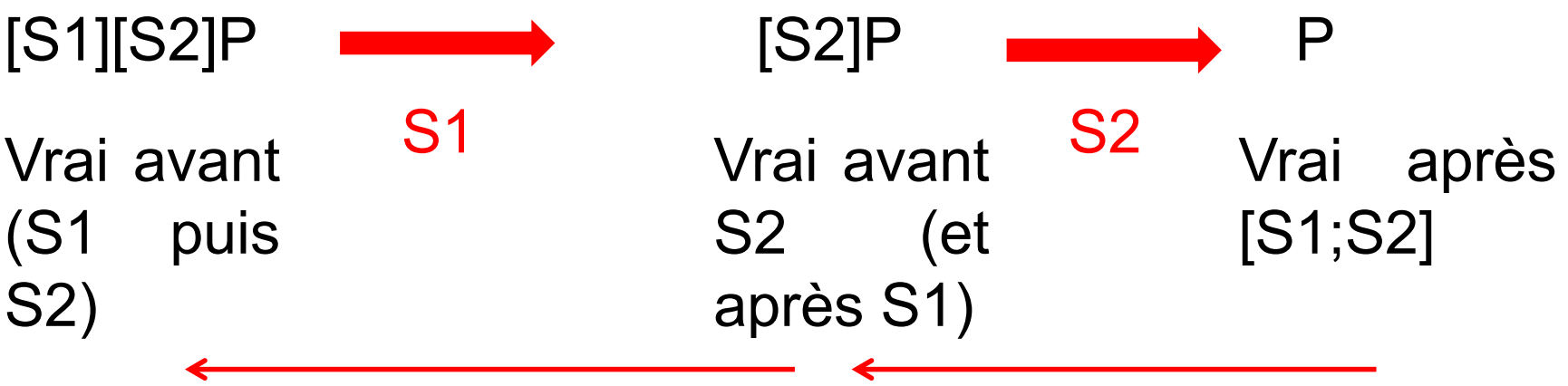
Substitutions généralisées

Notion de substitution généralisée

- En B une substitution peut être une affectation explicite d'une variable à une valeur (exemple [x:=1]) ou à une expression (exemple [x:=x+1])
- Le B se devant de préserver l'abstraction permet aussi :
  - De conditionner l'utilisation d'une substitution (précondition PRE ou sélection SELECT)
  - D'exprimer des substitutions offrant des choix (CHOICE x:=1 OR x:=2 END)
  - De manipuler des ensembles abstraits (x :: A_SET, à lire « x devient élément de l'ensemble A_SET »)
- …
- Ces substitutions généralisées (et les POs associées) sont décrites ci-après.

Substitutions séquentielles

- La substitution S1 puis S2 est notée [S1;S2]
- Correspond en notation mathématique à $S2 \circ S1$
- La WP associée à [S1;S2] s'obtient en remontant les WP associées à chaque substitution :



Par conséquent : [S1;S2]P=[S1][S2]P

Substitutions simultanées

Lorsqu'une opération comporte des substitutions multiples on peut se demander si l'invariant ne peut être violé transitoirement entre ces substitutions.

- La question ne se pose pas au moins pour les machines car les substitutions sont simultanées :
- Exemple x:=x+1 || y:=y+1 ou x,y:=x+1,y+1 (cette dernière forme déjà rencontrée)
- Cela présente en fait de nombreux avantages, par exemple [x:=y||y:=x](x>y) donne y>x
- Noter que dans ce cas cela n'apporte rien de fondamentalement nouveau car pouvant être représenté de manière équivalente par [z:=y][y:=x][x:=z](x>y) où z est une variable « fraiche » (non utilisée par ailleurs).

Substitutions préconditionnées

Dans l'exemple on souhaite ajouter un mécanisme précisant que AddTrain n'est utilisable que dans certaines conditions (ici nbTrains<MAXINT).

