Module A206

(version V2)

Programmation fichiers – COBOL

Chapitre 3

COBOL et la cinématique de fichiers

Alain Vailly

Alain.Vailly@univ-nantes.fr

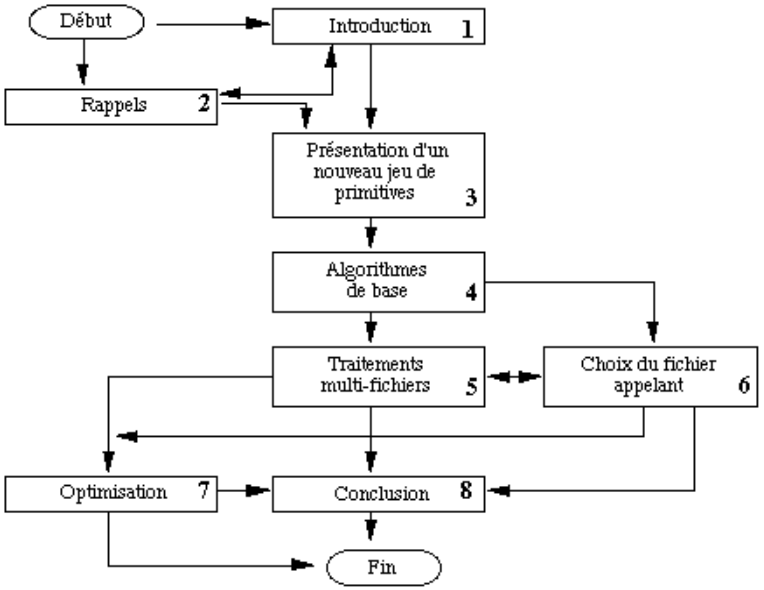
COBOL et la cinématique de fichiers

1. Introduction

De nombreux problèmes de gestion nécessitent un rapprochement entre plusieurs fichiers séquentiels. Résoudre ces problèmes met en jeu des techniques dites de cinématique de fichiers.

Ce chapitre présente tout d'abord un nouveau jeu de primitives de manipulation de ces fichiers séquentiels, plus standard que celles classiquement utilisées. Nous utilisons ensuite ces primitives dans un certain nombre d'algorithmes type. Nous montrons qu'il n'y a que quatre traitements et que tout problème se résoud à l'aide de l'un de ces algorithmes de base. Nous consacrons une section au problème crucial du choix du fichier appelant et terminons par des considérations d'optimisation.

La structuration de cette partie peut donc être schématisée comme suit :



la dernière section pouvant, dans une première lecture, être ignorée.

exercices d'application sont présentés dans le chapitre 6.

COBOL et la cinématique de fichiers

2. Rappels

La plupart des notions que nous utilisons dans ce chapitre sont empruntées au monde des bases de données relationnelles. Nous en rappelons rapidement la définition ici, renvoyant le lecteur au module B302, Méthodes systèmiques d'analyse et de conception, ou B210, Bases de données relationnelles, pour un approfondissement.

domaine

ensemble de valeurs prises par un attribut

relation

sous-ensemble du produit cartésien de domaines. Soient D1, D2... Dn, n domaines pas forcément distincts. Une relation R sera définie sur le produit cartésien D1 x D2 x ... x Dn.

n-uplet

élément d'une relation

attribut

association d'un domaine D et de sa place dans une relation. Afin d'éviter une notation trop lourde, on dote souvent ce couple d'un nom.

dépendance fonctionnelle

Soit R (X, Y, Z) une relation dans laquelle X, Y et Z sont des ensembles d'attributs. On dit que Y dépend fonctionnellement de X si et seulement si la formule suivante est vérifiée :

$$\forall x \in X; y, y' \in Y; z, z' \in Z \mid (x, y, z) \in R \wedge (x, y', z') \in R \Rightarrow y = y'$$

Cette dépendance fonctionnelle est notée $X \rightarrow Y$

dépendance fonctionnelle élementaire

dépendance fonctionnelle dans laquelle la partie gauche (X) ne peut être réduite. Plus formellement :

$$\forall X' \subset X, \neg (X' \rightarrow Y)$$

clé d'une relation

Soient R une relation et K un ensemble d'attributs de R. K est une clé de R si et seulement si tout attribut Y n'appartenant pas à K est en dépendance fonctionnelle élémentaire de K.

identifiant

Soient R une relation et K un ensemble d'attributs. K est un identifiant s'il existe un sous-ensemble K' de K tel que K' soit une clé de R. On peut donc dire qu'un identifiant est un sur-ensemble de la clé.

coupe d'une relation

Soient une relation R (X, Y, Z) et X, Y et Z trois ensembles d'attributs. Soit *val* une valeur de X. On appelle coupe de R selon *val* la relation R' définie comme suit :

$$\forall x \in X; y \in Y; z \in Z \mid (x, y, z) \in R' \Rightarrow ((x, y, z) \in R \wedge x = val)$$

La coupe est constituée de l'ensemble des n-uplets de la relation qui ont *val* pour valeur de X.

projection d'une relation

Soient R (X, Y, Z) une relation et trois ensembles d'attributs X, Y et Z. On appelle projection de R sur X, Y par rapport à Z la relation R' (X, Y), définie par :

$$\forall x \in X ; y \in Y \mid (x, y) \in R' \Rightarrow (\exists z \in Z \mid (x, y, z) \in R)$$

NB : projection et coupe sont deux opérations orthogonales sur les relations. La première procède à une sorte de sélection horizontale, tandis que la seconde correspond à une sélection verticale :

Relation MAISON

Catégorie	Puissance	Type	Couleur
Petite	5	309 PEUGEOT	Blanc
Petite	5	RENAULT 25	Noir
Moyenne	6	RENAULT 25	Blanc
Grande	9	309 PEUGEOT	Noir
Familiale	12	806 PEUGEOT	Bleu

PROJECTION

COUPE

Projection de MAISON sur Type

Type
309 PEUGEOT
RENAULT 25
RENAULT 25
309 PEUGEOT
806 PEUGEOT

Coupe de MAISON sur petite

Petite	5	309 PEUGEOT	Blanc
Petite	5	RENAULT 25	Noir

Il est tout à fait possible de représenter des fichiers séquentiels ou séquentiels indexés par des relations. Inversement, une relation est modélisable par un fichier. Nous pouvons donc introduire quelques définitions complémentaires :

fichier permanent

Un fichier permanent est l'association d'une relation et d'un ensemble d'adresses. La relation donne la structure des enregistrements, l'ensemble des adresses permet de repérer les enregistrements.

Fichier MAISON

Relation MAISON

Petite	5	309 PEUGEOT	Blanc	1
Petite	5	RENAULT 25	Noir	2
Moyenne	6	RENAULT 25	Blanc	3
Grande	9	309 PEUGEOT	Noir	4
Familiale	12	806 PEUGEOT	Bleu	5

Dans un fichier permanent, il n'y a pas de doubles. Traditionnellement, les adresses ne sont pas figurées lorsque l'on donne le contenu d'un fichier. Elles sont implicites.

Petite	5	309 PEUGEOT	Blanc
Petite	5	RENAULT 25	Noir
Moyenne	6	RENAULT 25	Blanc
Grande	9	309 PEUGEOT	Noir
Familiale	12	806 PEUGEOT	Bleu

fichier mouvement

Un fichier mouvement est l'association d'un multi-ensemble construit sur une relation et d'un ensemble d'adresses. Le multi-ensemble autorise l'existence de doublons. On peut, ainsi, avoir deux mouvements de versement d'une somme d'argent sur un compte bancaire. Dans le fichier donnant le solde du compte, il n'y aura pas deux enregistrements identiques. Il s'agit donc d'un fichier permanent. Dans le fichier mouvements, plus il y aura d'enregistrements identiques correspond à des versements, plus le titulaire du compte sera content.

enregistrement

Un enregistrement d'un fichier R est un couple (e, i), où e est un n-uplet de la relation R et i la place occupée par celui-ci dans le fichier. Par défaut, il sera noté Ri.

tri de degré 1

Soient R (X, Y, Z) une relation, X et Z deux ensembles d'attributs, Y un attribut et p le cardinal de R. R est trié en ordre croissant par rapport à Y si et seulement si la formule suivante est vérifiée :

$$\forall i \in [1 .. \text{card}(R)], R_i . c \geq R_{i+1} . c$$

Y est souvent appelé critère du tri.

séquence

coupe d'une relation associée à une valeur (celle-ci étant appelée référence de la séquence).

Cette notion est parfaitement généralisable. Ainsi, le fichier suivant, trié sur l'attribut Puissance :

Petite	5	309 PEUGEOT	Blanc
Petite	5	RENAULT 25	Bleu
Moyenne	6	RENAULT 25	Noir
Grande	9	309 PEUGEOT	Blanc
Familiale	12	309 PEUGEOT	Blanc

contient les séquences suivantes :

- séquence associée à la référence (Petite), de degré 1 :

Petite	5	309 PEUGEOT	Blanc
Petite	5	RENAULT 25	Bleu

- séquence associée à la référence (Blanc, 309 Peugeot), de degré 2 :

Petite	5	309 PEUGEOT	Blanc
Grande	9	309 PEUGEOT	Blanc
Familiale	12	309 PEUGEOT	Blanc

tri de degré 2

Soient une relation R (X, Y, Z) et trois attributs X, Y, Z. R est trié par rapport à (X, Y) si et seulement si R est trié par rapport à X et si toute séquence de R par rapport à X est triée par rapport à Y. (X, Y) est souvent appelé critère du tri.

Cette définition est parfaitement généralisable à n attributs.

29/02/04

COBOL et la cinématique de fichiers

---

3. Un nouveau jeu de primitives

Nous montrons, dans ce paragraphe, les limites des primitives de manipulation de fichiers séquentiels classiquement utilisées, puis proposons de les remplacer par un nouveau jeu. Après avoir montré comment exprimer les nouvelles par les anciennes, nous en donnons une implémentation COBOL. Nous les appliquons ensuite à deux cas, celui d'un fichier unique avec erreurs et celui d'un fichier constitué de deux fichiers sans erreur. Dans les deux cas, il s'agit de procéder à l'impression des contenus. Dans les deux cas, le fait de se servir des nouvelles primitives réduit au strict nécessaire les modifications. Une conclusion termine le paragraphe.

