Cours 3

Boulanger Jean-Louis

RATP

Décembre 2000

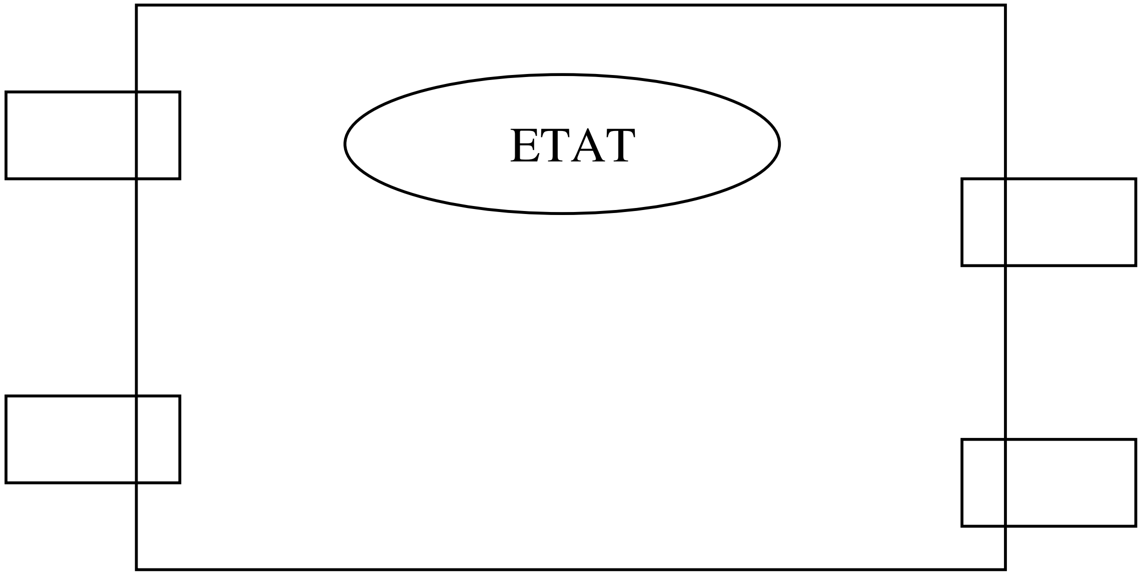
Abstract Machine Notation

Boulanger Jean-Louis

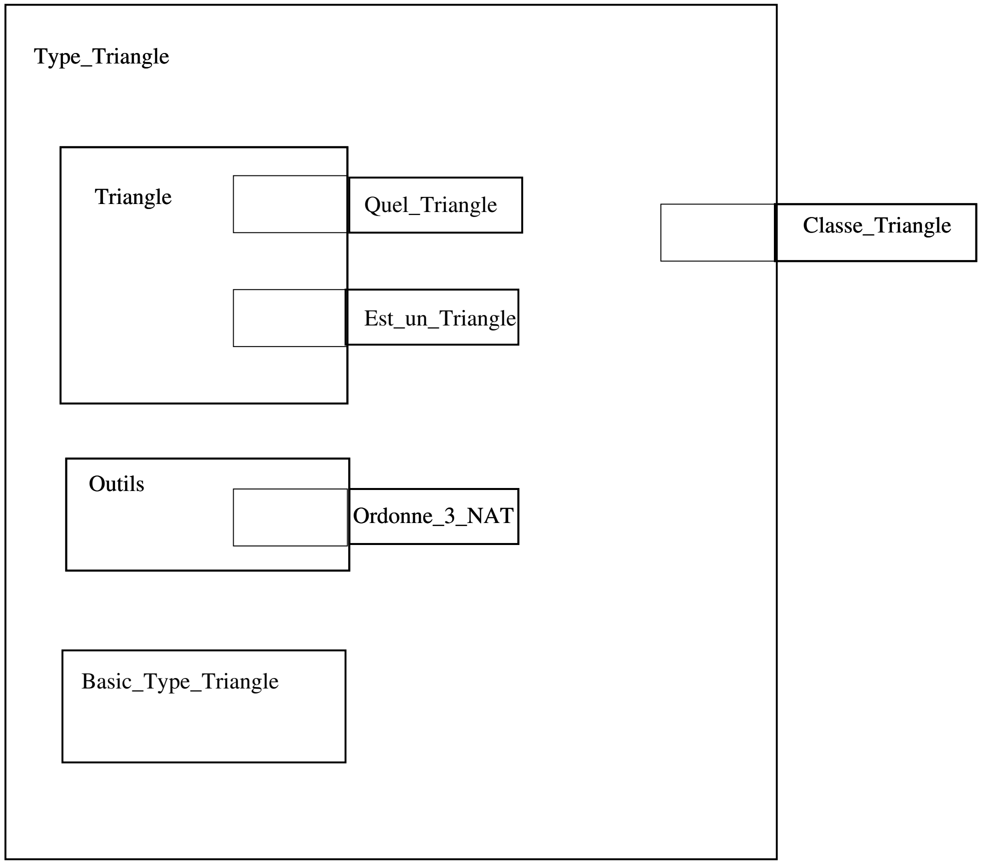
RATP

Décembre 2000

Encapsulation



Encapsulation



Machine abstraite

La méthode B utilise un concept (et une notation) unique :

la machine abstraite

Ce concept intègre les notions :

- de composition de modules,
- d'encapsulation des données et
- d'opérations.

Machine Abstraite

Toute **machine abstraite** est composée de trois parties :

- La partie *déclarative* qui décrit l'état de la machine et ses propriétés.
- La partie *exécutive* qui introduit le comportement au travers d'opérations qui font évoluer l'état de la machine.
