

Génie logiciel avancé

Mihaela Sighireanu

UFR d'Informatique Paris Diderot, LIAFA, 175 rue Chevaleret, Bureau 6A7
<http://www.liafa.jussieu.fr/~sighirea/cours/genielog/>

Spécification formelle des comportements: la méthode B

Résumé

- 1 **Introduction**
 - Objectifs
 - Motivation
- 2 **Méthode B**
 - Principes
 - Machines abstraites
 - Ensembles
 - Substitutions généralisées
 - Obligations de preuve
 - Raffinements
 - Implémentation
 - Modularité
 - B événementiel

Objectifs

- Montrer comment les techniques de spécification formelles aident à découvrir des problèmes dans la spécification du système.
- (Cours précédent :) Définir et utiliser les techniques algébriques de spécification (ADT) pour spécifier les interfaces.
⇒ le modèle mathématique est une algèbre de termes.
- Définir et utiliser des techniques basées sur les modèles pour la spécification des comportements.
 - comportements séquentiels : VDM, Z, B
 - comportements parallèles : algèbre de processus, automates hiérarchiques, réseaux de Petri, etc.

Motivation

Spécification formelle des comportement :

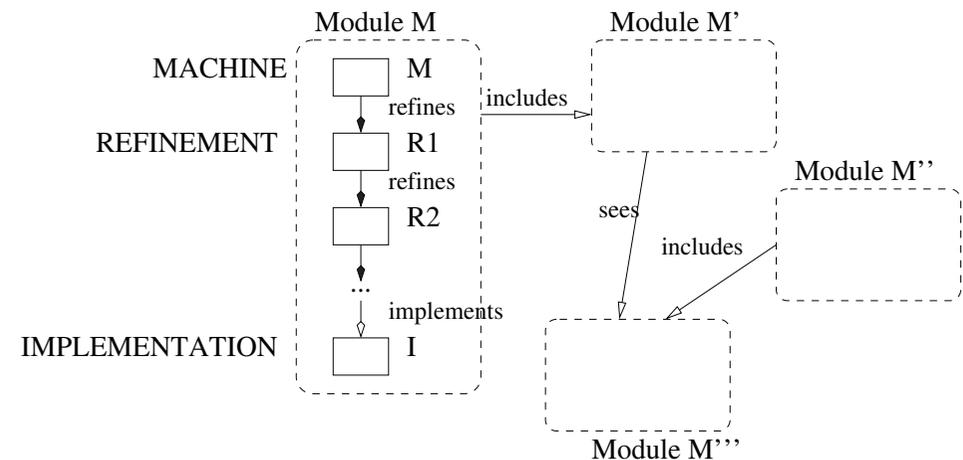
- La spécification d'interfaces permet de spécifier le "quoi" (sortes, opérations et leur propriétés algébriques) sans nous dire le "comment" (état, implémentation) !
- La spécification de comportements s'approche de l'implémentation et formalise l'essentiel d'un langage de programmation :
 - VDM (Johnes), Z (Abrial) : logique de Hoare avec opérations = prédicats
 - B (Abrial) : logique de Hoare avec opérations = substitutions généralisées
- Les deux types de spécifications sont complémentaires.
Par exemple, la spécification d'un système de fichiers Unix :
 - Spécification d'interface : sorte arbre et opérations générales sur un arborescence.
 - Spécification de comportement : état = (arborescence, home dir, crt dir) et des opérations qui changent l'état.

Méthode B

Principes :

- Ecriture de spécifications à état
 - formalisation rigoureuse mais accessible car basée sur la théorie des ensembles et la logique du premier ordre
- Preuve de propriétés sur les données
 - en utilisant le calcul de plus faibles pré-conditions
- Raffinement et génération de code
 - introduction progressive de la complexité ...
 - ... jusqu'à une représentation informatique des données
- Construction incrémentale
 - des spécifications, des développements et des preuves
- Méthodologie et outillage
 - vaste documentation disponible : www.atelierb.eu
 - outil en licence libre : Atelier B
 - génération de code Ada et C

Structure de la spécification



Structure de la spécification

Une unité de compilation en B est :

MACHINE décrit un comportement de manière abstraite par son état et ses opérations ; c'est la base de la spécification d'un comportement.

