Programmation procédurale
Preuve de programmes

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Programmation procédurale / Preuve de programmes

Sommaire

Preuves de programmes
- Introduction
- Correction partielle
- Correction totale

Preuve sur les D-schémas(Hoare)
- Axiome d'affectation
- Séquentielle,
- Implication,
- Conditionnelle
- Répétitive
- Problème de l'arrêt
- Exemple1
- Exemple 2

Preuve sur les R-schémas(Hoare)
- Règle de l'appel récursif
- Exemple

Preuve sur les B-schémas(FLOYD)
- Interprétation
- Vérification
- Exemple

Automatisation

Programmation procédurale / Preuve de programmes

Introduction

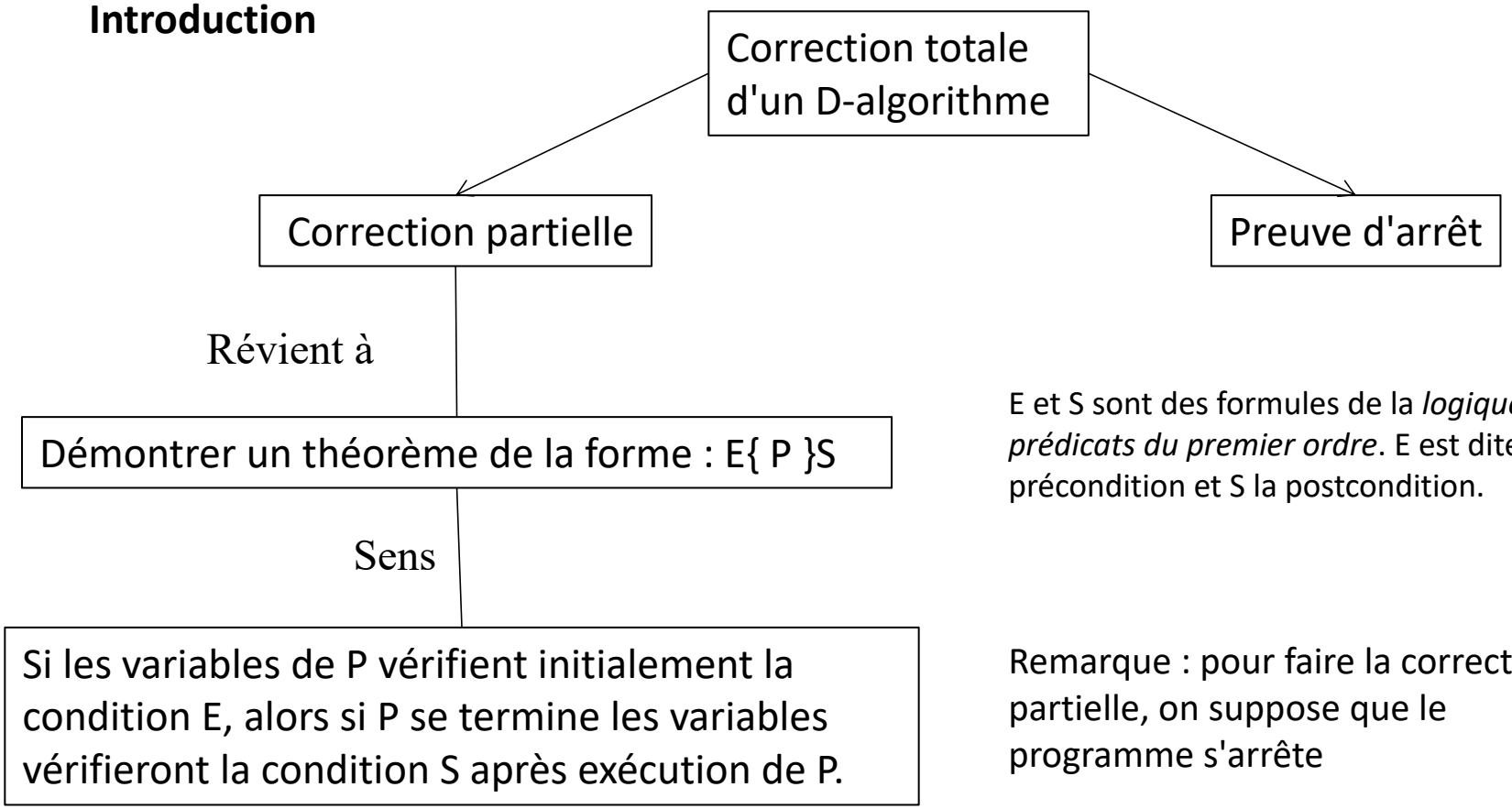
La preuve dépend du type de schéma.