3.1 Les primitives anciennes

Lorsque l'on veut lire un fichier séquentiel, on utilise habituellement un jeu de primitives qui ressemble à celui-ci :

- Ouvrir (f, lecture)
- Lire (f)
- ff (f)
- Fermer (f)

Elles ont l'effet suivant :

- Ouvrir (f, lecture) : associe le fichier au programme, l'ouvre (ie. rend disponible ses enregistrements et positionne le curseur sur le premier enregistrement).
- Lire (f) : rend l'enregistrement sur lequel pointe le curseur, puis déplace ce curseur d'un cran. Cette instruction n'est valide que si le curseur ne pointe pas sur la marque de fin de fichier.
- ff (f) : cette fonction renvoie la valeur vrai si le curseur est positionné sur la marque de fin de fichier.
- Fermer (f) : remet le fichier à disposition des autres utilisateurs.

Voici un algorithme typique :

```
Ouvrir(f, lecture)
Tantque non ff (f)faire
    Lire (f)
    TRAITER (f.tamp)
fintantque
Fermer (f)
```

Ces primitives se révèlent particulièrement bien adaptées pour parcourir un fichier non trié ou bien encore un fichier trié sur une clé de degré 1. Elles se révèlent peu efficaces dans tous les autres cas et notamment :

- quand on doit traiter les séquences qui sont associées à des références qui ne sont pas une clé ;
- quand le fichier à parcourir est composé de plusieurs fichiers ;
- ...

Dans la mesure où nous voulons disposer de procédures standard, il nous faut changer ce jeu de primitives et en adopter un autre.

3.2 Les nouvelles primitives

Avant d'introduire ces primitives, nous devons faire une hypothèse :

Nous supposons que, pour tout attribut de la relation, il existe un majorant et que cette valeur ne peut être utilisée pour constituer un n-uplet. Nous appelerons *sentinelle* le n-uplet composé des majorants de chaque attribut.

Le principe qui régit le nouveau jeu de primitives est le suivant :

- il y a toujours une lecture par anticipation, ce qui permet de détecter la fin d'une séquence avant de commencer la suivante.
- la lecture de la marque de fin de fichier correspond à un chargement de la sentinelle dans le tampon du fichier.

Moyennant tout ce qui précède, les nouvelles primitives peuvent être définies ainsi :

- Ouverture (f) : ouvre le fichier en mode lecture et appelle la procédure Lecture.
- Lecture (f) : rend l'enregistrement sur lequel pointe le curseur, puis déplace celui-ci d'un cran. Si le curseur pointe sur la marque de fin de fichier, place la sentinelle dans le tampon.
- maxind (f) : rend la valeur vrai si le tampon contient la sentinelle.
- Fermeture (f) : ferme le fichier.

Le parcours d'un fichier séquentiel devient alors :

```
Ouverture(f)
Tantque non maxind (f) faire
    TRAITER (f.tamp)
    Lecture (f)
fintantque
Fermeture (f)
```

La différence entre cet algorithme et le précédent peut sembler minime (lire puis traiter dans un cas, traiter puis lecture dans l'autre). Elle est néanmoins capitale, comme nous allons le montrer dans les paragraphes suivants et nous ne saurions trop recommander aux apprentis programmeurs et aux programmeurs de jeter l'ancien jeu au profit du nouveau.

3.3 Les nouvelles par les anciennes

Il est aisé de définir chaque primitive nouvelle en fonction des anciennes :

maxind (f) :

```
Début
  délivre  f.tamp = sentinelle
Fin
```


Lecture (f) :

```
Début
   Si ff(f) alors
      f.tamp := sentinelle
   sinon
      Lire (f)
   finsi
Fin
```

Ouverture (f) :

```
Début
   Ouvrir (f,lecture)
   Lecture(f)
Fin
```

Fermeture(f) :

```
Début
   Fermer (f)
Fin
```

3.4 Implémentation en COBOL

Il est également aisé de proposer des implémentations dans les langages de programmation. À titre d'exemple, voici comment on peut les instancier en COBOL.

La traduction en COBOL de ces primitives nécessite la mise en place de quelques élements. Nous faisons, en particulier, les hypothèses suivantes :

- le nom COBOL du fichier est F, celui de son tampon FTAMP.
- le majorant de tout attribut existe ; il s'agit de la constante figurative `HIGH-VALUE`.

Moyennant ceci, les primitives se traduisent ainsi :

Lecture (f)

```
1       7 A   B
          OBTENIR-F.
              READ F
                  AT END MOVE HICH-VALUE TO FTAMP
              END-READ.
```

Ouverture (f)

```
1       7 A   B
```

```
OUVERTURE-F.
    OPEN INPUT F.
    PERFORM OBTENIR-F.
```

Fermeture (f)

```
1       7 A   B
FERMETURE-F.
    CLOSE F.
```

La fonction maxind (f) peut aisément être remplacée par le test `FTAMP = HIGH-VALUE`. L'algorithme précédent peut maintenant être traduit :

```
1       7 A   B
    OPEN INPUT F.
    PERFORM OBTENIR-F.
    PERFORM UNTIL FTAMP = HIGH-VALUE
        PERFORM TRAITER
        PERFORM OBTENIR-F
    END-PERFORM.
    CLOSE F.
```

La version présentée ici est une simplification du code que nous aurions dû obtenir. Nous avons remplacé, en effet, certaines instructions d'appels de procédure par leur contenu. Ainsi, par exemple, les deux premières lignes sont-elles l'implémentation (la substitution) de `PERFORM OUVERTURE`.

3.5 Application au cas d'un fichier avec erreurs

Prenons le cas d'un fichier qui contient des erreurs (au sens de la logique du programme). Le principe que nous tentons de mettre en place est celui de l'indépendance de la production des enregistrements de la partie consommations. Nous pouvons appliquer ce principe en gardant le même algorithme de parcours (ie. que le fichier soit "pur" ou plein d'erreurs, le traitement des articles corrects reste le même) et en adaptant la production (ie. la lecture).

Imprimer un fichier -unique- sans erreur peut s'écrire comme suit :

```
Ouverture(f)
Ouvrir (fsort, écriture)
Tantque non maxind (f) faire
    Ecrire (f.tamp)
    Lecture (f)
fintantque
Fermer (fsort)
Fermeture (f)
```

La procédure Lecture (f) fournit les enregistrements de f, les uns après les autres. Son "code" est décrit dans le paragraphe précédent.

Faire le même travail avec un seul fichier erroné ne nécessite que le changement de la procédure de lecture (le reste est inchangé) :

```
WRESU := faux
Obtenir (f)
Tantque non maxind (f) et non WRESU faire
    CONTROLER (f.tamp, WRESU)
    Si non WRESU alors
        Obtenir (f)
    finsi
fintantque
```

La procédure Obtenir (f) correspond à ce que nous avions associé à la lecture dans le paragraphe précédent. Son "code" est fourni, sous l'appellation Lecture (f), dans un paragraphe précédent. La procédure CONTROLER (non détaillée ici) correspond aux différents tests à faire subir aux enregistrements pour définir s'il s'agit d'enregistrements corrects ou d'enregistrements erronés. Elle rend la valeur vrai si le contenu du tampon est exempt d'erreurs.

3.6 Application au cas de deux fichiers en entrée, sans erreur

Prenons le cas d'un fichier constitué, en fait, de deux fichiers A et B ayant la même structure. Écrivons l'algorithme qui permet d'imprimer, alternativement, un enregistrement de A suivi d'un enregistrement de B, ce jusqu'à la fin de l'un (auquel cas on imprimera, les uns à la suite des autres, le contenu de l'autre).

L'algorithme principal (celui qui se charge d'imprimer un enregistrement) est déjà écrit, dans le paragraphe précédent. Comme dans le cas précédent, la procédure de lecture va être modifiée :

```
Selon WDER
qd O:
    Obtenir (A)
    Obtenir (B)
    Si non maxind(A) alors
        WDER := A
        f.tamp := A.tamp
    sinon
        WDER := B
        f.tamp := B.tamp
    finsi
qd A:
    Obtenir (A)
    Si non maxind(B) alors
        WDER := B
        f.tamp := B.tamp
    sinon
        WDER := A
        f.tamp := A.tamp
    finsi
qd B:
    Obtenir (B)
    Si non maxind(A) alors
        WDER := A
        f.tamp := A.tamp
    sinon
        WDER := B
        f.tamp := B.tamp
    finsi
finselon
```

La procédure d'ouverture doit, elle aussi, être modifiée :

```
Ouvrir (A, lecture)
Ouvrir (B, lecture)
WDER := O
Lecture(f)
```

3.7 Conclusion

Nous avons montré que les anciennes primitives pouvaient assez facilement être remplacées par d'autres, plus pratiques. Ce remplacement permet de bien séparer production d'enregistrements et consommation de ceux-ci. Les deux exemples que nous avons donné sont, sur ce point, assez parlants. L'avantage de ce nouveau jeu est tel qu'il faut impérativement, selon nous, abandonner l'ancien au profit du nouveau.

Ce nouveau jeu est certes un peu déroutant pour des programmeurs confirmés qui ont, pendant des années, pensé autrement. Le changement de contexte est d'autant plus difficile que l'expérience s'est accumulée. Il faut toutefois le faire.

Pour avoir expérimenté cette "mutation" par nous-même, nous pouvons en garantir les bénéfices et, plus surprenant peut-être, le côté irréversible. Nous ne pouvons en effet plus rédiger d'algorithme qui ne respecterait pas les principes présentés ici. Il s'agit donc bien d'une mutation.

08/03/04

COBOL et la cinématique de fichiers

4. Algorithmes de base

Nous présentons dans ce paragraphe quatre algorithmes de base qui pourront être utilisés par la suite pour exprimer des traitements plus complexes. Après avoir décrit l'environnement de travail dans lequel nous nous plaçons, nous abordons successivement les algorithmes permettant :

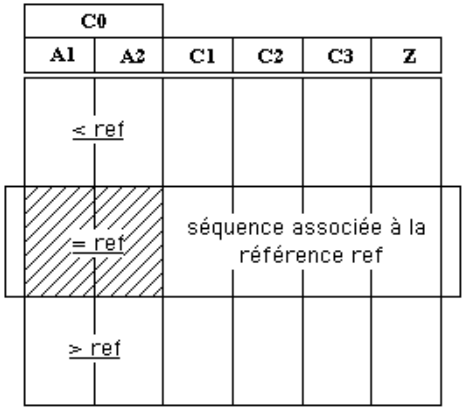
- le traitement d'une séquence référencée,
- l'élimination d'une séquence référencée,
- le traitement de toute séquence située avant une référence,
- l'élimination de toute séquence située avant une référence.