REFINEMENT (raffinement) est une étape du développement d'une machine ; il détaille la structure et les opérations de la machine ; il génère des obligations de preuve de cohérence du raffinement.

IMPLEMENTATION est le dernier maillon du développement d'une machine, le plus proche du codage ; il génère des obligations de preuve pour les blocs d'instruction utilisés.

Un module = chaîne de développement constitué

d'une MACHINE M ,

0 ou plusieurs REFINEMENT successifs de M (R_M^1, \dots, R_M^n) et une IMPLEMENTATION I .

- Les modules communiquent par appel d'opérations.
- Ils permettent une spécification (et preuve) incrémentale.

Machines abstraites : structure de la spécification

- Partie entête :
 - nom de la machine
 - (optionnel) paramètres de la machine
 - (optionnel) contraintes sur les paramètres
- Partie statique :
 - déclaration d'ensembles
 - déclaration de constantes
 - propriétés des constantes
 - variables (état)
 - invariant (caractérisation de l'état)
- Partie dynamique :
 - initialisation de l'état
 - opérations

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

$r \leftarrow op(p) = \text{PRE } P \text{ THEN } K \text{ END};$

...

END

Exemple : machine de la division entière

MACHINE

DIVISION

OPERATIONS

$qq, rr \leftarrow div(aa, bb) = /* qq$ et rr sont le quotient resp. le reste */

PRE /* de la division entière de aa par bb */

$aa \in \mathbf{NAT} \wedge bb \in \mathbf{NAT}_1$

THEN

ANY ss, tt **WHERE**

$ss \in \mathbf{NAT} \wedge tt \in \mathbf{NAT} \wedge aa = bb * ss + tt \wedge tt < bb$

THEN

$qq, rr := ss, tt$

END

END

END

Note : ci-dessus on utilise la syntaxe concrète mathématique, voir démo Atelier B pour la syntaxe concrète et la feuille mémo B.

Définition de base d'ensembles

Un nouveau ensemble peut être défini **explicitement** :

- soit de manière abstraite (fini ou infini), en définissant uniquement son nom : **SETS ADRESSES**
- soit de manière concrète (fini) en énumérant ses valeurs : **SETS FEUX = { rouge, jaune, vert }**

Ensembles prédéfinis : **BOOL, NAT, NAT1, INT, CHAR, STRING** et les sous-ensembles d'entiers $m..n$.

Prédicats et opérations sur les ensembles

Les prédicats usuels sont disponibles sur les ensembles :

\in	appartient à	\notin	n'appartient pas à
\subseteq	est inclus dans	$\not\subseteq$	n'est pas inclus dans
\subset	est strictement inclus dans	$\not\subset$	n'est pas strictement inclus dans

Les opérateurs à résultat entier sur les ensembles :

card	nb d'éléments d'un ensemble	card(E)
min	minimum d'un ensemble non vide d'entiers	min(E)
max	maximum d'un ensemble non vide d'entiers	max(E)

Expressions d'ensemble

Les ensembles peuvent être définis **implicitement** en utilisant des expressions d'ensemble E .