- La partie *composition* qui introduit des liens de visibilité avec d'autres machines abstraites.

Abstract Machine Notation

Toute machine abstraite est exprimée dans

l'**A**bstract **M**achine **N**otation

L'AMN est basée

- sur la théorie des ensembles,
- sur la logique du premier ordre (prédicat),
- sur les **S**ubstitutions **G**énéralisées (transformation de prédicats).

Les entités manipulées

Ensembles Les *ensembles* doivent être finis,

Séquences Les suites d'éléments, (principe des listes),

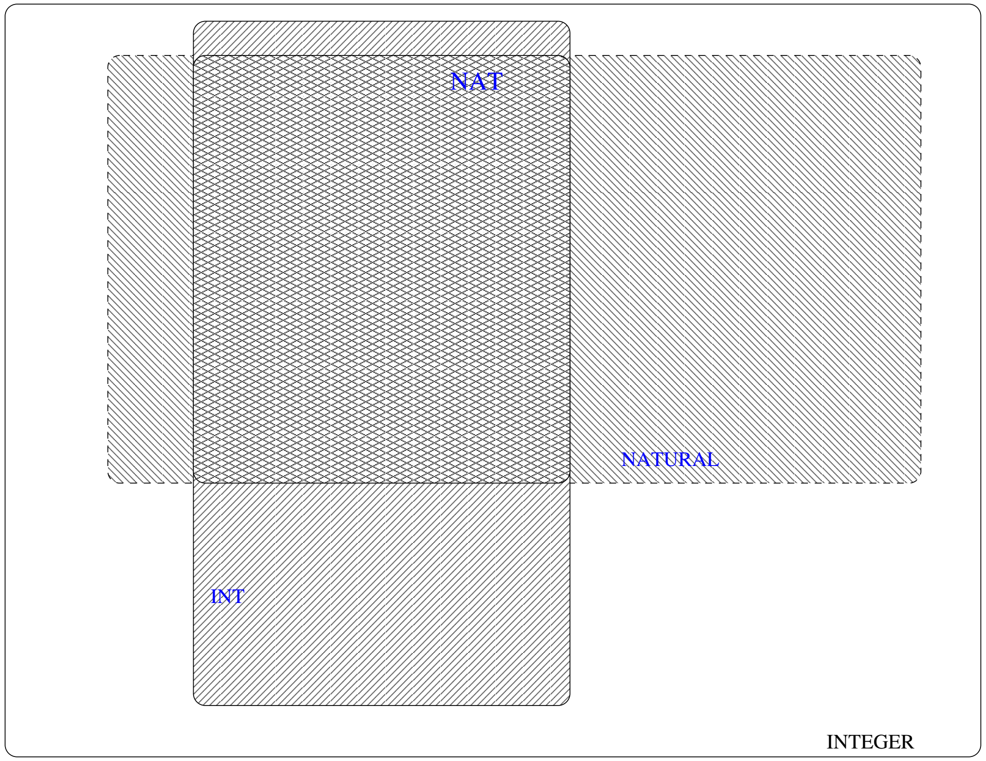
Tableaux Les tableaux informatiques,

Entiers Les Entiers Naturels, les entiers relatifs finis ou non.

