Spécifications Formelles, M1 2012-2013

Serenella Cerrito et Francesco Belardinelli

Laboratoire Ibisc, Université d'Evry Val d'Essonne, France

2012-2013

Introduction, 1

- Spécification formelle d'un logiciel = description précise mais abstraite de ce que le logiciel doit faire.
- Aide au développement.
- But : Vérifier formellement (en partie automatiquement) la correction de la conception
- But : La corriger, par étapes, avant d'investir dans l'implémentation.
- Résultat final : implémentation correcte par construction.
- Un exemple concret d'application : ligne 14 du métro parisien

Introduction, 2

- Plusieur méthodes de spécification formelle, fondées sur : automates, réseaux de Petri, logiques (équationnelles, des prédicats, temporelles etc.)
- Ce cours : la méthode B (utilisée pour la ligne 14 du métro).
  - Fondée sur la notion de machine abstraite
  - Notation AMN (*Abstract Machine Notation*)
  - Abstraction, modularité.
  - AMN contient des notations ensemblistes et logiques (logique (classique) des prédicats).

Abstract Machine

Une machine abstraite (AM) est une spécification d'une partie du système (logiciel) à réaliser.

Boîte noire qui devra se composer avec d'autres boîtes (modules).

Décrit les opérations que cett partie du logiciel doit effectuer, leur entrées, leur effets, etc.

Format :

MACHINE ...
VARIABLES...
INVARIANT...
INTIALISATION..
OPERATIONS...
END

Ici, le nom de la machine.

Ex. : une machine **Ticket** qui distribue des billets (dans un supermarché, ou à la poste) pour ordonner la queue des clients.

Vague spécification : *A l'entrée, chaque client prend un ticket avec un nombre. Un écran montre le nombre du prochain client à servir.*

MACHINE Tickets
VARIABLES...
INVARIANT...
INITIALISATION...
OPERATIONS...
END

Abstract Machine : VARIABLES

Déclaration des variables qui décrivent l'état courant de la machine.
Pas ici : ni les types des variables, ni d'autres informations.

Dans notre exemple, deux variables
serve : le numéro du client à servir (montré sur l'écran)
next : le numéro du prochain ticket à donner.

MACHINE Tickets
VARIABLES *serve*, *next*
INVARIANT...
INITIALISATION...
OPERATIONS...
END

Abstract Machine : INVARIANT 1

INVARIANT : un énoncé qui donne le type des variables de la machine et une propriété des valeurs des variables qui doit toujours rester vraie, pendant l'éxécution.

Abstract Machine : INVARIANT 2

Formes possible d'une déclaration du type d'une variable x dans l'invariant :

- $x \in TYPE$ (*la valeur de x appartient à l'ensemble TYPE*)
- $x \subseteq TYPE$ (*la valeur de x est un sous-enseble de l'ensemble TYPE*)
- x = expression

Exemple machine Tickets : $serve \in \mathbb{N}$ et $next \in \mathbb{N}$.

Abstract Machine : INVARIANT 3

Expression de la condition qu'une propriété P doit rester toujours vraie : par une formule logique.

Exemple machine Tickets : serve $\leq$ next.

L'invariant est donc une formule logique complexe.

MACHINE Tickets
VARIABLES *serve*, *next*
INVARIANT $serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge serve \leq next$
INITIALISATION...
OPERATIONS...
END

Liste de déclarations de la forme :
$outputs \leftarrow name\ (inputs)$

name = nom de l'opération
inputs = liste des variables d'entrée
outputs = liste des variables de sortie.

Inputs et/ou outputs : optionnels.

MACHINE Tickets
VARIABLES $serve$, $next$
INVARIANT $serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge serve \leq next$
INITIALISATION...
OPERATIONS
$ss \leftarrow serve\_next$
$tt \leftarrow take\_next$ **END**

NB : Pas d'inputs, ici !

Spécifier une opération = indiquer :
sa precondition
son body= effet sur les variables outputs et, éventuellement, mise à jour de l'état de la machine.

Pour serve_next :

MACHINE Tickets
VARIABLES *serve*, *next*
INVARIANT $serve \in \mathbb{N} \;\wedge\; next \in \mathbb{N} \;\wedge\; serve \leq next$
INITIALISATION...
OPERATIONS
$ss \leftarrow serve\_next \;\widehat{=}$
  PRE $serve < next$ $(precondition)$
  THEN $ss, serve := serve + 1, serve + 1$ $(body)$
  END
$tt \leftarrow take\_next$
END

Modification de Tickets et violation de l'invariant :

MACHINE Tickets
VARIABLES $serve, next$
INVARIANT $serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge serve \leq next$
INITIALISATION...
OPERATIONS
$ss \leftarrow serve\_next \ \widehat{=}$
  PRE $True$ ($precondition\ trop\ faible\ !$)
  THEN $ss, serve := serve + 1, serve + 1$ ($body$)
  END
$tt \leftarrow take\_next$
END

Abstract Machine : OPERATIONS 4

Completons avec take_next la machine correcte :

MACHINE Tickets
VARIABLES *serve*, *next*
INVARIANT $serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge serve \leq next$
INITIALISATION...
OPERATIONS
$ss \leftarrow serve\_next \ \widehat{=}$
  PRE $serve < next$ $\quad (precondition)$
  THEN $ss, serve := serve + 1, serve + 1$ $\quad (body)$
  END
$tt \leftarrow take\_next \ \widehat{=}$
  PRE $True$ $\quad (precondition)$
  THEN $tt, next := next, next + 1$ $\quad (body)$
END

NB : PRE true peut même être omise

Valeurs initiales pour les variables de VARIABLES $\leadsto$ l'état initial de la machine.

MACHINE Tickets
VARIABLES $serve, next$
INVARIANT $serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge serve \leq next$
INITIALISATION $serve, next := 0, 0$ **OPERATIONS**
$ss \leftarrow serve\_next \ \widehat{=}$
PRE $serve < next$ $(precondition)$
THEN $ss, serve := serve + 1, serve + 1$ $(body)$
END
$tt \leftarrow take\_next \ \widehat{=}$
PRE $True$ $(precondition)$
THEN $tt, next := next, next + 1$ $(body)$
END

Utilisent des notations ensemblistes et des notations logiques

Notations Ensemblistes

PETITS_PAIRS $= \{0, 2, 4, 6\}$
PETITS_PAIRS $= \{x \mid x \in \mathbb{N}$ et x est pair et $x < 8\}$
DE_0_à_5$=$ $0..5 = \{0, 1, 2, 3, 4, 5\}$.

Pour E et E' qui sont des ensembles :

$E \subset E'$, $E \subseteq E'$
$E \cup E'$, $E \cap E'$, $E - E'$
$E \times E'$, $\mathcal{P}(E)$
$\mid E \mid \ = \ card(E) =$ taille de E

Logique des prédicats (LP), 1

- Alphabet pour les TERMES de LP :
  - un ensemble infini dénombrable de variables :
  $X = \{x_1, x_2, x_3, ...\}$,
  - un ensemble C de constantes,
  - un ensemble F de symboles de fonction, chacun muni de son nombre $n > 0$ d'arguments,
  - les parenthèses () et la virgule.
- Grammaire pour le langage des termes :

$$T := v \mid c \mid f(\underbrace{T, ..., T}_{n})$$

où $v \in X$, $c \in C$, $f \in F$ et a n arguments, et chaque argument T est un terme.
NB : définition récursive.

Notation infixe : si $f \in F$ a 2 arguments, on écrira $x \ f \ y$ à la place de f(x,y).

Logique des prédicats (LP), 2

- Alphabet pour les FORMULES ATOMIQUES de LP :
  on ajoute à l'alphabet des termes un ensemble R de symboles de relation (ou prédicats), chacun muni de son nombre m d'arguments
- Grammaire pour le langage des formules atomiques (atomes) :

$$ATOMES := True \mid False \mid r(\underbrace{T, ..., T}_{m})$$

  où $r \in R$ et a m arguments, et chaque argument T est un terme.

Notation infixe : si $r \in R$ a 2 arguments, on écrira $x\ r\ y$ à la place de r(x,y)

Logique des prédicats (LP), 3

- Alphabet pour les FORMULES de LP :
  on ajoute à l'alphabet des formules atomiques les connecteurs booléens $\neg, \wedge, \vee, \Rightarrow$ et les quantificateurs : existentiel $\exists$ et universel $\forall$.
- Grammaire pour le langage des formules :

$$F := A \mid (\neg F) \mid (F \wedge F) \mid (F \vee F) \mid (F \Rightarrow F) \mid (Qv\ F)$$

où $A \in ATOMES$, $v \in X$, $Q \in \forall, \exists$ est un quantificateur et F est une formule.

NB : définition récursive.

Exemples de formules de la logique des prédicats et leur sémantique :
au tableau.

Logique des prédicats (LP), 5

Etant donnée une formule de la forme $\exists x_i F$ ou $\forall x_i F$, F est la portée du quantificateur.

Une occurrence d'une variable est dite libre si elle n'est pas dans la portée d'un quantificateur, et elle dite liée (par le quantificateur) sinon.

Ex.: pour la formule $(\forall x_1(\exists x_2(x_1 < x_2))) \wedge pair(x_1)$,
la première occurrence de x_1 est liée par la quantification $\forall x_1$,
la deuxième occurrence de x_1 est libre.

Les occurrences des variables liées peuvent être renommées :
$(\forall x_1(\exists x_2(x_1 < x_2))) \wedge pair(x_1) \quad \equiv (\forall x_3(\exists x_2(x_3 < x_2))) \wedge pair(x_1)$

Convention pour épargner des () :

1. Droit de ne pas écrire les () le plus exterieurs.
2. Droit d'écrire $A \wedge B \wedge C$ à la place de $A \wedge (B \wedge C)$ ou $(A \wedge B) \wedge C$ (et idem pour le $\vee$). Justification : associativité des fonctions booléennes $\wedge$ et $\vee$
3. Droit d'écrire $Q_1 v_1 Q_2 v_2\ F$ à la place de $Q_1 v_1 (Q_2 v_2\ F)$, où Q_i est un quantificateur.
   Par ex., droit d'écrire $\forall x_1 \exists x_2\ x_1 < x_2$ à la place de $\forall x_1 (\exists x_2\ x_1 < x_2)$.

Substitution, 1

Si E est une expression (un terme) et F est une formule, la formule obtenue à partir de F en remplaçant toute occurrence libre de la variable v par E est notée : $F[E/v]$. On lit : "F avec E à la place de la variable v.

Ex. : pour $F = x_1 < x_2$ (où $<$ est un symbole de relation à 2 arguments), on a :

$F[2/x_1]$=$x_1 < x_2[2/x_1]$= $2 < x_2$

La notation "substitutions'multiples" est permise :
$F[2, 3/x_1, x_2] = x_1 < x_2[2, 3/x_1, x_2] = 2 < 3$
NB : substitutions en //.

Substitution,2

$members \subseteq PERSON[(members \cup new)/members] =$
$members \cup new \subseteq PERSON$

$\exists p(p \in PERSON \;\wedge\; age(p) > limit)[oldimit + 2/limit] =$
$\exists p(p \in PERSON \;\wedge\; age(p) > oldlimit + 2)$

NB : $\exists n(n \in \mathbb{N} \;\wedge\; n > limit)[n + 3/limit] \neq$
$\exists n(n \in \mathbb{N} \;\wedge\; n > n + 3)$!
Pourquoi ?
Comment réécrire ?

Espace d'états

C: collection de variables, avec leur types. L'espace d'états associé à C est l'ensemble des combinaisons que ces variables peuvent prendre.
Par ex. si $C = \{x, y\}$ avec types : $x \in \{0, 1, 2\}$ et $y \in \{0, 1, 2\}$, l'espace d'états associé contient 9 couples.