4.1 Cadre méthodologique et algorithmique

Soient les hypothèses suivantes :

- On a un fichier F (A1... , AM, C1... , CN, Z), dans lequel les Ai et Ci sont des attributs et Z un ensemble éventuellement vide d'attributs.
- On suppose que M est supérieur strictement à zéro et que N est supérieur ou égal à zéro.
- Le fichier F est trié en ordre croissant sur (A1... , AM, C1... , CN).
- On dispose d'une référence sur A1 x A2 x ... x AM ; on appelle *ref* cette référence.
- On appelle C0 l'ensemble constitué des attributs A1... , AM.
- F a été ouvert (par Ouverture (F)) et a fait l'objet d'un nombre quelconque (éventuellement nul) d'appels à la primitive Lecture (F).

Ce fichier peut être schématisé ainsi :



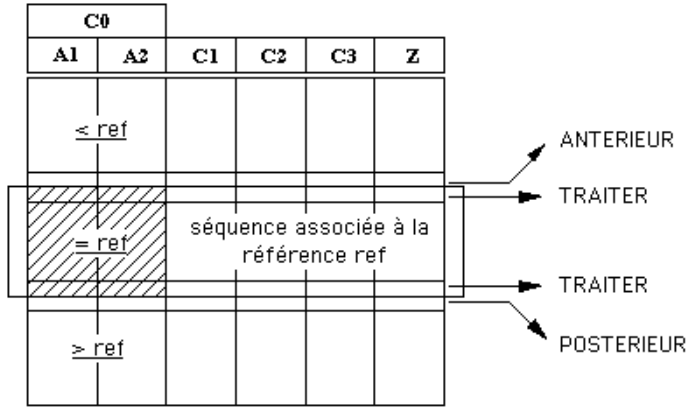
Les algorithmes que nous présentons sont au nombre de quatre, chacun correspondant à un cas précis. La sélection de l'algorithme à appliquer dépend de la réponse donnée aux deux questions suivantes :

- Tient-on compte de la structuration en séquence de la partie à traiter ?
- La référence est-elle comprise dans la séquence courante ?

		Q1	
		OUI	NON
Q2	OUI	rupture	alignement
	NON	ruptavant	alignavant

4.2 Traitement d'une séquence référencée

Il s'agit de prendre en compte la séquence associée à la référence et chacune des sous-séquences la composant, tout en encadrant le traitement de la séquence par deux traitements spécifiques, correspondant l'un au début (procédure antérieur), l'autre à la fin (procédure postérieur).



F est par hypothèse aligné sur la référence *ref*, ce qui veut dire que le prochain enregistrement disponible respecte la condition C0 >= ref.

Les traitements que nous présentons sont des algorithmes de traitement de ruptures (ie. de changements de valeurs). Il y en a quatre :

- rupture0 : le critère de tri (C0) est un identifiant et N = 0
- rupture1 : le critère de tri (C0) est un identifiant et N = 1 OU le critère de tri n'est pas un identifiant et N = 0
- rupture2 : le critère de tri (C0) est un identifiant et N = 2 OU le critère de tri n'est pas un identifiant et N = 1
- rupture3: le critère de tri (C0) est un identifiant et N = 3 OU le critère de tri n'est pas un identifiant et N = 2

Nous ne développons pas davantage, bien que l'algorithme soit parfaitement généralisable. Nous en donnons la forme générale. Nous tenons à la disposition des lecteurs la "solution" permettant de traiter la situation dans laquelle le critère n'est pas un identifiant et N = K ou bien le critère de tri est un identifiant et N = K + 1, K ayant une valeur quelconque supérieure à zéro.

4.2.1 Traitement d'une rupture de degré 0

Ce cas correspond à la situation dans laquelle C0 est un identifiant et les Ci (i > 0) n'existent pas. La séquence à traiter ne comprend donc qu'un seul enregistrement. L'algorithme est réduit à ceci :

```
Procédure ruptureO (f)
   Début
      TRAITER (f.tamp)
      Lecture(f)
   Fin
Fin
```

4.2.2 Traitement d'une rupture de degré 1

L'algorithme correspondant est le suivant :

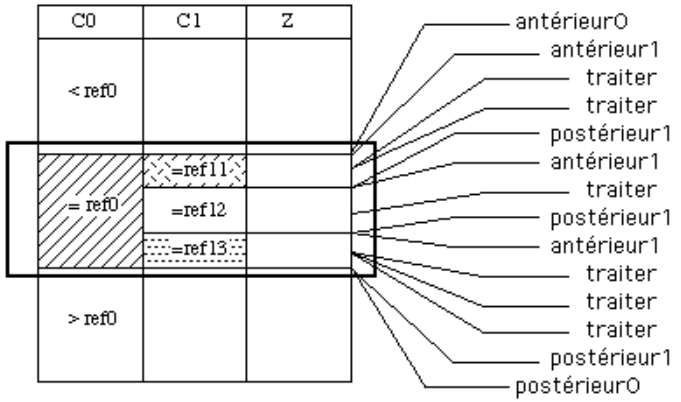
```
Procédure rupture1 (f)
   Début
      ANTERIEURO (ref)
      Tantque f.tamp.CO = ref faire
            TRAITER (f.tamp)
            Lecture (f)
      fintantque
      POSTERIEURO (ref)
   Fin
Fin
```

4.2.3 Traitement d'une rupture de degré 2

L'algorithme proposé reprend les principes mis en évidence dans le paragraphe précédent. Il ressemble à ceci :


```
Procédure rupture2 (f)
   Début
      ANTERIEURO (ref)
      Tantque f.tamp.CO = ref faire
            fen1 := f.tamp.C1
            ANTERIEUR1 (ref, fen1)
            Tantque f.tamp.CO = ref et f.tamp.C1 = fen1 faire
                  TRAITER (f.tamp)
                  Lecture (f)
            fintantque
            POSTERIEUR1 (ref, fen1)
      fintantque
      POSTERIEURO (ref)
   Fin
Fin
```

Le fichier sur lequel on travaille est dans la situation suivante :



On peut remarquer qu'il y a, dans ce cas précis, deux boucles Tantque.

4.2.4 Traitement d'une rupture de degré 3

Juste à titre d'exemple, voici à quoi ressemble le traitement dans le cas d'une rupture de degré 3 :

```
Procédure rupture3 (f)
   Début
        ANTERIEURO (ref)
        Tantque f.tamp.CO = ref faire
              fen1 := f.tamp.C1
              ANTERIEUR1 (ref, fen1)
              Tantque f.tamp.CO = ref et f.tamp.C1 = fen1 faire
                   fen2 := f.tamp.C2
                   ANTERIEUR2 (ref, fen1, fen2)
                   Tantque f.tamp.CO = ref et f.tamp.C1 = fen1 et f.tamp.C2 = fen2 faire
                        TRAITER (f.tamp)
                        Lecture (f)
                   fintantque
                   POSTERIEUR2 (ref, fen1, fen2)
              fintantque
              POSTERIEUR1 (ref, fen1)
        fintantque
        POSTERIEURO (ref)
   Fin
Fin
```

4.2.5 Quid des ruptures de degré supérieur ?

Dans le cas général, il y aura K + 1 boucles imbriquées, le traitement d'un enregistrement étant à l'intérieur de la plus interne. Une seule lecture, également à l'intérieur. K + 1 boucles. Chaque boucle est précédée d'une procédure antérieur et suivie d'une procédure postérieur. Plus la boucle est interne, plus il y a de paramètres dans antérieur et dans postérieur, plus il y a de conditions à tester pour sortir de la boucle.

4.3 Traitement de toute séquence située AVANT une référence

Le traitement d'une séquence se trouvant AVANT une séquence donnée se démarque du traitement vu précédemment par la condition d'arrêt des boucles. La plus externe teste le cas d'infériorité :

pour un traitement AVANT	pour un traitement PENDANT
Tantque f.tamp.CO < ref faire fintantque	Tantque f.tamp.CO = ref faire fintantque

La comparaison avec son homologue vu précédemment est significative.

4.4 Élimination d'une séquence référencée

Ce traitement, encore appelé alignement, revient à parcourir le fichier en "éliminant" les enregistrements composant la séquence. Il y a donc, à la fin, positionnement sur le premier enregistrement situé APRÈS la séquence. En ce sens, il y a eu élimination. Il n'y a que deux cas à envisager :

- alignement0 : le critère de tri (C0) est un identifiant et N = 0
- alignementN : les autres cas (TOUS les autres cas).

4.4.1 Élimination d'une séquence de degré 0

L'algorithme est simple :

```
TRAITER (f.tamp)
Lecture(f)
```

4.4.2 Cas général

L'algorithme est le suivant :

```
Tantque f.tamp.CO = ref faire
      TRAITER (f.tamp)
      Lecture (f)
fintantque
```

4.5 Élimination de toute séquence située AVANT une référence

Comme pour le traitement, l'élimination de toute séquence située AVANT une référence donnée va différer de celle d'une séquence précise par la condition dans la boucle :

pour un traitement AVANT	pour un traitement PENDANT
Tantque f.tamp.CO < ref faire fintantque	Tantque f.tamp.CO = ref faire fintantque

L'algorithme est le suivant (il n'y a qu'un seul cas) :

```
Tantque f.tamp.CO < ref faire
      TRAITER (f.tamp)
      Lecture (f)
fintantque
```

4.6 Conclusion

TOUS les problèmes dans lesquels il y a manipulation de fichiers séquentiels peuvent se résoudre par l'un des algorithmes-types suivants :

- le traitement d'une séquence référencée,
- l'élimination d'une séquence référencée,

- le traitement de toute séquence située avant une référence,
- l'élimination de toute séquence située avant une référence.

Leurs principes sont simples, ce qui en rend la maîtrise aisée. Il est donc particulièrement important de les comprendre, pour pouvoir être complètement opérationnel. Il n'y a "plus" qu'à analyser le problème posé, à déterminer le type de l'algorithme à appliquer et à adapater celui-ci au cas particulier.

La programmation devient un travail facile !

10/03/09

COBOL et la cinématique de fichiers

---

5. Traitements multi-fichiers

5.1 Introduction

Il est souvent nécessaire de mettre en présence plusieurs fichiers séquentiels contenant des informations que l'on souhaiter manipuler. Les traitements qui parcourent les fichiers en cherchant à aligner les enregistrements sur des attributs en commun sont des algorithmes de concordance ou de cinématique de fichiers.