Théorie basée sur

des systèmes formels de preuve, avec des démonstrateurs automatiques.	le point fixe (pour les algorithmes récursifs)

Types de preuve considérés : D, R et B-schémas

Introduction



E et S sont des formules de la *logique des prédicats du premier ordre*. E est dite précondition et S la postcondition.

Remarque : pour faire la correction partielle, on suppose que le programme s'arrête

Introduction

Soit le programme DIV suivant qui calcule le quotient(q) et le reste(r) de la division entière de 2 entiers a et b.

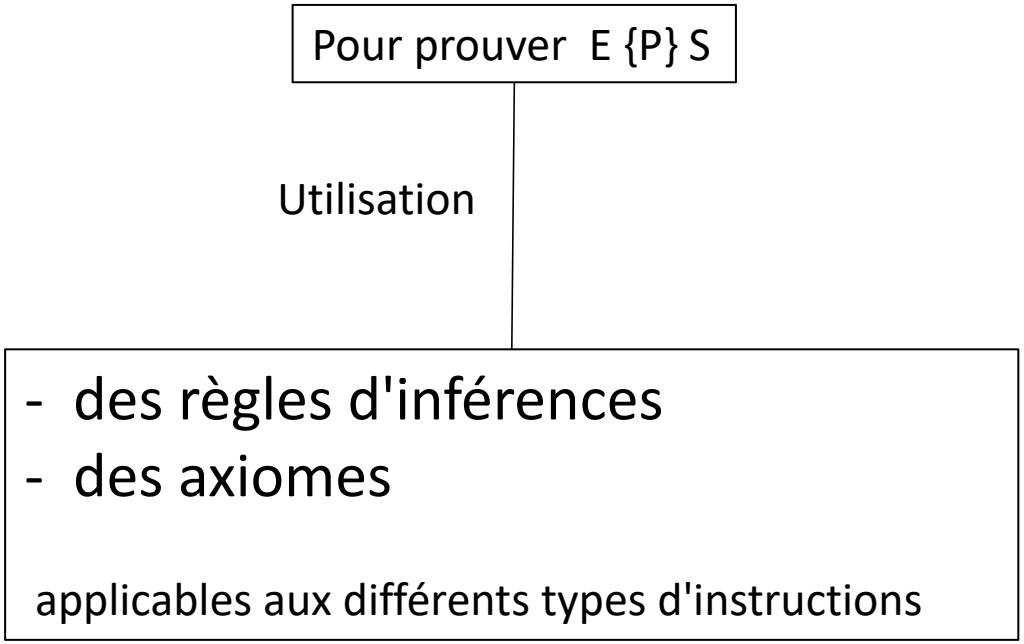
```
q := 0
r := a
Tantque r >= b :
        r := r - b
        q := q + 1
Fintantque
```

La preuve partielle de DIV consiste à démontrer le théorème:

(a ≥0 & b>0) { DIV } (a=bq+r & r<b)

Programmation procédurale / Preuve de programmes

Introduction



SYSTEME FORMEL
de preuve
(HOARE)

Preuve sur les D-schémas (HOARE)

Axiome d'affectation :

Pour toute affectation de la forme " x := exp ", on peut inférer grâce à l'axiome de l'affectation (AFF) un théorème de la forme :

T(exp) { x := exp } T(x)

où le prédicat T(exp) est obtenu à partir de T(x), en substituant (dans la post-condition) toutes les occurrences libres de x par exp.

Programmation procédurale /

Preuve de programmes

Preuve sur les D-schémas (HOARE)

Exemple : (x+1 > 0){ x := x + 1 }(x > 0)

Interprétation :Si " x+1 > 0 " est vraie avant l ’exécution de " x := x+1 ", alors après son exécution, la condition " x>0" devient vraie.