\emptyset	ensemble vide	$E \times E$	produit cartésien
$\mathbb{P}(E)$	ensemble de sous-ens.	$\mathbb{P}_1(E)$	ens. de sous-ens. non vides
$\{x \mid P\}$	déf. en compréhension	$\{E, \dots, E\}$	déf. en extension
$E - E$	différence		
$E \cup E$	union	$E \cap E$	intersection
union(E)	union des sous-ens. de E	inter(E)	inter. des sous-ens. de E
$\bigcup Id \cdot (P \mid E)$	union quantifiée	$\bigcap Id \cdot (P \mid E)$	intersection quantifiée

Exemples :

$$\text{union}(\{\{1,2\}, \{1,2,3\}, \{2,4,5\}\}) = \{1,2,3,4,5\}$$

$$\text{inter}(\{\{1,2\}, \{1,2,3\}, \{2,4,5\}\}) = \{2\}$$

$$\bigcup x \cdot (x \in \{2,4\} \mid \{y \mid y \in \mathbb{N} \wedge x - 1 \leq y \leq x + 1\}) = \{1,2,3,4,5\}$$

Relations

Mathématiquement, les relations sont un cas particulier de construction d'ensemble, car r est une relation entre s et t si $r \in \mathbb{P}(s \times t)$.

Principaux constructeurs de relations (voir mémo B pour une liste complète) :

Expression	Définition	Sémantique & Condition
$s \leftrightarrow t$	$\mathbb{P}(s \times t)$	les relations entre s et t
$\text{dom}(r)$	$\{x \mid x \in s \wedge \exists y. y \in t \wedge (x, y) \in r\}$	domaine de $r \in s \leftrightarrow t$
$\text{ran}(r)$	$\{y \mid y \in t \wedge \exists x. x \in s \wedge (x, y) \in r\}$	co-domaine de $r \in s \leftrightarrow t$
$r[u]$	$\{y \mid y \in t \wedge \exists x. x \in u \wedge (x, y) \in r\}$	image par u , $r \in s \leftrightarrow t \wedge u \subseteq s$
r^{-1}	$\{(y, x) \mid (x, y) \in r\}$	inverse d'une relation $r \in s \leftrightarrow t$
$\text{id}(s)$	$\{(x, x) \mid x \in s\}$	relation identité sur l'ens. s
$r_1; r_2$	$\{(x, z) \mid \exists y. (x, y) \in r_1 \wedge (y, z) \in r_2\}$	composition séquentielle $r_1 \in s \leftrightarrow t, r_2 \in t \leftrightarrow u$

Exercice : $r_1 = \{(0, 2), (1, 5), (2, 5), (2, 7)\}$ et $r_2 = \{(0, 0), (2, -1), (5, 8), (6, 9)\}$
alors $r_1; r_2 = \dots$?

Séquences

Mathématiquement, une séquence est un type particulier de fonction dont le domaine est un intervalle d'entiers et le co-domaine est l'ensemble des éléments de la séquence.

Principaux constructeurs de séquences :

Expression	Définition	Sémantique
$\text{seq}(E)$	$\bigcup n \cdot (n \in \text{NAT} \mid 1..n \rightarrow E)$	seq. finies d'éléments de E
$\text{seq}_1(E)$	$\text{seq}(E) - \emptyset$	seq. non vides
$\text{perm}(E)$		seq. bijectives ou permutations

Valeurs de séquences : $[]$ (séq. vide), $[a, b, c]$

Opérateurs classiques : size , \wedge (concaténation), \leftarrow (ajout à droite), rev (inversion), first , last , tail , front (tous sauf le dernier).

Exercice : Montrer que $\text{front}(s) \leftarrow \text{last}(s) = s$.

Fonctions

Mathématiquement, une fonction est une relation dont chaque élément du domaine est associé à au plus un élément du co-domaine.

Principaux constructeurs de fonctions :

Expression	Définition	Sémantique
$s \mapsto t$	$\{r \mid r \in s \leftrightarrow t \wedge \forall x, y, z. (x, y) \in r \wedge (x, z) \in r \Rightarrow (y = z)\}$	f. partielle
$s \twoheadrightarrow t$	$\{f \mid d \in s \mapsto t \wedge \text{dom}(f) = s\}$	f. totale
$s \gg t$	$\{f \mid f \in s \mapsto t \wedge f^{-1} \in t \mapsto s\}$	f. injective partielle
$s \rightarrow t$	$s \gg t \cap s \twoheadrightarrow t$	f. injective totale
$\lambda f \cdot (P \mid E)$		déf. explicite

Exemple : $\lambda x \cdot (x \in \text{NAT} \mid x + 2)$.