Dans cette section, nous mettons en évidence les outils permettant de mettre en oeuvre de tels traitements. Après avoir précisé les hypothèses, nous étudions le cas le plus simple de deux fichiers, puis esquissons le cas général dans lequel un nombre quelconque de fichiers peuvent être rapprochés.

5.2 Hypothèses de travail

Soient les hypothèses suivantes :

- On a un fichier F (C1... CN, Z), dans lequel les Ci sont des attributs définis sur des domaines Di et Z un ensemble éventuellement vide d'attributs.
- On a un fichier G (C'1... C'M, Z'), dans lequel les C'i sont des attributs définis sur des domaines D'i et Z' un ensemble éventuellement vide d'attributs.

Nous proposons les définitions suivantes :

fichiers compatibles

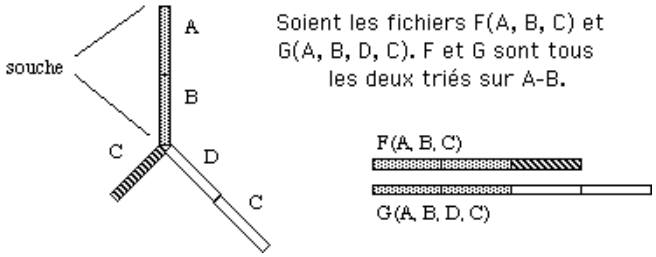
F et G sont compatibles si et seulement si D1 = D'1.

souche de fichiers compatibles

La souche de deux fichiers compatibles est constituée de l'ensemble des domaines de définition des attributs des fichiers compatibles F et G. La souche est un ensemble D1... Di tel que :

- Dk = D'k pour tout k appartenant à l'intervalle [1, i]
- Dk+1 ≠ D'k+1

k est le degré de la souche.



Cet arbre est appelé l'**arbre des fichiers**.

fichier appelant, fichier appelé

Le fichier appelant sera le fichier dont on utilise tout ou partie du critère de tri comme référence pour traiter (ou éliminer) toutes les séquences de l'autre fichier. Cet autre fichier sera le fichier appelé.

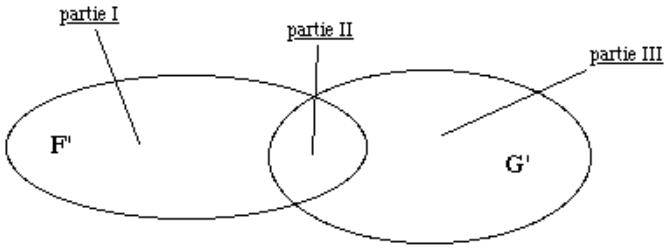
fichier primaire

Lorsqu'il y a plus de deux fichiers compatibles, celui des fichiers qui n'est jamais appelé est le fichier primaire (on utilise aussi, de temps en temps, le terme "principal").

5.3 Cas de deux fichiers

5.3.1 Principe

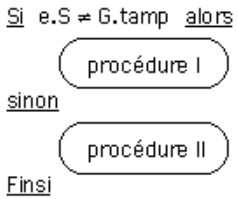
- Soit le fichier F (S1... Sp, C1... Cn, Z) le fichier appelant. F est trié sur (S1... Cn).
- Soit le fichier G (S1... Sp, C'1... C'm, Z') le fichier appelé. G est trié sur (S1... C'm). On sait que C1 est différent de C'1. (S1... Sp) est la souche des fichiers F et G.
- Soit F' (S1... Sp) la projection de F sur la souche et G' (S1... Sp) celle de G sur cette même souche. Les positions respectives de F' et G' peuvent se représenter comme suit :



Le problème revient à déterminer dans quelle portion de l'espace on est situé. Pour cela, et pour chaque élément e de F (e est composé de p valeurs correspondant chacune à un Si donné), il faut :

- traiter les éléments de la partie III qui sont inférieurs à e (l'algorithme appliqué est de type ruptavant -s'il s'agit simplement de les éliminer- alignavant -s'il faut les traiter-) ;

- distinguer dans quelle portion, I ou II, on se trouve, en appliquant une procédure adaptée à chacun des cas. La partie II correspondra soit à un alignement, soit à une rupture, ce qui permettra d'aligner G sur la première valeur ne concernant pas e.



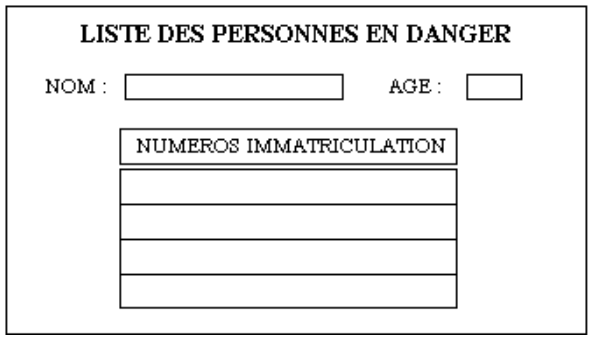
Selon le nombre d'attributs de type Ci et la nature (ie. identifiant ou pas) du critère de tri, les différentes procédures se situeront soit dans un ANTERIEUR quelconque, soit dans TRAITER.

5.3.2 Exemple

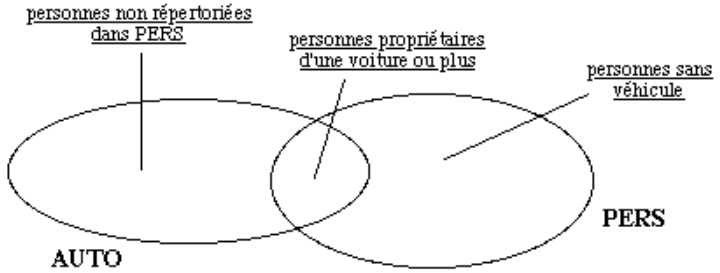
Afin d'illustrer ce principe, prenons un exemple. On dispose d'un fichier PERS de personnes, décrites par leur numéro d'INSEE, leur nom et leur âge. Il est trié selon le numéro d'INSEE, qui constitue une clé. On dispose, par ailleurs, d'un fichier AUTO de véhicules, chacun décrit par le numéro d'INSEE du propriétaire et le numéro d'immatriculation de la voiture. Il est trié sur le couple (numéro d'INSEE, numéro d'immatriculation), qui constitue une clé.

On veut obtenir :

- la liste ERR des personnes (numéro d'INSEE) qui possèdent au moins un véhicule, mais qui ne sont pas répertoriées dans le fichier PERS (ce sont des personnes en situation irrégulière) ;
- la liste DANGER des personnes de plus de 70 ans qui possèdent au moins un véhicule et de leur(s) véhicule(s).



Les deux fichiers sont compatibles, la souche correspond au numéro d'INSEE :

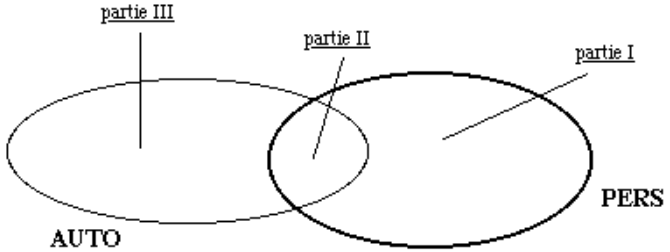


Dans l'ensemble correspondant à l'intersection, on sélectionne les personnes ayant plus de 70 ans. Il y a deux solutions :

- PERS est le fichier appelant,
- AUTO est le fichier appelant.

Pour l'instant, nous n'avons aucun argument pour choisir entre les deux solutions. Ceux-ci sont présentés dans la section suivante.

a) PERS est le fichier appelant



Le critère de tri est un identifiant de degré 1. L'algorithme de parcours contient donc une seule boucle :

```
Début
    Ouverture (PERS)
    ANTERIEURO
    Tantque non maxind (PERS) faire
          TRAITER (ENR-PERS)
          Lecture (PERS)
    fintantque
    POSTERIEURO
    Fermeture (PERS)
Fin
```

La procédure ANTERIEUR0 ne fait que l'ouverture des trois fichiers AUTO, ERR et DANGER.


```
Début
    Ouverture (AUTO)
    Ouvrir (ERR, écriture)
    Ouvrir (DANGER, écriture)
Fin
```

La procédure POSTERIEUR0 (elle est exécutée alors que toutes les personnes enregistrées ont été traitées) prend en compte des erreurs de AUTO, c'est-à-dire les personnes citées dans AUTO et non enregistrées dans PERS.

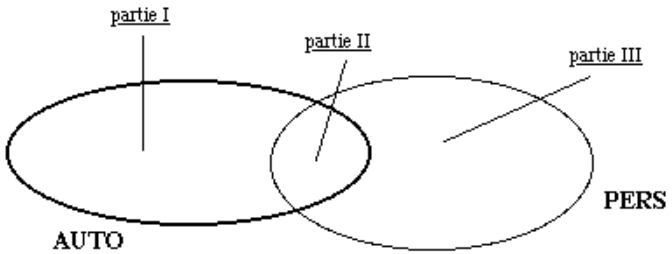
```
Début
    Tantque ENR-AUTO.N° INSEE < sentinelle faire
        fenO := ENR-AUTO.N° INSEE
        ENR-ERR := fenO
        Ecrire (ENR-ERR)
        Tantque ENR-AUTO.N° INSEE = fenO faire
            Lecture(AUTO)
        fintantque
    fintantque
    Fermeture (AUTO)
    Fermer (DANGER)
    Fermer (ERR)
Fin
```

NB : il s'agit d'un ruptavant de degré 2, sur la sentinelle, marque de fin de fichier.