Ou bien
Pour que la condition " x>0" soit vraie après l'exécution de " x:=x+1", il est nécessaire que la condition "x+1 > 0" soit vérifiée, avant l'exécution de l'affectation.

La pré-condition obtenue : la plus faible condition (contraintes minimales que doivent vérifier les variables du programme, pour que la post-condition soit vérifiée après l'exécution de l'affectation)

Preuve sur les D-schémas (HOARE)

Règle de la séquentielle.

$$\frac{E\ \{P1\}\ F \quad \&\quad F\ \{P2\}\ S}{E\ \{P1;P2\}\ S}\ (SEQ)$$

Si on a déjà prouvé les deux théorèmes E {P1} F et F {P2} S, alors on peut déduire par cette règle le théorème E {P1;P2} S

Programmation procédurale /

Preuve de programmes

$$\frac{E\ \{P1\}\ F \quad \&\quad F\ \{P2\}\ S}{E\ \{P1;P2\}\ S}\ (SEQ)$$

Preuve sur les D-schémas (HOARE)

Exemple : soit à prouver (t>0) { t :=t+1; t := t-1 } (t>0)

D'après SEQ, il faut prouver

a) (t>0) { t := t+1 } F

b) F { t := t-1 } (t>0)

En appliquant AFF à b) on trouve F = (t-1) > 0 (en substituant dans la post-condition t-1 à t).

F est donc t > 1

On a ainsi (t>1) {t:=t-1} (t>0)

Appliquant de nouveau AFF à a). Ce qui donne (t+1 > 1) { t := t+1} (t>1)

Ce qui est équivalent à (t>0) { t := t+1 } (t>1).

Les deux prémices de la règle SEQ étant prouvées, l'énoncé (t>0){ t:=t+1; t := t-1 } (t>0) est par conséquent un théorème.

Preuve sur les D-schémas (HOARE)

Règle de l'implication

$$\frac{E \rightarrow F \;\&\; F\,\{P\}\,S}{E\,\{P\}\,S}\ (IMP1)$$

$$\frac{E\,\{P\}\,F \;\&\; F \rightarrow S}{E\,\{P\}\,S}\ (IMP2)$$

Exemple : prouver l'énoncé (x>0) {x:=x+1} (x>0)

(AFF) donne (x+1 >0) { x:=x+1} (x>0)

De plus l'implication (x>0) --> (x+1) > 0 est toujours vraie

Donc d'après (IMP1), on a

(x>0) {x:=x+1} (x>0)

Preuve sur les D-schémas (HOARE)

En utilisant les règles (SEQ), (IMP1), (IMP2) et l'axiome de l'affectation, on peut facilement prouver un théorème de la forme : E {P1; P2; ....Pn} S où les Pi sont des affectations.

Il suffirait de commencer la preuve par la dernière affectation et de trouver de proche en proche la plus faible pré-condition.

T1{P1}T2 &T2{P2}T3 ..& Tn{Pn}S

<----------

Sens de la preuve

Si de plus, on arrive à prouver E --> T1, alors l'énoncé E {P1; P2; ....Pn} S est prouvé.

Preuve sur les D-schémas (HOARE)

Règle de conditionnelle

(E & B){P1}S et (E & Non B){P2}S

----------(COND1)

E { SI B : P1 SINON P2 FSI } S

(E & B){P}S et (E & Non B) --> S

----------(COND2)

E { SI B : P FSI } S

Programmation procédurale /

Preuve de programmes

(E & B){P1}S et (E & Non B){P2}S

----------(COND1)

E { SI B : P1 SINON P2 FSI } S

Preuve sur les D-schémas (HOARE)

Exemple : prouver
(x>10) { SI x=30 : y:=x-1 SINON y:=x FSI}
(y>10)

Il faut donc montrer
a) (x>10) & (x=30) { y := x-1 } (y>10)
b) (x>10) & (x#30) { y := x } (y>10)

Montrons a)
D'après AFF : (x>11) { y := x-1 } (y>10)

L'implication
(x>10) & (x=30) --> (x > 11) est toujours vraie

Donc d'après (IMP1) on a a).

Montrons b)

D'après AFF : (x>10) { y := x } (y>10)

De même, l'implication
(x>10) & (x#30) --> (x > 10) est toujours vraie.