Exemples

Ex. 1 : On veut spécifier une opération d'allocation d'un mot dans la mémoire ; elle retourne l'adresse de l'emplacement alloué, s'il y a de la place en mémoire.

Modélisation des ensembles et de l'état :

MACHINE MEMORY

SETS

ADDRESSES ensemble abstrait d'adresses

CONSTANTS

mem, null

PROPERTIES

$\text{mem} \subseteq \text{ADDRESSES} \wedge$ les adresses à allouer
 $\text{null} \in \text{ADDRESSES} \wedge$ une adresse particulière
 $\text{null} \notin \text{mem}$ l'adresse null n'est pas dans la mémoire

VARIABLES

free

INVARIANT

$\text{free} \subseteq \text{mem}$ l'ensemble des adresses libres

Exemples

Ex. 2 : Un ascenseur qui a une porte à chaque étage et on ne distingue pas les appels intérieurs de ceux extérieurs; pas de pannes.

Modélisation des ensembles et de l'état :

MACHINE LIFT

SETS

$MODE = \{stop, moving\}$ état de l'ascenseur

CONSTANTS

$max_floors, FLOORS$

PROPERTIES

$max_floors \in \mathbb{N}T_1 \wedge FLOORS = 0..max_floors$ ensemble d'étages

VARIABLES

$openf, callf, pos, mode$

INVARIANT

$openf \subseteq FLOORS \wedge$ ensemble portes ouvertes
 $callf \subseteq FLOORS \wedge$ ensemble d'appels faits
 $pos \in FLOORS \wedge$ position courante
 $mode \in MODE$ mode courant

Spécification d'opérations

Rappels :

- Logique des programmes (Hoare) : correction partielle

$$P \{S\} Q$$

Si l'état satisfait P avant S et si S termine, alors l'état satisfait Q après.

- Plus faible précondition : correction totale

$$wp(S, Q)$$

Si l'état satisfait $wp(S, Q)$ avant S alors S termine et l'état satisfait Q après.

Remarque : $P\{S\}Q$ en correction totale est équivalent à $P \Rightarrow wp(S, Q)$.

- En B, la plus faible précondition $wp(S, Q)$ est notée sous la forme d'une substitution $[S]Q$.

Langage des substitutions généralisées

Notation	Syntaxe	$wp(S, Q)$	Description
$x := E$		$[x := E]Q$	subst. simple
$x, y := E, F$		$[z := F]$	subst. multiple
skip		$[x := E][y := z]Q$	
$P \mid S$	PRE P THEN S END	Q	subst. sans effet
$P \Rightarrow S$	SELECT P THEN S END	$P \wedge [S]Q$	subst. préconditionnée
$S \parallel T$	CHOICE S OR T END	$P \Rightarrow [S]Q$	subst. gardée
$@z \cdot S$	VAR z IN S END	$[S]Q \wedge [T]Q$	subst. de choix borné
$S; T$		$\forall z. [S]Q$	subst. de choix non borné
$\mathcal{W}(P, S, J, V)$	WHILE P DO S	$[S][T]Q$	séquencement
	INVARIANT J		subst. d'itération
	VARIANT V END		

Langage de substitutions généralisées (suite)

Traduction	Syntaxe dérivée	Description
$(P \Rightarrow S) \parallel$	IF P THEN S	subst. conditionnelle
$(\neg P \Rightarrow T)$	ELSE T END	
$@z \cdot (P \Rightarrow S)$	ANY z WHERE P	choix gardé
	THEN S END	
$S \parallel T$...	subst. multiple généralisée
$x := \text{bool}(P)$	x := IF P THEN TRUE ELSE FALSE	conversion P en valeur booléenne
$f(x) := E$	f := f <+ { (x, E) }	surcharge fonction

Exemples

Ex. 1 : On veut spécifier une opération d'allocation d'un mot dans la mémoire ; elle retourne l'adresse de l'emplacement alloué, s'il y a de la place en mémoire.