Un énoncé AMN décrit une transformation d'états. Ex : l'affectation $y := max\{0, y - x\}$ associe à chacun de ces 9 états un nouvel état (figure au tableau).

Postcondition

Si P est une formule qui décrit un ensemble d'états qui peuvent être atteints après l'éxécution d'un énoncé AMN S (une instruction), alors P est une postcondition de S.

Weakest Precondition, 1

Soit P' une postcondition qu'on veut obtenir après l'éxécution d'un S (par ex. un invariant).
Il est important d'identifier quel est l'ensemble le plus large d'états qui assurent que l'éxécution de S aménera à des états vérifiant P.

La notation $[S]P$ indique une formule qui décrit cet ensemble d'états. C'est la *weakest precondition* (la condition la plus faible) afin que S puisse améner à P : elle vaut pour tous les états qui certainement arriveront à P par le biais de S !

Weakest Precondition, 2

Dans l'exemple précedént de $C = \{x, y\}$ avec $x, y \in \{0, 1, 2\}$:

$$\underbrace{[y := max\{0, y - x)\}}_{S}\underbrace{(y > 0)}_{P} =$$

$$(x = 0 \ \wedge \ y = 1) \ \vee \ (x = 0 \ \wedge \ y = 2) \ \vee \ (x = 1 \ \wedge \ y = 2)$$

NB :
$(x = 0 \ \wedge \ y = 1) \ \vee \ (x = 0 \ \wedge \ y = 2)$ serait une precondition assurant P, mais pas la plus faible ! Elle interdirait l'état initial $(x = 1 \ \wedge \ y = 2)$.
En revanche, *True* serait une precondition trop faible et elle n'assurerait pas P.

Lois pour $[S]P$

- $[S](P \wedge Q) \ = \ [S]P \ \wedge \ [S]Q$
- $[S]P \ \vee \ [S]Q \ \Rightarrow \ [S](P \vee Q)$ est valide
  NB : L'implication réciproque ne l'est pas !
  Penser, intuitivement, à : S = lancer une pièce, P = obtenir pile, Q= obtenir face. Alors $[S](P \vee Q) = True$, car $P \ vee \ Q \ = \ True$, mais $[S](P) \neq True$ et $[S](Q) \neq True$!
- Si $P \Rightarrow Q$ est vraie, alors $[S]P \ \Rightarrow \ [S]Q$ aussi.
  NB : Pas vrai que $[S]P = [S]Q$.Il y a juste implication.

Calcul de la weakest precondition

Etant donné S et P on a des règles permettant de calculer $[S]P$

R1= règle pour l'**affectation** :

$$[x := E]P \;=\; P[E/x]$$

Exemple : machine Ticket, P est $serve \leq next$ (partie de l'invariant), S est $serve := serve + 1$.

$[S]P=$
$[serve := serve + 1](serve \leq next) =$ (par R1)
$(serve \leq next)[serve + 1/serve] =$ (substitution)
$serve + 1 \leq next$
qui, par l'arithmétique, est équivalent à : $serve < next$.

Calcul de la weakest precondition

R1G : Généralisation de R1 aux **affectations multiples (en //)**

$$[x_1, \cdots, x_n := E_1, \cdots, E_n]P \; = \; P[E_1, \cdots, E_n/x_1, \cdots, x_n]$$

où si $i \neq j$ alors $x_i \neq x_j$

Exemple

$[serve, next := serve + 1, next - 1](serve \leq next) =$ (R1G)
$(serve + 1 \leq next - 1) =$ c'à.d. :
$serve + 2 \leq next$

Signification : l'invariant est preservé à coûp sûr par cette affectation seulement si on part d'un état où $serve + 2 \leq next$ est vrai.

L'expression AMN *skip* = affectation vide (**ne rien faire**).

R2 : Règle pour *skip*

$$[skip]P = P$$

Calcul de la weakest precondition

R3= règle pour le conditionnel :

S a la forme *IF E THEN S1 ELSE S2 END*, où E est une formule qui s'évalue à vrai ou faux, et $S1$, $S2$ sont elles-mêmes des instructions.

$$[\mathit{IF}\ E\ \mathit{THEN}\ S1\ \mathit{ELSE}\ S2\ \mathit{END}] \ = \ (E \wedge [S1]P) \vee (\neg E \wedge [S2]P)$$

Calcul de la weakest precondition

Exemple pour R3 combinée avec R1

$$\underbrace{[IF\ x < 5\ THEN\ x := x + 4\ ELSE\ x := x - 3\ END]}_{S}\underbrace{(x < 7)}_{P} =_{R3}$$

$$(x < 5\ \wedge\ \underbrace{[x := x + 4]}_{S1}\underbrace{(x < 7)}_{P})\ \vee\ (\neg(x < 5)\ \wedge\ \underbrace{[x := x - 3]}_{S2}\underbrace{(x < 7)}_{P} =_{R1}$$

$$(x < 5\ \wedge\ (x + 4) < 7)\ \vee\ (x \geq 5\ \wedge\ (x - 3) < 7) = (\text{arithm})$$

$$(x < 5\ \wedge\ x < 3)\ \vee\ 5 \leq x < 10 \text{ c.à.d}$$

$$x < 3\ \vee\ 5 \leq x < 10$$

Calcul de la weakest precondition

Cas particulier de R3

$$[IF\ E\ THEN\ S\ END]P \quad = \quad (E \wedge [S]P) \vee (\neg E \wedge P)$$

Calcul de la weakest precondition

Instruction *CASE OF*. Format

CASE E OF
EITHER e_1 *THEN* T_1
OR e_2 *THEN* T_2
OR...
OR e_n *THEN* T_n
ELSE V
END

NB : *ELSE* optionnel : si absent, ne pas changer l'état si aucun des n cas est applicable. Exemple :
CASE dir OF
EITHER north THEN partner $:=$ *south*
OR south THEN partner $:=$ *north*
OR east THEN partner $:=$ *west*
OR west THEN partner $:=$ *east*
END

Calcul de la weakest precondition

Instruction *CASE OF* : et si nombre infini de cas ? Couvrir les $n + i$ cas avec le *ELSE*.

Exemple

$CASE\ sizeoforder\ OF$
$EITHER\ 0\ THEN\ discount := 0$
$OR\ 1\ THEN\ discount := 0$
$OR\ 2\ THEN\ discount := 5$
$OR\ 3\ THEN\ discount := 10$
$ELSE\ discount := 15$
END

Calcul de la weakest precondition

R4 : Règle pour *CASE OF*

$$\begin{bmatrix} CASE & E & OF & \\ EITHER & e_1 & THEN & T_1 \\ OR & e_2 & THEN & T_2 \\ OR & \dots & & \\ EITHER & e_n & THEN & T_n \\ ELSE & V & & \\ END & & & \end{bmatrix} P =$$

$$(E = e_1 \;\Rightarrow\; [T_1]P) \wedge$$
$$(E = e_2 \;\Rightarrow\; [T_2]P) \wedge$$
$$\dots \wedge$$
$$(E = e_n \;\Rightarrow\; [T_n]P) \wedge$$
$$((E \neq e_1 \;\wedge\; E \neq e_2 \;\wedge\; \cdots \;\wedge\; E \neq e_n) \;\Rightarrow\; [V]P)$$

$$[BEGIN\ S\ END] = [S]P$$

C'est juste une notation permettant de mettre des parenthèses !

La AM M est-elle cohérente ?

Si oui, certaines conditions doivent être respectées.

Chacune engendre une *obligation de preuve*.

Format de M consideré ci-dessous :

MACHINE M
VARIABLES v
INVARIANT I
INTIALISATION T
OPERATIONS
  $y \leftarrow op(x) =$
    PRE P
    THEN S
    END;

...
END

Cohérence d'une AM

Il faut que l'invariant I soit satisfiable : des états légitimes de la machine pour lesquels I est vrai **doivent exister**.

C_INV : $\exists v \; I$ doit être vraie.
(Variables v : celles déclarées dans la partie "VARIABLES" de M)

Obligation de preuve : vérifier que des valeurs adéquates pour les v existent.

Cohérence d'une AM

Exemple de la AM **Tickets** :

C_INV :
$\exists serve\ \exists next\ (serve \in \mathbb{N}\ \wedge\ next \in \mathbb{N}\ \wedge\ serve \leq next)$
doit être un énoncé valide de l'arithmétique (standard).

Obligation de preuve : vérifier que $n, m \in \mathbb{N}$ tels que $n \leq m$ existent.
Ici, banal : par ex., $n = m = 0$.

NB : En général, utilité de méthodes de preuve automatique ou des assistants de preuve !

Cohérence d'une AM

Il faut que l'état T déclaré (par une affectation) dans la partie INITIALISATION de M respecte l'invariant I

C_INIT : $[T]I$ doit être toujours vraie.

Obligation de preuve : prouver la formule $[T]I$.

Exemple pour Tickets

$[T]I =$

$[serve, next := 0, 0](serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge \ serve \leq next)) =$

(par R1)

$(0 \in \mathbb{N} \ \wedge \ 0 \in \mathbb{N} \ \wedge \ 0 \leq 0)$

OK !

Cohérence d'une AM

Deux autres initialisations possibles (?) pour Tickets.

1. $T = serve, next := 1, 0$. Alors : $[T]I =$ $(0 \in \mathbb{N} \ \wedge \ 1 \in \mathbb{N} \ \wedge \ 1 \leq 0))$. False !
2. $T = (serve, next := serve + 1, next + 1)$. Alors : $[T]I =$ $(serve + 1 \in \mathbb{N} \ \wedge \ next + 1 \in \mathbb{N} \ \wedge \ serve + 1 \leq next + 1)) =$??

   Signification ? Tickets démarre dans un état aléatoire, avant de faire T ! Donc faux (cas ci-dessus, par ex.).

Cohérence d'une AM

Soit *op* une opération avec PRE=P= et BODY=S.

On effectue S seulement si P.
On suppose que l'invariant I est vrai dans l'état où on effecue I;

Sous ces conditions, effectuer *op* se réduit à faire S, et cette action doit préserver I.

C_OP : $(I \ \wedge \ P) \Rightarrow [S]I$ doit être toujours vraie.

$\rightsquigarrow$ obligation de prouver $(I \ \wedge \ P) \Rightarrow [S]I$

Cohérence d'une AM

Exemple de C_OP pour Tickets, avec op = serve_next.
Il faut prouver :

$$(\underbrace{(serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge \ serve \leq next)}_{I} \ \wedge \ \underbrace{serve < next}_{P})$$

$$\Rightarrow$$

$$\underbrace{[ss, serve := serve + 1, next + 1](serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge \ serve \leq next)}_{[S]I}$$

c.à.d., en utilisant R1 pour calculer $[S]I$:

$$(\underbrace{(serve \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge \ serve \leq next)}_{I} \ \wedge \ \underbrace{serve < next}_{P})$$

$$\Rightarrow$$

$$(serve + 1 \in \mathbb{N} \ \wedge \ next \in \mathbb{N} \ \wedge \ serve + 1 \leq next).$$

Un théorème arithmétique !

Cohérence d'une AM

Encore un exemple de AM, qui doit être cohérente.