Donc d'après (IMP1) on a b).

a) et b) étant démontrés, donc d'après (COND1) l'énoncé de départ est démontré.

Preuve sur les D-schémas (HOARE)

Règle de l'itération(répétitive)

(E & B) { P } E

----------(ITE)

E { TQ B : P FTQ } (E & Non B)

Cette règle stipule que si une condition E reste inchangée après l ’exécution d'une itération (le corps de la boucle P), alors cette même condition restera inchangée durant toutes les itérations de la boucle.

E est appelée Invariant de la boucle.

Preuve sur les D-schémas (HOARE)

Montrer que (a =bq+r) est un invarant pour la boucle

TANTQUE r >=b :
    r := r-b ; q := q+1
FINTANTQUE

Il faut montrer

(a=bq+r) { r := r-b ; q := q+1 } (a=bq+r)

D'après (SEQ), il faut montrer

E { q := q + 1 } (a=bq+r) (1)

(a= bq + r) { r := r-b } E (2)

Par (AFF) (1) donne

E= (a= b(q+1) + r)

E =(a= bq+b + r)

Par (AFF) (2) donne

(a= bq + b + r-b)

C'est à dire

(a= bq + r)

Preuve sur les D-schémas (HOARE) : Exemple 1

Exemple : Prouver

(a ≥0 & b>0)

```
q := 0
r := a
TQ r >=b :
    r := r-b ; q := q+1
FTQ
```

(a=bq+r) & r<b)

Programmation procédurale /

Preuve de programmes

$$\frac{(E \,\&\, B)\ \{P\}\ E}{E\ \{TQ\ B : P\ FTQ\}\ (E \,\&\, \text{Non } B)}\ (ITE)$$

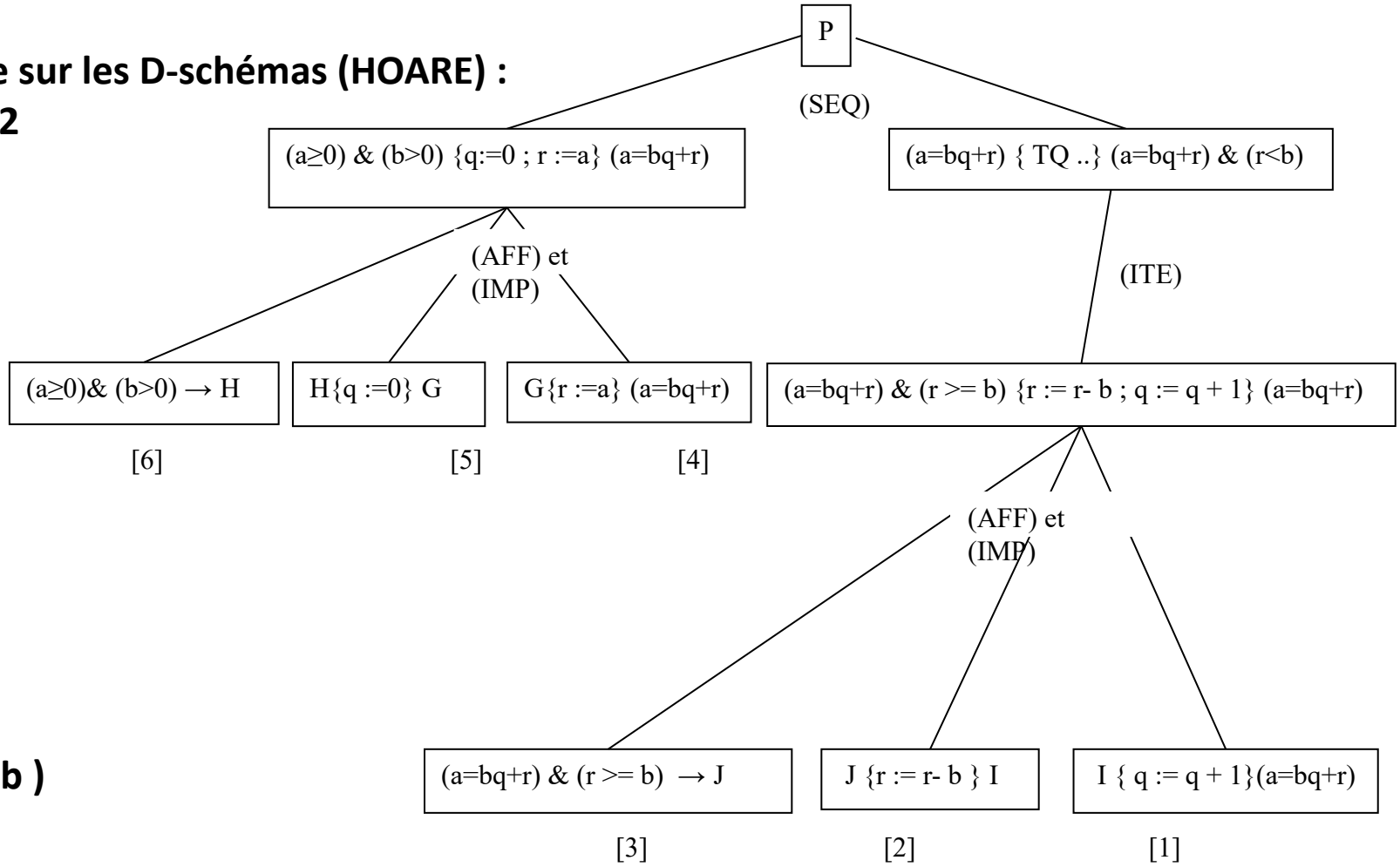
Preuve sur les D-schémas (HOARE) :

