Programmation procédurale

Transformation de programmes

D.E ZEGOUR
École Supérieure d'Informatique
ESI

Programmation procédurale / Transformation de programmes

Sommaire

Programmation procédurale / Transformation de programmes

Introduction

Transformation de programmes = modification source à source.

Une transformation doit être

- compréhensible,
- assurer l'équivalence,
- précise pour permettre une automatisation.

On distingue :
- les transformations syntaxiques
- les transformations sémantiques (éliminer les tests inutiles, détecter et éliminer les parties inaccessibles d'un programme , ...)

Toutes les transformations que nous développons ici sont fonctionnelles.

Programmation procédurale / Transformation de programmes

Introduction : Pourquoi transformer des algorithmes ?

Type de construction = mode de pensée.

Algorithme

- exprimé dans un formalisme (selon l'utilisateur).
- transfromé en une forme structurée (but : le rendre plus lisible, plus facile à prouver et a mettre au point)

Domaines d'utilisation :

. Synthèse des programmes

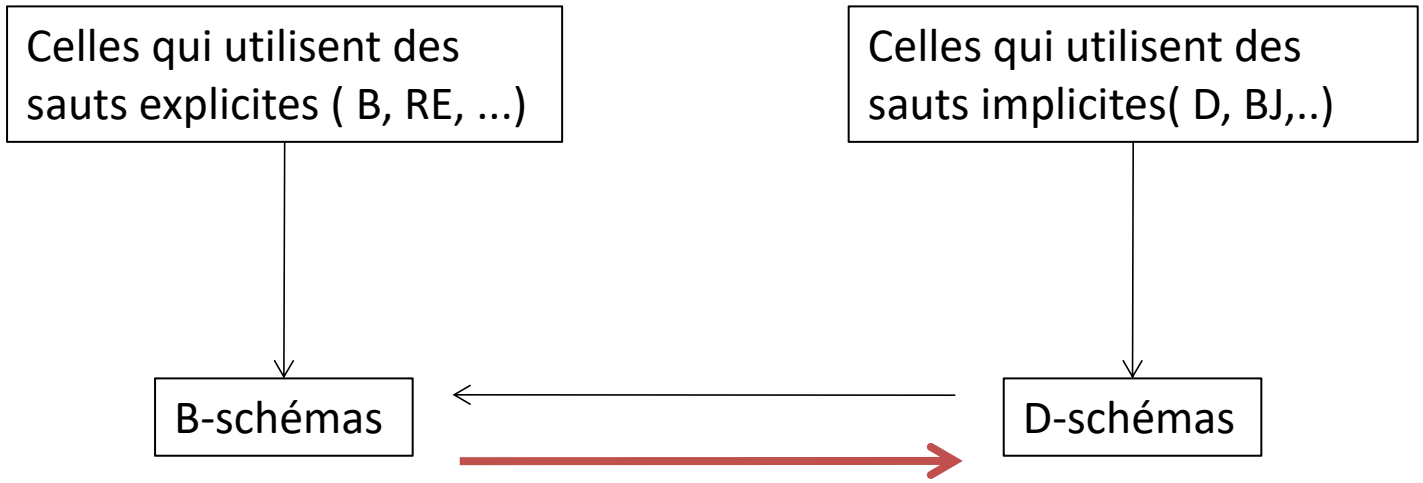
(Partir d'une spécification mathématique et arriver par une série de transformations à un programme)

.Structuration d'algorithmes

Programmation procédurale / Transformation de programmes

Introduction

Toutes les structures que nous avons présentées (des langages procéduraux) peuvent être répartie en deux catégories :



Programmation procédurale / Transformation de programmes

Introduction

Deux articles contradictoires méritent d'être évoqués :

DIJKSTRA(1968) critique l'utilisation des GOTO

L'instruction GOTO a des effets désastreux"

KNUTH (1974) "Programmation structurée avec GOTO".

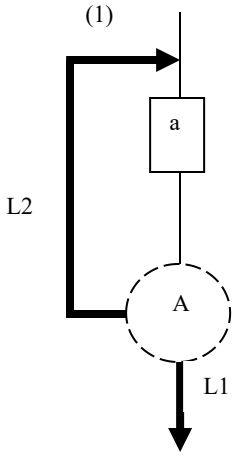
Le GOTO pourrait être un outil efficace s'il est bien utilisé.

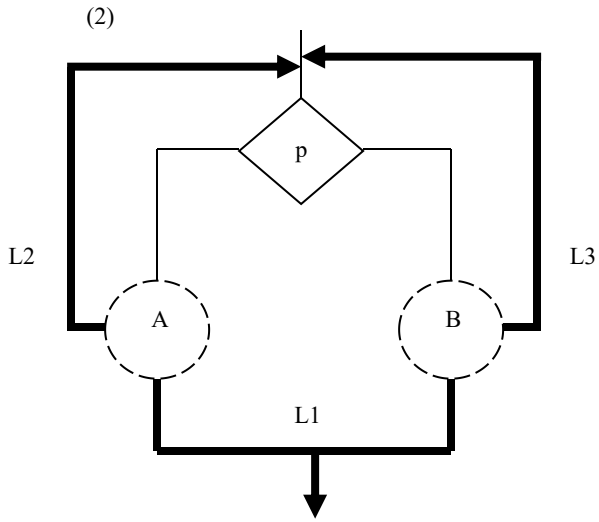
Programmation procédurale / Transformation de programmes