MACHINE PaperRounds
VARIABLES $papers, magazines$
INVARIANT $papers \subseteq 1..163 \ \wedge \ magazines \subseteq papers \ \wedge \ card(papers) \leq 60$
INTIALISATION $papers, magazines := \emptyset, \emptyset$
OPERATIONS
addpaper $(hh) \widehat{=}$ % pas de output
PRE $hh \in 1..163 \ \wedge \ card(papers) < 60$
THEN $papers := papers \cup \{hh\}$
END;
addmagazine $(hh) \widehat{=}$ % pas de output
PRE $hh \in papers$
THEN $magazines := magazines \cup \{hh\}$
END;
remove $(hh) \widehat{=}$ % pas de output
PRE $hh \in 1..163$
THEN $papers, magazines := papers - \{hh\}, magazines - \{hh\}$
END;
END

Cohérence d'une AM

Obligation de preuve pour C_INV. Prouver :

$$\exists papers\ \exists magazines\ (\\ papers \subseteq 1..163 \wedge \\ magazines \subseteq papers \wedge \\ card(papers) \leq 60)$$

Vrai, car : $\emptyset \subseteq 1..163\ \wedge \emptyset \subseteq papers\ \wedge card(\emptyset) \leq 60)$

ce qui prouve aussi C_INIT.

Obligation de preuve pour C_OP, avec $op = addpapers$. Prouver :

$$(\underbrace{(papers \subseteq 1..163 \ \wedge \ magazines \subseteq papers \ \wedge \ card(papers) \leq 60)}_{I} \ \wedge$$

$$\underbrace{hh \in 1..163 \ \wedge \ card(papers) < 60)}_{P})$$

$$\Rightarrow$$

$papers \cup \{hh\} \subseteq 1..163 \ \wedge \ magazines \subseteq papers \cup \{hh\} \wedge$
$card(papers \cup \{hh\}) \leq 60)$

où R1 a été utilisée pour simplifier le conséquent de l'implication, c.à.d. $[S]I$.

Théorème arithmetique (et ensembliste) !

Cohérence d'une AM

De façon analogue, il faut analyser addmagazine et remove.

Cohérence d'une AM

Raisons possible de l'incohérence d'une AM :

1. La PRE d'une opération est trop faible.
2. Le BODY est incorrect (ne preserve pas l'invariant désiré).
3. C'est I qui est trop restrictive (exclut des états OK).
4. Au contraire, I est trop permissive : permet d'atteindre des "mauvais" états.

Modification de I à cause de 3 ou 4 : il faut refaire <u>toutes</u> les obligations de preuve.

Machines plus expressives

On va étudier l'usage de :

- paramètres $\leadsto$ machines génériques
- constantes (analogues aux constantes d'un programme)
- ensembles $\leadsto$ types abstraits

dont l'implémentation va être différée.

On va utiliser l'exemple d'une machine Club.

Machine Club

MACHINE Club $(capacity)$
CONSTRAINTS $capacity \in \mathbb{N}_1 \;\wedge\; capacity \leq 4096$
SETS $REPORT = \{yes, no\}$ **;** $NAME$
CONSTANTS $total$
PROPERTIES $card(NAME) > capacity \;\wedge\; total \in \mathbb{N}_1 \;\wedge\; total > 4096$
VARIABLES $member, waiting$
INVARIANT
$member \subseteq NAME \;\wedge\; waiting \subseteq NAME \;\wedge\; member \cap waiting = \emptyset$
$\wedge\; card(member) \leq 4096 \;\wedge\; card(waiting) \leq total$
INITIALISATION $member, waiting := \emptyset, \emptyset$
OPERATIONS
$join(nn) \widehat{=}$
PRE $nn \in waiting \;\wedge\; card(member) < capacity$
THEN $member, waiting := member \cup \{nn\}, member \setminus \{nn\}$
END ;
$join\_queue(nn) \widehat{=}$
PRE $nn \in NAME \;\wedge\; nn \notin member \;\wedge\; nn \notin waiting \;\wedge\; card(waiting) < total$
THEN $waiting := waiting \cup \{nn\}$
END ;
$remove(nn) \widehat{=}$
PRE $nn \in member$
THEN $member := member \setminus \{nn\}$
END ;
$semi\_reset \widehat{=} member, waiting := \emptyset, member$ **;**
$ans \leftarrow query\_membership \widehat{=}$
PRE $nn \in NAME$
THEN
IF $nn \in member$
THEN $ans := yes$
ELSE $ans := no$
END
END
END

MACHINE Club (*capacity*)

capacity est un paramètre

Les paramètres sont donnés après le nom de la machine.

Deux sortes de paramètres : à valeur scalaire, comme pour Club, ou ensembliste (alors en majuscules) :

MACHINE Store (*ITEM*)
VARIABLES *elements*
INVARIANT $elements \subseteq ITEM$
INTIALISATION $elements := \emptyset$
OPERATIONS
    input (ii)$\widehat{=}$
        PRE $ii \in ITEM$
        THEN $elements := elements \cup \{ii\}$
        END ;
....
END

On peut utiliser un paramètre ensembliste comme type (généricité) et il faudra l'instancier avec un ensemble non-vide.

Le type d'un paramètre est donnée dans la clause CONSTRAINTS.

MACHINE Club (*capacity*)
CONSTRAINTS $capacity \in \mathbb{N}_1 \ \wedge \ capacity \leq 4096$

Dans CONSTRAINTS, les types des paramètres scalaires et autres contraintes sur les paramètres.
Même relation avec les paramètres que INVARIANT par rapport aux variables de la machine.
Mais on ne peut pas contraindre le type des paramètres ensemblistes !!

Erreurs :

MACHINE Store (*ITEM*)
CONSTRAINTS $ITEM \subseteq \mathbb{N}$

MACHINE Bijouterie ($PIERRE, METAL$)
CONSTRAINTS $PIERRE \cap METAL = \emptyset$

MACHINE Club $(capacity)$

...

SETS $REPORT = \{yes, no\}$ **;** $NAME$

Un type ensembliste générique peut aussi être introduit dans SETS (en majuscules). Alors :

1. On peut le nommer sans donner d'autre information;
2. On peut aussi l'expliciter comme type *énuméré*, comme $REPORT$ dans la machine Club;
3. On peut l'introduire comme abbreviation pour un sous-type (par ex. PAIR par rapport à $\mathbb{N}$).

MACHINE Club (*capacity*)

....

CONSTANTS *total*

PROPERTIES

$card(NAME) > capacity \ \wedge \ total \in \mathbb{N}_1 \ \wedge \ total > 4096$

- Analogie : constantes globales dans un programme.
- Type : tout type connu par la machine, et on l'indique dans la clause PROPERTIES.
- Dans PROPERTIES on peut aussi indiquer des relations entre des SETS et des paramèters (par ex. entre *NAME* et *capacity*).

Constantes, 2

Différence entre le paramètre *capacity* et la constante *total* ?

On doit pouvoir instancier la machine à *n'importe lequel* nombre naturel valeur de *capacity* (généricité) et *au moins une* valeur appropriée de *total*.

Pour valider les PROPERTIES il faut prouver une formule de la forme :
$\forall capacity\ \exists total\ \ldots$

Voir après les obligations de preuve associées à la clause PROPERTIES.

Opérations de Requête

MACHINE Club (*capacity*)

...

OPERATIONS

...

$ans \leftarrow query\_membership(nn) \widehat{=}$

PRE $nn \in NAME$

THEN

IF $nn \in member$

THEN $ans := yes$

ELSE $ans := no$

END

END

....

query_membership : *opération de requête* : produit une information sur l'état de la machine mais ne change pas l'état de la machine.

Nouvelles Obligations de Preuve, 1

Pour une clause CONSTRAINTS $C(p)$ avec paramètres p, on doit démontrer : $\exists p \; C(p)$

Pour Club :
$\exists capacity(capacity \in \mathbb{N}_1 \;\wedge\; capacity \leq 4096)$

Puis : toujours instancier un paramètre ensembliste à un ensemble $\neq \emptyset$!

Nouvelles Obligations de Preuve, 2

Pour une clause PROPERTIES $Prop(p, s, c)$ avec paramètres p, constantes c et ensembles (sets) s on doit démontrer que : $\forall$ valeur de p qui satisfait les contraintes $C(p)$, il existe des valeurs pour s et c tels que $Prop(p, s, c)$ vaut vrai

Obligation : prouver le théorème : $C(p) \Rightarrow (\exists c\ \exists s\ Prop(p, s, c))$

N.B. Point de vue logique : paramètre p comme une variable libre x, et prouver $\forall x(A(x) \Rightarrow \exists y(B(x, y))$ revient à prouver $A(x) \Rightarrow \exists y(B(x, y))$.

Pour Club, il faut prouver :

$$(capacity \in \mathbb{N}_1 \ \wedge \ capacity \leq 4096) \Rightarrow \exists NAME\ \exists total$$

$$(card(NAME) > capacity \ \wedge \ total \in \mathbb{N}_1 \ \wedge \ total > 4096)$$

Nouvelles Obligations de Preuve, 3

Il faut prouver :
l'INVARIANT $I(v)$ contenant les variables v est est satisfiable par des valeurs de v, et cela pour toutes les valeurs des paramètres p, des constantes c et des ensembles (sets) s telles que les contraintes $C(p)$ et les PROPERTIES $Prop(p, s, c)$ sont vraies.

Obligation de preuve de :

$$(Prop(p, s, c) \ \wedge \ C(p)) \ \Rightarrow \ \exists v \ I$$

Pour Club, prouver :
$(card(NAME) > capacity \ \wedge \ total \in \mathbb{N}_1 \ \wedge \ total > 4096 \wedge$
$capacity \in \mathbb{N}_1 \ \wedge \ capacity \leq 4096) \Rightarrow$

$$\exists member \ \exists waiting$$

$$\begin{aligned}(&member \subseteq NAME \ \wedge \ waiting \subseteq NAME \wedge \\ &member \cap waiting = \emptyset \\ &\wedge \ card(member) \leq 4096 \ \wedge \ card(waiting) \leq total)\end{aligned}$$

Nouvelles Obligations de Preuve, 4

Il faut prouver : l'invariant $I(v)$ contenant les variables v est préservé par l'exécution T de l'INITIALISATION, et cela pour toutes les valeurs des paramètres p, des constantes c, des ensembles (sets) s et des variables v) telles que les contraintes $C(p)$ et les *properties* $Prop(p, s, c)$ sont vraies.

Obligation de preuve de :

$$(Prop(p, s, c) \ \wedge \ C(p)) \ \Rightarrow \ [T]I$$

Pour Club ceci signifie prouver (par R1) :
$(card(NAME) > capacity \ \wedge \ total \in \mathbb{N}_1 \ \wedge \ total > 4096 \wedge$
$capacity \in \mathbb{N}_1 \ \wedge \ capacity \leq 4096) \Rightarrow$
$(\emptyset \subseteq NAME \ \wedge \ \emptyset \subseteq NAME \ \wedge \ \emptyset \cap \emptyset = \emptyset \ \wedge \ card(\emptyset) \leq 4096$
$\wedge \ card(\emptyset) \leq total)$

Pour toute opération $op \widehat{=}$ **PRE** *prec* **THEN** S **END** il faut prouver :
l'invariant $I(v)$ est préservé par l'éxécution de S, et cela pour toutes les valeurs des paramètres, des constantes, des ensembles (sets) et des variables telles que les contraintes $C(p)$, les *properties* $Prop(p, s, c)$ et la *prec* de *op* sont satisfaites.

Pour toute opération $op \widehat{=}$ **PRE** *prec* **THEN** S **END** on a l'obligation de preuve de :

$$(Prop(p, s, c) \ \wedge \ C(p) \ \wedge \ I \wedge prec) \ \Rightarrow \ [S]I$$

Obligations de preuve pour les opérations de Club : au tableau.

Nouvelles Obligations de Preuve, 6

Cas particulier des opérations de requête : elles ne modifient pas les variables de VARIABLES –les seules variables de l'invariant –.

Soit *op* une opération de requête qui execute S. Puisque $[S]I = I$, cohérence banale avec l'invariant I :
$(Prop(p, s, c) \ \wedge \ C(p) \ \wedge \ I \wedge prec) \ \Rightarrow \ [S]I$ est une tautologie.

