \mathrel{+\!\!\!\twoheadrightarrow} T}$ = ensemble des fonctions surjectives partielles de S dans T

  $S \mathrel{+\!\!\!\twoheadrightarrow} T = \{f \mid f \in S \mathrel{+\!\!\!\to} T \wedge \mathrm{ran}(f)= T\}$

  $\mathbf{S \twoheadrightarrow T}$ = ensemble des fonctions **surjectives totales** de S dans T

- **fonction injective** : si deux éléments de S ne peuvent avoir la même image

  $\mathbf{S \mathrel{\rightarrowtail\!\!\!+}T}$ = ensemble des fonctions injectives partielles de S dans T

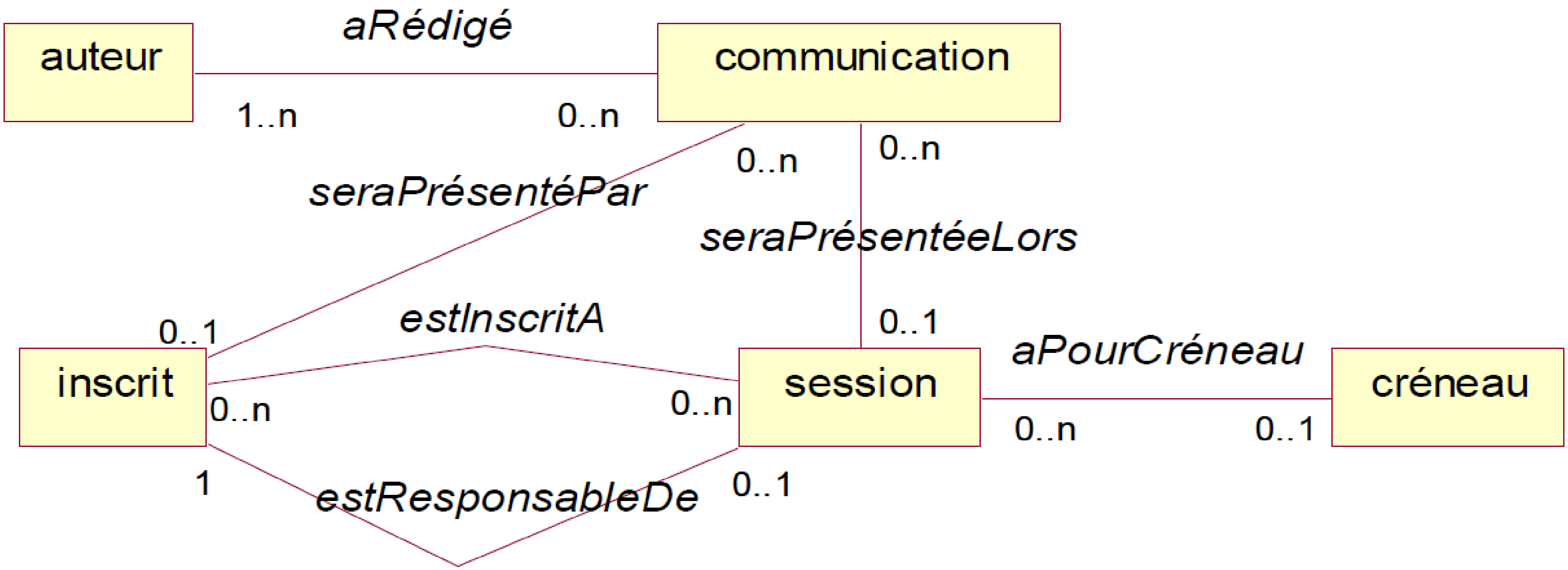
  $S \mathrel{\rightarrowtail\!\!\!+} T = \{f \mid f \in S \mathrel{+\!\!\!\to} T \wedge f^{-1} \in T \mathrel{+\!\!\!\to} S \}$

  $\mathbf{S \rightarrowtail T}$ = ensemble des fonctions **injectives totales** de S dans T

- **fonction bijective** = injective et surjective totale

  $\mathbf{S \rightarrowtail\!\!\!\!\twoheadrightarrow T}$ = ensemble des fonctions bijectives de S dans T

Exemple : conférence



Partie statique

SETS	PERSONNE, CRENEAU, SESSION, COMMUNICATION
VARIABLES	inscrit, session, estInscritA, estResponsableDe, aPourCréneau, aRédigé, seraPrésentéePar, seraPrésentéeLors, communication, auteur