Ensembles de base (1)

BOOL	
INTEGER	Ensemble des Entiers naturels
INT	Ensemble fini des Entiers naturels
NATURAL	Ensemble des Entiers relatifs
NAT	Ensemble fini des Entiers relatifs
NAT1	Ensemble fini des Entiers relatifs différents de 0
STRING	Ensemble des chaînes de caractère
CHAR	Ensemble des caractères

Ensembles de base (2)



Constructeur ensembliste

$E * F$		Produit Cartésien
$\mathbb{P}(E)$	$POW(E)$	Ensemble des sous-ensembles
$aa..bb$		Intervale de valeur

E et F sont des ensembles.

Constructeur ensembliste : Exemple

$\{aa, bb\} * \{1, 2\}$	$\{(aa, 1), (bb, 2)\}$
$\mathbb{P}(\{aa, bb\})$	$\{\{\}, \{aa\}, \{bb\}, \{aa, bb\}\}$
$1..3$	$\{1, 2, 3\}$

Opérateur Ensembliste

	$\{\}$	Empty Set
$\{xx_1, xx_2, \ldots, xx_n\}$		Set of xx_1 to xx_n
$xx \in aa$	$xx : aa$	xx is in aa
$xx \notin aa$	`xx /: aa`	xx is not in aa
$aa \cup bb$	`aa \/ bb`	Set Union
$aa \cap bb$	`aa /\ bb`	Set Intersection
$uu - tt$		Set Difference
$card(uu)$		Cardinalty of set uu
$FIN(uu)$		Finite sets of uu

Opérateur Ensembliste de comparaison (2)

$aa = bb$	aa = bb
$aa \neq bb$	aa /= bb
$aa \subset bb$	aa « : bb
$aa \subseteq bb$	aa < : bb

Exemple d'ensembles (1)

Soit C l'ensemble des chiffres de la numération décimale qui est défini par **énumération** $C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Soit N l'ensemble des nombres naturels $N = \{0, 1, 2, 3, 4, ...\}$

Et $vide = \{\}$

$card(C) = 10$	on a bien 10 chiffres
$card(N) =?$	on a une infinité d'éléments
$card(vide) = 0$	on a un ensemble vide
$1 \in C$	on dit que "1 appartient à C"
	ou que "1 est élément de C"

Exemple d'ensembles (2)

Soit C l'ensemble des chiffres de la numération décimale qui est défini par **énumération** $C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
Soit N l'ensemble des nombres naturels $N = \{0, 1, 2, 3, 4, ...\}$
Et $vide = \{\}$

$vide \subset C$ on dit que l'ensemble *vide* est inclus dans C

$C \subset N$

Exemple d'ensembles définis en compréhension (3a)

Soit C' l'ensemble des chiffres impairs

$C' = \{x \mid x \in C \wedge impair(x)\}$

$x \in C$	C est dit ensemble de référence
$impair(x)$	*impair* est appelé **propriété caractéristique**
$x \mid P$	se lit "x tel que P"

Exemple d'ensembles définis en compréhension (3b)

$$NATURAL = \{xx \mid xx : INTEGER \wedge xx > 0\}$$

Exemple d'ensembles définis en extension (4)

Les trois définitions de E suivantes sont équivalentes :