Modélisation de l'opération :

INITIALISATION

$free := mem$

OPERATIONS

$rr \leftarrow alloc =$

entête de l'opération

IF $free \neq \emptyset$ **THEN**

test dynamique

ANY vv **WHERE**

$vv \in free$

choix d'une adresse libre

THEN

$free, rr := free - \{vv\}, vv$

modification de l'état
et retour de l'adresse allouée

END

ELSE

$rr := null$

retour valeur de non allocation

END ;

Navigation icons

Exemples

Ex. 2 : Un ascenseur qui a une porte à chaque étage et on ne distingue pas les appels intérieurs de ceux extérieurs ; pas de pannes.

Modélisation des ensembles et de l'état :

INVARIANT

$openf \subseteq FLOORS \wedge callf \subseteq FLOORS \wedge$

$pos \in FLOORS \wedge mode \in FLOORS \wedge$

$openf \neq \emptyset \Rightarrow (openf = \{pos\} \wedge mode = stop) \wedge$ sûreté

$mode = stop \Rightarrow pos \notin callf$

INITIALISATION

$openf := \emptyset \parallel callf := \emptyset \parallel pos := 0 \parallel mode := stop$

OPERATIONS

$move =$

actualise pos

PRE $openf = \emptyset$ **THEN**

bouge que si porte non ouverte
pas d'appel, arrêt

IF $callf = \emptyset$ **THEN** $mode := stop$

ELSE

ANY nn **WHERE**

$nn : callf \wedge nn \neq pos$

un étage avec appel

THEN

$mode, pos := moving, nn$

END END END ;

Navigation icons

Obligations de preuve

MACHINE

$M(X, u)$

CONSTRAINTS

C

SETS

$S; T = \{a, b\}$

CONSTANTS

c

PROPERTIES

R

VARIABLES

x

INVARIANT

I

INITIALISATION

U

OPERATIONS

$r \leftarrow op(p) = \text{PRE } P \text{ THEN } K \text{ END ;}$

Notations :

$A = C \wedge x \in \mathbb{P}_1(\text{INT})$

$B = R \wedge S \in \mathbb{P}_1(\text{INT})$

$\wedge T \in \mathbb{P}_1(\text{INT})$

$\wedge T = \{a, b\} \wedge a \neq b$

Preuve que l'initialisation établit
l'invariant :

$A \wedge B \Rightarrow [U]I$

Preuve de préservation d'invariant pour
chaque opération op :

$A \wedge B \wedge I \wedge P \Rightarrow [K]I$

Navigation icons

Obligations de preuve : exemple

Ex. 1 : Allocation mémoire.

$A = true$

$B = mem \subseteq ADDRESSES \wedge null \in ADDRESSES \wedge null \notin mem$
 $\wedge ADDRESSES \in \mathbb{P}_1(\text{INT})$

Preuve que l'initialisation établit l'invariant :

$B \Rightarrow [free := mem](free \subseteq mem)$

Preuve de préservation d'invariant pour l'opération $alloc$:

$B \wedge free \subseteq mem \Rightarrow [alloc](free \subseteq mem)$

\Leftrightarrow

$B \wedge free \subseteq mem \Rightarrow \left(\begin{array}{l} (free \neq \emptyset \Rightarrow (\forall vv \in free \Rightarrow (free - \{vv\} \subseteq mem))) \\ \wedge \\ (free = \emptyset \Rightarrow (free \subseteq mem)) \end{array} \right)$

Navigation icons

Relation définie par une substitution

Un substitution généralisée S définit une relation entre les valeurs (en b) d'une variable x avant et après S , c'est-à-dire $\text{prd}_x(S) \in b \leftrightarrow b$.