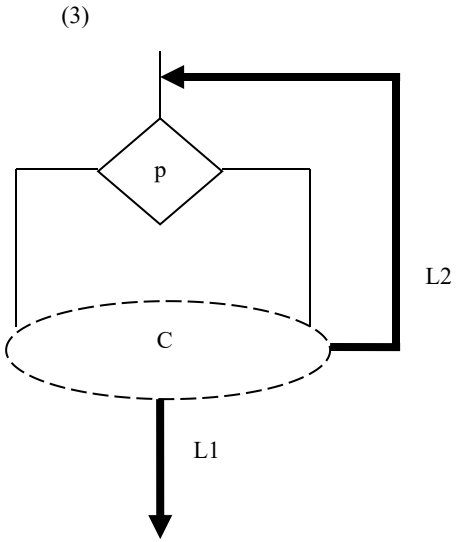
Transformation de Bohm & Jaccopini

Historiquement, la première (1966)

3 Types d'organigrammes



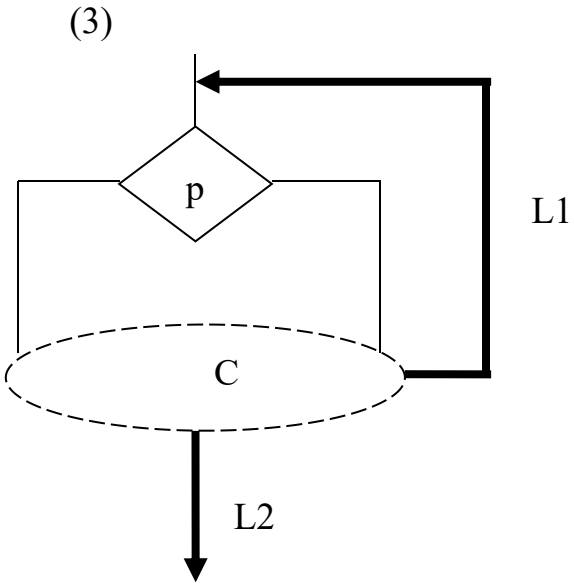




Programmation procédurale / Transformation de programmes

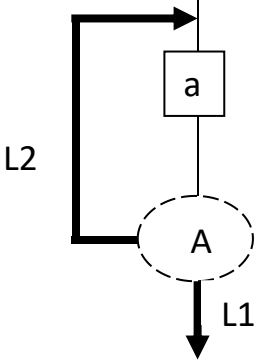
Transformation de Bohm & Jaccopini

Cas 3 peut se transformer en cas 2 avec duplication des actions et des tests

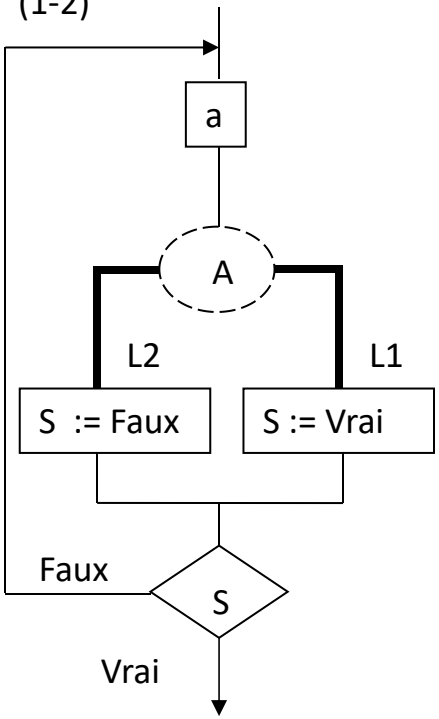


Transformation de Bohm & Jaccopini (Cas 1)

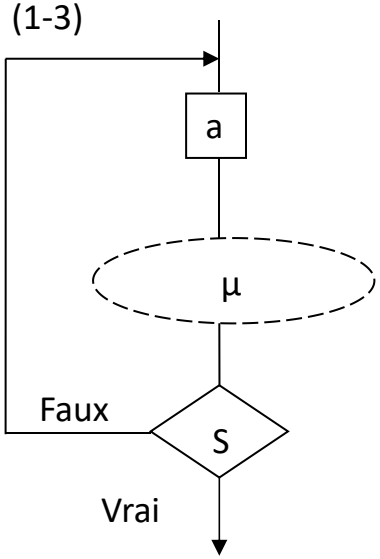
(1-1)



(1-2)



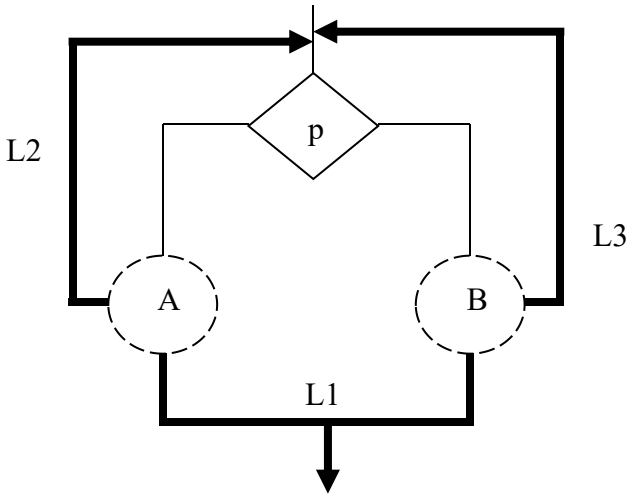
(1-3)