$E = \{a, b, c, d, e\}$

$E = \{e, b, a, d, c\}$

$E = \{d, a, a, e, b, a, c, b, d, d, e, e\}$

$E = F$ Deux ensembles sont égaux

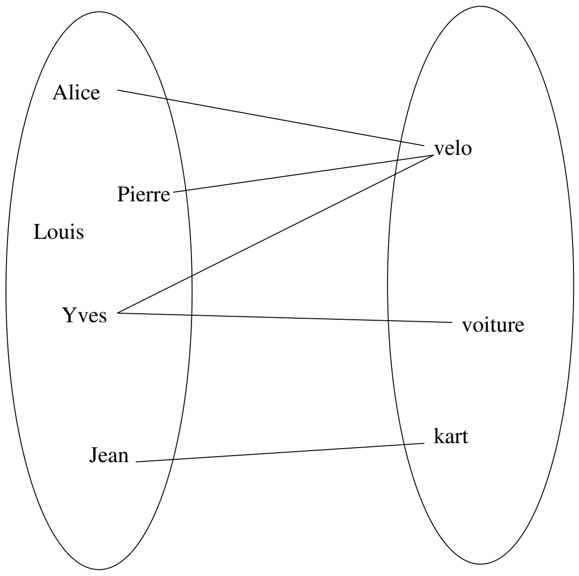
s'ils sont formés des mêmes éléments

Exemple d'ensembles exprimés en B

```
MACHINE
   EX_SET_1
SETS
   EE = { aa, bb, cc, dd, ee}
CONSTANTS
   impaire, CC, CC_PRIME
PROPERTIES
   CC        = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
&  impaire   : CC --> BOOL
&  CC_PRIME = {xx | (xx:CC) & impaire(xx)=TRUE}
END
```

Relation (1)

Une relation binaire ($A \leftrightarrow B$) entre deux ensembles A et B est un sous-ensemble du produit cartésien $A * B$.C'est le sous-ensemble dont les éléments vérifient un certain prédicat défini.



conduit : A <–> B

Relation (2)

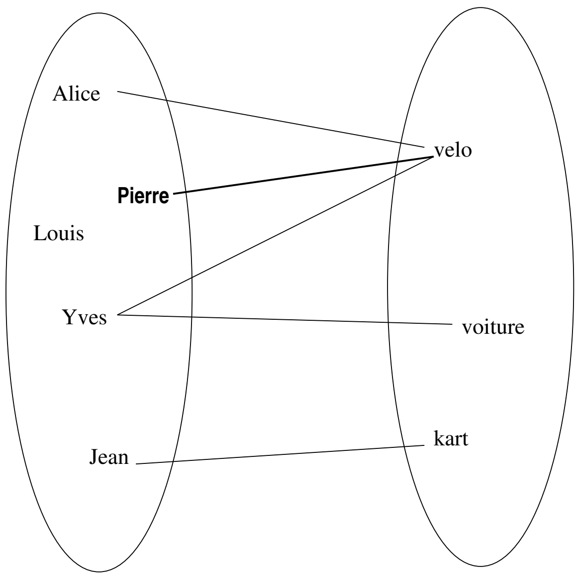
```
MACHINE
   EX_SET_2
SETS
   PERSONNE = {Jean, Alice, Pierre, Louis, Yves}
;  VEHICULE = {Velo, Voiture, Kart}
CONSTANTS
   Conduit
PROPERTIES
   Conduit : PERSONNE <-> VEHICULE
&  Conduit = { (Jean,Kart), (Alice, Velo), (Pierre,Velo),
              (Yves, Velo), (Yves, Voiture) }
END
```

Relation (2)

```
MACHINE
   EX_SET_2
SETS
   PERSONNE = {Jean, Alice, Pierre, Louis, Yves}
;  VEHICULE = {Velo, Voiture, Kart}
VARIABLES
   Conduit
PROPERTIES
   Conduit : PERSONNE <-> VEHICULE
INITIALISATION
   Conduit = {(Jean,Kart), (Alice, Velo), (Pierre,Velo),
              (Yves, Velo), (Yves, Voiture) }
END
```

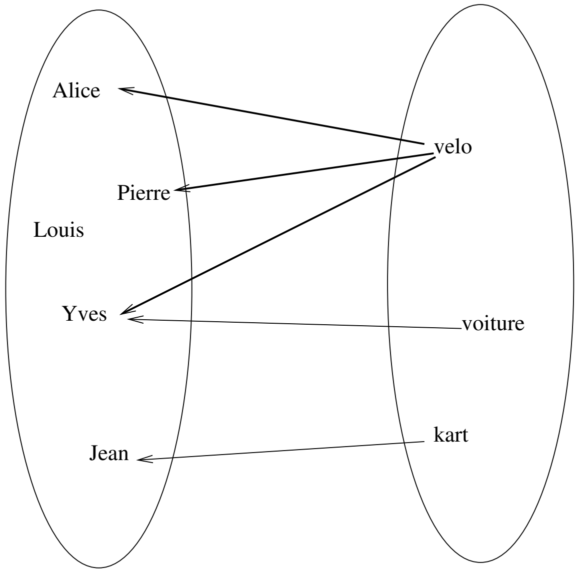
Manipulation de Relation (1)