Notation : x variables avant, x' variables après.

Syntaxe : $x\$0$ dénote la valeur de x avant la substitution dans les prédicats P .

$$\begin{aligned} \text{prd}_x(x := E) &\Leftrightarrow x' = E \\ \text{prd}_{x,y}(x := E) &\Leftrightarrow x' = E \wedge y' = y \\ \text{prd}_x(\text{skip}) &\Leftrightarrow x' = x \\ \text{prd}_x(P \mid S) &\Leftrightarrow P \Rightarrow \text{prd}_x(S) \\ \text{prd}_x(P \Rightarrow S) &\Leftrightarrow P \wedge \text{prd}_x(S) \\ \text{prd}_x(S \parallel T) &\Leftrightarrow \text{prd}_x(S) \vee \text{prd}_x(T) \\ \text{prd}_x(@z \cdot S) &\Leftrightarrow \exists z \cdot \text{prd}_x(S) \quad \text{si } z \text{ n'est pas modifié par } S \\ \text{prd}_x(@y \cdot S) &\Leftrightarrow \exists(y, y') \cdot \text{prd}_{x,y}(S) \quad \text{si } y \text{ est modifié par } S \\ \text{prd}_x(x \in E) &\Leftrightarrow x' \in E \\ \text{prd}_x(x : P) &\Leftrightarrow [x\$0, x := x, x']P \\ \text{prd}_x(S; T) &\Leftrightarrow S \text{ termine} \Rightarrow \exists x'' \cdot ([x' := x'']\text{prd}_x(S) \wedge [x := x'']\text{prd}_x(T)) \\ \text{prd}_x(\mathcal{W}(P, S, J, V)) &\Leftrightarrow \mathcal{W} \text{ termine} \Rightarrow [x\$0, x := x, x'](J \wedge \neg P) \end{aligned}$$

Raffinements de machines

- Raffinement = étape de développement d'une machine.
- Un raffinement détaille et précise les structures de données et les opérations de la machine.
- Les obligations de preuve du raffinement assurent que le raffinement est cohérent avec la sémantique de la machine.

Trois types :

- **Simple** : sans changement de variables d'états
- **De composante** : avec changement (ajout, suppression) de variables d'état
- **Pas à pas** : composition (en séquence) de raffinements

Raffinement simple

Soit x l'ensemble de variables d'état.

Definition

Une substitution S est raffinée par un substitution T , notation $S \sqsubseteq T$, ssi pour tout prédicat R on a $[S]R \Rightarrow [T]R$.

\Leftrightarrow l'effet de T sur les variables x est un des effets de S , c'est-à-dire qu'en T se produit :

- une diminution du non-déterminisme et/ou
- un affaiblissement des pré-conditions

Exemples : Prouver que

$$\begin{aligned} (x := 1 \parallel x := 2) &\sqsubseteq x := 1 \\ (x > 4 \mid x := x + 1) &\sqsubseteq (x \geq 0 \mid x := x + 1) \end{aligned}$$

Raffinement avec changement de représentation

Soient :

- x l'ensemble de variables d'état (initial) avec valeurs en b .
- y un ensemble de variables d'état (raffiné) avec valeurs en c .
- v une relation telle que $v \in c \leftrightarrow b \wedge \text{dom}(v) = c$

Definition

Une substitution S est raffinée par un substitution T via la relation v , notation $S \sqsubseteq_v T$, ssi $v(y, x) \wedge \text{prd}_y(T) \Rightarrow \exists x' \cdot (\text{prd}_x(S) \wedge v(y', x'))$.

La relation v peut être décrite en extension $\{(y, x) \mid L\}$.

