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

Abstract Machine : MACHINE

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.

Abstract Machine : INVARIANT 4

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

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 $\rightsquigarrow$ 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 : $serve < next$.

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.

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

Calcul de la weakest precondition

Exemple pour R3 combinée avec R1

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

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

$(x < 5\ \wedge\ (x + 4) < 7)\ \vee\ (x \geq 5\ \wedge\ (x - 3) < 7) =$ (arithm)
$(x < 5\ \wedge\ x < 3)\ \vee\ 5 \leq x < 10$ c.à.d
$x < 3\ \vee\ 5 \leq x < 10$

Cas particulier de R3

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

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

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

R4 : Règle pour *CASE OF*

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

$$\begin{array}{l} (E = e_1 \;\Rightarrow\; [T_1]P) \wedge \\ (E = e_2 \;\Rightarrow\; [T_2]P) \wedge \\ \dots \wedge \\ (E = e_n \;\Rightarrow\; [T_n]P) \wedge \\ ((E \neq e_1 \;\wedge\; E \neq e_2 \;\wedge\; \cdots \;\wedge\; E \neq e_n) \;\Rightarrow\; [V]P) \end{array}$$

Calcul de la weakest precondition

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

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

Cohérence d'une AM

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.

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

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 !

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.

Cohérence d'une AM

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) !

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 toutes les obligations de preuve.

Machines plus expressives

On va étudier l'usage de :

- paramètres $\rightsquigarrow$ machines génériques
- constantes (analogues aux constantes d'un programme)
- ensembles $\rightsquigarrow$ 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 <u>type</u> (généricité) et il faudra l'instancier avec un <u>ensemble non-vide</u>.

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

Paramètres, 2

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 $\mathbb{N}$).

Constantes, 1

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*).

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) :

$(card(NAME) > capacity \ \wedge \ total \in \mathbb{N}_1 \ \wedge \ total > 4096 \wedge$
$capacity \in \mathbb{N}_1 \ \wedge \ capacity \leq 4096) \Rightarrow$
$(\emptyset \subseteq NAME \ \wedge \ \emptyset \subseteq NAME \ \wedge \ \emptyset \cap \emptyset = \emptyset \ \wedge \ card(\emptyset) \leq 4096$
$\quad \wedge \ card(\emptyset) \leq total)$

Nouvelles Obligations de Preuve, 5

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 $r[\{4\}] = \{0, 1, 2, 3\}$.

Fonctions, 1

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.

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
$\wedge \ reading \in READER \not\rightarrow COPY$
$\wedge \ (reading ; copyof) \cap hasread = \emptyset$

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 $\{t \mid t \in T \text{ et } \exists u \in S\ (s, t) \in r\}$.

Exemple, suite

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.

$resp \leftarrow precurrentquery(rr) \widehat{=}$
  PRE $rr \in READER$
  THEN
    IF $rr \in dom(reading)$ **THEN** $resp := yes$ **ELSE** $resp := no$
  END ;
$bb \leftarrow currentquery(rr) \widehat{=}$
  PRE $rr \in READER \;\wedge\; rr \in dom(reading)$
  THEN $bb := copyof(reading(rr))$
  END ;
$resp \leftarrow hasreadquery(rr, bb) \widehat{=}$
  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 : $\boxed{3 \mid 7 \mid 5 \mid 0}$
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 $t(i) := nouvelle\_valeur$ 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 $i = j$!

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 ?

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**] $P =$
$\forall x_1 ... \forall x_n \ (Q \Rightarrow [T]P)$

ANY 6

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

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

ANY 7

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

$[\textbf{CHOICE } S_1 \textbf{ OR} ... \textbf{ OR } S_n]\ P = [S_1]\ P \wedge .... \wedge [S_1]$

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** $\neg Q_1 \wedge \ldots \neg Q_n$ **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 :
$[\textbf{PRE } Q \textbf{ THEN } S \textbf{ END}]P = Q \wedge [S]P$