`Conduit[{Pierre}]`



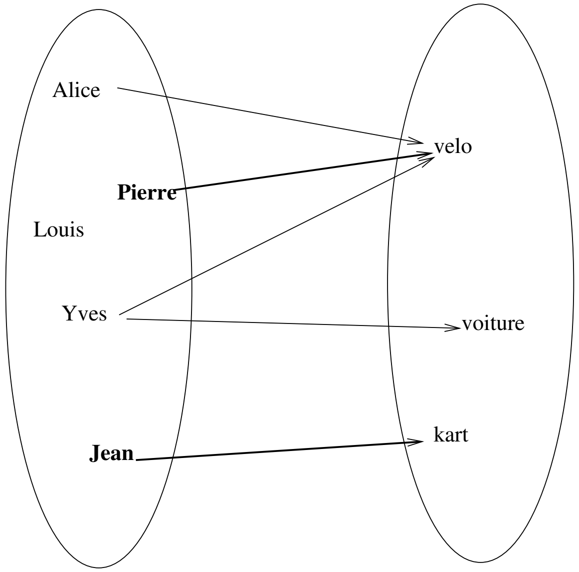
Manipulation de Relation (2)

```
Conduit~[{Velo}]
```



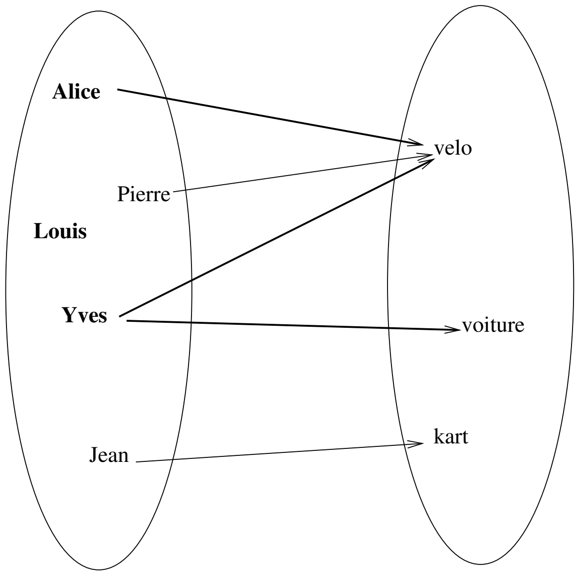
Manipulation de Relation (3)

`{Jean,Pierre}<|Conduit`



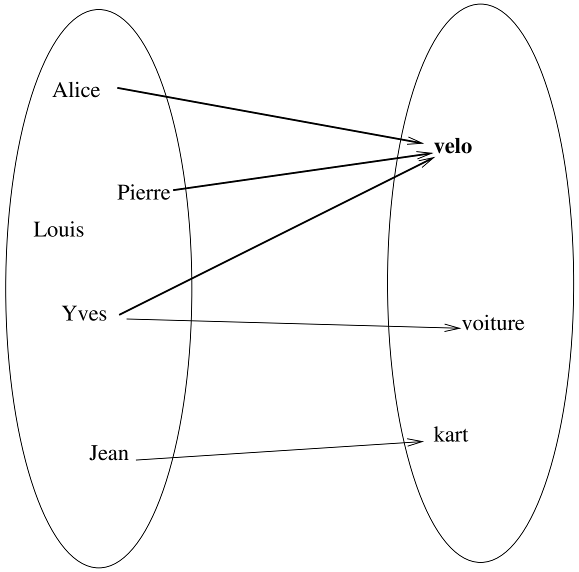
Manipulation de Relation (4)

```
{Jean,Pierre}<<|Conduit
```