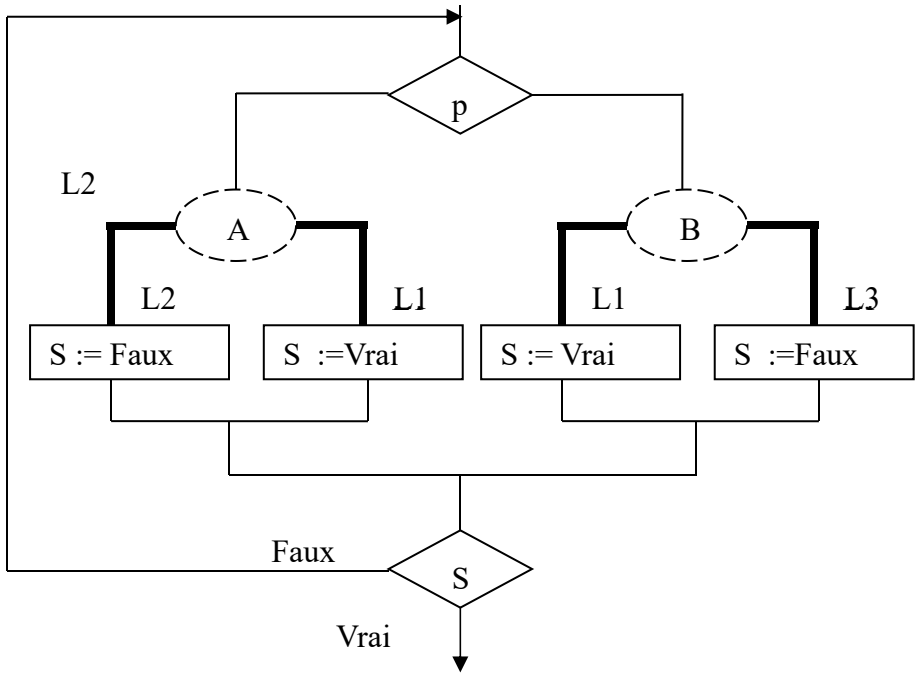
Repeat
a ; μ
Until S

Transformation de Bohm & Jaccopini (Cas 2)

(2-1)

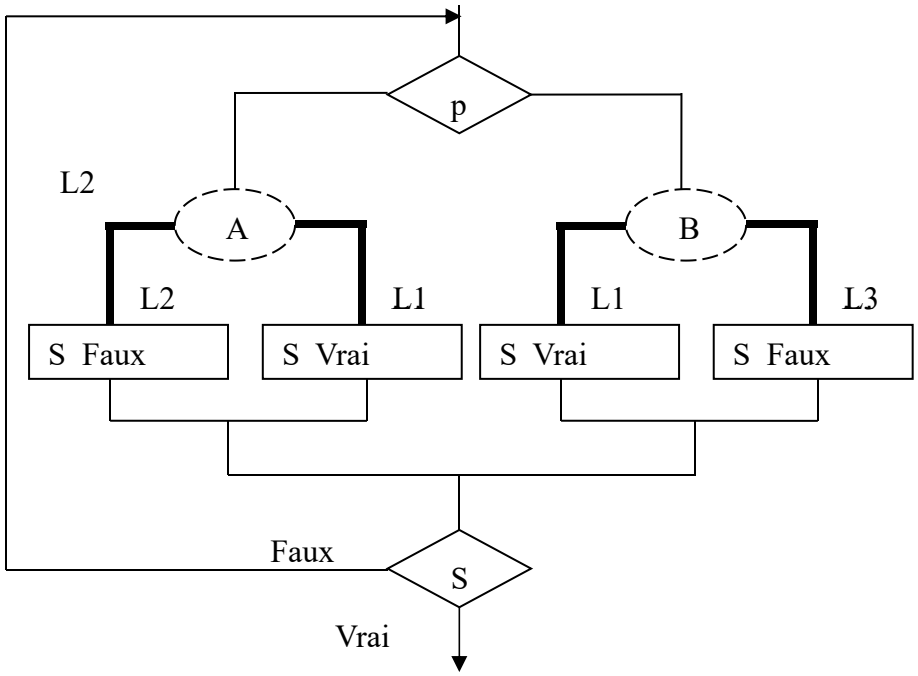


(2-2)

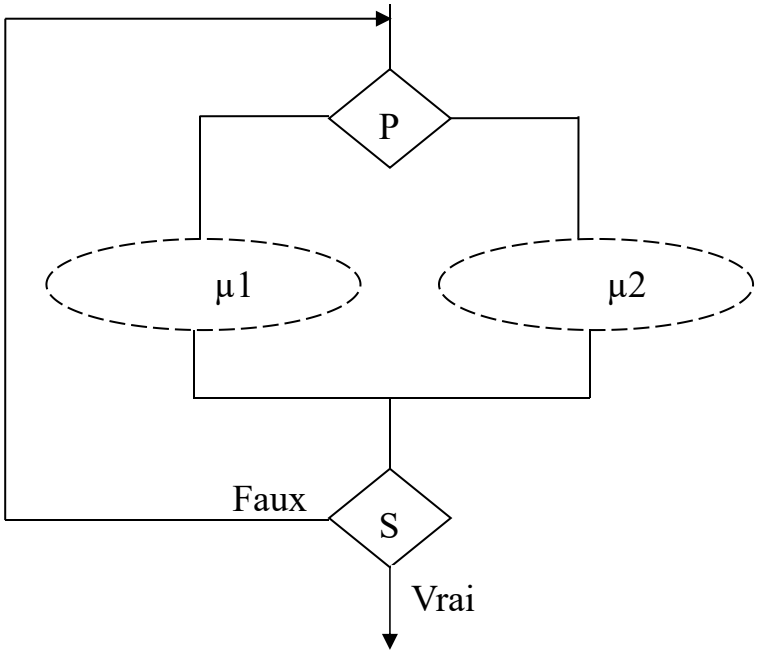


Transformation de Bohm & Jaccopini (Cas 2)