Exemple :

$$\begin{aligned} mstate &: mem \rightarrow \text{BOOL} \\ v &= \{(mstate, free) \mid free = mstate^{-1}\{\{FALSE\}\}\} \\ (free := mem) &\sqsubseteq_v (mstate := mem \times \{FALSE\}) \end{aligned}$$

Machine et raffinement : syntaxe

MACHINE $M(X, u)$	REFINEMENT $N(X, u)$
CONSTRAINTS C	REFINES M
SETS G	SETS H
CONCRETE_CONSTANTS c	CONCRETE_CONSTANTS d
ABSTRACT_CONSTANTS a	ABSTRACT_CONSTANTS b
PROPERTIES R	PROPERTIES O
VARIABLES x	VARIABLES y
INVARIANT I	INVARIANT J
INITIALISATION U	INITIALISATION V
OPERATIONS $r \leftarrow op(p) = \text{PRE } P \text{ THEN } K \text{ END};$	OPERATIONS $r \leftarrow op(p) = \text{PRE } Q \text{ THEN } L \text{ END};$
...	...
END	END

Machine et raffinement : contraintes

- les paramètres doivent être répétés dans l'entête et pas de contraintes ;
- les ensembles H doivent être nouveaux ;
- les constantes abstraites a doivent être raffinées (en utilisant b ou d) par la relation donnée par le prédicat O ;
- les constantes d ont des nouveaux noms ;
- la relation de raffinement est $v = \{(y, x) \mid I \wedge J\}$;
- les opérations redéfinies en N gardent les mêmes entêtes (liste résultats, nom, liste paramètres).

Machine construite par le raffinement

Soit $B_N = O \wedge H \in \mathbb{P}_1(\text{INT})$.
S'il existe une preuve pour les obligations de preuve suivantes :

- Preuve de l'initialisation :
 $A \wedge B \wedge B_N \Rightarrow [M] \neg [U] \neg J$
- Preuve de pré-condition raffinée :
 $A \wedge B \wedge B_N \wedge I \wedge J \wedge P \Rightarrow Q$
- Preuve conservation de l'invariant par une opération sans résultat :
 $A \wedge B \wedge B_N \wedge I \wedge J \wedge P \Rightarrow [L] \neg [K] \neg J$
- Preuve conservation de l'invariant par une opération avec résultat :
 $A \wedge B \wedge B_N \wedge I \wedge J \wedge P$
 $\Rightarrow [[r := r'] L] \neg [K] \neg (J \wedge r = r')$

alors par raffinement on définit
"virtuellement" la machine ci-contre.

MACHINE $N_M(X, u)$
CONSTRAINTS C
SETS $G; H$
CONCRETE_CONSTANTS c, d
ABSTRACT_CONSTANTS b
PROPERTIES $\exists a \cdot (R \wedge O)$
VARIABLES y
INVARIANT $\exists x \cdot \exists a \cdot (R \wedge O \wedge I \wedge J)$
INITIALISATION V
OPERATIONS $r \leftarrow op(p) =$ PRE $Q \wedge \exists x \cdot \exists a \cdot (R \wedge O \wedge I \wedge J \wedge P)$ THEN L END;

Raffinement : exemple

Ex. 1 : Allocation mémoire.

REFINEMENT $MEMORY_r1$
REFINES $MEMORY$
VARIABLES $mstate$
INVARIANT $mstate : mem \rightarrow \text{BOOL} \wedge$ $free = mstate^{-1}\{\{FALSE\}\}$
INITIALISATION $mstate := mem \times \{FALSE\}$
OPERATIONS $rr \leftarrow alloc =$ IF $mstate\{\{FALSE\}\} \neq \emptyset$ THEN ANY vv WHERE $vv : mstate^{-1}\{\{FALSE\}\}$ THEN $mstate(vv) := \text{TRUE} \parallel rr := vv$ END ELSE $rr := null$ END;
END

Raffinements pas à pas

Lemma

La relation de raffinement est transitive, c'est-à-dire :

$$S \sqsubseteq_{v_1} T \wedge T \sqsubseteq_{v_2} Y \Rightarrow S \sqsubseteq_{v_2; v_1} U$$