Pas d'obligations de preuve !

Visibilité entre Clauses d'une AM

- CONSTRAINTS voit les paramètres;
- PROPERTIES voit les paramètres, les ensembles (sets) et les constantes;
- INVARIANT voit les paramètres, les ensembles (sets), les constantes et les variables de VARIABLES;
- OPERATIONS voit les paramètres, les ensembles (sets), les constantes et les variables de VARIABLES;

Relations

Une AM peut utiliser des variables (déclarées dans VARIABLES) et/ou des constantes (déclarées dans CONSTANTS) pour des relations.

Une relation (binaire) r entre l'ensemble S et l'ensemble T est un sous-ensemble de $S \times T$, donc un ensemble de couples.

Le type d'une variable x pour une relation est donné dans l'INVARIANT, et celui d'une constante c dans PROPERTIES.
Notation : $x \in S \leftrightarrow T$, $c \in S \leftrightarrow T$.
N.B. : le type de x et c est un ensemble (de couples). La méthode B sait raisonner en logique et en théorie des ensembles !

Si r est une relation $\in S \leftrightarrow T$ et $U \subseteq S$, la notation $r[U]$ indique ces éléments de T qui sont reliés à des éléments de U(*image relationnelle* de U par r).
Ex: si r est $>$ sur $\mathbb{N}$, alors $r[\{4\}] = \{0, 1, 2, 3\}$.

Une fonction partielle f de S à T est une relation $\in S \leftrightarrow T$ telle que $\forall s \in S$ $\exists$ au maximum un $t \in T$ tel que $(s, t) \in f$.

Domaine de f $=$ le plus grand sous-ensemble S' de S tel que, $\forall s \in S'$ $\exists! t \in T$ tel que $(s, t) \in f$. Notation : $dom(f)$

Ensemble d'arrivé de f $=$ le plus grand sous-ensemble de T contenant des éléments t tels que $(s, t) \in f$ (pour quelques $s \in dom(f)$). Notation : $ran(f)$ (*range* de f).

Une fonction totale f de S à T est une fonction partielle de S à T telle que le domaine de f est S, c'est-à-dire que $\forall s \in S$ $\exists! t \in T$ tel que $(s, t) \in f$.

Fonctions, 2

Notation	Signification
$f \in S \nrightarrow T$	f est une fonction (partielle) de S à T
$f \in S \rightarrow T$	f est une fonction <u>totale</u> de S à T
$f \in S \not\rightarrowtail T$	f est une fonction (partielle) et <u>injective</u> de S à T
$f \in S \rightarrowtail T$	f est une fonction totale et <u>injective</u> de S à T
$f \in S \twoheadrightarrow T$	f est une fonction totale et <u>sujective</u> de S à T
$f; f'$	la fonction composée de f par f' ($f' \circ f$)
$dom(f)$	le domaine de la fonction f
$ran(f)$	l'ensemble d'arrivé de la fonction f
$f[U]$	l'image de $U \subseteq S$ par f.

Exemple

Machine utilisant déclarations de relations et fonctions : Reading.

Partie Statique

MACHINE Reading
SETS $READER$ **;** $BOOK$ **;** $COPY$ **;** $RESPONSE \ = \ \{yes, no\}$
CONSTANTS $copyof$
PROPERTIES $copyof \in COPY \twoheadrightarrow BOOK$
VARIABLES $hasread, reading$
INVARIANT

$hasread \in READER \leftrightarrow BOOK$ % $hasread$ relation
$\wedge \ reading \in READER \nrightarrow COPY$
$\wedge \ (reading; copyof) \cap hasread = \emptyset$

INITIALISATION $hasread, reading := \emptyset, \emptyset$

Partie Dynamique de l'AM Reading : les Opérations, 1

L'opération *start* ajoute un nouveau couple ($lecteur, copie$) à la fonction *reading*, qui dit qui est en train de lire quoi.

$start(rr, cc) \widehat{=}$
PRE
$rr \in READER \wedge cc \in COPY \wedge copyof(cc) \notin hasread[\{rr\}] \wedge$
$rr \notin dom(reading) \wedge cc \notin ran(reading)$
THEN $reading := reading \cup \{rr \mapsto cc\}$

où :
- $a \mapsto b$ est une façon de noter le couple $(a, f(a))$ d'une relation (fonctions incluses)
- Rappel : $r[U]$ est l'image relationnelle de U, c'est-à-dire $\{t \mid t \in T \text{ et } \exists u \in S \ (s, t) \in r\}$.

Partie Dynamique de l'AM Reading : les Opérations, suite

L'opération *finished* fait si qu'un lecteur *rr* soit consideré comme ayant terminé de lire une copie *cc* d'un livre donné.

$finished(rr, cc) \widehat{=}$

PRE

$rr \in READER \ \wedge \ cc \in COPY \ \wedge \ cc = reading(rr)$

THEN $hasread := hasread \cup \{rr \mapsto copyof(cc)\} \ ||$

$reading := \{rr\} \ remove\_de \ \ reading$

où $||$ est une façon de noter deux actions faites en parallèle et, si E est un sous-ensemble du domaine d'une fonction f, alors $E \ remove\_de \ f$ note la fonction f privée des couples $(x, f(x))$ où $x \in E$.

Partie Dynamique de l'AM Reading : les Opérations, suite

L'opération de requête *precurrentquery*(*rr*) teste si *rr* est en train de lire un livre ou pas;
L'opération de requête *currentquery*(*bb*) produit *bb*, le livre que *rr* est en train de lire;
L'opération de requête *hasreadquery*(*rr*, *bb*) teste si *bb* est un livre déjà lu par *rr*.

$resp \longleftarrow precurrentquery(rr) \widehat{=}$
  PRE $rr \in READER$
  THEN
    IF $rr \in dom(reading)$ **THEN** $resp := yes$ **ELSE** $resp := no$
  END ;
$bb \longleftarrow currentquery(rr) \widehat{=}$
  PRE $rr \in READER \ \wedge \ rr \in dom(reading)$
  THEN $bb := copyof(reading(rr))$
  END ;
$resp \longleftarrow hasreadquery(rr, bb) \widehat{=}$
  PRE $rr \in READER \ \wedge \ bb \in BOOK$
  THEN
    IF $bb \in hasread[\{rr\}]$ **THEN** $resp := yes$ **ELSE** $resp := no$
  END ;

Tableaux

Dans la méthode B : un tableau avec indices dans I et valeurs dans V est vu comme une fonction partielle de I à V. N.B : la méhode B ne sait raisonner logiquement que sur les ensembles !

Exemples

Le tableau de taille 4 :

3	7	5	0

peut être vu comme une fonction partielle de $\mathbb{N}$ à $\mathbb{N}$, ou comme une fonction totale de 1..4 à $\mathbb{N}$ ou comme une fonction totale de 1..4 à $\{0, 1, ..., 17\}$ etc.

Le tableau vide existe, et c'est la fonction vide (pas de couples).

$\leadsto$ si on met à jour la valeur de la case i du tableau t, on écrit $t(i) := \mathit{nouvelle\_valeur}$ mais ce que l'on modifie c'est la fonction t, qui change de valeur pour l'argument i et reste identique pour tout autre argument.

Les affectations multiples dans le même tableau $(t(i), t(j) := 3, 4)$ sont interdites, car problème si $i = j$!

Tableaux : Weakest Precondition

Soit t un tableau, i un index, P une postcondition souhaitée après la mise à jour de $t(i)$. On obtient, comme cas particulier de la weakest précondition pour les affectations :

$$[t(i) := E]P = P[(t <\!+ \{i \mapsto E\}) \,/ t\}$$

où la fonction $t <\!+ \{i \mapsto E\}$ qui remplace t est définie par : $(t \ <\!+ \ \{i \mapsto E\})(j)$ est égal à E si $i = j$ et à $t(j)$ sinon.

Exemples

$[t(3) := 6](t(3) = 6) \rightsquigarrow_{R1} (t \ <\!+ \ \{3 \mapsto 6\})(3) = 6 \rightsquigarrow 6 = 6$ par déf. de $<\!+$.

$[t(4) := 6](t(3) = 6) \rightsquigarrow_{R1} (t \ <\!+ \ \{4 \mapsto 6\})(3) = 6 \rightsquigarrow t(3) = 6$ par déf. de $<\!+$. Cet énoncé sera vrai si, déjà, $t(3) = 6$ était vrai avant l'affectation.

Ici, noté par $\rightsquigarrow$ la réécriture due à $R1$ ou aux définitions.

Swap et Σ

Les affectations multiples modifiant $t(i)$ et $t(j)$ sont interdites, en général, mais le swap, qui échange les valeurs de $t(i)$ et $t(j)$, est OK.

swap : $\quad t := t \ <+ \ \{i \mapsto a(j), j \mapsto a(i)\}$
où : $(t \ <+ \ \{i \mapsto t(j), j \mapsto a(i\})(n)$ est $t(j)$ si $n = i$, est $t(i)$ si $n = j$ et c'est $t(n)$ sinon.
Pourquoi le cas $i = j$ ne pose pas de problèmes, ici ?

Notation $\Sigma x.(P(x) \ | \ E(x) =)$: somme de toutes les valeurs de $E(x)$ pour lesquelles $P(x)$ est vraie.

Si t est un tableau ayant les indices dans $1..\mathbb{N}$, la notation

$$\Sigma i.(i \in 1..\mathbb{N} \ | \ t(i))$$

signifie la somme de tous les valeurs du tableau t.

Machine Hotelguests

MACHINE Hotelguests (sze)
CONSTRAINTS $sze \in \mathbb{N}_1$
SETS $ROOM$; $NAME$; $REPORT = \{present, nopresent\}$
CONSTANTS $empty$ % nom d'un client inexistant
PROPERTIES $card(ROOM) = sze \ \wedge \ empty \in NAME$
VARIABLES $guests$
INVARIANT $guests \in ROOM \rightarrow NAME$ % $guests$ est un tableau
INITIALISATION $guests := ROOM \times \{EMPTY\}$
OPERATIONS
$guestcheckin(rr, nn) \widehat{=}$
PRE $rr \in ROOM \ \wedge \ nn \in NAME \ \wedge \ nn \neq empty$
THEN $guest(rr) = nn$
END ;
$guestcheckout(rr) \widehat{=}$
PRE $rr \in ROOM$
THEN $guests(rr) = empty$
END ;
$nn \leftarrow guestquery(rr) \widehat{=}$
PRE $rr \in ROOM$
THEN $nn = guests(rr)$
END
$rr \leftarrow presentquery(nn) \widehat{=}$
PRE $nn \in NAME \ \wedge \ nn \neq empty$
THEN
IF $nn \in ran(guests)$
THEN $rr := present$
ELSE $rr := nopresent$
END
END
$guestswap(rr, ss) \widehat{=}$
PRE $rr \in ROOM \ \wedge \ ss \in ROOM$
THEN $guests := guests \ <+ \{rr \mapsto guests(ss), ss \mapsto guests(rr)\}$
END

Indéterminisme

Toutes les constructions AMN vues jusqu'à ici étaient déterministes : une seule valeur des sorties possible pour une entrée donnée, un seul état final pour un état initial donné.

Mais utiliser de l'indéterminisme dans les spécifications est utile : liberté pour le programmeur, possibilité de retarder certain choix.

Indéterminisme = sous-spécification

La syntaxe AMN offre plusieurs opérateurs non-déterministes : **ANY**, **CHOICE**, **SELECT**, **PRE** (on verra après en quel sense **PRE** est indéterministe).