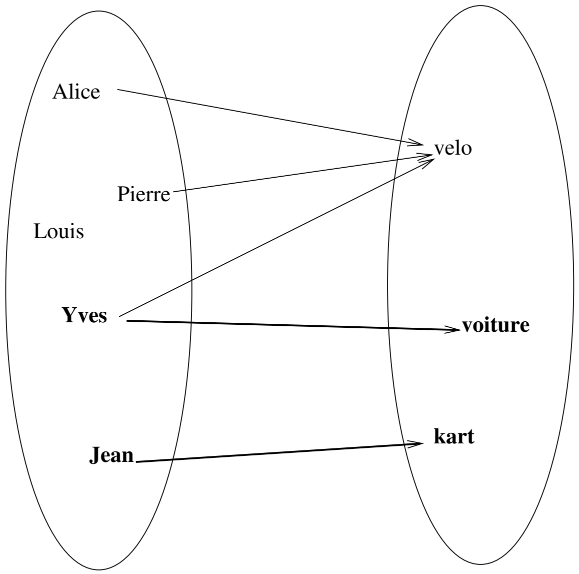
Manipulation de Relation (5)

```
Conduit|>{Velo}
```



Manipulation de Relation (5)

`Conduit|>>{Velo}`



Manipulation de Relation (2)

Soit $f = \{(1,2),(2,3)\}$ et $g = \{(2,4),(3,5)\}$

`f;g` composition des fonctions

on a pour résultat la fonction $\{(1,4),(2,5)\}$

Fonction

Une relation de A et B est une fonction lorsque chaque élément a au plus une image par la relation

Fonction totale	$x_1 \rightarrow x_2$
Fonction partielle	$x_1 \nrightarrow x_2$
Fonction injective	$x_1 \rightarrowtail x_2$
Fonction surjective	$x_1 \twoheadrightarrow x_2$
Fonction bijective	$x_1 \rightarrowtail\!\!\!\!\twoheadrightarrow x_2$

Liens entre relation et fonction

La figure suivante présente un graphe introduisant une hiérarchie entre les différents types de flèche.

$$
\begin{tikzcd}
 & & \text{↔} \arrow[d] & & \\
 & & \text{⇸} \arrow[dll] \arrow[d] \arrow[drr] & & \\
\text{⤔} \arrow[dr] & & \text{→} \arrow[dl] \arrow[dr] & & \text{⤀} \arrow[dl] \\
 & \text{↣} \arrow[dr] & & \text{↠} \arrow[dl] & \\
 & & \text{⤖} & &
\end{tikzcd}
$$

Exemples (1)

$$ff = \{1 \mid -- > 2, 2 \mid -- > 3, 3 \mid -- > 5\}$$

$ff : N \rightarrowtail\!\!\!\!\!+ N$	ff est une fonction partielle sur son domaine
$ff : 1..3 \longrightarrow N$	ff est une fonction totale sur son domaine
$dom(ff) = 1, 2, 3$	le domaine de ff
$ran(ff) = 2, 3, 5$	le co-domaine de la fonction ff
$card(ff) = 3$	le cardinal de l'ensemble de définition de ff

Exemples (2)

$$ff = \{1 \mid -- > 2, 2 \mid -- > 3, 3 \mid -- > 5\}$$

$ff < +\{2 \mid -- > 500\}$ on crée une nouvelle fonction par surcharge

$ff \sim$ on inverse la fonction

Pour conclure sur les fonctions

Une Fonction est une Relation. Une Relation est un Ensemble.

Connecteur Logique

5 connecteurs de bases NON (not , $\neg$), ET (& , $\wedge$), OU (*or*, $\vee$), implique ($\Rightarrow$) et *si et seulement si* ($\Longleftrightarrow$). Chacun de ces connecteurs peut être définit au travers d'une table de vérité.

x_1	x_2	$x_1 \wedge x_2$
T	T	T
T	F	F
F	F	F
F	F	F

Opérateur Arithmétique

$$+, -, *, /, mod, div$$

Calcul des Prédicats

Les constructeurs :

$$=, \neq, >, \geq, <, \leq$$

Les quantificateurs logiques :

$$\forall, \exists$$

Substitution

Une substitution est un remplacement des occurences libres d'une variable dans une expression par une autre expression donnée

Elle est notée : $P[v/x]$

Une variable est dite libre si elle n'est pas quantifiée.