(2-2)

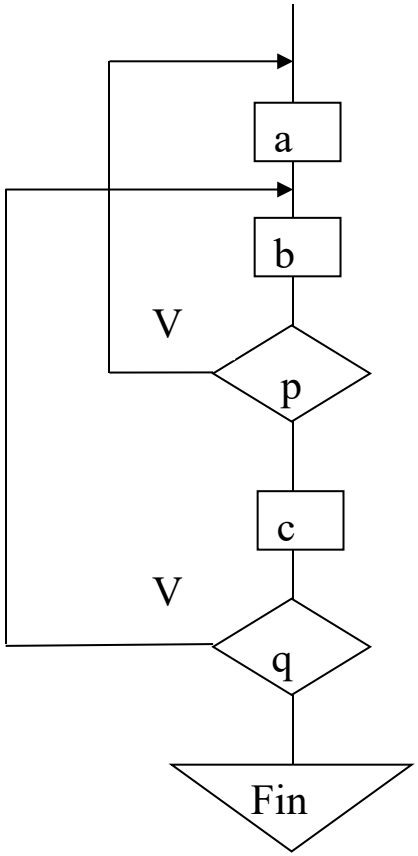


(2-3)



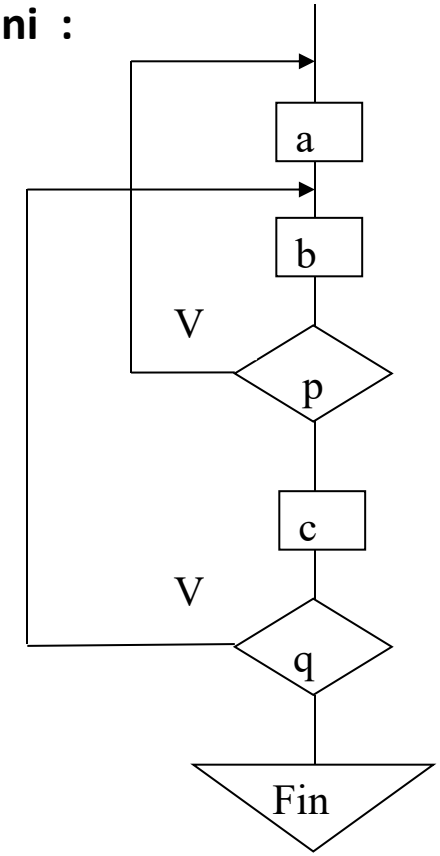
Repeat
Si P : µ1 Sinon µ2 Fsi
Until S

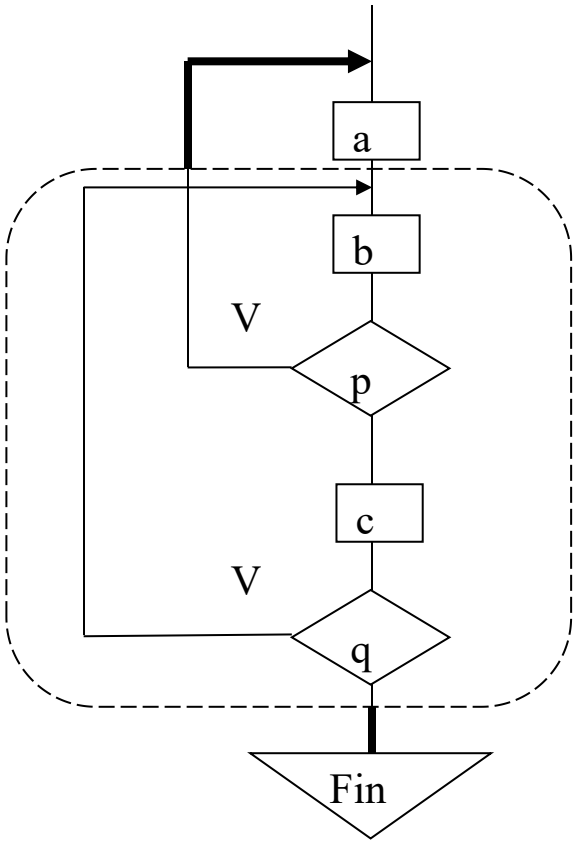
Transformation de Bohm & Jaccopini : Exemple