Conséquence :

- si la spécification R_2 raffine R_1 via la relation $\{(x_2, x_1) \mid J_1\}$
 - et la spécification R_3 raffine R_2 via la relation $\{(x_3, x_2) \mid J_2\}$
- alors on peut montrer que R_3 raffine R_1 pour la relation $\{(x_3, x_1) \mid \exists x_2 \cdot (J_1 \wedge J_2)\}$

Implémentation

Dernier maillon d'une chaîne de raffinement d'une machine, une implémentation est un cas particulier de raffinement car les conditions suivantes sont satisfaites :

- une clause spéciale **VALUES** doit fixer les valeurs des ensembles (abstrait) et des constantes tel que l'invariant raffiné est satisfait :
VALUES
 $ADDRESSES = 0..200000$; $mem = 1000..20000$; $null = 0$
- les variables doivent être typées par des ensembles **concrets** :
 - des entiers représentable (**INT**)
 - des booléens (**BOOL**)
 - des types énumérés
 - des tableaux (fonction totale) indexé par un (produit d') ensemble concret simple (non tableau) types avec des éléments dans un ensemble concret simple.
- les substitutions doivent être déterministes, c'est-à-dire des **instructions** : affectation, appel d'opération, séquençement, bloc, déclaration de variables, **IF**, **CASE** et **WHILE**
- les prédicats utilisés dans les instructions doivent être évaluable opérationnellement (sans choix non-déterministe)

Implémentation : exemple

Ex. 1 : Allocation mémoire.

IMPLEMENTATION

MEMORY_I

REFINES

MEMORY_r1

VALUES

$ADDRESSES = 0..200000$; $mem = 1000..20000$; $null = 0$

CONCRETE_VARIABLES

mstate

INITIALISATION

$mstate := mem \times \{FALSE\}$

OPERATIONS

$rr \leftarrow alloc =$

VAR *ind, val* **IN**

$ind := 1000$; $val := mstate(ind)$;

WHILE $ind \leq 20000 \wedge val = TRUE$ **DO**

$ind := ind + 1$; $val := mstate(ind)$

INVARIANT

$ind \in 1000..20000 \wedge val = mstate(ind) \wedge mstate[1..ind - 1] \subseteq \{TRUE\}$

VARIANT $20000 - ind$

END;

IF $ind \leq 20000$ **THEN** $rr := vv$; $mstate(vv) := TRUE$

ELSE $rr := null$ **END**

END;

Modularité

Primitives de structurations de modules :

- **IMPORTS** M_1, \dots, M_n : que dans une implémentation, permet d'utiliser les services des machines ou des instanciations de machines importés; pas d'importation multiple.
- **SEES** M_1, \dots, M_n : dans tout composant, mais sans instanciation, permet d'utiliser en lecture les informations d'une machine; pas de dépendences circulaires.
- **INCLUDES** M_1, \dots, M_n : dans machine ou raffinement, permet inclusion des définitions d'un composant ou de son instanciation; pas d'inclusion multiple.

Des obligations de preuves peuvent être générées pour assurer la cohérence de la composition.

Pour rendre visibles les opérations d'une machine importée ou incluse, utiliser :

- la clause **PROMOTES** op_1, \dots, op_n pour sélectionner les opérations
- la clause **EXTENDS** M_i pour inclure puis promouvoir toutes les opérations de M_i

B événementiel

Il s'agit une machine spéciale, appelée **SYSTEM**, utilisée pour spécifier des automates :

- les opérations sont remplacées par des **événements** (= changement atomique de l'état)
- un événement a un nom mais pas de paramètres ou résultat
- les substitutions utilisées dans les événements ont une forme spéciale :
 - **BEGIN S END**
 - **SELECT $G(x)$ THEN S END**
 - **ANY v WHERE $G(t, x)$ THEN S END**(avec x les variables d'état).