C'est l'objet d'une substitution généralisée appelée substitution préconditionnée notée

- [P|S] (notation abrégée dite GSL : Generalized Substitution Language)
- PRE P THEN S END (notation des outils B dite AMN : Abstract Machine Notation, faite pour se rapprocher des langages de programmation).

INVARIANT nbTrains:NAT

OPERATIONS AddTrain=

PRE nbTrains<MAXINT THEN nbTrains:=nbTrains+1 END

Substitutions préconditionnées

6 Attention la précondition n'est pas un IF, elle ne fait que dire que l'opération n'est utilisable que si la précondition est vérifiée. Si ce n'est pas le cas, aucun résultat n'est garanti.

Formellement, la Weakest Precondition (WP) associée à [P|S]R (R étant un prédicat) est : $P \wedge [S]R$

- Si P est vrai la WP devient [S]R et si elle est vraie, S établit R
- Si P est faux la WP sera toujours fausse et donc quelle que soit R elle ne sera jamais établie. Dans ce cas on dit que la substitution ne se termine pas (ou avorte), elle ne peut établir aucun prédicat.
- En pratique une substitution qui avorte peut être n'importe quel comportement du programme (crash ou autre…)

IF

Le IF existe toutefois en B mais exprime deux choses que le B permet d'appréhender séparément :

- La garde : une substitution n'est faisable que si une condition (prédicat) est vérifiée (vrai) : substitution SELECT.
- Le choix : une substitution peut avoir plusieurs possibilités : substitution CHOICE.

Le IF (ou le CASE) est une combinaison CHOICE / SELECT (même dans le cas de IF sans ELSE, il y a un choix car implicitement, ELSE : rien…)

Substitutions gardées : SELECT

8 Le SELECT diffère du PRE car dit cette fois que la substitution n'est faisable que dans certaines conditions (ici nbTrains<MAXINT).

- [P⇒S] (Notation GSL) : attention ce n'est pas le même ⇒ que dans un prédicat.
- SELECT P THEN S END (Notation AMN)

INVARIANT nbTrains:NAT

OPERATIONS AddTrain=

SELECT nbTrains<MAXINT THEN nbTrains:=nbTrains+1 END

Le SELECT est la base des substitutions gardées (nous en verrons différentes variantes (IF, CASE etc.)

Substitutions gardées SELECT

Le SELECT exprime que l'opération n'est faisable que si la condition est vérifiée. Si ce n'est pas le cas, aucune substitution n'est effectuée, la substitution est dite non faisable.

Formellement, la WP associée à [P⇒S]R (R étant un prédicat) est : P⇒[S]R

- Si P est vrai la WP devient [S]R (comme pour la substitution PRE) et si elle est vraie, S établit R
- Si P est faux, la WP est toujours vraie (au lieu de toujours fausse dans le cas du PRE). La substitution est dite non faisable, mais d'un point de vue logique comme F⇒V et F⇒F sont tous deux vrais, une substitution non faisable peut établir tout prédicat (et son contraire).

Terminaison et faisabilité

Il est possible d'exprimer formellement le fait qu'une substitution ne se termine pas et ne soit pas faisable

- Une substitution qui ne se termine pas ne peut établir aucun prédicat donc en particulier, ne peut établir x=x (la réciproque se prouve facilement) donc :
  S avorte $\Leftrightarrow \neg[S](x=x)$, S se termine $\Leftrightarrow [S](x=x)$
- Une substitution qui n'est pas faisable peut établir tout et son contraire donc en particulier peut établir $x \neq x$ (la réciproque se prouve facilement) donc :
  S non faisable $\Leftrightarrow [S](x \neq x)$, S faisable $\Leftrightarrow \neg[S](x \neq x)$

PRE vs SELECT

Il est important de bien voir la différence :

- Pour PRE pour prouver P ∧ [S]R, il faut prouver P *et* [S]R *indépendamment* (vrai ssi P et [S]R sont vrais) : il faudra prouver P un jour même si on diffère cette preuve…
- Pour le SELECT pour prouver P⇒[S]R, il faut prouver *si* P *alors* [S]R (donc que [S]R est vrai si P l'est également car si P est faux P⇒[S]R l'est quelle que soit la valeur de vérité de [S]R) : il n'est pas nécessaire de prouver P.