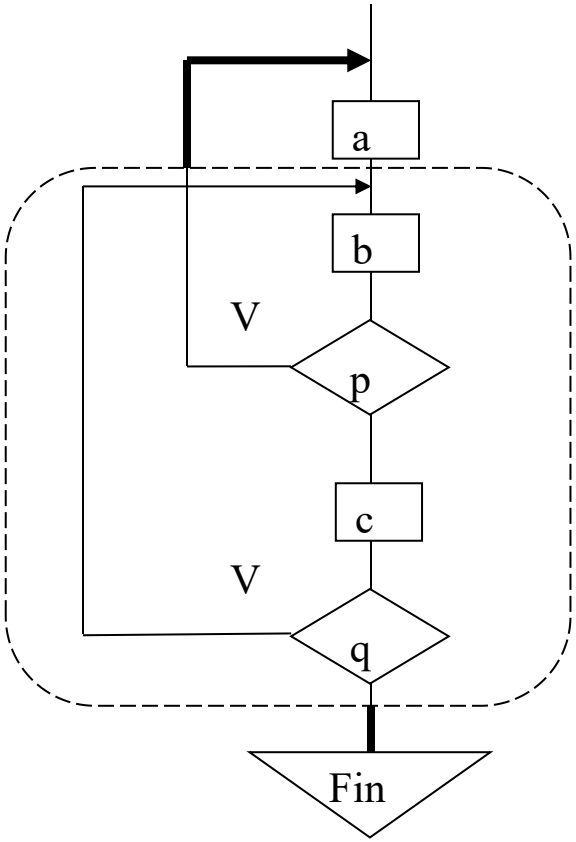

```
1 : a
2 : b
Si p Aller à 1
c
Si q Aller à 2
Fin
```

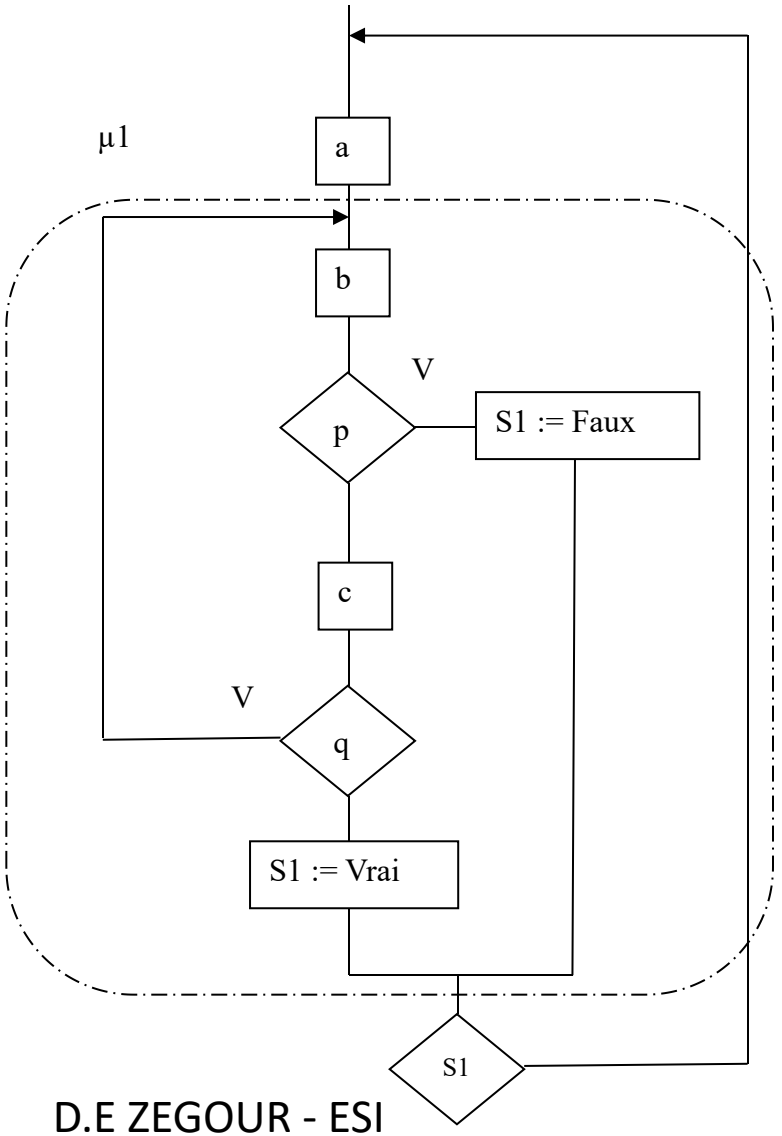
Transformation de Bohm & Jaccopini : Exemple





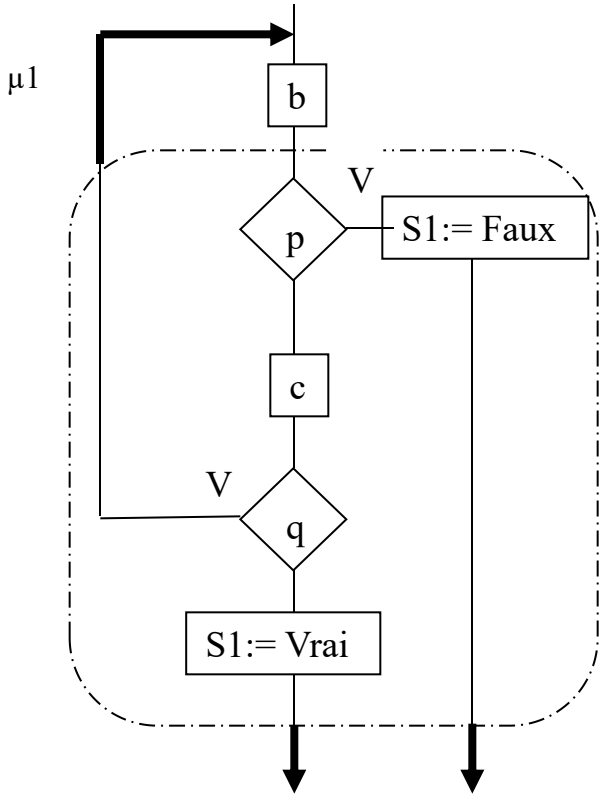
Transformation de Bohm & Jaccopini : Exemple