Exemple 2

(a ≥0 & b>0)

q := 0
r := a
TQ r >=b :
  r := r-b ;
  q := q+1
FTQ

(a=bq+r) & r<b)



Preuve sur les D-schémas (HOARE) : Exemple 1

Il faut donc prouver [1], [2], ..

[1] donne par (AFF) I = (a= bq+b + r)

[2] donne par (AFF) J= (a=bq + r)

L'implication [3] (a=bq+r & r >=b) --> (a=bq+r) est toujours vraie.

[4]donne par (AFF) G= (a=bq + a) = (bq = 0)

[5] donne par (AFF) H= (0=0)

L'implication [6] (a≥0)& (b>0) → (0=0) est toujours vraie

Programmation procédurale / Preuve de programmes

Preuve sur les D-schémas (HOARE) :

Exemple 1

Preuve d'arrêt.

Pour prouver l'arrêt d'un D-algorithme, il suffit de prouver l'arrêt de chaque instruction "TANTQUE".

Pour cela il faut trouver une application m qui décroît à chaque itération.

```
q := 0
r := a
TQ r >= b :
    r := r - b
    q := q + 1
FTQ
```

m : r, r-b, r-2b,...r-ib, ...
avec r-ib >= 0

Preuve sur les D-schémas (HOARE) : Exemple 2

Prouver que le programme suivant

```
x1 := 0 ;
x2 := 0 ;
TQ x2 # a
  x1 := x1 + 2x2 + 1
  x2 := x2  + 1
FTQ
```

calcule le carré d'un nombre a positif ou nul donné.

Hypothèse : $(a \geq 0)$

Conclusion : $(x1 = a^2)$

Programmation procédurale

/ Preuve de programmes

$$\frac{(E \,\&\, B)\ \{P\}\ E}{E\ \{TQ\ B : P\ FTQ\}\ (E \,\&\, \text{Non } B)}\ (ITE)$$