ANY 1

ANY x **WHERE** Q **THEN** T **END**

- x est une variable nouvelle et locale à l'énoncé ANY;
- Q est une formule qui donne le type de x et d'autres infos, et peut faire référence à d'autres variables;
- T le *body* de l'énoncé, est une n'importe quelle instruction AMN dont le résultat dépend de la valeur de x.

Sémantique : Une x arbitraire pour laquelle P vaut vrai est choisie, et T est éxécutée pour cette x.

Il faudra s'assurer que au moins une telle x existe.

Exemple 1 : "Enoncé de diminution"
ANY t **WHERE** $t \in \mathbb{N} \wedge t \leq total \wedge 2 \times t \geq total$ **THEN** $total := t$ **END**

Que se passe-t-il pour des états initiales où *total* est un nombre dans 1...6 ? Au tableau.

ANY 2

Exemple 2 : "Achats raisonnables"
ANY a **WHERE** $a \subseteq articles\_choisis \wedge prix(a) \leq limite$ **THEN** $achats := a$ **END**

On choisit un ensemble d'articles tels que le prix globale est au dessous d'une certaine limite.

Exemple 2 : Comment écrire une opération qui, étant donné un $x \in \mathbb{N}$, affecte la variable *div* à n'importe quel diviseur d de x ?

ANY 3

Weakest Precondition pour **ANY**

Soit la weakest precondition :
WP = [**ANY** x **WHERE** Q **THEN** T **END**] P
Cette precondition doit dire que peu importe la valeur de x telle que Q, l'execution de T assure P.

Donc WP $= \forall x \, (Q \Rightarrow [T]P)$

Exercice au tableau : calculer la weakest precondition pour l'exemple 1 (énoncé de diminution) quand P est $total > 1$:

[**ANY** t
WHERE $t \in \mathbb{N} \wedge t \leq total \wedge 2 \times t \geq total$
THEN $total := t$ **END**]
$(total > 1)$

Pour quelle condition sur l'état initial elle vaut vrai ?

ANY 4

La syntaxe AMN permet des énoncés de la forme :

LET x **BE** $x = E$ **IN** S **END**

Les occurrence de x doivent être évaluées par E, et alors S est executée.

Il s'agit d'une macro pour un cas particulier d'énoncé **ANY** : lequel ?

ANY 5

En général, un énoncé **ANY** peut utiliser une liste de variables :

ANY $x_1, ..., x_n$ **WHERE** Q **THEN** T **END**

et alors Q est une formule qui donne les types de toutes les x_i et d'autres infos, peut faire référence à d'autres variables et peut aussi spécifier des contraintes sur les combinaisons de valeurs permises (voir après l'exemple loto, version 2).

Donc la forme générale de la weakest precondition pour **ANY** est :

[**ANY** $x_1, ..., x_n$ **WHERE** Q **THEN** T **END**] $P =$
$\forall x_1 ... \forall x_n \ (Q \Rightarrow [T]P)$

Exemple : opération loto, version 1, avec 1 variable, TT, de type ensembliste

$SS \leftarrow loto \mathrel{\widehat{=}}$
PRE $True$
THEN ANY TT
    WHERE $TT \subseteq 1...49 \wedge card(TT) = 6$
    THEN $SS := TT$
    END
END

Exemple : opération loto, version 2, avec 6 variables, TT, de type $\mathbb{N}$

$a, b, c, d, e, f \leftarrow loto \quad \widehat{=}$
PRE *True*
THEN ANY $a0, b0, c0, d0, e0, f0$
WHERE $a0 \in 1...49 \wedge b0 \in 1...49 \wedge c0 \in 1...49 \wedge$
$d0 \in 1...49 \wedge e0 \in 1...49 \wedge f0 \in 1...49 \wedge$
$card(\{a0, b0, c0, d0, e0, f0\}) = 6$
THEN $a, b, c, d, e, f := a0, b0, c0, d0, e0, f0$
END
END

Les 6 variables du **ANY** sont contraintes à prendre des valeurs différentes entr elles.

CHOICE 1

On peut faire un choix parmi un nombre fixé d'alternatives :

CHOICE S **OR** T ... **OR** U **END**

Exemple 1 : test pour le permis de conduire

CHOICE $resultat := success \ || \ permis\_donnes := permis\_donnes \cup \{candidat\}$
OR $resultat := echec$

Exemple 2 : variation de l'exemple "achats raisonnables"

CHOICE
ANY a **WHERE** $a \subseteq articles\_choisis \wedge prix(a) \leq limite$ **THEN** $achats := a$ **END**
OR $limite := prix(articles\_choisis) + 100 \ || \ achats := articles\_choisis$

NB : indéterminisme dans les alternatives, aussi.

CHOICE 2

Weakest Precondition pour CHOICE

[**CHOICE** S_1 **OR**... **OR** S_n] $P = [S_1]\ P \wedge \ldots.\wedge\ [S_1]$

NB : c'est bien un AND !

Exercice au tableau :
Calculer la weakest precondition pour l'énoncé "test pour le permis de conduire" et la postcondition :
$P = permis\_donnes \subseteq bonne\_age$.
On suppose de stocker dans la variable *bonne_age* un ensemble de personnes ayant la bonne age pour conduire.
Que se passe-t-il si le candidat n'a pas l'age réquise et il échoue le test ?

SELECT 1

Chaque alternative est contrôlée par une condition permettant de la déclancher.

SELECT Q_1 **THEN** T_1
WHEN Q_2 **THEN** T_2
$\vdots$
WHEN Q_n **THEN** T_n
ELSE V
END

- Plusieurs Q_i peuvent être vraies, à un état. Alors une S_i quelconque parmi celles pouvant être déclanchés, est executée (indéterminisme !)
- Si aucune des S_i est vraie, c'est V qui est executée.
- Le **ELSE** est optionnel mais, si absent, alors les $Q_1, \ldots Q_n$ doivent couvrir tous les cas possibles !

CHOICE 2

Exemple 1 : choix d'un assistant

$a \leftarrow assistant \ \widehat{=}$
PRE *True*
THEN
  SELECT $anne \in presents$
    THEN $a := anne$
  WHEN $bernard \in presents$
    THEN $a := bernard$
  WHEN $tarek \in presents$
    THEN $a := tarek$
  ELSE $a := damien$
  END
END

Si plusieurs personnes présentes, choix indéterministe de l'assistant.

Exemple 2 : la position d'une pièce sur un échiquier 4×4 est representée par ses coordonnées (x, y) où $1 \leq x \leq 4$ et $1 \leq y \leq 4$. On bouge la pièce d'une carré à la fois, de façon horizontale ou verticale, mais on ne peut pas la sortir de l'échiquier.

SELECT $x > 1$ **THEN** $x{:=}x - 1$
WHEN $x < 4$ **THEN** $x{:=}x + 1$
WHEN $y > 1$ **THEN** $y{:=}y - 1$
WHEN $y < 4$ **THEN** $y{:=}y + 1$
END

Que se passe-t-il si $x = 2$ et $y = 4$?

Weakest Precondition pour SELECT

Le cas où **ELSE** est présent se réduit au cas où c'est absent, en ajoutant une autre clause de la forme **WHEN** $\neg Q_1 \wedge ... \neg Q_n$ **THEN** V.

Donc la formulation suivante est générale :

[**SELECT** Q_1 **THEN** T_1...**WHEN** Q_m **THEN** T_m **END**] P $=$
$Q_1 \Rightarrow [T_1]P \wedge \cdots \wedge Q_m \Rightarrow [T_m]P$

Exercice au tableau :
Calculer la weakest precondition pour l'exemple de l'échiquier et $P \ : \ y > 2$.

Exercice au tableau :
L'expression **IF** E **THEN** T est équivalente à un énoncé **SELECT**. Lequel ?

PRE

L'opérateur **PRE**, déjà vu pour indiquer la precondition d'une opération est indéterministe au sens que

PRE Q **THEN** S **END**

est tel que si Q est fausse alors toute exécution est permise, même une qui ne termine pas ! Si pas de terminaison, aucune postcondition est assurée, même pas *True* !

Donc :
R5 : $[\textbf{PRE } Q \textbf{ THEN } S \textbf{ END}]P = Q \wedge [S]P$
(une autre règle de calcul de la weakest precondition)

Structuration des machines et modularité

- Structurer une "grosse" machine en modules est un principe de base du GL;
- Reutiliser une machine dont la cohérence a été déjà prouvée comme composante d'une nouvelle machine est utile;
- Des méchanismes de structuration par rapport aux machines composantes de la "grande" machine permettent une forme d'abstraction dite "*semi-hiding*;
- $\rightsquigarrow$ Etude de quelques operateurs de structuration.

INCLUDES, 1

Une machine M1 peut être incluse dans M2 avec la clause :

INCLUDES M1

1. Si p est un paramètre de M1, p doit être istancié à une valeur v au moment de l'inclusion, et parmi les obligations de preuve pour M2 on aura la vérification que les CONSTRAINTS de M1 sont respectés par v;
2. Les SETS et les CONSTANTS de M1 sont visibles par M2, exactement comme les SETS et CONSTANTS "natifs" de M2. Les PROPERTIES de M2 peuvent les utiliser.

INCLUDES, 2

3 L'état de M1 devient une partie de l'état de M2 $\leadsto$ l'INVARIANT de M2 peut contraindre les variables d'état de M1, et leur relation avec les variables d'état natives de M2:

4 Les operations de M2 ont un acces de lecture à l'état de M1, dans leur PRE et dans leur *body*:

5 L'INVARIANT de M2 "suppose" l'INVARIANT de M1;

6 Mais M2 ne peut pas affecter (écrire) les variables de M1, car la cohérence de M1 (déjà prouvée) doit être preservée *a priori*;

7 L'INITIALISATION de M2 commence par initialiser M1, mais au sens que une opération de *query* s'informe sur comment M1 a été initialisée au moment de l'inclusion.

8 Tout aspect de M1 est visible à M2, mais toute mise à jour de l'état de M1 doit être effecuée par une opération de M1;

9 M1 est une machine independente $\leadsto$ pas de référence à M2, dans M1;

10 Une opération *op* de M1 peut être rendue disponible à M2 avec la clause de M2 :

PROMOTES *op*

NB : *op* change seulement l'état de M1, mais doit présever l'INVARIANT de M2.
On peut aussi juste appeler une opération de M1, dans une opération de M2 (voir la suite pour la $\neq$).

INCLUDES, 4

Les relations entre M2 et M1 à traver d'une figure.

INCLUDES, 5 et EXTENDS

Cas particulier : M2 PROMOTES toute opération de M1. On peut déclarer, dans M2 :

EXTENDS M1

Exemple

Machine M1 : la machine Doors

MACHINE Doors
SETS $DOOR$ **;** $POSITION = \{open, closed\}$
VARIABLES $position$ % var fonctionnelle
INVARIANT $position \in DOOR \rightarrow POSITION$ % fonc totale
INITIALISATION $position := DOORS \times \{closed\}$
OPERATIONS
$opening(d) \widehat{=}$
PRE $d \in DOOR$
THEN $position(d) := open$
END ;
$closedoor(d) \widehat{=}$
PRE $d \in DOOR$
THEN $position(d) := closed$
END
END

Machine M2 : la machine Locks, contrôlant les portes des coffreforts d'une banque. Partie statique

MACHINE Locks
INCLUDES Doors
PROMOTES closedoor
SETS $STATUS = \{locked, unlocked\}$
VARIABLES $status$
INVARIANT
$status \in DOOR \rightarrow STATUS$ % fonc totale
$\wedge\ position^{-1}[\{open\}] \subseteq status^{-1}[\{unloked\}]$
INITIALISATION $status := DOORS \times \{locked\}$

Dans l'INVARIANT, on dit que les portes ouvertes ne sont pas verrouillées. Il porte sur la variable native de $M2$, $status$, **et** celle de $M1$ ($= Doors$), qui est $position$.

