3. Substitution généralisée

Les plus faibles préconditions

Soient P et R des prédicats et S une instruction

- **P{S}R** [Logique de Hoare]

Le prédicat R décrit le résultat de l'opération S.

Le prédicat P représente un ensemble d'états tels que l'exécution de S commençant par un d'entre eux se termine en un temps fini dans un état satisfaisant R.

- Exemple : S est l'affectation i:=i+1, et R le prédicat i<=1.

Donnez un prédicat P vérifiant P{S}R :

i<=-10, mais aussi i<-4...

Parmi tous ces prédicats, le plus faible (le moins contraignant, le prédicat donnant le plus d'états...) est i<=0.

Ce prédicat sera appelé **la plus faible précondition de S par rapport à R**. On le notera **wp(S,R)** pour 'weakest precondition'.

Les plus faibles préconditions

Exercices :

- wp(i:=i + 1, i<=1) = ?

- wp(if x >= y then z:=x else z:=y, z=max(x,y))= ?

Le sens de wp(S,R) peut être précisé par deux propriétés :

1- wp(S, R) est une précondition garantissant R après l'exécution de S, c'est à dire que : wp(S,R){S}R

2- wp(S, R) est la plus faible des telles préconditions, c'est à dire que :

si P{S}R alors P $\Rightarrow$ wp(S,R)

B et plus faible précondition…

wp est une fonction à deux arguments : une instruction (ou programme) S et un prédicat Q.

Pour un S fixé, on peut voir wp(S, Q) comme une fonction à un seul argument $wp_S(Q)$.

La fonction wp_S est appelé transformateur de prédicats : c'est la fonction qui associe à tout prédicat Q la plus faible précondition P telle que P{S}Q.

$P\{S\}Q \equiv P \Rightarrow wp_S(Q)$ pour S qui termine

Notation B : $wp_S(R)$ sera noté **[S]R**

[S]P

Plus généralement (autrement dit):

- si on interprète S comme un programme alors [S]P représente la plus faible précondition pour que après n'importe quelle exécution de S, la propriété P soit vérifiée.
- [S]P est la condition initiale la plus large pour que, après avoir “exécuté” S, P devienne vrai.

Le prédicat [S]P se lit : “S établit le prédicat P”.

Exemple : [x := x+1] x=5 ≡ ?

Langage des substitutions généralisées

Les constructions du langage pour la spécification sont des substitutions : les programmes sont mathématiquement représentés par des substitutions généralisées qui sont définies par leur action sur les prédicats.

2 remarques importantes :

1. Toute substitution est vue comme un **transformateur** de prédicat : la méthode B permet de manipuler des programmes qui sont vus comme des transformations de la mémoire.

2. Une substitution généralisée S est complètement déterminée (**sémantique**) par la définition de la formule [S]P pour une formule P arbitraire.

Sémantique du langage des substitutions généralisées

Soit la substitution S≡n:=n+3 (affectation n:=3). On a :

- $[n:=n+3]n>2 \equiv n>-1$
- $[n:=n+3]n>7 \equiv n>4$
- $[n:=n+3]n^2>9 \equiv (n+3)^2>9$

Plus généralement :

$[n:=n+3]P(n) \equiv Q(n)$ avec Q(n) obtenu en remplaçant chaque occurrence de n dans P par n+3

La connaissance de la formule [S]P pour une formule P arbitraire détermine complètement cette substitution S.

=> **Sémantique** de S

Sémantique du langage des substitutions généralisées

Pour chaque construction (substitution) du langage, nous définirons sa syntaxe, ainsi que sa sémantique de la façon suivante :

Syntaxe : x:=e

Sémantique : [x:=e]P ≡ P(e/x) où P(e/x) représente le résultat de la substitution des occurrences libres de x dans P par e (ie. ‘e renomme les x libres dans P’).

Substitution simple (ou affectation)

Syntaxe : x:=e (cas particulier : f(x) :=y).