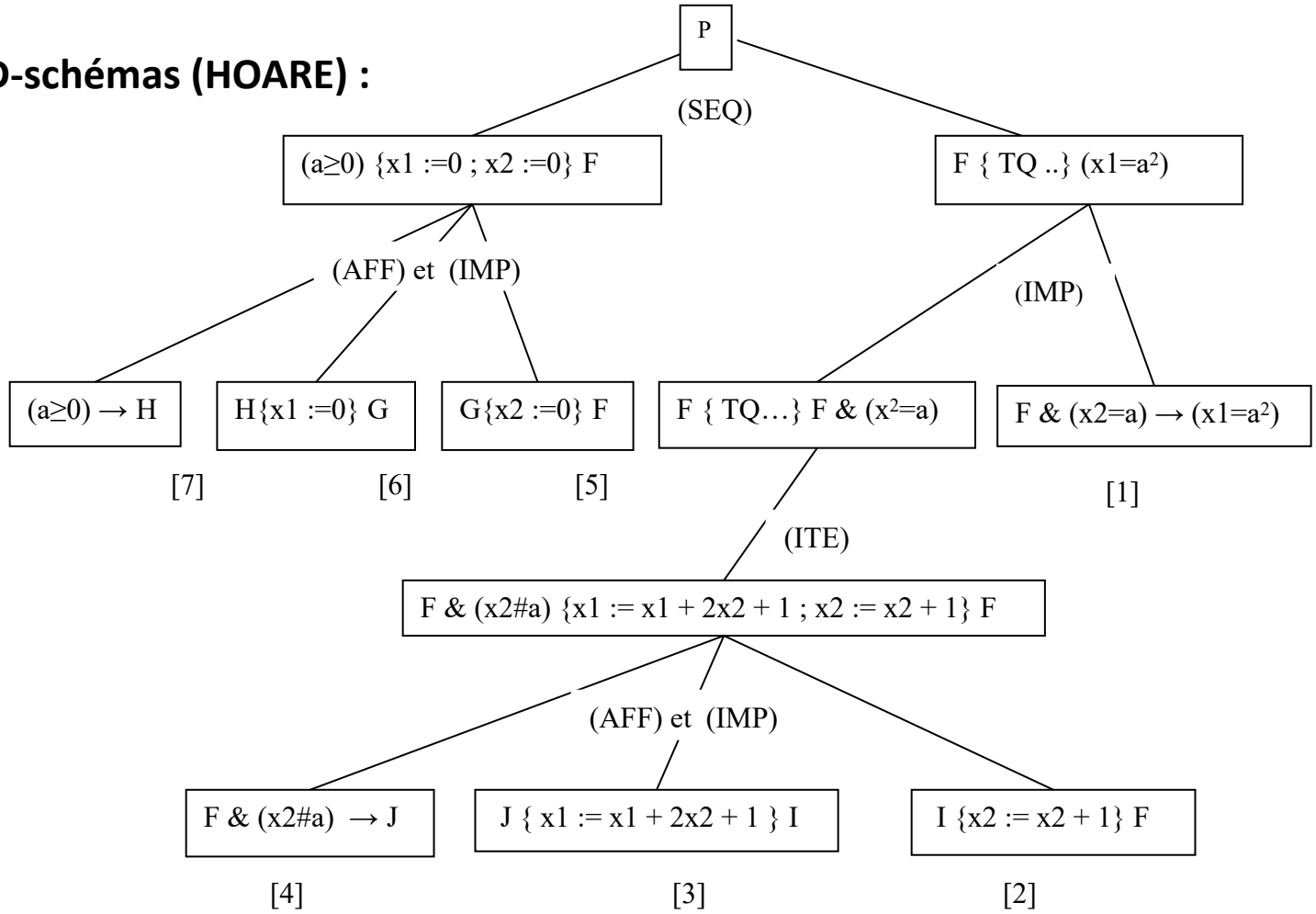
Preuve sur les D-schémas (HOARE) :

Exemple 2

(a≥0)

x1 := 0 ;
x2 := 0 ;
TQ x2 # a
  x1 := x1 + 2x2 + 1
  x2 := x2 + 1
FTQ

(x1 = a²)



Preuve sur les D-schémas (HOARE) : Exemple 2

En général, la preuve est faite selon l'ordre postordre inverse. Il faut donc prouver [1], [2], etc.
F désigne l'invariant de la boucle et constitue ainsi la clé du problème.
On doit donc proposer un invariant et essayer de faire la preuve.

Posons F = $(x1=x2^2)$
F est bien un invariant car
$(x1=x2^2)$ {x1 := x1 + 2x2 + 1 ; x2 := x2 + 1} $(x1=x2^2)$

En effet d'après (SEQ) :
I {x2 := x2 + 1} $(x1=x2^2)$ (1)
Z {x1 := x1 + 2x2 + 1} I (2)
C'est-à-dire il faut montrer que Z est bien F.

De (1) on déduit I = $(x1 = (x2+1)^2)$ et de (2) on déduit Z = $(x1=x2^2)$ qui est bien égal à F.