INVARIANT

$communication \subseteq COMMUNICATION \wedge$

$inscrit \subseteq PERSONNE \wedge session \subseteq SESSION \wedge créneau \subseteq CRENEAU \wedge$

$auteur \subseteq PERSONNE \wedge présent \subseteq inscrit \wedge$

$estInscritA \in inscrit \leftrightarrow session \wedge estResponsableDe \in session \rightarrowtail inscrit \wedge$

$aPourCréneau \in session \mathrel{+\!\!\!\to} créneau \wedge$

$aRédigé \in auteur \leftrightarrow communication \wedge ran(aRédigé) = communication \wedge$

$seraPrésentéePar \in communication \mathrel{+\!\!\!\to} inscrit \wedge$

$seraPrésentéeLors \in communication \mathrel{+\!\!\!\to} session$

Langage pour la dynamique

- Langage de description des opérations
- Langage de substitutions généralisées
- Sémantique définie en termes de transformateurs de prédicats

Langage de substitutions généralisées

Symboles: x, y variables; E, F expressions; P, Q prédicats; G,H,I substitutions; S ensemble

- vide: **skip** ne fait rien
- affectation : **x := E** substitution simple

  cas particulier : f(x):=y
- composition en parallèle : **G ‖ H** faire G et H simultanément

  G ‖ H = H ‖ G
- alternative : **IF** P **THEN** G **ELSE** H **END**
- substitution préconditionnée : **PRE** P **THEN** G **END**

  si P est vrai alors l'effet de la substitution préconditionnée est celui de G.

  si P est faux, on ne peut rien dire sur l'effet de la substitution préconditionnée.

Substitutions non déterministes :

- choix non borné : **ANY** x **WHERE** P **THEN** G **END**

  variantes : x : $\in$ S **ANY** y **WHERE** y $\in$ S **THEN** x:=y **END**

  x : P **ANY** y **WHERE** P **THEN** x:=y **END**

- choix borné : **CHOICE** G **OR** H … **OR** I **END**

- choix borné gardé : **SELECT** P1 **THEN** G
  WHEN P2 **THEN** H
  WHEN P3 **THEN** I
  …
  END

  P1,P2,P3 ne sont pas nécessairement disjoints

Opérations

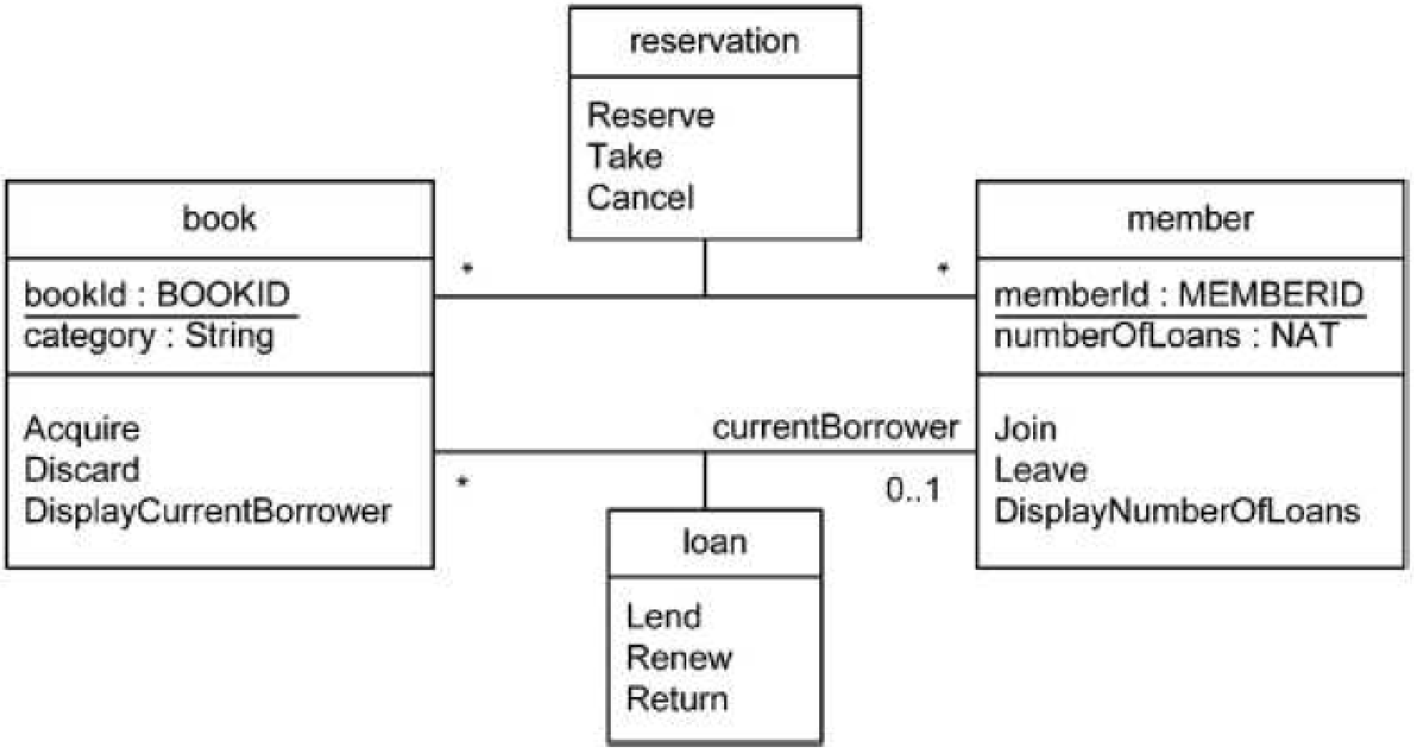
paramètres de sortie ← **nom_opération**(*paramètres d'entrée*) =