Traiter une personne ayant au moins un véhicule est l'oeuvre de la procédure TRAITER (e).

```
Début
   Tantque ENR-AUTO.N° INSEE < e.N° INSEE faire
         fen0 := ENR-AUTO.N° INSEE
         ENR-ERR := fen0
         Ecrire (ENR-ERR)
         Tantque ENR-AUTO.N° INSEE = fen0 faire
               Lecture (AUTO)
         fintantque
   fintantque
   Si ENR-AUTO.N° INSEE ≠ e.N° INSEE alors
         -- personne sans voiture
   sinon
         -- personnes ayant au moins un véhicule
         Si e.AGE > 70 alors
               ENR-DANGER := e
               Ecrire (ENR-DANGER)
         Finsi
         Tantque ENR-AUTO.N° INSEE = e.N° INSEE faire
               Si e.Age > 70 alors
                     ENR-DANGER := e
                     Ecrire (ENR-DANGER)
               Finsi
               Lecture (AUTO)
         fintantque
   Finsi
Fin
```

b) AUTO est le fichier appelant



Le critère de tri est un identifiant, il est de degré 2. Il y a donc deux boucles.


```
Début
    Ouverture (AUTO)
    ANTERIEURO
    Tantque non maxind (AUTO) faire
         fen1 := ENR-AUTO.N° INSEE
         ANTERIEUR1 (fen1)
         Tantque ENR-AUTO.N° INSEE = fen1 faire
               TRAITER (ENR-AUTO)
               Lecture (AUTO)
         fintantque
         POSTERIEUR1 (fen1)
    fintantque
    POSTERIEURO
    Fermeture (AUTO)
Fin
```

La procédure ANTERIEUR0 ne fait qu'ouvrir les différents fichiers :

```
Début
    Ouverture (PERS)
    Ouvrir (ERR, écriture)
    Ouvrir (DANGER, écriture)
Fin
```

la procédure POSTERIEUR0 les refermant.

```
Début
    Fermeture (PERS)
    Fermer (ERR)
    Fermer (DANGER)
Fin
```

La procédure ANTERIEUR1 (x) correspond à la première apparition d'une personne x.

```
Début
   Tantque ENR-PERS.N° INSEE < x faire
         -- personne sans véhicule
         Lecture (PERS)
   fintantque
   Si ENR-PERS.N° INSEE ≠ x alors
         -- personne non répertoriée
         ENR-ERR := x
         Ecrire (ENR-ERR)
   sinon
         -- personne en voiture
         Si ENR-PERS.AGE >= 70 alors
               ENR-DANGER := ENR-PERS
               Ecrire (ENR-DANGER)
         Finsi
         Lecture (PERS)
   Finsi
Fin
```

POSTERIEUR1 (x) ne joue aucun rôle. Cette procédure est vide. TRAITER (e), quant à elle, s'occupe des autres voitures de x.

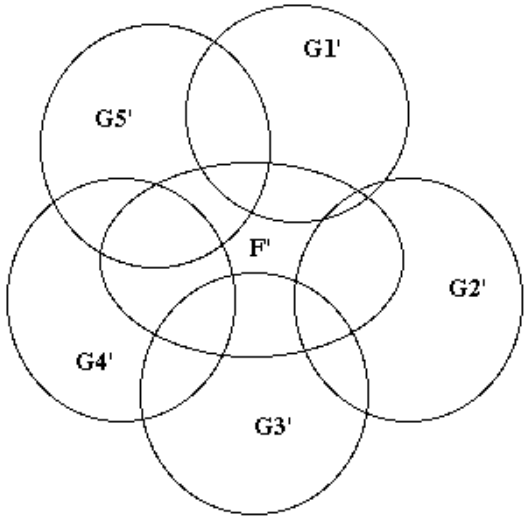
```
Début
   Si ENR-PERS.Age >= 70 alors
      ENR-DANGER := e
      Ecrire (ENR-DANGER)
   Finsi
Fin
```

5.4 Cas général

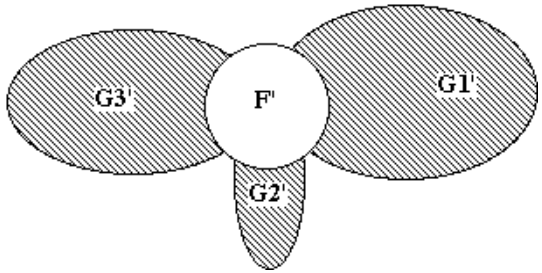
5.4.1 Hypothèses

- Soit F le fichier appelant. Il est composé des attributs S1... Sp, C1... Cn et Z. Il est trié sur (S1... Cn).
- Soient G1, G2... Gx, x fichiers appelés de la forme Gi (S1... SP, Ci1... Cim, Zi), tous triés sur (S1... Sp) qui constitue la souche.

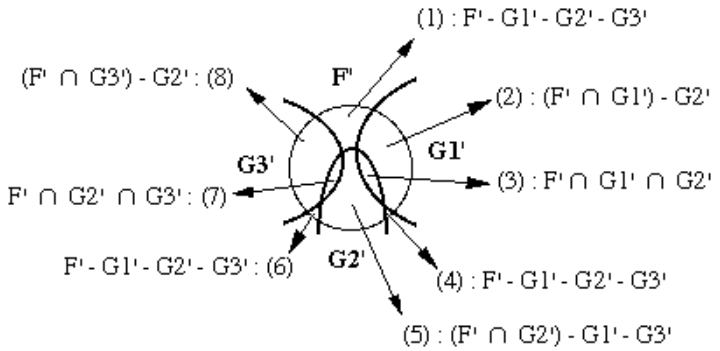
Les projections sur cette souche donnent le diagramme de Venn suivant :



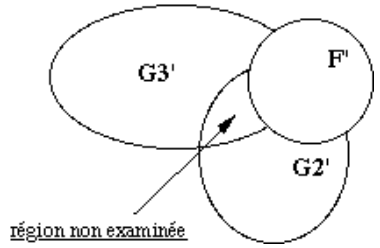
Le cas général se résume à une distinction entre les sous-ensembles Gi' - F' (ie. les pétales de la fleur) :



et entre les 2**x sous-ensembles partitionnant F' :



Par hypothèse, nous ne nous préoccupons pas des sous-ensembles situés à l'extérieur de F' et correspondant à une intersection entre deux ou plus Gi' :



5.4.2 Principe

Le traitement consiste, pour chaque élément e de F', à :

- appliquer, pour chaque pétale de la corolle, alignavant (Gi, e) ou ruptavant (Gi, e) ;
- distinguer dans lequel des 2**x sous-ensembles du calice on est situé et appliquer le traitement spécifique. Celui-ci est de type alignement ou rupture, selon les cas.


```
Si e.S ≠ G1.tamp et e.S ≠ G2.tamp et ... et e.S ≠ Gn.tamp alors
     procédure 1
sinonsi e.S = G1.tamp et e.S ≠ G2.tamp et ... et e.S ≠ Gn.tamp alors
     procédure 2
sinonsi

      - - - - - - - - - -

sinon
   -- (e.S = G1.tamp et e.S = G2.tamp et ... et e.S = Gn.tamp)
     procédure 2**n
Finsi
```

5.4.3 Exemple

Une manufacture vend, sur catalogue, des produits finis. Certains de ces produits sont fabriqués sur place et stockés au magasin. D'autres sont achetés chez un grossiste. Dans ce dernier cas, on peut malgré tout disposer d'un fond de roulement en stock.

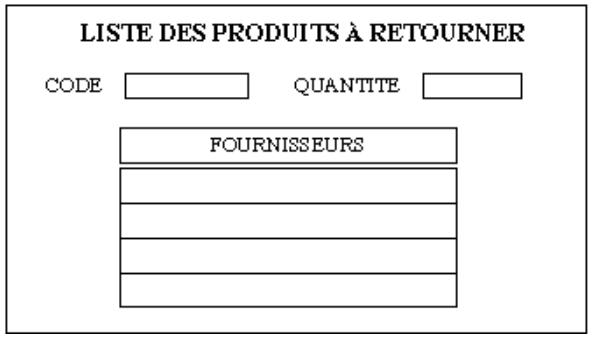
On dispose des fichiers suivants :

- un fichier FOUR des fournisseurs, contenant pour chacun le numéro du produit et le code du grossiste. Il est trié sur le couple (numéro produit, code grossiste), qui est une clé.
- un fichier STOCK qui décrit chaque produit par son numéro (c'est la clé) et sa quantité en stock. Il est trié sur le numéro de produit.

À l'occasion d'une restructuration, certains produits vont disparaître du catalogue. Ces produits sont référencés dans un fichier MOUVE (il ne contient que des numéros de produits).

On souhaite obtenir cinq états :

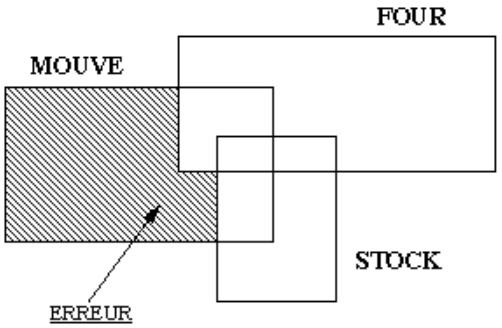
- RETOUR : état des produits à rendre aux grossistes, avec les informations suivantes :



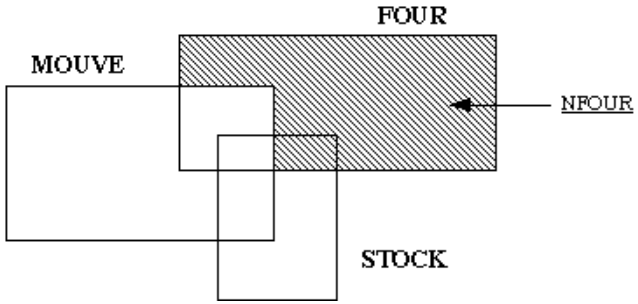
- ERREUR : fichier des erreurs de MOUVE (code produit n'existant ni dans FOUR, ni dans STOCK) ;
- NFOUR : fichier des fournisseurs dans lequel les produits retirés n'apparaissent plus ;
- SOLDES : fichier des stocks à écouler rapidement, correspondant aux produits retirés mais encore en stock ;
- NSTOCK : fichier des stocks mis à jour, les produits retirés n'apparaissant plus.