Preuve sur les D-schémas (HOARE)

Preuve de [1] : F & (x2=a) $\rightarrow$ (x1=a^2)
En remplaçant F par (x1=$x2^2$), nous trouvons (x1=a^2) (Ce qu'il faut démontrer)

Preuve de [2] : I {x2 := x2 + 1} F
Il s'agit de trouver I. I est donc (x1 = $(x2+1)^2$)

Preuve de [3] : J {x1 := x1 + 2x2 + 1} I
Il s'agit de trouver J. J est donc (x1=$x2^2$)

Preuve de [4] : F & (x2#a) $\rightarrow$ J
Il est facile de remarquer que (x1=$x2^2$) & (x2#a) $\rightarrow$ (x1=$x2^2$)

Preuve de [5] : G{x2 :=0} F
G est donc (x1=0)

Preuve de [6] : H{x1 :=0} G
H est (0=0)

Preuve de [7] : (a≥0) $\rightarrow$ H
Comme H= (0=0) est vraie, l'implication est vraie.

Programmation procédurale /

Preuve de programmes

R = A ∪ *R;R* ∪ Si P *R* [Sinon *R*]
Fsi + appels récursifs
A : actions // P : Prédicats

Preuves sur les R-schémas (HOARE)

Existence d'une régle dans le système de Hoare pour démontrer les algorithmes récursifs.

Soit f(x;y) : μ une déclaration de procédure de corps μ.

x : listes des paramètres valeurs(entrées)
y : listes des paramètres résultats(sorties)

Nous voulons démontrer p(x) {f(x;y):μ} q(x,y)
(Si p(x) est vérifiée à l'entrée de la procédure alors q(x,y) est vérifiée à la sortie)

Théorème :

Si l'on peut démontrer la correction du corps μ relativement au prédicat d'entrée p(x) et au prédicat de sortie q(x, y)

sous l'hypothèse de la correction par rapport à p et q de tous les appels internes

alors

p(x) {f(x;y) : μ} q(x, y)

est vraie

Preuves sur les R-schémas (HOARE)

Exemple : Soit la procédure récursive :

$F_{91}(x,y)$ =si x > 100 : y := x-10
Sinon $F_{91}(x+11,z)$; $f_{91}(z, y)$ Fsi

Démontrer la correction partielle de cette procédure relativement aux prédicats p et q suivants:

p(x) : (x>=0)

q(x, y) : (x>100 --> y =x-10)et (x<=100 --> y=91)

Programmation procédurale /

Preuve de programmes

(x>=0)

$F_{91}(x,y)$ =si x > 100 : y := x-10

sinon $F_{91}(x+11,z)$; $f_{91}(z, y)$ Fsi

(x>100 --> y =x-10)et (x<=100 --> y=91)

Preuves sur les R-schémas (HOARE)

Cas x > 100, le résultat est immédiat.

Cas 0 <= x <= 100, supposons la correction des appels internes, c'est à dire :

1) (x+11) >=0 {F91(x+11; z)}

(x>89 -->z=x+1) et x<=89 --> z=91

2) z>=0 {F91(z;y)}

(z>100 -->y=z-10)et (z<=100 --> y=91)

D'après 1) deux cas apparaissent:

x>89 : donc z = x+1

et d'après 2) deux sous cas

1-1) x+1 > 100, c'est à dire x>99, donc x =100 et donc y=x+1-10=91 (q(x,y) vérifiée)

1-2) x+1 <=100, c'est à dire x <= 99 donc y=91 (q(x,y) vérifiée)

x<=89 donc z=91 d'après 1) et 2) y=91 (q(x, y) vérifiée)

Preuves sur les B-langages(Floyd)

- Définir une interprétation P_i pour chaque instruction S_i du B-schéma

Interprétation : formule en logique mathématique

Instruction = affectation, branchement conditionnel ou branchement inconditionel

P_i est l'antécédent pour l'instruction S_i et P_{i+1} est le conséquent de l'instruction S_i

$P_i \wedge [Si] \Rightarrow P_{i+1}$.