PRE vs SELECT

Au niveau des POs :

- Pour un PRE l'utilisateur de l'opération est averti que l'opération n'est *utilisable* que sous condition P (qu'il devra prouver *par ailleurs* pour la faire disparaître dans l'implantation).
- Pour la PO associée à l'opération préconditionnée [P|S] on rajoute donc intentionnellement P comme hypothèse dans l'état de machine avant l'opération ce qui donne la PO suivante :
  $I \wedge P \Rightarrow P \wedge [S]I$ soit finalement : $I \wedge P \Rightarrow [S]I$
- Toutefois comme P est une précondition, le fait qu'il faille aussi prouver P est conservé pour la suite dans le mécanisme de preuves tant que l'on n'a pas raffiné (ce qui pourra comme on le verra affaiblir cette précondition).

PRE vs SELECT

Au niveau des POs :

- Pour un SELECT la PO est $I \Rightarrow (P \Rightarrow [S]I)$, dont il est facile de voir qu'une formule équivalente est $I \wedge P \Rightarrow [S]I$
- En effet $(A \Rightarrow (B \Rightarrow C)) \Leftrightarrow (A \wedge B \Rightarrow C)$ les deux n'étant faux que dans le cas VVF)
- On obtient donc la même PO que pour un PRE mais cette fois il n'a pas été nécessaire de rajouter artificiellement P comme hypothèse.
- P apparaît ici naturellement dans la PO qu'on n'a à prouver que sous condition P (il ne sera pas nécessaire de prouver P par la suite).

PRE vs SELECT

- SELECT : programmation dite défensive (l'opération se défend de tout utilisation inappropriées)
- PRE : programmation dite généreuse ou offensive (les conditions d'utilisation appropriée de l'opération sont précisées, il revient à l'utilisateur de l'opération de les vérifier).
- SELECT : se propage jusqu'au code source (sous forme de IF), y compris en méthode B (protection certes, mais pouvant être coûteuse en performances).
- PRE : la vocation de PRE est de s'affaiblir de plus en plus au fur et à mesure des raffinements (les choses se précisant certaines PRE apparaissent trop fortes) et si possible de disparaître (PRE est interdit dans les implantations)… ou de se transformer en IF en dernier recours…

Substitutions potentiellement non déterministes

- L'avantage des machines abstraites est de permettre d'exprimer le besoin de manière très abstraite (le quoi sans préciser le comment).
- L'une des possibilités est l'utilisation de substitution laissant des choix.
- Il ne s'agit pas de choix au sens tirage au hasard dans l'application finale, mais de choix qui seront précisés ultérieurement (lors des raffinements).
- On verra successivement
  - Choix borné (CHOICE… OR …)
  - Choix non borné (ANY… WHERE… THEN) et leurs cas particuliers (devient tel que, devient élément de…)
- De telles substitutions peuvent donc être non déterministes (non unicité de l'expression substituée)

Choix borné (CHOICE)

Le choix borné laisse ouvert le choix entre deux (ou plus généralement un nombre fini) de possibilités. Ainsi si S et T sont des substitutions :

- [S[]T] (Notation GSL)
- CHOICE S OR T END (Notation AMN)

Exprime que soit S soit T sont exécutées. L'une ou l'autre est acceptable. Il s'agit donc clairement d'une substitution non déterministe.

La postcondition doit être établie dans l'un ET l'autre cas, il faut donc bien faire le ET des weakest preconditions.