Machine M2 : la machine Locks, Partie dynamique

OPERATIONS

$opendoor(d) \widehat{=}$ % $\neq$ *opening*, op de Doors

PRE $d \in DOOR \land status(d) = unlocked$

THEN $opening(d)$ % op de Doors

END ;

$unlock(d) \widehat{=}$

PRE $d \in DOOR$

THEN $status(d) := unlocked$

END

$lock(d) \widehat{=}$

PRE $d \in DOOR \land$

$position(d) = closed$ % position = var d'état de Doors

THEN $status(d) := locked$

END

Example, commentaires

- L'opération *opening* est appelée par M2 (dans *opendoor*), tandis que *closedoor* est *promoted*.
  Différence ?
  closedoor est importée par **PROMOTES** car son utilisation ne peut pas violer l'INVARIANT de M2.
  opening peut être utilisée seulement si le status de la porte est "unloked" ($\in$ PRE de *opendoor*), sinon l'INVARIANT de M2 serait violée : il faut que $M2$ contrôle cette opération !
- Pourquoi si on declarait *Locks* **EXTENDS** *Doors* on aurait une machine incohérente ?

Exemple, suite

Les relations entre *Locks* et *Doors* peuvent être visualisées :

Plus sur les opérations de la machine incluse, 1

En général :

A l'appel d'une opération de la machine incluse M1, ses variables input et output doivent être instanciées à des valeurs, qui peuvent être $=$, dans le cas de 2 variables input $\neq$.
On peut même utiliser la même variable dans les inputs et l'outputs.

Par exemple, $M1$ déclares l'opération :

$$z \leftarrow mult(x, y) \widehat{=}$$
$$\textbf{PRE } x \in \mathbf{N} \ \wedge \ y \in \mathbf{N} \textbf{ THEN } z := x \times y \textbf{ END}$$

et $M2$ l'utilise pour calculer $n^2 - 1$ pour un $n > 0$:

$$n \leftarrow mult(n - 1, n + 1)$$

Une opération *op*2de M2 qui appele une operation *op*1 de la machine incluse $M1$ doit préserver l'invariant de M2.

Pour obtenir l'obligation de preuve pour *op*2 il faut remplacer les occurrences de *op*1 dans *op*2 par sa défintion **PRE** P **THEN** S **END**.

Opérations de M1, cohérence, 2

Example

*op*1 est mult, I2 est l'invariant de M2, et *op*2 est :

IF $n > 0$ **THEN** $n \leftarrow mult(n-1, n+1)$ **END**

On calcule [op2] I2, c'est-à-dire :

[**IF** $n > 0$ **THEN** $n \leftarrow mult(n-1, n+1)$ **END**] I2 = (R3)

$n > 0 \Rightarrow ([n \leftarrow mult(n-1, n+1)]$ I2) $\wedge (\neg(n > 0) \Rightarrow$ I2) =

$n > 0 \Rightarrow$

([**PRE**] $n - 1 \in N \ \wedge \ n + 1 \in N$

THEN $n := n - 1 \times n - 1$ **END**] I2)

$\wedge (\neg(n > 0) \Rightarrow$ I2)

= (par R5)

$n > 0 \Rightarrow$

(($n - 1 \in \mathbb{N} \ \wedge \ n + 1 \in \mathbb{N}$) $\wedge$ ([$n := n - 1 \times n + 1$] I2)

$\wedge (\neg(n > 0) \Rightarrow$ I2)

Inclusion Multiple

Plusieurs machines peuvent être incluses dans une autre.

On peut aussi : M2 inclue un machine M1, qui inclue une machine M1', qui inclue une machine M1"... etc. La relation d'inclusion est transitive.

Mais l'acces aux opérations de machines incluses n'est pas transitif : les opérations de M1' sont accessibles seulement à M1, celles de M1" seulement à M1', etc.

Opérations en $||$

Quand 2 machines M1 et M3 sont inclues dans M2 au même niveau, on peut appeler une opération $op1$ de M1 et une opération $op2$ de façon parallèle et :

PRE P_1 **THEN** S_1 **END** $||$ **PRE** P_2 **THEN** S_2 **END** $=$
PRE $P_1 \wedge P_2$ **THEN** $S_1 || S_2$.

Ensuite, on peut réduire $S_1 || S_2$.

$\rightsquigarrow$

Réductions d'instructions en $||$

$S \mid\mid \textbf{skip} = S$

$S \mid\mid T = T \mid\mid S$

$(x_1, \cdots x_n := E_1, \cdots, E_n) \;\mid\mid\; (y_1, \cdots y_m := F_1, \cdots, F_m) =$
$x_1, \cdots x_n, y_1, \cdots y_m := E_1, \cdots, E_n, F_1, \cdots F_m$

$(\textbf{IF } E \textbf{ THEN } S_1 \textbf{ ELSE } S_2 \textbf{ END}) \mid\mid T =$
$\textbf{IF } E \textbf{ THEN } S_1 \mid\mid T \textbf{ ELSE } S_2 \mid\mid T$

$(\textbf{CHOICE } S_1 \textbf{ OR } S_2 \textbf{ END}) \mid\mid T =$
$\textbf{CHOICE } S_1 \mid\mid T \textbf{ OR } S_2 \mid\mid T \textbf{ END}$

$(\textbf{PRE } P \textbf{ THEN } S \textbf{ END}) \mid\mid T = \textbf{PRE } P \textbf{ THEN } S \mid\mid T \textbf{ END}$

$(\textbf{ANY } x \textbf{ WHERE } E \textbf{ THEN } S \textbf{ END}) \mid\mid T =$
$\textbf{ANY } x \textbf{ WHERE } E \textbf{ THEN } S \mid\mid T \textbf{ END}$

Question utile d'un étudiant

Pourquoi, dans la réduction de **PRE** P **THEN** S **END**) $||$ T, la precondition P de S devient une precondition de $S \,||\, T$?

Parce-que la sémantique de $||$ est telle que $S \,||\, T$ termine ssi les 2, S et T terminent. Si P est fausse, à l'appel de **PRE** P **THEN** S on pourrait ne pas terminer, et que se passerait alors si on avait **PRE** P **THEN** S suivi (notation : **;**) par un'instruction T_2 ?

Exemples de réduction

Exercices

Réduire :

1. **ANY** x **WHERE** $x \in \mathbb{N}$ **THEN** $x := x^2$ **END** $||$ $y := y_3$
2. **IF** $x = 1$ **THEN** $y{:=}y + 1$ **ELSE** $y{:=}y - 1$ $||$ $z := z + 3$
3. **IF** $x = 1$ **THEN** $y{:=}y + 1$ **ELSE** $y{:=}y - 1$ **END** $||$ **ANY** w **WHERE** $w \in 1..10$ **THEN** $z{:=}z + w$ **END**
4. **IF** $x > y$ **THEN** $y, z := y + z, 0$ **END** $||$ **CHOICE** $x := x + y$ **OR** $x := x + z$ **END**
5. **ANY** x **WHERE** $x \in 4..y$ **THEN** $z := x^2$ **END** $||$ **PRE** $y > 3$ **THEN** $y := y - 3$ **END**

Exemple de machine complexe, 1

On utilise la machines Locks, qui inclue Doors, et une autre machine encore, Keys (Clés), dans une large machine Safes (Coffreforts).

MACHINE Keys
SETS *KEY*
VARIABLES *keys*
INVARIANT $keys \subseteq KEY$
INITIALISATION $keys := \emptyset$
OPERATIONS

$insertkey(k) \widehat{=}$ % une var input, pas de var output
PRE $k \in KEY$
THEN $keys := keys \cup \{k\}$
END ;
$removekey(k) \widehat{=}$ % une var input, pas de var output
PRE $k \in KEY$
THEN $keys := keys \setminus \{k\}$
END
END

Exemple de machine complexe, suite

Machine Safes (coffreforts), partie statique

MACHINE Safes
INCLUDES Locks, Keys % au même niveau
PROMOTES *opendoor*, *closedoor*, *lockdoor*
CONSTANTS *unlocks* % association des clés à leur portes
PROPERTIES $unlocks \in KEY \rightarrowtail\!\!\!\!\twoheadrightarrow DOOR$ % par une bijection
INVARIANT $status^{-}1[\{unlocked\}] \subseteq unlocks[keys]$

N.B : l'opération importée *closedoor* est 2 niveaux + bas.
La variable *unlocks* associe les clés aux portes qu'elle déverrouillent (étant insérées dans les verrous respectifs), avec une bijection.
L'invariant dit que toute porte déverrouillée doit avoir sa clé (insérée).

Exemple de machine complexe, suite

Machine Safes (coffreforts), partie dynamique

OPERATIONS

$insert(k, d) \widehat{=}$

PRE $k \in KEY \;\wedge\; d \in DOOR \;\wedge\; unlocks(k) = d$ % d a sa clé

THEN $insertkey(k)$ % opération de *Keys*

END ;

$extract(k, d) \widehat{=}$

PRE $k \in KEY \;\wedge\; d \in DOOR \wedge$

$unlocks(d) = k \;\wedge status(d) = locked$

THEN $removekey(k)$ % opération de *Keys*

END ;

La precondition de *extract* demande que la porte d soit vérrouillée.

Exemple de machine complexe, suite

Encore des opérations de *Safes* :

$unlock(d) \widehat{=}$
PRE $d \in DOOR \ \wedge \ unlocks^{-}1(d) \in keys$
THEN $unlockdoor(k)$ % opération de *Locks*
END ;
$quicklock(d) \widehat{=}$
PRE $d \in DOOR \ \wedge \ position(d) = closed$
THEN $lockdoor(d) \ || \ removekey(unlocks^{-}1(d))$
END ;

La precondition de *unlock* demande que la clé soit dans la porte (condition sur *unlocks*).
L'opération *quicklock* met à jour, au même temps, les deux machines (directement) incluses : une porte est vérrouillée **et** sa clé est enlévée. Sa précondition utilise *position*, variable (fonctionnelle) de la machine *Doors*, qui est incluse dans la machine *Locks*. Son *body* utilise l'opération *lockdoor* de *Locks* et l'opération *removekey* de *Key*.

Illustration des relations entre *Safe* et ses composantes.

NB : La méthode impose que si appele deux opérations de machines incluses en $||$, alors ces opérations appartiennent à 2 machines distinctes.

Obligations de Preuve spécifiques à l'inclusion

Soit M1 incluse dans M1 et soient : C_1 les CONSTRAINTS de $M1$, C_2 celles de M2, ST_1 les SETS de M1, ST_2 ceux de M2, k_1 les CONSTANTS de M1, k_2 celles de M2, B_1 les PROPERTIES de M1, $B2$ celles de M2, v_1 les VARIABLES de M1, v_2 celles de M2, I_1 l'INVARIANT de M1, I_2 celui de M2, T_1 l'instruction de l'INITIALISATION de M1, T_2 celle de M2.

Il faut prouver :

1. $(C_1 \ \wedge \ C_2) \ \rightarrow \ \exists ST_1, ST_2, k_1, k_2 \, (B_1 \ \wedge \ B_2)$
2. $(C_1 \ \wedge \ C_2 \ \wedge \ B_1 \ \wedge B_2) \ \rightarrow \ \exists v_1, v_2 (I_1 \wedge I_2)$
3. $(C_1 \ \wedge \ C_2 \ \wedge \ B_1 \ \wedge B_2) \ \rightarrow \ [T_1; T_2] I_2$
   où ";" est la séquence.
4. $(C_1 \ \wedge \ C_2 \ \wedge \ B_1 \ \wedge B_2 \ \wedge \ I_1 \ \wedge \ I_2 \ \wedge \ P) \rightarrow$
   $[S] I_2$
   pour toute opération native de M2, et pour toute opération citée dans PROMOTES, ayant la forme
   PRE P **THEN** S **END**.

