I.2: Le test fonctionnel
I.2.2 : Le test fonctionnel de logiciel

Introduction

- Notre contexte : pas possible d'exprimer toutes les combinaisons de DT.
- Le test fonctionnel est basé sur la spécification/interface du programme : il n'examine pas le programme, mais son comportement.
- Chaque technique de test fonctionnel fournit une méthode avec un critère de sélection/production des DT (à partir des spécifications) :
  1. Analyse partitionnelle
  2. Test aux limites
  3. Graphe Cause-Effet
  4. Test syntaxique
  5. Etc…

Analyse partitionnelle : un exemple

Élaboration d'un logiciel de traitement des notes du BAC

- On distingue les différents profils de lycéens,
- On suppose que si le calcul est bon pour un lycéen A avec un certain profil, il sera exact pour tout lycéen B de même profil
- Exemple de profil : lycéen redoublant sa terminale ES option math, ne repassant pas l'épreuve anticipée de Français, mais repassant l'épreuve anticipée SVT.

Problème : comment construire ces différentes classes ?

- domaine des valeurs d'entrées

Analyse partitionnelle

- Un programme P traite des données en entrée :

  $P : D \rightarrow R$

- On partitionne D en différentes classes D_i telles que :
  1. $D = D_0 \cup D_1 \cup \ldots/\ldots \cup D_N$
  2. $D_i \cap D_j = \varnothing$ si $i \neq j$
  3. On sélectionne un seul cas de test par classe D_i

- Hypothèse de test :

  $\forall i \in [1,N]$, $\forall d,d' \in D_i$: P(d) correct $\Leftrightarrow$ P(d')correct

Analyse partitionnelle : exercices

Soit les spécifications suivantes :

1 - « Si la valeur n est négative : un message d'erreur est affichée. Si n est dans [1,20[on affiche la valeur exacte de Factoriel(n). Si n est dans [20,200] on affiche une approximation de Factoriel(n) en virgule flottante avec une précision de 0,1%. Si n>200 un message d'erreur est affichée. »

2 - « Écrire un programme qui calcule $F(x)=(1/x)^{1/2}$ *»*

*3 - « L'instruction FOR n'accepte qu'**un seul paramètre** en tant que variable auxiliaire. Son nom ne doit pas dépasser **2 caractères non blancs**. Une borne supérieure et une borne inférieure doivent être précisées: **la borne inférieure est précédée du mot-clé '='** et **la borne supérieure est précédée par le mot-clé 'TO'**. Les bornes sont des **entiers positifs**. »*

On se propose de tester des programmes correspondants à ces spécifications : donner un jeu de test associé à chaque spécification.

Analyse partitionnelle : exercices/solutions

1 - « Si la valeur n est négative : un message d'erreur est affichée. Si n est dans [1,20[on affiche la valeur exacte de Factoriel(n). Si n est dans [20,200] on affiche une approximation de Factoriel(n) en virgule flottante avec une précision de 0,1%. Si n>200 un message d'erreur est affichée. »

5 classes : entiers négatifs ; {0} ; [1,20[; [20,200] ; entiers supérieurs à 200.

2 - « Écrire un programme qui calcule $F(x)=(1/x)^{1/2}$ »

3 classes : réels négatifs, {0}, réels positifs

*3 - « L'instruction FOR n'accepte qu'**un seul paramètre** en tant que variable auxiliaire. Son nom ne doit pas dépasser **2 caractères non blancs**. Une borne supérieure et une borne inférieure doivent être précisées: **la borne inférieure est précédée du mot-clé '='** et **la borne supérieure est précédée par le mot-clé 'TO'**. Les bornes sont des **entiers positifs**. »*

FOR A=1 TO 10 ; FOR A=10 TO 10 ; FOR AA=1 TO 10 ;

FOR A=1,B=1 TO 10 ; FOR AAA=1 TO 10 ; FOR A=100 TO 10 ;

FOR =1 TO 10 ; FOR A=1 TO 10 ; FOR A=1.5 TO 10 ;

FOR A=1 TO 10.5 ; FOR A=1 TO ;

Etc.

Test aux limites

Constat : erreurs souvent dues au fait que le programmeur n'avait pas prévu le comportement du logiciel pour des valeurs aux limites (aussi appelées valeurs de bord):

- Valeurs très grandes
- Valeur de boucle nulle, négative, maximale
- Données non valides
- Etc…

Remarque : Souvent utilisée avec la technique de partitionnement : les valeurs aux limites peuvent être des valeurs aux frontières des partitions

Exemple de choix des valeurs de test.