G

;

- Les paramètres de sortie et d'entrée sont optionnels
- Le passage des paramètres se fait par valeur
- Dans une opération d'une machine abstraite, la substitution G est en général une substitution préconditionnée. La précondition permet de fixer les conditions sous lesquelles l'opération doit être appelée. Le résultat d'une telle opération n'est garanti que si sa précondition est valide.

Exemple : bibliothèque



SETS $MEMBER; \ MEMBERID$

VARIABLES $members, \ memberId$

INVARIANT
$members \subseteq MEMBER \wedge memberId \in members \rightarrowtail MEMBERID$

INITIALISATION $members := \emptyset \parallel memberId := \emptyset$

OPERATIONS

$B\_Join(mId) =$
pre
$mId \in MEMBERID - \mathrm{RAN}(memberId) \wedge$
$members \subseteq MEMBER$
then
any $memb$ **where** $memb \in MEMBER - members$
then
$memberId := memberId \cup \{memb \mapsto mId\} \parallel$
$members := members \cup \{memb\}$
end
end;

$B\_Leave(mId) =$
pre $mId \in \mathrm{RAN}(memberId)$
then
$members := members - \{memberId^{-1}(mId)\} \parallel$
$memberId := memberId \rhd \{mId\}$
end

Cohérence des machines

- Cohérence : « la dynamique doit respecter la statique »
- Vérification de la cohérence : obligations de preuve
  - Initialisation
  - Opération

Cohérence des machines

- Cohérence : « la dynamique doit respecter la statique »
- Vérification de la cohérence : obligations de preuve
  – Initialisation $[Init]\ INV$
  – Opération $P \wedge INV \Rightarrow [S]\ INV$

Machine *Nom(param)*
Sets *S*
Variables *V*
Invariant *INV*
Initialisation *Init*
Operations Op = $P \mid S$
End

Sémantique wp

On note **[S] P** le prédicat “la substitution S établit le prédicat P”.

[S] P : condition initiale la plus large pour que, après avoir “exécuté” S, P devienne vrai.

Exemple :

[x := x+1] x = 5

se calcule en remplaçant x par x+1 dans le prédicat x=5.

On obtient : x = 4

Toute <u>substitution</u> est vue comme un <u>transformateur de prédicats</u>.

Permet de donner une sémantique aux différentes substitutions.