Sémantique : [x:=e]P ≡ P(e/x)

P(e/x) représente le résultat de la substitution des occurrences libres de x dans P par e (ie. ‘e renomme les x libres dans P’).

Exemple :

[n:=3] n>m ≡ ?

[i:=i+1] i<=1 ≡ ?

Substitution simple (ou affectation)

- ATTENTION !

Un même nom de variable peut avoir simultanément des occurrences libres et des occurrences liées dans une formule :

$n>0 \Rightarrow \forall n.\exists m.m>=n$

Lors de la substitution [var:=E]P, il faut être attentif à ne remplacer que les occurrences libres de var dans P et de plus, il ne faut pas que les variables libres de E se retrouvent liées par des quantificateurs de P

- Exemple : Soit P défini par $n>0 \Rightarrow \forall n.\exists m.m>n$.

$[n:=m+1]P \equiv$?

Substitution multiple

Permet de substituer de manière simultanée plusieurs variables.
Soient x1 et x2 2 variables distinctes.

- **Syntaxe** : x1,x2:=e1,e2

(se généralise sous la forme : x1,...,xn:=e1,...,en avec toutes les variables xi distinctes)

- **Sémantique** : [x1,x2:=e1,e2]P ≡ P(e1,e2 /x1,x2)

où P(e1,…, en /x1,…xn) représente le résultat de la substitution dans P de toutes les occurrences libres de x1 par e1 et simultanément de toutes les occurrences libres de x2 par e2.

Exemple : [x,y:=y,x]x=y ≡ ?

Opération vide

Ne fait rien.

- **Syntaxe** : skip
- **Sémantique** : [skip]P ≡ P

Les substitutions généralisées sont formées à partir des substitutions élémentaires (substitution simple, multiple, skip) en utilisant différents combinateurs.

Pour la suite, P et Q désignent des formules et S, S1 et S2 des substitutions généralisées.

Composition séquentielle

Permet de faire une séquence de substitutions.

- **Syntaxe** : S1;S2
- **Sémantique** : [S1;S2]P ≡ [S1]([S2]P)

Exemple : Calculer [x:=y;y:=x]x=y ≡ ?

La substitution simultanée (en particulier x,y:=y,x) n'est pas équivalente à la substitution séquentielle (x:=y;y:=x) :

[x:=y;y:=x]x=y ≡ ?

[x,y:=y,x]x=y ≡ ?

Alternative

- **Syntaxe** : IF Q THEN S1 ELSE S2 END
- **Sémantique** : [IF Q THEN S1 ELSE S2 END]P ≡ (Q⇒[S1]P) /\ (not(Q)⇒ [S2]P)

Remarque : partie ELSE optionnelle.

IF *P1* THEN *S1* END ≡ IF *P1* THEN *S1* ELSE skip END

Exemples :

IF $x \in \{ 2, 4, 8 \}$ THEN $x := x / 2$ END ;

IF $y + z < 0$ THEN $y := - z$ ELSE $y := 0$ END ;

Garde

Cette substitution généralise l'alternative sous les 2 formes suivantes : un seul choix (Si Q Alors I) ou plusieurs choix (selon Q1 faire I1, Q2 faire I2…)

- **Syntaxe** : SELECT Q THEN S END
- **Sémantique** : [SELECT Q THEN S END]P ≡ (Q ⇒ [S]P)

Exemple : SELECT x>0 THEN y:=y+x END

Plus généralement, on aura :

- **Syntaxe** : SELECT Q1 THEN S1 WHEN Q2 THEN S2 WHEN Q3 THEN S3 END
- **Sémantique** : [SELECT Q1 THEN S1 WHEN Q2 THEN S2 WHEN Q3 THEN S3 END]P ≡ (Q1 ⇒ [S1]P)/\ (Q2 ⇒ [S2]P) /\ (Q3 ⇒ [S3]P)

Q1, Q2, et Q3 n'étant pas nécessairement disjoints (non déterministe)

Exemple : SELECT $x \geq 0$ THEN $y := x^2$ WHEN $x \leq 0$ THEN $y := -x^2$ END

Pré-condition

- **Syntaxe** : PRE Q THEN S END
- **Sémantique** : [PRE Q THEN S END]P ≡ Q /\ [S]P

Si Q est vrai alors l'effet de la substitution préconditionnée est celui de S, sinon on ne peut rien dire…

Exemple : PRE x>0 THEN y:=y+x END

Choix borné (ou choix fermé)

Substitution non déterministe.