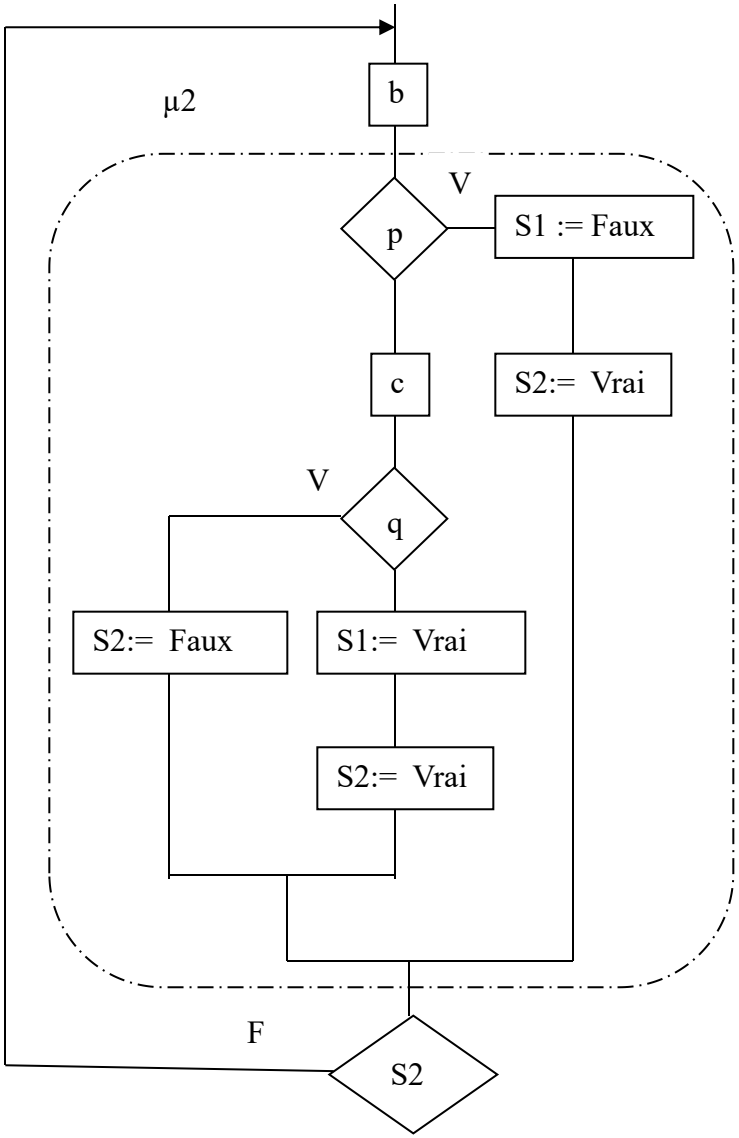




Répéter
a ;µ1
Jusqu'à S1

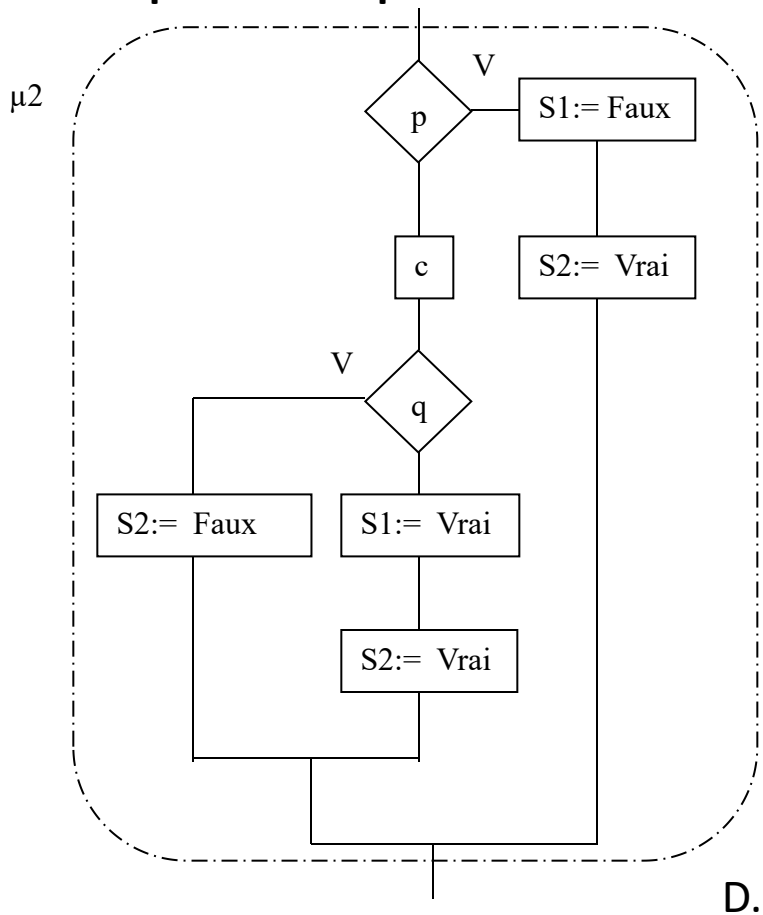
Transformation de Bohm & Jaccopini : Exemple