Symboles: x, y variables; E, F expressions; P, Q prédicats; G,H,I substitutions;

[skip]P	$\Leftrightarrow$ P	
[x := E] P	$\Leftrightarrow$ $P^{[E/x]}$	substitution de toutes les occurrences libres de x par E dans P
[x:=E ‖ y:=F] P	$\Leftrightarrow$ $P^{[E,F/x,y]}$	substitution dans P de toutes les occurrences libres de x par E et simultanément de toutes les occurrences libres de y par F

[G ‖ H] P se ramener à des substitutions multiples

[**IF** Q **THEN** G **ELSE** H **END**] P $\Leftrightarrow (Q \Rightarrow [G]P \wedge \neg Q \Rightarrow [H]P)$

[**PRE** Q **THEN** G **END**] P $\Leftrightarrow (Q \wedge [G]P)$

[**CHOICE** G **OR** H **END**]P $\Leftrightarrow ([G]P \wedge [H]P)$

[**ANY** x **WHERE** Q **THEN** G **END**]P $\Leftrightarrow \forall x \bullet (Q \Rightarrow [G]P)$

[**SELECT** P1 **THEN** G $\Leftrightarrow$ $(P1 \Rightarrow [G]P \wedge$

WHEN P2 **THEN** H $P2 \Rightarrow [H]P \wedge$

WHEN P3 **THEN** I $P3 \Rightarrow [I]P \wedge$

… …)

END] P

Modularité des spécifications

- Modularité en B : spécification incrémentale
- Machines abstraites :
  - **SEES** : accès aux ensembles et constantes, accès aux variables en lecture dans les opérations
  - **USES** : accès aux ensembles et constantes, accès aux variables en lecture dans les opérations et dans l'invariant
  - **INCLUDES** : accès aux ensembles et constantes, accès aux variables en lecture dans les opérations et dans l'invariant, appel des opérations en lecture

Méthode B :

- Systèmes ouverts
- Opérations préconditionnées
- Opérations toujours disponibles
- Spécification par contrat

Méthode B :

- Systèmes ouverts
- Opérations préconditionnées
- Opérations toujours disponibles
- Spécification par contrat

Event B :

- Systèmes fermés
- Evénements gardés
- Evénements disponibles uniquement lorsque la garde est vraie
- Spécification par observation

SYSTEM *Exemple_Machine*

SETS $ETATS = \{0, 1, 2\}$

VARIABLES *Etat*

INVARIANT $Etat \in ETATS$

INITIALISATION $Etat := 0$

EVENTS

change01 $=$

select $Etat = 0$

then $Etat := 1$

end

change12 $=$

select $Etat = 1$

then $Etat := 2$

end

change20 $=$

select $Etat = 2$

then $Etat := 0$

end

$$
\begin{array}{l}
\textbf{Event} \quad movement\_act \mathrel{\widehat{=}} \\
\quad \textbf{any} \\
\qquad tt \\
\quad \textbf{where} \\
\qquad \texttt{gu1} : tt \in TRAIN \\
\qquad \texttt{gu2} : tt \in dom(position) \\
\quad \textbf{then} \\
\qquad \texttt{act1} : position :| (\exists pp \cdot (pp \in TRACK \land \\
\qquad\qquad position' = position \mathbin{\lhd\!\!-} \{tt \mapsto pp\} \land \\
\qquad\qquad (\forall t2 \cdot (t2 \in dom(position') \land t2 \neq tt \Rightarrow pp \neq position'(t2))) \land \\
\qquad\qquad (\forall t2 \cdot (t2 \in dom(position) \land \\
\qquad\qquad\quad position(tt) \mapsto position(t2) \in is\_behind \Rightarrow \\
\qquad\qquad\qquad pp \mapsto position(t2) \in is\_behind)) \land \\
\qquad\qquad (pp = position(tt) \lor position(tt) \mapsto pp \in is\_behind)))
\end{array}
$$

Raffinement

- Première intuition : Wirth en 1971
- Plusieurs travaux : Dijkstra, Hoare, Milner, Morgan, Back dans les années 70
- Problème initial : correction d'un programme
- Abstraction du problème : correction d'une spécification vis-à-vis d'une autre spécification

Idée

- Spécification : ce que fait le logiciel
- Programme : comment fait le logiciel
- Raffinement : passage du quoi au comment

Raffinement

- Démarche de conception formelle
- On se donne une spécification formelle SP_1 qui exprime en toute abstraction ce que le programme doit réaliser
- Génération progressive du code du programme :

$$SP_1 \longrightarrow SP_2 \longrightarrow \quad \longrightarrow SP_n$$

- Correction de chacune des étapes de raffinement (obligations de preuve)