Par conséquent : $[S \,[]\, T]\, R \Leftrightarrow [S]\, R \wedge [T]\, R$

Il peut y avoir plus de deux choix que l'on peut écrire :

- S [] T [] … [] U ou CHOICE S OR T OR … OR U END

IF… THEN… ELSE

- Nous avons maintenant les ingrédients pour construire le IF… THEN… ELSE du langage B (garde et choix)
- IF P THEN S ELSE T (Notation AMN) est défini par :
  - (P⇒S) [] (¬P⇒T) (Notation GSL) ou encore
  - CHOICE
    SELECT P THEN S END
    OR
    SELECT not(P) THEN T END
    END

IF… THEN… ELSE

Exemple de WP avec un IF… THEN… ELSE

MACHINE M
VARIABLES x
INVARIANT x:[0..1]
INITIALISATION x:=0
OPERATIONS S=IF x=0 THEN x:=x+1 ELSE x:=x-1 END

[S]I : [IF x=0 THEN x:=x+1 ELSE x:=x-1](x:[0..1]) soit :
[(x=0⇒x:=x+1) [] (¬x=0⇒x:=x-1)] (x:[0..1]), soit :
(x=0⇒(x+1:[0..1]))&(x/=0⇒(x-1:[0..1]))
La PO complète est : I⇒[S]I soit
x:[0..1]⇒(x=0⇒(x+1:[0..1]))&(x/=0⇒(x-1:[0..1]))
qui est vraie (preuve à suivre)

IF… THEN… ELSE

Exemple de WP avec un IF… THEN… ELSE

Soit à prouver :

$x:[0..1] \Rightarrow (x=0 \Rightarrow (x+1:[0..1])) \& (x/=0 \Rightarrow (x-1:[0..1]))$

Comme $A \Rightarrow B \& C \Leftrightarrow A \Rightarrow B \ \& \ A \Rightarrow C$ (les deux sont faux seulement si A vrai et (B ou C faux))

Et $A \Rightarrow (B \Rightarrow C) \Leftrightarrow A \& B \Rightarrow C$ (déjà vu, les deux sont faux seulement dans le cas VVF)

La PO se récrit donc

$(x:[0..1] \& x=0 \Rightarrow (x+1:[0..1])) \ \&$

$(x:[0..1] \& x/=0 \Rightarrow (x-1:[0..1]))$

Soit enfin $(x=0 \Rightarrow (x+1:[0..1])) \ \& \ (x=1 \Rightarrow (x-1:[0..1]))$

skip, IF

- Comme dit plus haut, un IF sans ELSE est supposé (implicitement) ne rien faire.
- C'est l'une des raisons de l'existence de la substitution sans effet skip.
- skip ne modifie pas l'état de machine (toutes les variables conservent leur valeur).
- Une opération simple opération de sortie (printf…) ne modifie pas l'état de machine et est donc modélisée par un skip
- IF P THEN S est par conséquent défini comme :
  - (P⇒S) [](¬P⇒skip) (Notation GSL) ou encore
  - CHOICE (SELECT P THEN S END) OR (SELECT not(P) THEN skip END) END

IF… ELSIF

- Le B connaît le IF… ELSIF avec le sens usuel
- IF P THEN S
  ELSIF Q THEN T
  ELSE U
  END
  est défini comme :
- IF P THEN S
  ELSE
  IF Q THEN T
  ELSE U
  END
  END

SELECT… WHEN

- Le B reconnaît aussi la garde composée (SELECT à plusieurs prédicats, il peut y en avoir plus de deux) :
- SELECT P THEN S
  WHEN Q THEN T
  END
  est défini comme :
- CHOICE
  SELECT P THEN S END
  OR
  SELECT Q THEN T END
  END
- Noter que cette construction peut être infaisable (si P et Q sont tous deux faux) ou non déterministe (si P et Q sont tous deux vrais)

SELECT… WHEN

- Le SELECT… WHEN peut avoir un ELSE
- SELECT P THEN S
  WHEN Q THEN T
  ELSE U
  END
  est défini comme :
- SELECT P THEN S
  WHEN Q THEN T
  WHEN ¬(PvQ) THEN U
  END
  Construction donc toujours faisable mais pouvant être non déterministe.