- **Syntaxe** : CHOICE S1 OR S2 OR … OR Sk END
- **Sémantique** : [CHOICE S1 OR S2 OR … OR Sk END]P ≡ [S1]P /\ [S2]P /\ … [Sk]P

Exemple :

CHOICE

x1 := x1 + 1

OR

x1 := x1 - 1

END

Choix non borné (ou choix libre)

Substitution non déterministe.

- **Syntaxe** : ANY x WHERE Q THEN S END
- **Sémantique**:[ANY x WHERE Q THEN S END]P ≡ ∀x (Q ⇒[S] P)

Exemples :

ANY x WHERE
    x*(x+1)=2

THEN
    x1,x2:=1,-2

END

ANY *r1*, *r2* WHERE
    r1 ∈ NAT ∧ *r2* ∈ NAT ∧ *r12* + *r22* = 25

THEN
    SommeR := *r1* + *r2*

END

Quelques propriétés…

- x1,x2:=e1,e2 ≡ x2,x1:=e2,e1
- S1||S2 ≡ S2||S1 (exécution simultanée de deux substitutions)
- x1,...,xn:=e1,...,en ≡ x1:=e1 || ... || xn:=en
- [S1]P1 , [S2]P2 ⇒ [S1||S2](P1/\P2)

Opération

Syntaxe :

paramètres de sortie ← nom_opération(paramètres d'entrée) = G ;

- Les paramètres de sortie et d'entrée sont optionnels.
- Le passage des paramètres se fait par valeur.
- Dans une opération d'une machine abstraite :
  - G est en général une substitution préconditionnée,
  - La précondition permet de fixer les conditions sous lesquelles l'opération doit être appelée.
  - Le résultat de G n'est garanti que si sa précondition est valide.

Opération - Exemple

```
MACHINE
  ExempleCinema
SETS
  ACTEURS
VARIABLES
  acteurs
INVARIANT
  acteurs <: ACTEURS
INITIALISATION
  acteurs:={}
OPERATIONS
  AjouterActeur(a) =
    PRE a : ACTEURS - acteurs
    THEN acteurs:= acteurs \/ {a}
    END ;
  SupprimerActeur(a) =
    PRE a : acteurs
    THEN acteurs:= acteurs - {a}
    END ;
END
```

Cohérence des machines

```
MACHINE Nom
SETS S
VARIABLES V
INVARIANT INV
INITIALISATION Init
OPERATIONS
  Op = PRE Q THEN S END;
END
```

Vérification de la cohérence -> obligations de preuve

- Initialisation : *[Init] INV*
- Chaque opération : *Q /\ INV => [S] INV*

Exercices

[xx := 1] (xx /= yy) ≡ ?

[xx := yy] (xx = yy) ≡ ?

[xx := 1] (!(xx).(xx:NAT => xx>=yy) or xx >= 0) ≡ ?

[xx := 0] (yy > 0) ≡ ?

Exercice

Rappel : [PRE Q THEN S END]P ≡ Q /\ [S]P

[PRE xx > 0 THEN xx := xx – 1 END] (xx > yy) ≡ ?

Exercices

Rappel :

[IF Q THEN S1 ELSE S2 END]P ≡ (Q⇒[S1]P) /\ (not(Q)⇒ [S2]P)

`[if x >= y then z:=x else z:=y] (z=max(x,y)) ≡ ?`