Exemple : tri d'un tableau

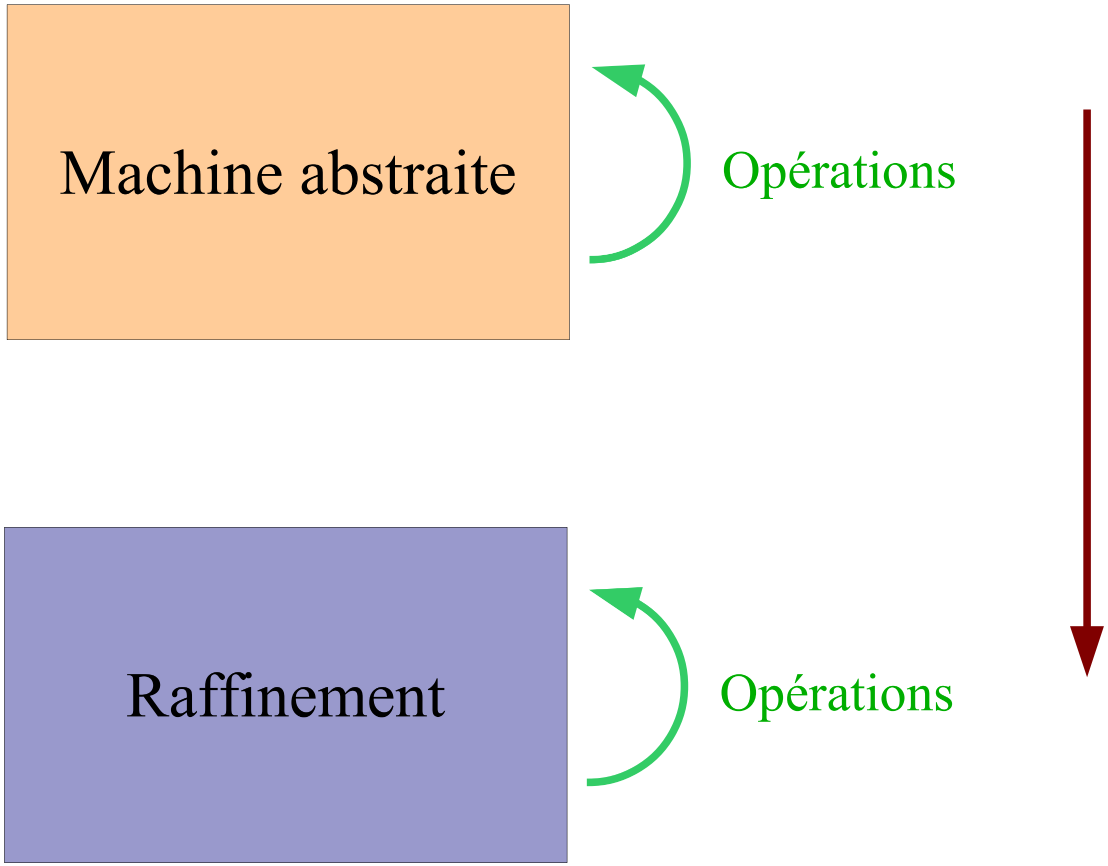
On veut développer un programme qui réalise le tri par ordre croissant d'un tableau *Tab* d'entiers de taille *n*. Le tableau trié est *Tab_Trié*.

Spécification abstraite : donner les propriétés du tableau trié sans faire allusion à aucun algorithme ou méthode de tri

- les éléments de *Tab* et *Tab_Trié* sont les mêmes
- pour chaque indice *i* (*i*=2..*n*):

$$Tab[i\text{-}1] \leq Tab\_Trié[i]$$

Spécification concrète : mettre en œuvre une des méthodes de tri existantes



On n'observe aucune différence entre une machine abstraite et son 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 :**

$$(SP_1 \sqsubseteq SP_2 \quad \text{et} \quad SP_2 \sqsubseteq SP_3) \Rightarrow SP_1 \sqsubseteq SP_3$$

$\Rightarrow$ génération de programmes complexes en plusieurs étapes élémentaires

- **Le raffinement B est monotone :**

$$(SP_1 \sqsubseteq SP_2 \quad \text{et} \quad SP_3 \sqsubseteq SP_4) \Rightarrow C(SP_1,SP_3) \sqsubseteq C(SP_2,SP_4)$$