Description du GSL

x := E	Simple substitution
S || T	Parallel substitution
skip	Empty substitution or no-op
P | S	Preconditionning
S [] T	Bounded choice
P ⇒ S	Guarding
@ x.S	Unbounded choice, non-detminism

x, S, T, P sont respectivement une variable, deux substitutions et un prédicat.

Description du GSL (suite)

$S\,;T$	Sequencing
$WHILE\ Q\ DO\ S$ $INVARIANT\ I$ $VARIANT\ P\ END$	Looping

S, T, I, P, Q sont respectivement deux substitutions et trois prédicats.

Règle d'utilisation du GSL

$[skip]R$	R
$[x := E]R$	
$[S \| T](P \wedge Q)$	$[S]P \wedge [T]Q$
$[P \mid S]R$	$P \wedge [S]R$
$[S [] T]R$	$[S]R \wedge [T]R$
$[P \Rightarrow S]R$	$(P \Rightarrow [S]R)$
$[@\, x.S]R$	$\forall z.[S]R$
$[x, y := E, F]R$	$[x := E \| y := F]R$

Exemple

$$[x > 5 \Rightarrow y := y + 1 \mid\mid z := z + 1](x, y, z : NAT)$$

$$[x > 5 \Rightarrow y := y + 1](x, y : NAT)$$

$$\wedge\ [z := z + 1](z : NAT)$$

$$(x > 5 => [y := y + 1](x, y : NAT))$$

$$\wedge\ (z + 1 : NAT)$$

$$(x > 5 => (y + 1 : NAT \wedge x : NAT))$$

$$\wedge\ (z + 1 : NAT)$$

Sucre syntaxique (1)

`BEGIN S END`	`S`
`PRE Q THEN S END`	`P | S`
`a,b := E,F`	`a:=E || b:=F`
`IF P THEN S END`	`P=>S[]not(P)=>skip`

`IF P THEN S ELSE T END`
`IF P THEN S`
`ELSIF Q THEN T ELSE U END`
	`P=>S[]not(P)=>T`
	IF P THEN S ELSE
	IF Q THEN T ELSE U END

Sucre syntaxique (2)

SELECT P THEN S END	P=>S[]not(P)=>skip
SELECT P THEN S WHEN P1 THEN S1 END	P=>S[] P=>S[] not(P)&not(P1)=> skip
SELECT P THEN S WHEN P1 THEN S1 ELSE T END	P=>S[] P1=>S1[] not(P)&not(P1)=> T

Sucre syntaxique (3)

CHOICE S OR .. OR U END	S [] .. [] U
CASE E OF EITHER L1 THEN S1 OR L2 THEN S2 ... OR Ln THEN Sn ELSE T END	SELECT E :L1 THEN S1 WHEN E :L2 THEN S2 WHEN E :Ln THEN Sn ELSE T END

Sucre syntaxique (4)

ANY z WHERE P THEN S END	@ z.(P=>S)
VAR x IN S END	@ x.S
f(x) := E	f:=f <+ {x|->E}

Exemples de machines abstraites

Une Personne

```
MACHINE        Person2 ( maxage)
CONSTRAINTS    maxage:NAT & maxage > 100
SETS           GENDER = {female, male}
CONSTANTS      date_of_birth , gender
PROPERTIES     date_of_birth : NAT &
               gender : GENDER
VARIABLES      age
INVARIANT      age : 0..maxage
INITIALISATION age := 0
OPERATIONS
       birthday    = PRE age<maxage THEN age:=age +1 END;
ca <-- current_age = ca := age
END
```

Une Pile

```
MACHINE      STACK (max_object)
CONSTRAINTS max_object : NAT1
SEES         OBJECT
VARIABLES    stack
INVARIANT    stack:seq(Object) &
             size(stack)<=max_object
INITIALISATION  stack := []
OPERATIONS
PUSH(XX) = PRE XX:Object & size(stack)<max_object
           THEN stack := stack <- XX END;
XX <-- POP = PRE size(stack) > 0
           THEN XX,stack:=last(stack),front(stack) END
END
```