Le problème peut être schématisé ainsi :

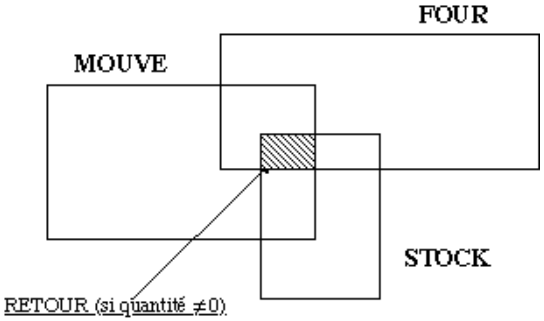
- ERREUR :



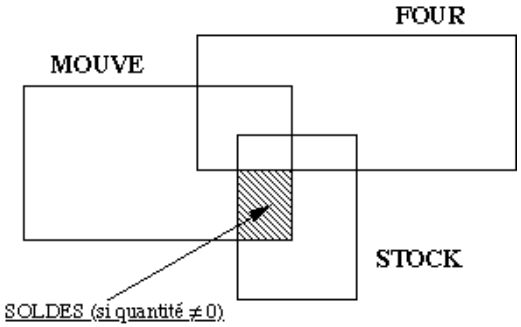
- NFOUR :



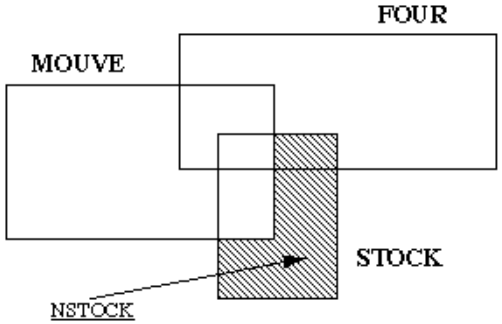
- RETOUR :



- SOLDES :



- NSTOCK :



la projection ayant eu lieu sur le numéro de produit (c'est la souche des trois fichiers).

L'analyse du problème conduit à choisir MOUVE comme fichier appelant. Le traitement peut alors se modéliser avec une seule boucle :

```
Début
    Ouverture (MOUVE)
    ANTERIEURO
    Tantque non maxind (MOUVE) faire
        TRAITER (ENR-MOUVE)
        Lecture (MOUVE)
    fintantque
    POSTERIEURO
    Fermeture (MOUVE)
Fin
```

La procédure ANTERIEUR0 ouvre les fichiers :

```
Début
    Ouverture (FOUR)
    Ouverture (STOCK)
    Ouvrir (ERREUR, écriture)
    Ouvrir (NFOUR, écriture)
    Ouvrir (NSTOCK, écriture)
    Ouvrir (SOLDES, écriture)
Fin
```

POSTERIEUR0 produisant une partie des fichiers NFOUR et NSTOCK, puis fermant les fichiers :


```
Début
    -- élimination du fichier des fournisseurs
    -- régénération dans NFOUR
    Tantque ENR-FOUR.N° produit < sentinelle faire
        ENR-NFOUR := ENR-FOUR
        Ecrire (ENR-NFOUR)
        Lecture (FOUR)
    fintantque
    -- élimination du fichier des stocks
    -- régénération dans NSTOCK
    Tantque ENR-STOCK. N° produit < sentinelle faire
        ENR-NSTOCK := ENR-STOCK
        Ecrire (ENR-NSTOCK)
        Lecture (STOCK)
    fintantque
    Fermeture (FOUR)
    Fermeture (STOCK)
    Fermer (NFOUR)
    Fermer (NSTOCK)
    Fermer (SOLDES)
    Fermer (ERREUR)
Fin
```

Le traitement d'un mouvement e quelconque est l'oeuvre de TRAITER (e) :

```
Début
    Tantque ENR-FOUR.N° produit < e.N° produit faire
        ENR-NFOUR := ENR-FOUR
        Ecrire (ENR-NFOUR)
        Lecture (FOUR)
    fintantque
    Tantque ENR-STOCK.N° produit < e.N° produit faire
        ENR-NSTOCK := ENR-STOCK
        Ecrire (ENR-NSTOCK)
        Lecture (STOCK)
    fintantque
    Si ENR-FOUR.N° produit ≠ e.N° produit et ENR-STOCK.N° produit ≠ e.N° produit alors
        -- cas (1) : erreur ou doublon
        ENR-ERREUR := e
        Ecrire (ENR-ERREUR)
    sinonsi ENR-FOUR.N° produit = e.N° produit et ENR-STOCK.N° produit ≠ e.N° produit alors
        -- cas non répertorié : élimination de
        la séquence référencée de fournisseur
        Tantque ENR-FOUR.N° produit = e.N° produit faire
              Lecture (FOUR)
        fintantque
    sinonsi ENR-FOUR.N° produit ≠ e.N° produit et ENR-STOCK.N° produit = e.N° produit alors
        -- cas (4)
        Si ENR-STOCK.quantité > 0 alors
            ENR-SOLDES := ENR-STOCK
            Ecrire (ENR-SOLDES)
        Finsi
        Lecture (STOCK)
    sinon
        -- cas (3)
        difO := ENR-STOCK.quantité > 0
        Si difO alors
              ENR-SOLDES.N° produit := ENR-STOCK.N° produit
              ENR-SOLDES.quantité := ENR-STOCK.quantité
              Ecrire (ENR-SOLDES)
        Finsi
        Lecture (STOCK)
        Tantque ENR-FOUR.N° produit = e.N° produit faire
              Si difO alors
                  ENR-SOLDES := ENR-FOUR.code grossiste
                  Ecrire (ENR-SOLDES)
              Finsi
        fintantque
    Finsi
Fin
```

5.5 Conclusion

Il est possible d'appliquer les principes énoncés et de traiter n'importe quel problème dans lequel il s'agit de confronter N fichiers séquentiels. Ceci se fait en projetant les fichiers sur leur souche, puis en analysant les régions de la "fleur" correspondant au problème à traiter. Cette annalyse effectuée, il faut choisir l'algorithme-type approprié et l'adapter au cas particulier.

Ceci suppose que le fichier appelant ait été choisi. Nous devons donc être capable de le déterminer. Ceci fait l'objet de la section suivante.

11/05/04

COBOL et la cinématique de fichiers

6. Organisation des traitements

6.1 Introduction

Nous avons, dans les sections précédentes, étudié la structure des algorithmes à mettre en oeuvre lorsque l'on est en présence de plusieurs fichiers séquentiels. Nous avons surtout mis l'accent sur les fichiers d'entrée. Or le bon sens indique que la structure des algorithmes dépend à la fois de ces fichiers ET des résultats attendus.

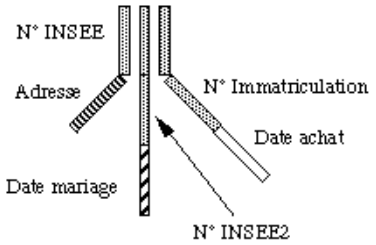
Nous développons ici cette idée pour mettre en évidence les critères de choix fichier appelant - fichiers appelés. Dans ce qui suit, nous supposons que l'hypothèse suivante est vérifiée.

Hypothèse

Tous les fichiers séquentiels en entrée d'un algorithme sont compatibles.

Le non respect de cette hypothèse conduirait à un coût prohibitif de cet algorithme (voir, pour s'en convaincre, le sujet d'examen proposé dans le chapitre 6).

Une conséquence du respect de cette hypothèse est que l'on peut représenter les situations respectives des différents fichiers sous forme d'un arbre (il est appelé **arbre des fichiers**) :



Cet exemple décrit trois fichiers, PERS, MARIAGE et AUTOS, chacun comprenant les attributs suivants :

- PERS : N° INSEE (clé), Adresse
- MARIAGE : N° INSEE, N° INSEE2 (les deux constituent la clé), Date mariage
- AUTOS : N° INSEE, N° Immatriculation (les deux forment la clé), Date achat

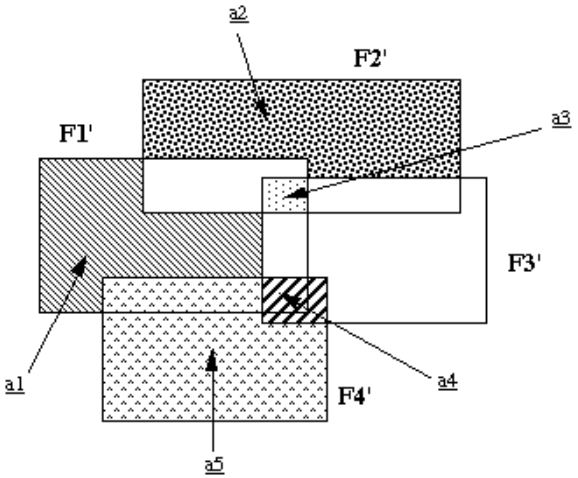
Cette section est structurée de la façon suivante : les critères de choix entre appelés et appelant sont d'abord présentés. L'usage de ces critères à plusieurs niveaux de l'arbre est ensuite abordé.

6.2 Critères de choix

6.2.1 Définitions

Soient F1... Fn les fichiers d'entrée nécessaires et F1'... Fn' leurs projections sur la souche. Les résultats attendus sont caractérisés par A = {a1... ak}, où chaque ai est un sous-ensemble de l'union généralisée des Fi. Chaque résultat n'est pas obligatoirement un de ces sous-ensembles, mais il peut toujours s'en déduire (par une union, une coupe...).

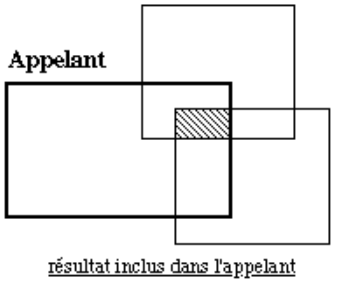
Prenons un exemple : soit le diagramme de Venn suivant, décrivant les positions respectives de quatre fichiers et des cinq résultats attendus :



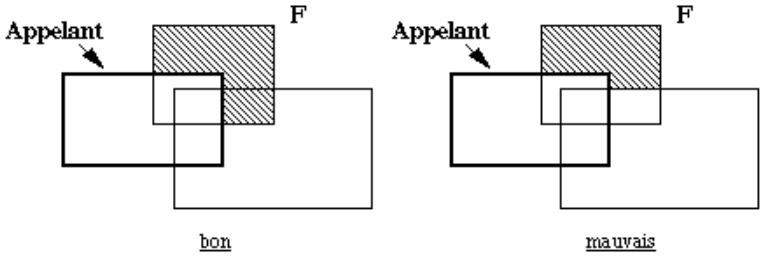
6.2.2 Principe

F peut être retenu comme fichier appelant si et seulement si il existe une partition de l'ensemble des résultats A en deux sous-ensembles, I et J, tels que :

- tous les éléments de I sont inclus dans la projection de F :



- tous les éléments de J sont inclus dans la projection d'un fichier E, différent de F, et leur complément à E est inclus dans la projection de F :

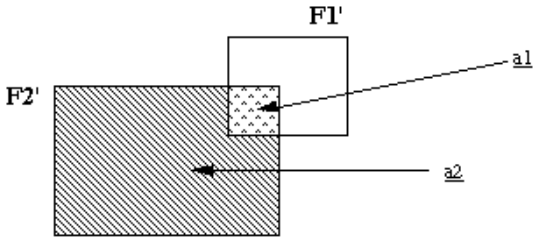


Il y a, en général, plusieurs solutions à ce problème, celles-ci pouvant être décrites par un arbre, appelé, lui, **arbre des appels** :

6.2.3 Exemples

Prenons plusieurs exemples pour illustrer ce qui précède.