$\Rightarrow$ raffinement indépendamment des constructeurs

Raffinement B

- Raffinement de données :
  - Introduction de variables et d'ensembles concrets
  - Réécriture des opérations pour la prise en compte des variables concrètes
  - Définition d'une relation d'abstraction entre l'espace d'état abstrait et l'espace d'état concret : **invariant de collage**
- Raffinement de contrôle
- Raffinement algorithmique

Raffinement B

- Raffinement de données
- Raffinement de contrôle :
  - Les opérations conservent la même signature
  - Elargissement des préconditions et réduction du non-déterminisme dans les substitutions
- Raffinement algorithmique

Raffinement B

- Raffinement de données
- Raffinement de contrôle
- Raffinement algorithmique :
  - Introduction de boucles while et d'itérations
  - Les substitutions en parallèle sont remplacées par des séquences

Machine MA
Sets C
Variables a
Invariant I
Initialisation $Init$
Operations
Op = $P \mid S$
End

Pas de uses

$\sqsubseteq$

Refinement MB
Refines MA
Sets D
Variables b
Invariant J
Initialisation $Init'$
Operations
Op = $P' \mid S'$
End

Syntaxe du raffinement

- Les ensembles abstraits C sont implicitement présents dans MB
- Les variables abstraites a sont raffinées par les variables concrètes b
- Les variables concrètes b contiennent :
  - les variables abstraites conservées par le raffinement
  - des variables concrètes introduites par le raffinement
- L'invariant de collage J permet de :
  - typer les variables concrètes introduites par le raffinement
  - exprimer des propriétés sur les variables concrètes
  - exprimer la relation liant les variables concrètes aux variables abstraites (d'où le nom d'invariant de *collage*)
- L'initialisation concrète *Init'* est le raffinement de l'initialisation abstraite *Init*
- L'opération abstraite *Op* est raffinée par une opération concrète de même signature

Machine abstraite

Machine Ensembles

Sets ENS

Variables ens1, ens2

Invariant ens1 $\subseteq$ ENS $\wedge$ ens2 $\subseteq$ ENS

Machine abstraite

Machine Ensembles

Sets ENS

Variables ens1, ens2

Invariant $ens1 \subseteq ENS \wedge ens2 \subseteq ENS$

Initialisation $ens1 := \varnothing \;||\; ens2 := ENS$

Operations

`ajout_ens1`(elem) =

pre $elem \in ENS$

then $ens1 := ens1 \cup \{elem\}$

end;

Operations

`retrait_ens1`(elem) =

pre elem $\in$ ens1

then ens1 := ens1 $-$ {elem}

end;

`union` =

begin ens1 := ens1 $\cup$ ens2

end;

`intersect` =

begin ens1 := ens1 $\cap$ ens2

end;

...

Comment raffiner ?

- Raffinement de données : remplacer les variables abstraites par des fonctions booléennes
- Raffinement algorithmique : introduire des itérations et des boucles while dans les opérations

Raffinement de données

Refinement Ensembles1

Refines Ensembles

Variables tab1, tab2

Invariant tab1 $\in$ ENS $\longrightarrow$ BOOL $\wedge$

tab2 $\in$ ENS $\longrightarrow$ BOOL $\wedge$

invariant de collage ?

Raffinement de données

Refinement Ensembles1

Refines Ensembles

Variables tab1, tab2

Invariant tab1 $\in$ ENS $\longrightarrow$ BOOL $\wedge$

tab2 $\in$ ENS $\longrightarrow$ BOOL $\wedge$

$tab1^{-1}[\{TRUE\}] = ens1 \ \wedge$

$tab2^{-1}[\{TRUE\}] = ens2$

Raffinement de données

Refinement Ensembles1

Refines Ensembles

Variables tab1, tab2

Invariant $tab1 \in ENS \longrightarrow BOOL \wedge$

$tab2 \in ENS \longrightarrow BOOL \wedge$

$tab1^{-1}[\{TRUE\}] = ens1 \wedge$

$tab2^{-1}[\{TRUE\}] = ens2$

Initialisation

tab1 := ??? $\|$

tab2 := ???

Raffinement de données

Refinement Ensembles1

Refines Ensembles

Variables tab1, tab2

Invariant $tab1 \in ENS \longrightarrow BOOL \wedge$

$tab2 \in ENS \longrightarrow BOOL \wedge$

$tab1^{-1}[\{TRUE\}] = ens1 \wedge$

$tab2^{-1}[\{TRUE\}] = ens2$

Initialisation

$tab1 := ENS \times \{FALSE\} \parallel$

$tab2 := ENS \times \{TRUE\}$

Operations

`ajout_ens1`(elem) =

pre ???

then ???

end;

`retrait_ens1`(elem) =

pre ???

then ???

end;

`union` =

begin

???

end;

...