Raffinement

L'atélier B permet de raffiner pas par pas une spécification, jusqu'au code (C ou Ada).

Niveau spécification pure : le QUOI. Raffinement : le COMMENT, de plus en plus.

Plusieurs operateurs utilisables dans les étapes de raffinement.

Composition Séquentielle 1

S **;** T : executer S, **puis** T.
Si S ne termine pas, S **;** T non plus !
Penser à nouveau aux obligations de preuve pour $S =$ **PRE** S_1 **END** $||$ S_2 dans le contexte de S **;** T.

Exemples

x:=x+2 **;** x:=x+4 est équivalent à x:=x+6
On peut composer plusieurs **;** par ex. t:=x **;** x:=y **;** y:=t (swap, t var temporaire)
Obligation de preuve :

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

S doit assurer d'arriver à un état où $[T]P$ est vraie.

Composition séquentielle 2

$[x := x + 2; x := x + 4](x > 9) =$
$[x := x + 2]([x := x + 4](x > 9)) = R1$
$[x := x + 2](x + 4 > 9) = R1$
$x + 2 + 4 > 9 = x + 6 > 9 = x + 3$

Exercice : Calculer $[x := y; y := x^2](y > x)$.

Dans le contexte de **;** c'est parfois utile de déclarer des variables locales à une instruction (comme t pour le swap entre x et y).
Instruction générale pour le faire :
VAR $x_1 \cdots x_n$ **IN** S

La Weakest Precondition est semblable à celle du **ANY** :
$[\textbf{VAR}\ x_1 \cdots x_n\ \textbf{IN}\ S]\mathrm{P} = \forall x_1 \cdots \forall x_n([S]P)$

Exemple (swap) :
$[\textbf{VAR}\ t\ \textbf{IN}\ t := x\ \textbf{;}\ x := y\ \textbf{;}\ y := t\ \textbf{END}](x = A\ \wedge\ y = B)$
$=$ (weakest precondition pour les variables locales)
$\forall t\ ([t := x\ \textbf{;}\ x := y\ \textbf{;}\ y := t]\ (x = A\ \wedge\ y = B))$
$=$ (weakest Precondition pour **;**)
$\forall t\ ([t := x]([x := y](y := t]\ (x = A\ \wedge\ y = B)) =$ (R1)
$\forall t\ ([t := x]\ ([x := y]\ (x = A\ \wedge\ t = B) =$ (R1)
$\forall t\ ([t := x]\ ((y = A\ \wedge\ t = B) =$ (R1)
$\forall t\ ((y = A\ \wedge\ x = B) =$ (logique)
$y = A\ \wedge\ x = B$

Une machine MR qui raffine une machine M est déclarée par :
REFINEMENT MR
REFINES M

Il faut établire un INVARIANT de liaison qui connecte les états de MR aux états (abstraits) de R.

Il faut aussi décrire ce que deviennent l'initialisation et les opérations de M.

Illustration par l'exemple du raffinement d'une machine abstraite Equipe qui stocke dans une variable *équipe* l'ensemble des joueurs sur le terrain (11) d'une équipe de football au sens large (22 joueurs).
On peut remplaçer un joueur qui est sur le terrain par un autre qui est assis.

Machine ABstraite Equipe

MACHINE Equipe
SETS $REPONSES = \{in, out\}$
VARIABLES *equipe*
INVARIANT $equipe \subseteq 1..22 \ \wedge \ card(equipe) = 11$
INITIALISATION $equipe = 1..11$
OPERATIONS
$remplacer(old, new) \widehat{=}$
PRE $old \in equipe \ \wedge \ new \in 1..22 \ \wedge \ new \notin equipe$
THEN $equipe := (equipe \cup \{new\}) \setminus \{old\}$
END ;
$rep \leftarrow query(j) \widehat{=}$
PRE $j \in 1..22$
THEN
IF $j \in equipe$
THEN $rep := in$
ELSE $rep := out$
END
END
END

N.B : Pour exécuter *query*, il faut stocker la valeur de *equipe*.
Comment ?

Raffinement d'Equipe, Version 1

REFINEMENT EquipeR
REFINES Equipe
VARIABLES *equipeR*
INVARIANT $equipeR \subseteq 1..11 \rightarrowtail 1..22$ % Fonct. inj. et totale, tableau
$\wedge\ ran(equipeR) = equipe$ % Invariant de liaison
INITIALISATION $equipeR = \lambda n.(n \in 1..11 \mid n)$
OPERATIONS % PRE des opérations implicites
$remplacer(old, new) \widehat{=}$
$equipeR(equipeR^{-1}(old)) := new$
END ;
$rep \leftarrow query(j) \widehat{=}$
IF $j \in ran(equipeR)$
THEN $rep := in$
ELSE $rep := out$
END
END
END

Notation λ pour les fonctions.

La fonction qui fait le carré d'un nombre naturel :

$carre = \lambda x.(x \in \mathbb{N} \mid x^2)$.

Ici, la fonction *equipeR* est l'identité sur $\mathbb{N}$.

Relations ente Equipe et EquipeR

- L'invariant de liaison dit que la valeur d'*equipe*, variable d'EQUIPE, est l'ensemble d'arrivée du tableau *equipeR*.
- On suppose que les **PRE** des opérations de EQUIPE sont valables :
  - *remplacer* déclanchée seulement si *equipe* contient *old* mais ne contient pas *new* et $new \in 1..22$;
  - *query* déclanchée seulement si $j \in 1..22$.
  Ceci assure, par ex, que *old* soit dans $Dom(equipeR^{-}1)$ et que $equipeR(equipeR^{-1}(old)) := new$ soit bien défini ($equipeR$ étant injective).
- Plusieurs états d'EQUIPER représentent le même état d'EQUIPE. Par ex., [3 | 5 | 11 | 1 | 2 | 9 | 4 | 10 | 7 | 8 | 6] et [1 | 2 | 11 | 5 | 3 | 9 | 4 | 10 | 7 | 8 | 6 |] correspondent à $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

Raffinement d'Equipe, Version 2

REFINEMENT EquipeRbis
REFINES Equipe
VARIABLES *equipeRep*
INVARIANT $equipeRep \subseteq 1..22 \rightarrow REPONSES$
$\wedge\ equipeRep^{-1}[\{in\}] = equipe$ % Invariant de liaison
INITIALISATION $equipeRep = (1..11) \times \{in\} \cup (12..22) \times \{out\}$
OPERATIONS % PRE des opérations implicites
$remplacer(old, new) \widehat{=}$
BEGIN $equipeRep(old)) := out$ **;** $equipeRep(new) := in$ **END**
$rep \leftarrow query(j) \widehat{=} rep := equipeRep(j)$
END

$equipeRep \subseteq 1..22 \rightarrow REPONSES$ est une fonction totale. C'est un tableaux indexé par les joeurs, disant si un joeur donné est sur le terrain ou pas, et utilisable dans *query*.

Machine Abstraite NotesExam

MACHINE NotesExam
SETS *ETUDIANTS*
VARIABLES *note*
INVARIANT $note \in ETUDIANTS \nrightarrow 0..20$ % fonct. partielle
INITIALISATION $note = \emptyset$
OPERATIONS
  $inserer(e, n) \mathrel{\widehat{=}}$
   PRE $e \in ETUDIANTS \ \wedge \ e \notin dom(note) \ \wedge \ n \in 0..20$
   THEN $note(e) := n$
   END ;
  $rep \leftarrow moyenne \mathrel{\widehat{=}}$
    PRE $note \neq \emptyset$
    THEN $\Sigma z.(z \in dom(note) \ | \ note(z))/card(dom(note))$
    END ;
  $n \leftarrow nombre \mathrel{\widehat{=}} n := card(dom(notes))$
END

N.B. L'idéntite de l'étudiant est irrelevante, par rapport aux opérations.
Quand on raffine, on peut l'éliminer : voir la machine NotesExamR

$\leadsto$

Machine NotesExamR 1

REFINEMENT NotesExamR
REFINES NotesExam
VARIABLES $total, num$
INVARIANT $num = card(dom(note))$ % Partie de l'invariant de liaison
$\wedge\ total = \Sigma z.(z \in dom(note)\ \mid\ note(z))/card(dom(note))$ % Partie de l'invariant de liaison
INITIALISATION $total := 0, nom = 0$
OPERATIONS
$inserer(e, n) \mathrel{\widehat{=}}$
BEGIN $total := total + n\ \ ||\ \ num := num + 1$ **END**
$rep \leftarrow moyenne \mathrel{\widehat{=}} total/num$ **;**
$n \leftarrow nombre \mathrel{\widehat{=}} n := num$
END

Un état donné de NotesExamR peut correspondre à plusieurs états de NotesExam. Par ex. $total = 25, num = 2$ à $note = \{(Marc, 7), (Julie, 18)\}$ mais aussi à $note = \{(Marie, 14), (Isabelle, 11)\}$.

Les opérations peuvent être déclanchées seulement si les **PRE** de la machine abstraite sont vraies. En particulier, $e \notin dom(note)$ et on ne récrira pas la note d'un étudiant. Ceci assure que $total$ et num se comportent bien.

Exercice.

On ajoute une opération à NotesExam :

$t \leftarrow top \mathrel{\widehat{=}} \textbf{PRE} \neq \emptyset \textbf{ THEN } t := max(ran(note))$.

Modifier NotesExamR.

Machine Abstraite Villes

MACHINE Villes (VILLE) % paramètre
SETS $REPONSES = \{oui, non\}$
VARIABLES *routes*
INVARIANT $routes \in VILLE \leftrightarrow VILLE$ % relation binaire
INITIALISATION $routes = \emptyset$
OPERATIONS
  $lier(v1, v2) \mathrel{\widehat{=}}$
   PRE $v1 \in VILLE \ \wedge \ v2 \in VILLE$
   THEN $routes := routes \ \cup \ \{v1 \mapsto v2\}$
   END ;
  $rep \leftarrow query\_connection(v1, v2) \mathrel{\widehat{=}}$
    PRE $v1 \in VILLE \ \wedge \ v2 \in VILLE$
    THEN
     IF $(v1 \mapsto v2 \in (routes \ \cup \ routes^{-1})^*) \ \vee \ v1 = v2$
     THEN $rep := oui$
     ELSE $rep := non$
     END
END

Si R est une relation binaire, R^n est R composée n fois, et R^* est la cloture transitive de R, c'est-à-dire $R^* = \bigcup_{n \geq 0} R^n$.

Premier pas de raffinement : on implémente en stockant une partition des villes en classes d'équivalence de villes réliées (composantes connexes du réseau) $\leadsto$

Partie Statique

REFINEMENT VillesR   % héritage du paramètres VILLE de Villes
REFINES Villes
VARIABLES *partition*, *classe*
INVARIANT   % contient l'invariant de liaison

$partition \in \mathbb{P}(VILLE) \wedge$
$\bigcup cl\ (cl \in partition \ \mid \ cl\} = VILLE \wedge$
$\forall cl1\ \forall cl2\ ((cl1 \in partition\ cl2 \in partition) \ \Rightarrow$
$\quad (cl1 = cl2 \ \vee \ cl1 \cap cl2 = \emptyset))$
$\wedge\ \forall v\ (v \in VILLE \ \Rightarrow$
$\quad (routes \cup routes^{-}1 \cup id(VILLE))^{*}[\{v\}] \in partition)$
$\wedge\ classe \in VILLE \rightarrow \mathbb{P}(VILLE))$
$\wedge\ \forall v\ (v \in VILLE \Rightarrow v \in classe(v))$
$ran(classe) = partition$
$\forall cl\ (cl \in partition \ \Rightarrow \ classe^{-1}[\{cl\}] = cl$

En général, la machine raffinée hérite toute la partie statique (paramètres, sets et constantes) de la machine abstraite.