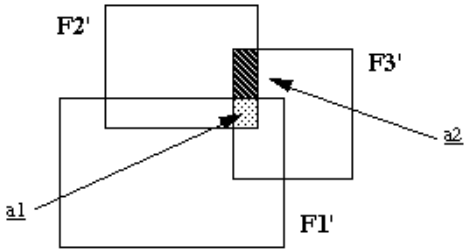
- Soit le diagramme suivant :



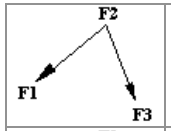
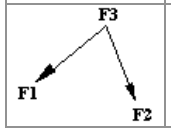
Il y a deux solutions :

F1 → F2	I = {a1} et J ={a2}
F2 → F1	I = {a1, a2} et J = {}

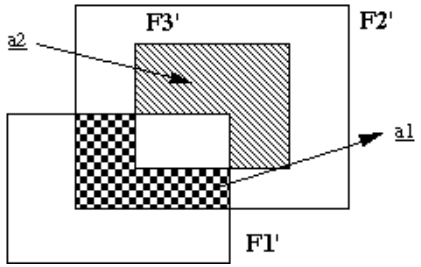
- Soit le diagramme suivant :



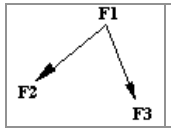
Il y a aussi deux solutions :

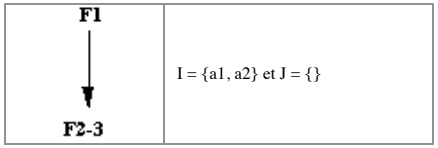
	I = {a1, a2} et J ={}
	I = {a1, a2} et J = {}

- Soit le diagramme suivant :

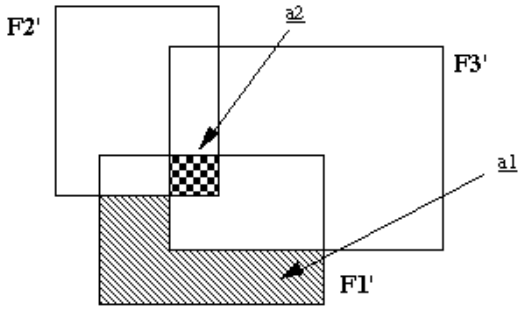


Il y a encore deux solutions :

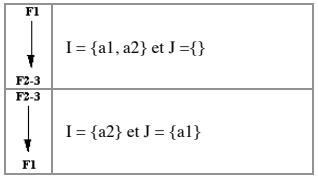
	I = {a1} et J ={a2}



- Soit le diagramme suivant :



Il n'y a aucune solution à ce problème... sauf à regrouper les deux fichiers F2 et F3 en un seul, appelé par exemple F2-3. Il y a alors deux solutions.



6.3 Usage de ces critères

6.3.1 Principe

Confrontés à une application développée sur la base de fichiers séquentiels, deux comportements sont envisageables :

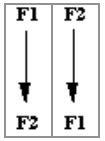
- décomposer le problème en sous-problèmes, chacun pouvant se résoudre en faisant appel aux techniques vues précédemment ;
- tenter de résoudre le problème globalement.

Dans ce second cas, on peut retrouver la situation évoquée dans le paragraphe précédent à chaque niveau de l'arbre des fichiers. Le problème que l'on a à résoudre consiste alors à combiner de manière cohérente les différents arbres des appels de chaque niveau, pour en faire un arbre des appels unique qui préfigure la structure du programme.

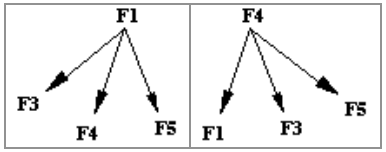
La méthode de résolution se décline en six points :

- À partir des résultats souhaités, répertorier les fichiers d'entrée nécessaires ;
- S'assurer que les traitements ne soient pas séparables (autrement dit que les fichiers nécessaires, pour deux résultats, ne soient pas disjoints) ;
- Construire l'arbre des fichiers ;
- Attacher, à chaque noeud de l'arbre des fichiers, un diagramme de Venn. Celui-ci doit mettre en évidence les sous-ensembles intéressants à ce niveau de projection.
- Pour chaque diagramme de Venn, produire l'ensemble des solutions possibles pour l'arbre des appels ;
- Vérifier qu'il existe un fichier qui soit la racine dans chacun des niveaux répertoriés et fusionner les arbres concernés. Dans le cas contraire, décomposer le problème en sous-problèmes.

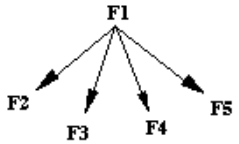
Ce dernier point peut s'illustrer ainsi : soient les fichiers F1, F2, F3, F4 et F5. Au premier niveau, on a obtenu deux arbres possibles, S1 et S2 :



Au second niveau, on dispose également de deux possibilités, S3 et S4 :



F1 se retrouve en tant que racine aux deux niveaux. On va donc fusionner les deux arbres pour obtenir finalement l'arbre des appels suivant :



6.3.2 Exemple

Considérons une application où interviennent quatre fichiers séquentiels, ENFANTS, FAMILLES, VACCINS et MALADIES. Ces fichiers comprennent les attributs suivants :

- ENFANTS :

n° famille de l'enfant

prénom de l'enfant

année de naissance de l'enfant

Numéro de famille et prénom de l'enfant constituent la clé de ce fichier.

- FAMILLES :

n° de la famille (c'est la clé)

nom de la famille

adresse de la famille

- MALADIES :

n° de famille

prénom de l'enfant malade

code maladie

date de début de la maladie

Numéro de famille, prénom de l'enfant et code maladie forment la clé du fichier.

- VACCINS :

n° de famille

prénom de l'enfant

code vaccination

date de dernier rappel

Numéro de famille, prénom de l'enfant et code vaccination constituent la clé du fichier.

ENFANTS et FAMILLES sont triés sur la clé, MALADIES et VACCINS sur le couple (numéro de famille, prénom de l'enfant). Tous ces fichiers sont supposés sans erreur.

On désire obtenir les résultats suivants :

- SANS : familles (numéro de famille, nom et adresse) sans enfants ;
- BCGvsCOQ : nom de famille et prénom des enfants vaccinés pour le BCG (code vaccination = "BCG") et ayant eu la coqueluche (code maladie = "1").

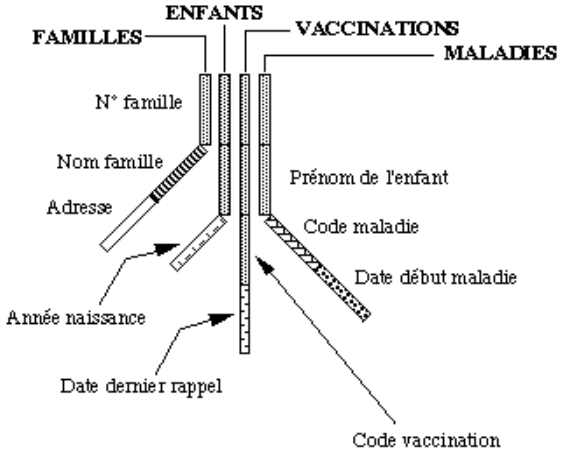
Déroulons la méthode préconisée sur l'exemple.

Les fichiers nécessaires sont :

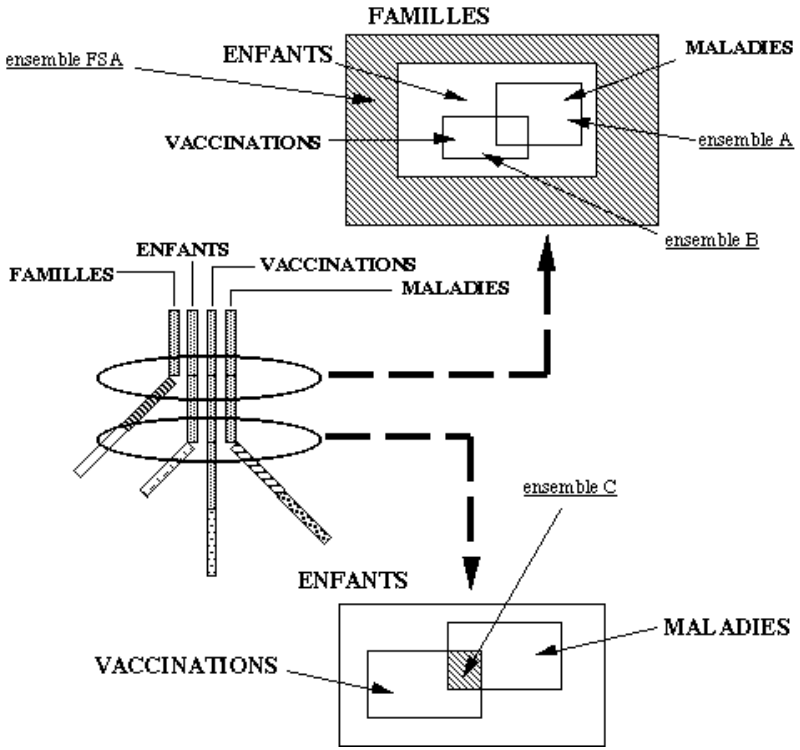
- pour SANS : fichiers ENFANTS et FAMILLES
- pour BCGvsCOQ : fichiers ENFANTS, FAMILLES, MALADIES et VACCINS

Dans la mesure où les deux résultats sont toujours souhaités simultanément, il peut être avantageux de les obtenir lors d'un seul traitement.