Operations

```
ajout_ens1(elem) =
begin
    tab1(elem) := TRUE
end;
retrait_ens1(elem) =
begin
    tab1(elem) := FALSE
end;
union =
begin
    ???
end;
...
```

Operations

```
ajout_ens1(elem) =
begin
    tab1(elem) := TRUE
end;
retrait_ens1(elem) =
begin
    tab1(elem) := FALSE
end;
union =
begin
    tab1 := tab1 <+ ( tab2 |> {TRUE} )
end;
...
```

Correction du raffinement

Un raffinement est correct *ssi* l'effet de la spécification concrète ne contredit pas l'effet de la spécification abstraite

ou bien :

Un raffinement est correct *ssi* à chaque effet de la spécification concrète correspond un effet de la spécification abstraite

Il faut **vérifier** cette définition pour :

1. l'initialisation
2. chaque opération de la spécification abstraite

Obligations de preuve

- Correction de chaque opération op :

$$I \wedge J \wedge P \Rightarrow P' \wedge [\,[s := s']\, S'\,] \neg\, [S] \neg\, (J \;\wedge\; s = s'\,)$$

avec op de la forme : $s \leftarrow$ op (param) = **PRE** P **THEN** S **END**
et op' de la forme : $s \leftarrow$ op (param) = **PRE** P' **THEN** S' **END**

- Correction de l'initialisation :

$$[\text{Init'}] \neg\, [\text{Init}] \neg\, J$$

Remarque : la double négation est utile pour le cas de substitutions indéterministes. Peut être éliminée dans le cas contraire.

Machine *Exemple*

Variables Res

Invariant $Res \subseteq NAT$

Initialisation $Res := \varnothing$

Operations

entrer (n) =

Pre $n \in NAT$

Then

$Res := Res \cup \{n\}$

End

Refinement $Exemple_1$

Refines *Exemple*

Variables Res_1

Invariant

$Res_1 = max(Res \cup \{0\})$

Initialisation $Res_1 := 0$

Operations

entrer (n) =

Begin

$Res_1 := \max(Res_1, n)$

End

Raffinement de l'opération « entrer »

$$I \wedge J \wedge P \Rightarrow P' \wedge [S'] \neg\, [S] \neg\, J$$

1. $Res \subseteq NAT \wedge Res_1 = max(Res \cup \{0\}) \wedge n \in NAT \Rightarrow \; True$

2. $Res \subseteq NAT \wedge Res_1 = max(Res \cup \{0\}) \wedge n \in NAT \Rightarrow$

$[Res_1 := max(Res_1, n)\,] \neg\, (\,[Res := Res \cup \{n\}]$

$\neg\, (Res_1 = max(Res \cup \{0\})\,)$

$\Leftrightarrow$

$Res \subseteq NAT \wedge \; Res_1 = max(Res \cup \{0\}) \wedge \; n \in NAT \Rightarrow$

$[Res_1 := max(Res_1, n)\,] \neg\, (Res_1 \neq max(Res \cup \{0\} \cup \{n\})$

$\Leftrightarrow$

$Res \subseteq NAT \wedge \; Res_1 = max(Res \cup \{0\}) \wedge \; n \in NAT \Rightarrow$

$\neg\, (max(Res_1, n) \neq max(Res \cup \{n\})$

Tautologie

Intérêt de la double négation

Montrer que

$$y := 0$$

est un raffinement de

$$\textbf{choice } x{:=}0 \textbf{ or } x{:=}1 \textbf{ end}$$

MACHINE *Exemple_Machine*

VARIABLES x

INVARIANT $x \in 0 \,..\, 20$

INITIALISATION $x := 10$

OPERATIONS

change $=$

pre

$x + 2 \leq 20 \wedge$

$x - 2 \geq 0$

then

choice $x := x + 2$

or $x := x - 2$

end

end

REFINEMENT *Exemple_Ref*

REFINES *Exemple_Machine*

VARIABLES y

INVARIANT

$y \in 0 \,..\, 10 \wedge$

$x = 2 \times y$

INITIALISATION $y := 5$

OPERATIONS

change $=$

begin

$y := y + 1$

end

Implémentation

- Raffinement : passage du quoi au comment
- Spécification abstraite : spécification des propriétés, pas de séquence, pas d'itération
- Implémentation : raffinement traduisible en un programme Ada, C, C++, etc ...
- Langage B0 : séquences, itérations, pas de non-déterminisme, pas de précondition