Remarques sur la partie statique de VillesR

L'INVARIANT de VillesR dit que l'ensemble des villes réliée à une ville *v* selon *routes* est une classe élément de *partition* : 4ème conjoint, une partie de l'INVARIANT de liaison.

La fonction *classe* associe à chaque ville sa classe d'équivalence (l'ensemble des villes qui lui sont réliées, directement ou indirectement.
Autant *partition* que *classe* contiennent toutes les informations sur les classes d'équivalence. Le 5ème et le 6ème conjoint donnent le reste de l'INVARIANT de liaison.
Le dernier conjoint de l'INVARIANT que l'ensembledes villes ayant *cl* comme valeur de la fonction *classe* est exactement *cl*.

Raffinement : VillesR, 3

Partie Dynamique

INITIALISATION

$\quad$ $\mathit{partition} := \{cl \mid cl \in (P)(\mathit{VILLE}) \ \wedge \ \mathit{card}(cl) = 1\}$ % villes isolées

$\quad$ $|| \ \mathit{classe} := \{(v, c) \mid (v, c) \in \mathit{VILLE} \times (P)(\mathit{VILLE}) \ \wedge \ c = \{v\}\}$

OPERATIONS

$\quad$ $\mathit{lier}(v1, v2) \mathrel{\widehat{=}}$

$\quad$ **IF** $\mathit{classe}(v1) \neq \mathit{classe}(v2)$

$\quad$ **THEN** $\mathit{partition} := (\mathit{partition} \setminus \{\mathit{classe}(V1), \mathit{classe}(v2)\}) \ \cup \ \{\mathit{classe}(1) \cup \mathit{classe}(v2)\} \ \{;\}$ % fusion

$\quad\quad$ $\mathit{classe} < +(\mathit{classe}(v1) \cup \mathit{classe}(v2)) \times \{\mathit{classe}(1) \cup \mathit{classe}(v2)\}$ % m.à.j. analogue de *classe*

$\quad$ **END ;**

$\quad$ $\mathit{rep} \leftarrow \mathit{query\_connection}(v1, v2) \mathrel{\widehat{=}}$

$\quad\quad$ **IF** $\mathit{classe}(v1) = \mathit{classe}(v2)$

$\quad\quad$ **THEN** $\mathit{rep} := \mathit{oui}$

$\quad\quad$ **ELSE** $\mathit{rep} := \mathit{non}$

$\quad\quad$ **END**

END

On peut implémenter VillesR en mémorizant juste l'association de chaque ville à la ville réprésentative de sa classe par une fonction *reponse* utilisée aussi dans la query. Ceci conduit à un pas ultérieur de raffinement, VillesRR $\rightsquigarrow$

REFINEMENT VillesRR
REFINES VillesR
VARIABLES *reponse* % une seule var stocke le réseau
INVARIANT % contient l'invariant de liaison avec VillesR
$reponse \in VILLE \rightarrow VILLE \wedge$ % fonct totale, tableau
$\forall v1\ \forall v2\ (v1 \in VILLE\ \wedge\ v2 \in VILLE)\ \Rightarrow$
$classe(v1) = classe(v2)\ \Leftrightarrow (reponse(v1) = reponse(v2)))$
INITIALISATION $rep := id(VILLE)$ % villes isolées
OPERATIONS
$lier(v1, v2) \mathrel{\widehat{=}}$
IF $reponse(v1) \neq reponse(v2)$
THEN $reponse := reponse < +((reponse^{-1}[\{reponse(v1)\}]) \times \{reponse(t2)\})$
END ;
$rep \leftarrow query\_connection(v1, v2) \mathrel{\widehat{=}}$
IF $reponse(v1) = reponse(v2)$
THEN $rep := oui$
ELSE $rep := non$
END
END

Ici on lie $v1$ à $v2$ en déclarant que la ville réprésentative de (la classe de) $v1$ est la même ville qui répresente (la classe de) $v2$.

Etude de l'évolutio des valeurs de *reponse* si on a les villes 1..6 d'abord isolées, puis on lie 1 à 2, 3 à 4 et 5 à 6, puis 5 à 4.

Encore un pas de Raffinement

On peut reffiner encore pour gagner en efficacité. Dans VillesRR, quand on lie $v1$ à $v2$ il faut mettre à jour toute case du tableau *reponse* ayant comme index une ville de la classe de $v1$, et il faut donc parcourir le tableau entier.

A la place d'associer chaque ville à sa ville réprésentative dans un tableau, on implémentera chaque classe par un *arbre* dont la racine est la ville réprésentative de la classe (structure de données de Galler-Fischer)

Chaque noeud interne aura un *parent* qui est une autre ville, et la racine (la ville réprésentative de la classe) sera son propre parent. On stockera juste la fonction *parent*, par un tableau $VILLE \rightarrow VILLE$, et la hauteur maximale n d'un arbre.

$\rightsquigarrow$

REFINEMENT VillesRRR
REFINES VillesRR
VARIABLES *parent*, *n*
INVARIANT % contient l'invariant de liaison avec VillesR
$parent \in VILLE \rightarrow VILLE \;\wedge\; n \in \mathbb{N}$ % parent est un tableau
$reponse = parent^n \;\wedge$
$\forall v\ (v \in VILLE \;\Rightarrow\; parent(reponse(v)) = reponse(v))$ % racine = représ. de la classe
INITIALISATION $parent =: id(VILLE) \;\|\; n = 0$ % tout arbre a 1 seul noeud
OPERATIONS
$lier(v1, v2) \mathrel{\widehat{=}}$
VAR $rep1, rep2$ % var locales
IN $rep1 := parent^n(v1)$; $rep2 := parent^n(v2)$;
IF $rep1 \neq rep2$
THEN $parent(rep1) := rep2$; $n := n + 1$ % fusion de 2 arbres
END ;
$rep \leftarrow query\_connection(v1, v2) \mathrel{\widehat{=}}$
IF $parent^n(v1) = parent^n(v2)$
THEN $rep := oui$
ELSE $rep := non$
END
END

N.B. $parent^n$ est la composition de la fonction *parent* *n* fois.
Si la distance d'un noeud *n* de sa racine *r* est *d* et $n \geq d$, alors $parent(n) = r$).

On fusionne 2 arbres a1 et a2 en déclarant que la racine de a2 est le parent de celle de a1 et incrémentant *n* de 1.

Raffinement du non-déterminisme

MACHINE Allouer   % alloue des num de tél aux usagers
SETS $ANS = \{vrai, faux\}$
VARIABLES *alloues*
INVARIANT $alloues \subseteq \mathbb{N}_1$
INITIALISATION $alloues := \emptyset$
OPERATIONS
  $rep \leftarrow query(n) \mathrel{\widehat{=}}$
    PRE $n \in \mathbb{N}_1$
    THEN
      IF $n \in alloues$ **THEN** $rep := vrai$ **ELSE** $rep := faux$
    END
  $n \leftarrow allouer \mathrel{\widehat{=}}$
    ANY m
    WHERE $m \in \mathbb{N}_1 \setminus alloues$
    THEN $n := m \;||\; alloues := alloues \cup \{m\}$
    END
END

On raffine en donnant un critére pour choisir le nouveau numéro à allouer : on choisit le min des nombres pas encores alloués $\leadsto$

Machine AllouerR

REFINEMENT AllouerR **REFINES** Allouer
VARIABLES *alloues*
INITIALISATION $alloues := \emptyset$
OPERATIONS

$rep \leftarrow query(n) \widehat{=}$

IF $n \in alloues$ **THEN** $rep := vrai$ **ELSE** $rep := faux$ **END**

$n \leftarrow allouer \widehat{=}$

BEGIN $n := min(\mathbb{N}_1 \setminus alloues)$ **;** $alloues := alloues \cup \{n\}$

END
END

N.B La VAR de Allouer et AllouerR étant la même, *alloues*, l'invariant de liaison inplicit est que les valeurs sont les mêmes.

Illustrées à l'aide d'un raffinement de la machine abstraite Couleurs

MACHINE Couleurs
SETS $COULEUR = \{rouge, vert, blue\}$
VARIABLES $cols$
INVARIANT $cols \subseteq COULEUR$
INITIALISATION $cols :\in \mathbb{P}(COULEUR \setminus \{blue\})$ % affectation non-détermiste
OPERATIONS
  $ajout(c) \mathrel{\widehat=}$ **PRE** $c \in COULEUR$ **THEN** $c :\in cols \cup \{c\}$ **END ;**
  $c \leftarrow query \mathrel{\widehat=}$
    PRE $cols \neq \emptyset$ **THEN** $c :\in cols$ % renvoi d'une coul arbitraire
  $change \mathrel{\widehat=} cols :\in (\mathbb{P} \setminus \{cols\}$ % modif arbitraire de *cols*
END

REFINEMENT CouleursR
REFINES Couleurs % set COULEUR hérité
VARIABLES $couleur$
INVARIANT $couleur \in cols$ % inv de liaison
INITIALISATION $couleur :\in (COULEUR \setminus \{blue\})$ % indéterminisme nouveau ! Mais seul choix indét.
OPERATIONS
  $ajout(c) \mathrel{\widehat=} couleur :\in \{couleur, c\}$
  $c \leftarrow query \mathrel{\widehat=} c := couleur$ % renvoie la coul courante, déterminisation
  $change \mathrel{\widehat=} couleur :\in COULEUR \setminus \{couleur\}$ % modif déterministe de *couleur*
END

Obligations de Preuve spécifiques au Raffinement, 2

En géneral, une machine M et son raffinement MR peuvent avoir plusieurs éxécutions (indéterminisme).

Il faut assurer que tout état que MR peut atteindre correspond , par l'invariant de liaison J, à quelque état que M peut atteindre.
NB : J est une formule qui porte sur un état de M et un état de MR.

Obligations pour l'INITIALISATION, 1

Dans le cas de l'INITIALISATION, il faut établir que pour n'importe quel état *er* atteignable par MR à partir de son initialisation *IR* il existe un état *e* atteignable par M à partir de son initialisation *I*, et tel que *J* est vraie pour *e* et *er*.

Il faut donc établir :
$[IR]\neg[I]\neg J$

Lire : Toute exécution de IR assure que c'est faux que toute exécution de *I* porte à un état où *J* est fausse, c.à.d. :
toute exécution de IR assure qu'il existe au moins une exécution de *I* portant à la vérité de *J*.

Obligations pour l'INITIALISATION, 2

Machines Couleurs et CouleursR

$[I]\neg J = [cols :\in \mathbb{P}(COULEUR \setminus \{blue\})]\neg couleur \in cols$, qui se lit :
Si *cols* est n'importe quel ensemble de couleurs ne contenant pas le blue, alors $couleur \notin cols$.
Ceci est vrai ssi $couleur = blue$.

Donc $\neg[I]\neg J$ dit : il existe au moins un état de Couleurs donné par son initialisation et tel que $couleur \in cols$.
C'est vrai ssi $couleur \neq blue$.

$[IR]\neg[I]\neg J \ \equiv$
$[couleur :\in (COULEUR \setminus \{blue\})](coleur \neq blue) \ \equiv$
$\forall couleur\ (couleur :\in (COULEUR \setminus \{blue\}) \ \Rightarrow \ coleur \neq blue \ \equiv$
$True$

Obligations pour les Opérations

Les obligations de preuve pour les opérations suivent la même philosophie :
Il faut s'assurer que toute exécution de MR correspond, par l'invariant de liaison J, à quelque exécution de M.