L'arbre des fichiers est le suivant :



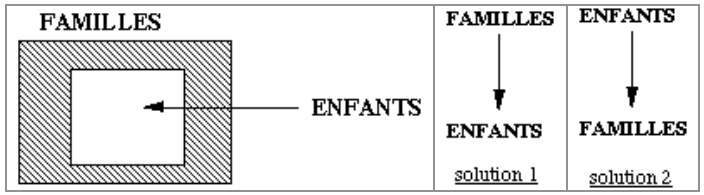
On associe ensuite à chaque niveau de l'arbre des fichiers un diagramme de Venn :



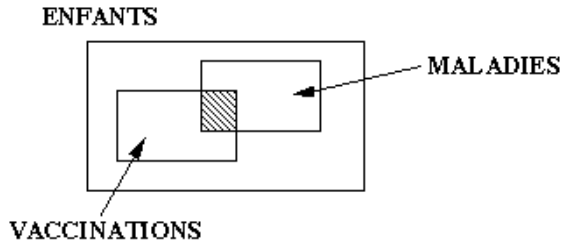
Le résultat SANS correspond au sous-ensemble FSA. Le résultat BCGvsCOQ est obtenu à partir du sous-ensemble C. Quant aux deux autres, ils correspondent pour A aux familles ayant au moins un enfant ayant été malade, pour B aux familles ayant au moins un enfant ayant été vacciné.

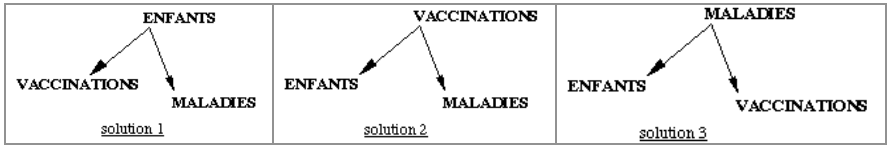
On associe à chaque diagramme de Venn les arbres des appels suivants :

- au premier niveau :

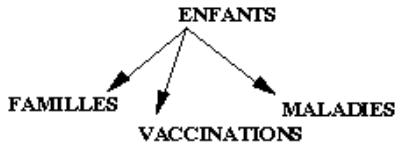


- au second niveau :





La solution 2 du premier niveau et la solution 1 du second fournissent l'arbre des appels final :



13/05/04

COBOL et la cinématique de fichiers

7. Optimisations

7.1 Introduction

Le problème analysé, la compatibilité des fichiers séquentiels vérifiée, le fichier appelant choisi, l'algorithme rédigé, on peut légitimement considérer que le travail est accompli et que l'on peut passer au codage en COBOL. Un certain nombre d'exercices, du reste, propose de traduire tel ou tel algorithme.

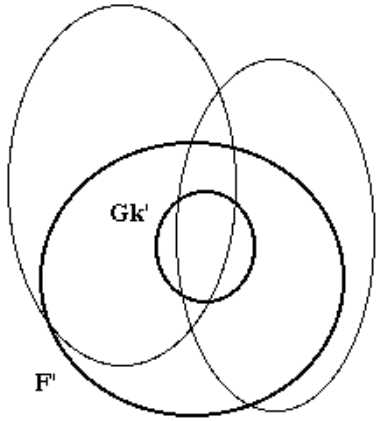
Ceci fait, la tâche sera terminée, pense-t-on. Et, de fait, elle peut être considérée comme achevée. Pour ceux qui veulent parachever leur "oeuvre", en soignant la qualité des algorithmes produits, il y a toutefois une dernière action à entreprendre, celle d'optimisation.

Il est en effet possible de retoucher les algorithmes. C'est ce que nous abordons dans cette section. Nous présentons deux types de retouche, l'une de nature sémantique qui se base sur une analyse des diagrammes de Venn, l'autre qui utilise l'algèbre de Boole en général et les tableaux de Karnaugh en particulier pour factoriser certaines branches des algorithmes.

7.2 Simplifications sémantiques

7.2.1 Introduction

Le principe que nous avons mis en évidence dans la section 5 (qui présente les différents traitements multifichiers) repose sur un diagramme de Venn correspondant à la projection de tous les fichiers sur leur souche :



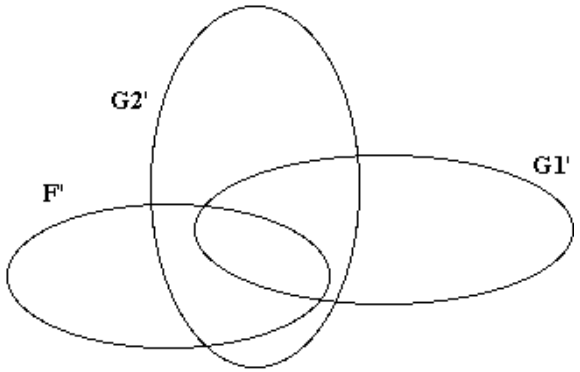
et sur un traitement spécifique des différentes "zones" géographiques de ce qui apparaît être une fleur, avec corolle (2**x cas) et calice (x cas).

Dans certaines cas (cela dépend du problème à résoudre), certaines de ces zones sont vides. Si deux fichiers n'ont pas d'intersection, par exemple, il est inutile d'envisager ce cas dans l'algorithme. L'alternative généralisée présentée dans la partie Principe de la section 5 peut alors être simplifiée. C'est le cas du fichier Gk qui est, selon le diagramme précédent, totalement inclus dans F. Il est donc inutile de chercher à aligner Gk sur F.

7.2.2 Exemples

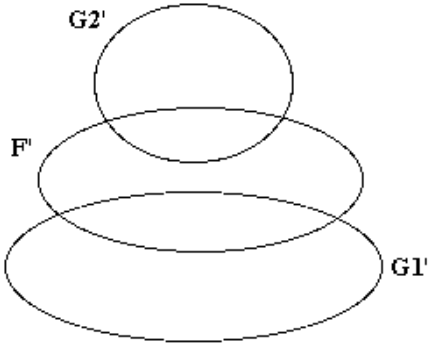
Prenons plusieurs exemples pour illustrer ces simplifications.

Soit le diagramme suivant :



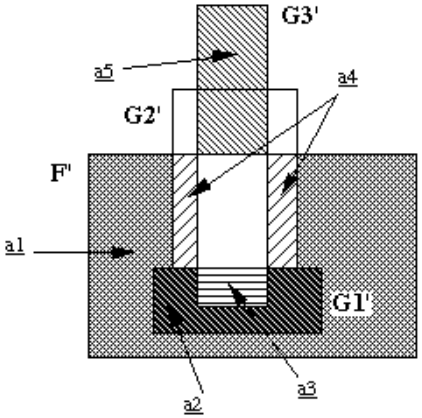
À l'intérieur de F', G1' est totalement inclus dans G2'. Il est donc inutile de traiter l'intersection entre F et (G1' - G2'). On peut alors supprimer de l'algorithme toutes les branches conditionnées par g1.tamp-- = e et g2.tamp-- ≠ e.

Soit le diagramme suivant :



À l'intérieur de F', l'intersection de G1' et de G2' est vide. Il faut éliminer de l'alternative généralisée les branches dépendant de la condition g1.tamp-- = e et g2.tamp-- = e.

Considérons le diagramme suivant, avec les cinq résultats {a1, a2, a3, a4, a5} matérialisés :



Sur les onze cas possibles (huit dans le calice, 3 dans la corolle), plusieurs sont à supprimer. Il s'agit des ensembles comprenant les éléments :

- appartenant à G1' mais pas à F' (G1' est totalement inclus dans F'),
- appartenant à F', à G1' et à G2', mais pas à G3' (G1' et G2' ont une intersection vide),
- appartenant à F', à G1', à G2' et à G3' (pour la même raison),
- appartenant à F', à G3', mais ni à G1', ni à G2' (ce cas n'existe pas, G3' étant, dans le calice, totalement inclus dans G1' et G2').

Il y a donc sept cas, deux en dehors, cinq en dedans. Sur ces cinq, celui appartenant à F', à G2' et à G3', mais pas à G1', sera ignoré, cet ensemble ne correspondant à aucun résultat attendu.

7.3 Simplification par factorisation

Le principe à appliquer consiste à mettre en évidence des séquences algorithmiques équivalentes, à les regrouper et à simplifier les conditions déclenchantes.

Soient les deux séquences suivantes :

- séquence 1 :


```
Si a et non b alors
    procédure A
sinonsi a et b alors
    procédure A
    Tantque b faire
        procédure B
    fintantque
Finsi
```

- séquence 2 :

```
Si a alors
    procédure A
    Tantque b faire
        procédure B
    fintantque
Finsi
```

Elles sont équivalentes. On peut les fusionner. Prenons un exemple. Soit l'alternative généralisée suivante :

```
Si a et f.tamp ≠ 5 alors
    g.tamp := r                          A
    Ecrire (g)                           *
sinonsi a et f.tamp = 5 alors
    g.tamp := r                          A
    Ecrire (g)
    Tantque f.tamp = 5 faire             B
        Lecture (f)
    fintantque
Finsi
```

Cette alternative se schématise sous la forme d'un tableau de Karnaugh :

		f.tamp = 5	
		F	V
a	F	ø	ø
	V	A	A B

Il est tout à fait possible de dupliquer la partie B en * sans altérer le sens de l'alternative.

```
Si a et f.tamp ≠ 5 alors
    g.tamp := r                          A
    Ecrire (g)
    Tantque  f.tamp = 5  faire           B
        Lecture (f)
    fintantque
sinonsi a et f.tamp = 5 alors
    g.tamp := r                          A
    Ecrire (g)
    Tantque  f.tamp = 5  faire           B
        Lecture (f)
    fintantque
Finsi
```

soit, sous forme de Karnaugh :

		f.tamp = 5	
		F	V
a	F	ø	ø
	V	A B	A B

simplifiable

L'alternative peut donc se reformuler ainsi :


```
Si a alors
    g.tamp := r                          A
    Ecrire (g)
    Tantque  f.tamp = 5  faire           B
        Lecture (f)
    fintantque
Finsi
```

Ce principe peut complètement se généraliser. Nous laissons le lecteur le faire.

17/05/04

COBOL et la cinématique de fichiers

8. Conclusion

Dans ce chapitre, nous avons introduit de nouvelles primitives de manipulation de fichiers séquentiels, présenté des algorithmes de base permettant de traiter ou d'éliminer une séquence donnée ou toute séquence située avant une séquence donnée.

Tous les problèmes de gestion peuvent se résoudre à l'aide de ces techniques. L'idée maîtresse qui sous-tend ce chapitre est que l'on doit se focaliser sur la partie analyse du contexte pour caractériser la problématique. Celui-ci étant identifié, il "suffit" de sélectionner le bon algorithme et de l'adapter au cas à traiter.

Le point central de la caractérisation est le choix du fichier appelant. Nous consacrons une section entière à cette sélection. Nous terminons en abordant l'optimisation de nos algorithmes.

"Avec la cinématique de fichiers, programmez facile"

17/05/04