CASE

- Le B connaît le CASE, fait pour être déterministe (cas supposés mutuellement exclusifs) et toujours faisable (ajout d'un ELSE skip si pas de ELSE spécifié) :

Syntaxe	Définition
CASE E OF EITHER C1 THEN S OR C2 THEN T ELSE U END	SELECT E=C1 THEN S WHEN E=C2 THEN T ELSE U END

- E est une expression et C1, C2… sont des constantes littérales (« valeurs explicites ») entières, booléennes ou énumérées, deux à deux distinctes (déterminisme)

CASE

Cas d'un CASE sans ELSE toujours faisable (ajout d'un ELSE skip)

Syntaxe	Définition
CASE E OF EITHER C1 THEN S OR C2 THEN T END	SELECT E=C1 THEN S WHEN E=C2 THEN T ELSE skip END

- E est une expression et C1, C2… sont des constantes littérales (« valeurs explicites ») entières, booléennes ou énumérées, deux à deux distinctes (déterminisme)

Choix non borné (ANY)

- Le choix non borné généralise le choix borné en laissant ouvert pour une substitution S_z dépendant d'une variable z le choix de n'importe quelle valeur de z.
- La variable z doit être « fraiche » (aucune occurrence libre nulle part, renommer z si nécessaire).
- $[@z.S_z]$ (Notation GSL)
- ANY z THEN S_z (Notation AMN)
- Exprime que l'on peut utiliser n'importe quel z et effectuer S_z. Tous les z sont acceptables. Il s'agit donc clairement d'une substitution non déterministe.
- La postcondition doit être établie dans pour tous les z, il faut donc bien faire le ET généralisé (∀) des weakest preconditions.
- Par conséquent : $[@z.S_z]R \Leftrightarrow \forall z.[S_z]\ R$

Choix non borné (ANY … WHERE)

- En pratique le choix non borné s'accompagne d'un prédicat P(z) concernant z (indiquant au minimum son « type » : ensemble éventuellement abstrait auquel il appartient).
- [@z.P(z)⇒ S_z] (Notation GSL)
- ANY z WHERE P(z) THEN S_z (Notation AMN)
- Comme on peut le voir (sur la définition GSL) il s'agit de la combinaison du ANY avec une substitution gardée (SELECT). Elle peut donc ne pas être faisable.
- [@z.P(z)⇒ S_z]R ⇔ ∀z.(P(z) ⇒ [S_z]R)

Substitution « devient tel que »

- Un cas particulier important du choix non borné est la substitution « devient tel que » :( ) où la substitution S_z pouvant concerner une ou plusieurs variables
- ANY z WHERE P(z) THEN x:=z
- Peut être noté de manière équivalente : x:(P(x))
- Exemple : a,b,c :(a∈NAT1∧b∈NAT1∧c∈NAT1∧$c^2=a^2+b^2$) rend les trois variables compatibles avec le fait d'être un triangle rectangle dont les longueurs de côté sont des entiers : infinité de solution dans ce cas… Pour un exposant >2 en revanche… (Fermat…).
- :( ) est souvent utilisée comme initialisation par défaut dans les machines : les variables deviennent telles que l'invariant (on précisera comment dans les raffinements)

Substitution « devient élément de »

- Un cas particulier important du cas particulier «devient tel que » est la substitution « devient élément de » (:∈ ou en notation ASCII ::) où le prédicat P(z) est simplement un prédicat d'appartenance à un ensemble (« typage »)
- ANY z WHERE z:trains THEN x:=z
- Ou encore x:(x:trains)
- Peut être noté de manière équivalente : x :: trains