- Entreprendre les vérifications

3 types : affectation, branchement conditionnel et branchement inconditionnel

Preuves sur les B-langages (Floyd) : Interprétation

1. i = 1
2. S = 0
3.
4. if (i > n) goto 8
5. S = S + ai
6. i = i + 1
7. goto 4
8. Halt

$P1 \equiv n \in N^+$ (N^+ est l'ensemble des entiers positifs)

$P2 \equiv n \in N^+ \wedge i = 1$

$P3 \equiv n \in N^+ \wedge i = 1 \wedge S = 0$

$P4 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n + 1 \wedge S = \sum(j=1, i-1)\ aj$

$P5 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n \wedge S = \sum(j=1, i-1)\ aj$

$P6 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n \wedge S = \sum(j=1, i)\ aj$

$P7 \equiv n \in N^+ \wedge i \in N^+ \wedge 2 \leq i \leq n + 1 \wedge S = \sum(j=1, i-1)aj$

$P8 \equiv n \in N^+ \wedge i = n + 1 \wedge S = \sum(j=1,i-1)\ aj$; cad., $S = \sum(j=1, n)\ aj$

Preuves sur les B-langages(Floyd) :

Vérification

Condition de vérification de l'affectation

- L'affectation x ← f(x, y) est vérifiée pour l'antécédent P1(x, y) et le conséquent Q1(x, y) si nous pouvons montrer que P1(x, y) ⇒ Q1(f(x, y), y)

(Substituer chaque occurrence de x dans Q1(x, y) par l'expression f(x, y))

(Substitution en sens inverse)

S = S + ai

Antécédent :

$P5 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n \wedge S = \sum_{j=1, i-1} a_j$

Conséquent

$P6 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n \wedge S = \sum_{j=1, i} a_j$

Montrer que P5 ⇒ P6

Remplacement de S par S+ai dans P6 donne P5

Preuves sur les B-langages(Floyd) : Vérification

Condition de vérification du branchement conditionnel

- Supposons que c est l'instruction de branchement conditionnel avec l'antécédent P1, le test Φ, et le consequent Q1 quand Φ est vrai et le conséquent Q2 quand Φ est faux.

- La condition de vérification notée Vc(P1; Q1, Q2) pour c est

  $((P1 \wedge \Phi) \Rightarrow Q1) \wedge ((P1 \wedge \neg\Phi) \Rightarrow Q2)$

if (i > n) goto 8

Antécédent:

$P4 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n + 1 \wedge S \sum j=1, i-1\ aj$

Conséquent si (i>n) vrai:

$P8 \equiv n \in N^+ \wedge i = n + 1 \wedge S = \sum j=1,i-1\ aj$; i.e., $S = \sum j=1, n\ aj$

Conséquent si (i>n) faux

$P5 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n \wedge S = \sum j=1, i-1\ aj$

Montrer que

- P4 et (i>n) ⇒ P8
- P4 et (i<=n)⇒ P5

Preuves sur les B-langages(Floyd) : Vérification

Condition de vérification du branchement conditionnel

- Est considéré comme instruction spéciale avec les antecedents P1, P2, ... et le conséquent Q1.

- La condition de vérification notée Vc(P1, P2, ...; Q1) est $(P1 \Rightarrow Q1) \wedge (P2 \Rightarrow Q1) \wedge \ldots$

  Ce qui est équivalent à

  $(P1 \vee P2 \vee \ldots) \Rightarrow Q1$

goto 4

Antécédent:

$P7 \equiv n \in N^+ \wedge i \in N^+ \wedge 2 \leq i \leq n + 1 \wedge S = \sum j=1, i-1\ aj$

Conséquent

$P4 \equiv n \in N^+ \wedge i \in N^+ \wedge i \leq n + 1 \wedge S = \sum j=1, i-1\ aj$

Montrer que

$P7 \Rightarrow P4$

Programmation procédurale / Preuve de programmes

Automatisation

- Pour un D-algorithme

--> prouveur semi-automatique (puisque on ne peut déterminer l'invariant de manière automatique)

- Pour un B-algorithme

Plus dur à réaliser car il faut définir préalablement l'interprétation (+ invariant)

- Pour un R-algorithme

--> prouveur entièrement automatique puisque pas de problème d'invariant.

Remarque : Pour D et B, implémenter un démonstrateur de théorèmes (a, b, c, ... --> but)

Pour R : simplificateur d'expressions