μ1=
Répéter
b ; μ2
Jusqu'à S2

Transformation de Bohm & Jaccopini : Exemple



μ2 est déjà un organigramme de type simple dont l'algorithme correspondant est le suivant :

```
μ2=
Si p
   S1 := Faux ; S2 :=Vrai
Sinon
   c
   Si q
      S2 := Faux
   Sinon
      S1 := Vrai ; S2:= Vrai
   Fsi
Fsi
```

Transformation de Bohm & Jaccopini : Exemple

```
1 : a
2 : b
Si p Aller à 1
c
Si q Aller à 2
Fin
```

```
Repéter
    a
    Répeter
        b
        Si  p
            S1 := Faux ; S2 := Vrai
        Sinon
            c
            Si q
                S2 := Faux
            Sinon
                S1 := Vrai ; S2 := Vrai
            Fsi
        Fsi
    Jusqu' à S2
Jusqu'à S1
```

Programmation procédurale / Transformation de programmes

Technique par automate (B vers D)

Pour tout B-algorithme, on associe un automate.

De l'automate découle l'algorithme correspondant.

Chaque étiquette dans le B-algorithme représente un état dans l'automate.

Quelques étiquettes additionnelles sont nécessaires au niveau du B-algorithme pour réaliser la transformation.

Programmation procédurale / Transformation de programmes

Technique par automate

Soit le B-algorithme suivant :

```
        a
        b
E1      c
        d
        IF t1 GOTO E2
        x
E3      IF t2 GOTO E4
        z
        GOTO E1
E2      r
        s
        u
        GOTO E3
        u
        v
E4      x
        y
        z
```

Programmation procédurale / Transformation de programmes

Technique par automate (*Etape 1 : Réecriture)*

	a
	b
E1	c
	d
	IF t1 GOTO E2
	x
E3	IF t2 GOTO E4
	z
	GOTO E1
E2	r
	s
	u
	GOTO E3
	u
	v
E4	x
	y
	z

Étiqueter la première instruction.

Étiqueter toute instruction conditionnelle

Regrouper les actions qui se suivent qui ne sont pas étiquetées

$0	[a, b]
E1	[c, d]
$1	IF t1 GOTO E2
	[x]
E3	IF t2 GOTO E4
	[z]
	GOTO E1
E2	[r, s, u]
	GOTO E3
	[u, v]
E4	[x, y, z]

Programmation procédurale / Transformation de programmes

Technique par automate (*Etape 1 : Réecriture)*

$0	[a, b]
E1	[c, d]
$1	IF t1 GOTO E2
	[x]
E3	IF t2 GOTO E4
	[z]
	GOTO E1
E2	[r, s, u]
	GOTO E3
	[u, v]
E4	[x, y, z]

Construction d'une table de correspondance (COND) entre les états et les conditions.

Table Cond :

$0	Vrai
E1	Vrai
$1	t1
E3	t2
E2	Vrai
E4	Vrai

Technique par automate (*Etape 2 : matrice de transition)*

$0 [a, b]
E1 [c, d]
$1 IF t1 GOTO E2
[x]
E3 IF t2 GOTO E4
[z]
GOTO E1
E2 [r, s, u]
GOTO E3
[u, v]
E4 [x, y, z]

Table Cond :

$0 Vrai
E1 Vrai
$1 t1
E3 t2
E2 Vrai
E4 Vrai

	Vrai	Faux
$0	[a ; b] → E1	
E1	[c ; d] → $1	
$1	→ E2	x → E3
E3	→ E4	z → E1
E2	[r ; s ; u] → E3	
E4	[x ; y ; z] → Stop	

Remarquer que la séquence [u, v] is supprimée comme elle n'est pas accessible.

Programmation procédurale / Transformation de programmes

Technique par automate (*Etape 3 : D-algorithme*)

Utilisation de "IF ELSE" en cascade + une variable State

Cas COND(etat) = vrai : donner simplement les actions correspondantes si elles existent suivie par la modification de l'état courant

Cas COND(etat) = t :

donner un choix entre les deux colonnes de la matrice selon la valeur logique de t.

	Vrai	Faux
$0	[a ; b] → E1	
E1	[c ; d] → $1	
$1	→ E2	x → E3
E3	→ E4	z → E1
E2	[r ; s ; u] → E3	
E4	[x ; y ; z] → Stop	

Exemple : à la ligne E3 de la matrice on associe l'alternative suivante :
If Cond(E3) : State := E4
Else z ; State := E1 Endif

Programmation procédurale / Transformation de programmes

Technique par automate (*Etape 3 : D-algorithme*)

	Vrai	Faux
$0	[a ; b] → E1	
E1	[c ; d] → $1	
$1	→ E2	x → E3
E3	→ E4	z → E1
E2	[r ; s ; u] → E3	
E4	[x ; y ; z] → Stop	

Table Cond :