Notations atelier B

- Attention l'atelier B exige des identifiants d'au moins deux caractères : x, y ou z ne conviennent donc pas, il faut mettre au minimum xx, yy et zz
- L'appartenance ∈ se note : et par conséquent le « devient élément de » :∈ se note ::
- L'inclusion se note <: et l'inclusion stricte <<:
- Le « devient inclus dans AA » (AA étant un ensemble) n'existe pas, mais peut être fait de manière équivalente par « devient élément des parties de A »,soit : xx::POW(AA)
- Le quantificateur universel ∀ se note !
- Le quantificateur existentiel ∃ se note #

POs et substitutions généralisées

Les POs introduites au chapitre précédent (avec des substitutions simples) s'écrivent de manière semblable avec des substitutions généralisées :

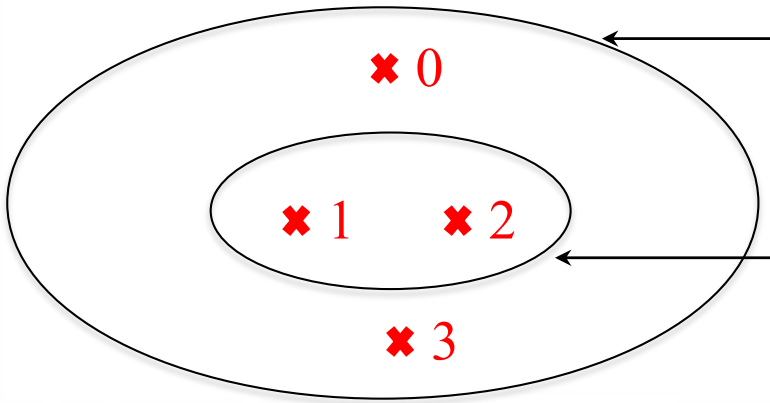
```
MACHINE Ex1
VARIABLES xx
INVARIANT xx : 0..3
INITIALISATION
CHOICE xx:=1 OR xx:=2 END
END
```

PO d'initialisation :

[xx:=1 [] xx:=2](xx : 0..3)

c'est-à-dire

$1 \in 0..3 \land 2 \in 0..3$



Etats respectant l'invariant

Etats possibles après initialisation

POs et substitutions généralisées

Les POs introduites au chapitre précédent (avec des substitutions simples) s'écrivent de manière semblable avec des substitutions généralisées :

MACHINE Ex2

VARIABLES xx

INVARIANT xx : 0..5

INITIALISATION

xx :: 1..3

END

PO d'initialisation :

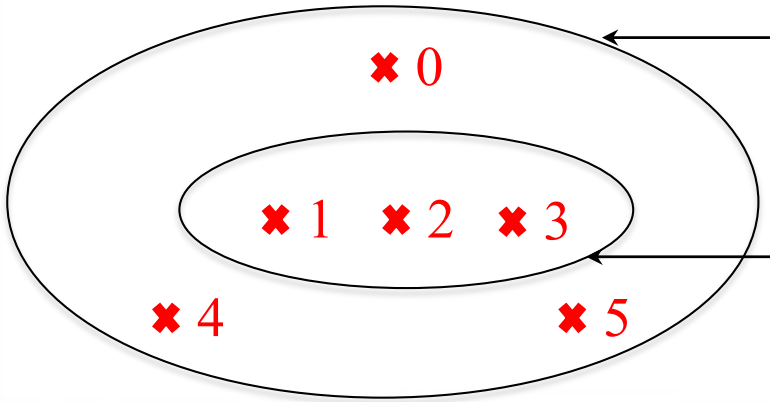
$[xx :: 1..3](xx : 0..5)$

$[@zz.(zz : 1..3) \Rightarrow xx:=zz](xx : 0..5)$

$\forall zz.(zz : 1..3) \Rightarrow [xx:=zz] (xx : 0..5)$

$\forall zz.(zz : 1..3) \Rightarrow (zz : 0..5)$

$1..3 \subseteq 0..5$



Etats respectant l'invariant

Etats possibles après initialisation

POs et substitutions généralisées

La PO d'initialisation d'une machine d'invariant INV et d'initialisation INIT est donc en toute généralité