Séquences et itérations

- Les séquences et itérations peuvent être utilisées dans un raffinement ou une implémentation B, mais jamais dans une machine abstraite.
- La séquence est représentée par le symbole ;
- Une itération B est de la forme suivante :

VAR *Définition de variables locales* **IN**
    WHILE *Condition de continuation* **DO**
    Séquence de substitutions
    INVARIANT *Liste de propriétés sous la forme de prédicats du 1er ordre*
    VARIANT *Variable entière strictement positive qui décroît à chaque pas de la boucle*
    END
END

Modularité des implémentations

- Construction de logiciels en couches
- Implémentations :
  - **SEES** : accès aux ensembles et constantes, appels des opérations en lecture
  - **IMPORTS** : accès aux ensembles et constantes, accès aux variables en lecture dans l'invariant, appels des opérations en lecture

Exemple d'implémentation

Implementation Ensembles2

Refines Ensembles1

Values ENS = 1..20

Initialisation ???

Initialisation

```
Var cpt In
cpt := 20;
While cpt ≥ 1 Do
    tab1(cpt) := FALSE;
    tab2(cpt) := TRUE;
    cpt := cpt – 1;
Invariant
    ???
Variant
    ???
End
```

Var cpt **In**

cpt := 20;

While cpt $\geq$ 1 **Do**

tab1(cpt) := FALSE;

tab2(cpt) := TRUE;

cpt := cpt – 1;

Invariant

$tab1 \in ENS \longrightarrow BOOL \wedge$

$tab2 \in ENS \longrightarrow BOOL \wedge$

$cpt \in ENS \cup \{0\} \wedge$

$ran((cpt+1)..20 \lhd tab1) = FALSE \wedge$

$ran((cpt+1)..20 \lhd tab2) = TRUE$

Variant

cpt

End

Raffinement

Méthode B :

- Signature du raffinement : idem que la machine abstraite

Event B :

- Signature du raffinement : possibilité de rajouter des événements

Raffinement

Méthode B :

- Signature du raffinement : idem que la machine abstraite
- Raffinement de données : invariant de collage

Event B :

- Signature du raffinement : possibilité de rajouter des événements
- Raffinement de données : invariant de collage

Raffinement

Méthode B :

- Signature du raffinement : idem que la machine abstraite
- Raffinement de données : invariant de collage
- Raffinement de contrôle : réduction du non-déterminisme et élargissement des préconditions

Event B :

- Signature du raffinement : possibilité de rajouter des événements
- Raffinement de données : invariant de collage
- Raffinement de contrôle : réduction du non-déterminisme et renforcement des gardes

Evénement B

NomEvenement =

select *Garde G*

then *Substitution S*

end

Méthode B :

- OP pour l'initialisation :

$$[\text{Init'}] \neg [\text{Init}] \neg J$$

- OP pour chaque opération :

$$I \wedge J \wedge P$$
$$\Rightarrow$$
$$P' \wedge$$
$$[[s := s']S'] \neg [S] \neg (J \wedge s = s')$$

Event B :

- OP pour l'initialisation :

$$[\text{Init'}] \neg [\text{Init}] \neg J$$

- OPs pour événement raffiné :

$$I \wedge J \wedge G' \Rightarrow G \wedge [S'] \neg [S] \neg J$$

$$I \wedge J \wedge G \Rightarrow G'_1 \vee G'_2 \vee \ \dots \vee \ G'_n$$

- OPs pour nouvel événement :

$$I \wedge J \wedge G' \Rightarrow [S'] J$$

$$I \wedge J \wedge G' \Rightarrow [S'] \mathrm{V} < \mathrm{V}$$

Références

- J.R. Abrial : *The B-Book.* Cambridge University Press, 1996
- J.R. Abrial : *Modeling in Event-B.* Cambridge University Press, 2010
- Atelier B : http://www.atelierb.eu/
- Rodin : http://www.event-b.org/
- ProB : http://www.stups.uni-duesseldorf.de/ProB/