$0	Vrai
E1	Vrai
$1	t1
E3	t2
E2	Vrai
E4	Vrai

```
State := $0
WHILE ( State # Stop) :
  IF State = $0 :
      a; b; State := E1
  ELSE
      IF State = E1 :
          c; d; State := $1
      ELSE
          IF State = $1 :
              IF t1 :
                  State := E2
              ELSE x; State := E3 ENDIF
```

Programmation procédurale / Transformation de programmes

Technique par automate

	Vrai	Faux
$0	[a ; b] → E1	
E1	[c ; d] → $1	
$1	→ E2	x → E3
E3	→ E4	z → E1
E2	[r ; s ; u] → E3	
E4	[x ; y ; z] → Stop	

Table Cond :

$0	True
E1	True
$1	t1
E3	t2
E2	True
E4	True

```
ELSE
   IF State = E2 :
      r; s, u; State := E3
   ELSE
      IF State = E3 :
         IF t2 :
            State := E4
         ELSE z; State := E1 ENDIF
      ELSE
         IF State = E4 :
            x; y; z; State := Stop
         ENDIF
       ENDIF
   ENDIF
  ENDIF
ENDIF
ENDIF
ENDWHILE
```

Programmation procédurale / Transformation de programmes

Arsac (1977) B --> REn

La technique revient à poser le programme sous la forme d'un système d'équations

Les inconnues = les étiquettes

La résolution du système donne un programme RE-n (exit multi-niveaux)

Méthode :

a) Etiquetter la première instruction.
b) Introduire les "sinon"
c) Mise en équation

d) Résoudre le système
d1)Substitution (élimination d'une variable) , revient à substituer les variables non récursives.
d2) Récursifier / Dérécursifier

Programmation procédurale / Transformation de programmes

Arsac (1977) B --> REn

Soit le B-algorithme suivant :

```
            a
4:          IF t GOTO5
2:          b
            IF u GOTO 3
            IF v GOTO 9
            c
            GOTO 2
3:          d
            GOTO 4
5:          e
            GOTO 4
9           w
```

REn_algorithme correspondant

```
a
Faire
  Si non t
     Faire
            b
            Si non u :
                 Si non v: c;Exit(0)
                 Sinon Exit(2) Fsi
            Sinon d; Exit(1) Fsi
      Fait
  Sinon e; Exit(0) Fsi
Fait
```

Programmation procédurale / Transformation de programmes

Ramshaw(1988) B--> D?

Préservation de la structure du programme.

C'est à dire éliminer tous les "GOTO" et les étiquettes vers lesquelles ils se branchent en insérant des instructions de sortie "Exit" et des boucles "Loop-Endloop" tout en gardant le reste du programme inchangé.

Donc plus de lisibilité.

Programmation procédurale / Transformation de programmes

Ramshaw(1988) B--> D?

Soit l'exemple suivant :

```
action1;
action2;
if test3 then Goto G endif;
action4;
action5;
G:action6;
```

```
action1;
action2;
Loop : L
    if test3 then Exit L endif;
    action4;
    action5;
    Exit L ;
Endloop L ;
action6;
```

Programmation procédurale / Transformation de programmes

Williams and Chen(1985) B-> D

Proposée par Williams et Chen dans un article intitulé "Restructuring Pascal programs containing Goto statements"

Principe basé sur la constitution d'une bibliothèque de schémas équivalents.

Un schéma pour une forme de branchement.

On procède par identification de forme de branchement et remplacement de celle-ci le schéma équivalent (sans branchement)

Programmation procédurale / Transformation de programmes

R (Récursif) vers B

Procédé sémantique : technique basée sur la sémantique de la récursion.

Rappel principe

1. Définir la zone de données = {Variables locales, paramètres appelés par valeur, adresse de retour}

2. Définir les points d'appel et de retour (On suppose qu'il existe un premier appel dans le programme principal)

3. Chaque Appel se traduit par :
   - Empiler la zone de données
   - Préparer la nouvelle
   - Se brancher au début de la procédure

4. Chaque retour se traduit par :
   - Récupérer l'adresse de retour, Ret.
   - Dépiler la zone de données
   - Se brancher à Ret.

5. Empiler une Zdd bidon au départ!

Programmation procédurale / Transformation de programmes

R(Récursif) -> B

Exemple

```
Fonction récursive
Fact(N)
X et Y variables locales
SI N = 0
     Fact := 1
  SINON
      X := N-1;
      Y := Fact(X);
     Fact := N * Y
FSI
```

```
   B-algorithme
   Lire(n) ;     Créerpile(P)
   { Empiler une zone de données bidon}
   Empiler(P,Zddc)
   { Initialiser Zddc pour le premier appel}
   Zddc.N := n;  Zddc.A := 1
   { Début de la fonction simulée }
10 : SI Zddc.N <> 0 Aller à 11
       Fact := 1
       { Simulation du retour de la fonction}
       I := Zddc.A;  Dépiler(P,Zddc)
       SI I=1 ALLERA 1
       ALLERA 2
11 : Zddc.X = Zddc.N - 1
       { Simulation de l'appel récursif }
       Empiler(P,Zddc);  Zddc.N := Zddc.X ;     Zddc.A:= 2
       ALLERA 10
2:     Zddc.Y := Fact ;   Fact := Zddc.N * Zddc.Y
       { Simulation du Retour de la fonction }
       I:= Zddc.A;  Dépiler(P,Zddc)
       SI I=1 ALLERA 1
       ALLERA 2
1:   Ecrire (Fact)
```