$$[INIT]INV$$

Si INIT est une substitution choix non borné type ANY variables WHERE PR_ANY THEN INIT où PR_ANY est un prédicat portant sur variables (INIT dépend donc de variables), on voit facilement que la PO peut se mettre sous la forme :

$$PR\_ANY \Rightarrow [INIT]INV$$

Ou plus exactement

$$\forall(\text{variables du PR\_ANY}).(PR\_ANY \Rightarrow [INIT]INV)$$

POs et substitutions généralisées

- L'obligation de preuve d'opération si l'invariant est INV et l'opération OP (supposée non préconditionnée)

$$INV \Rightarrow [OP]INV$$

- Par contre si l'opération est préconditionnée (définie par PRE|OP où OP est une substitution non préconditionnée) l'obligation de preuve ne doit être faite que sous l'hypothèse supplémentaire PRE. La PO est donc :

$$INV \wedge PRE \Rightarrow [PRE|OP]INV \text{ soit :}$$

$$INV \wedge PRE \Rightarrow PRE \wedge [OP]INV \text{ qui se réduit à}$$

$$INV \wedge PRE \Rightarrow [OP]INV$$

Ou plus exactement :

$$(\forall \text{variables de machine}).(INV \wedge PRE \Rightarrow [OP]INV)$$

POs et substitutions généralisées

Si l'opération est une substitution choix non borné type ANY variables WHERE PR_ANY THEN OP où PR_ANY est un prédicat portant sur variables (OP dépend donc de variables), on voit facilement que PR_ANY peut être rajouté en tant qu'hypothèse dans le membre de gauche de l'implication, de sorte que la PO devient en toute généralité

$$INV \wedge PRE \wedge PR\_ANY \Rightarrow [OP]INV$$

ou plus exactement

$$(\forall \text{variables de machine, variables du } PR\_ANY).(INV \wedge PRE \wedge PR\_ANY \Rightarrow [OP]INV)$$

Définition locale : substitution LET

Réservée aux machines et raffinements (interdite dans les implantations où il faut utiliser VAR) la substitution LET permet d'introduire des définitions locales utilisables en lecture seule dans une substitution.

Syntaxe :

- LET
  list_variables_locales BE Predicat IN Substitution
  END

Exemple :

- LET sum, diff BE
  sum=x1+x2 & diff=x1-x2 IN m1:=sum/2 || var=diff**2
  END

Variable locale : substitution VAR

Réservée aux raffinements et implantations (interdite dans les machines où il faut utiliser LET) la substitution VAR permet d'introduire des variables locales utilisables dans une substitution. En implantation (substitutions séquentielles, les variables locales doivent être initialisées avant d'être lues).

Syntaxe :

- VAR
  list_variables_locales IN Substitution
  END

Exemple :

- VAR sum, diff IN
  sum:=x1+x2 ; diff:=x1-x2 ; m1:=sum/2 ; var:=diff**2
  END

Boucle : Substitution WHILE

Réservée aux implantations (interdite dans les machines et raffinements) la substitution WHILE permet de réaliser une boucle.

Syntaxe : WHILE Prédicat DO Substitution INVARIANT I VARIANT V END

- L'invariant sert à typer les variables utilisées dans la boucle et à exprimer des propriétés les concernant
- Le variant doit être une expression entière positive qui décroit strictement à chaque itération (ainsi on garantit l'absence de boucle infinie).

Exemple : WHILE ii < 5 DO nn:=nn+ii ; ii:=ii+1 INVARIANT ii:NAT & ii<=5 VARIANT 5-ii END