- Variable dans un intervalle de valeurs [a,b] :

a, b, $a \pm \Delta$, $b \pm \Delta$ (Δ : plus petite variation possible)

- Variable dans un ensemble de valeurs ordonnées $\{a_1, a_2, a_3,\ldots/\ldots,a_n\}$:

a_1, a_2, a_{n-1}, a_n (premier, second, avant dernier, dernier)

Test aux limites : des exemples

- Une variable a dans le domaine [**1**,**10**]
  Valeurs à tester : 0,**1**,2,9,**10**,11
- Deux variables a et b dans le domaine [**1**,**10**]^2
  6x6=36 valeurs à tester : {0,**1**,2,9,**10**,11}^{2}
- La notion de limite n'est pas toujours évidente ! Deux variables a et b dans le domaine [-10,10]^2
  - Rajouter les valeurs au voisinage de 0,
  - les valeurs où a=b,
  - etc. (selon son intuition… en fonction de la spécification, conception…)

Test aux limites : Exercice

- Un programme de classification de triangles prend en entrée un triplet de réels (a,b,c) correspondants aux longueurs des 3 côtés d'un triangle. Le programme doit préciser la nature du triangle (équilatéral, isocèle, scalène, impossible)

  Donner des exemples de valeurs aux limites.

Test syntaxique

But : déterminer des DT à partir d'une description formelle (sous forme BNF ou automate à états finis) des données

Type d'application visé : application nécessitant des données d'entrée respectant une syntaxe (analyseur d'expression, interpréteur, compilateur, etc.)

Méthode :

1. Définir une grammaire
2. Construire l'arbre de dérivation 'générique'
3. Produire des DT en se basant sur l'arbre de dérivation

   Couverture des nœuds Terminaux, des nœuds Non Terminaux

   Deux cas :

   1. Entrées valides
   2. Entrées non valides

Test syntaxique : exercice

Interpréteur qui reconnaît des commandes du type :

```
copy <fic1> to <fic2>
rename <fic1> to <fic2>
```

Où `<fic1>` et `<fic2>` sont des noms de fichiers formés sur une ou deux lettres prises dans l'ensemble {a,b}.

1-Donner une grammaire

2-Donner un arbre de dérivation 'générique'

3-Produire des DT

Test syntaxique : exercice (solution)

1-Une grammaire

<commande>::=<nom_commande><nom_fichier>to<nom_fichier>
<nom_commande>::=copy|rename
<nom_fichier>::= <lettre>|<lettre><lettre>
<lettre>::=a|b

2-Un arbre de dérivation ‘générique’

A faire

3-Des DT

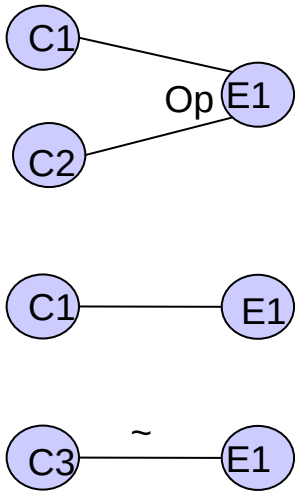
A faire

Graphes Cause-Effet

Approche proposée par G.J Myers

Méthode : proposer un graphe/réseau qui relie les effets du programme (sorties) aux causes (entrées) qui sont à leur origine.

Syntaxe: Op::=And|Or|Nand|Nor



Graphes Cause-Effet : exercice

P prend en entrée une longueur (entier entre 1 et 20), une chaîne de caractères de cette longueur, et un caractère. P retourne sa position dans la chaîne ou un message d'erreur. Il est possible de cherche d'autres caractères.

Précises les causes et les effets

Donner le graphe Cause-Effet

En déduire des DT

Exercice

Une procédure de classification de triangles reçoit en entrées 3 réels a, b et c qui sont les longueurs des côtés d'un triangle. La procédure retourne comme résultat 0 si le triangle défini par a, b et c est invalide, 1 si le triangle est équilatéral, 2 si le triangle est isocèle et 3 pour un triangle valide quelconque (ni isocèle, ni équilatéral).

a) Proposez un partitionnement des données en entrée pour tester cette procédure.

b) En déduire un jeu d'essai pour cette procédure.

Partie I : Le test.

3: Le test structurel

Statique / Dynamique

- Analyse **dynamique** : nécessite l'exécution du code binaire

Principe : à partir du code source et spécification, produire des DT qui exécuteront un ensemble de comportements, comparer les résultats avec ceux attendus…

1. Techniques de couverture du graphe de contrôle
   a) Couverture du flot de contrôle
   b) Couverture du flot de données
2. Test mutationnel (test par injection de défaut)
3. Exécution abstraite
4. Test évolutionniste (algorithme génétique)
5. …

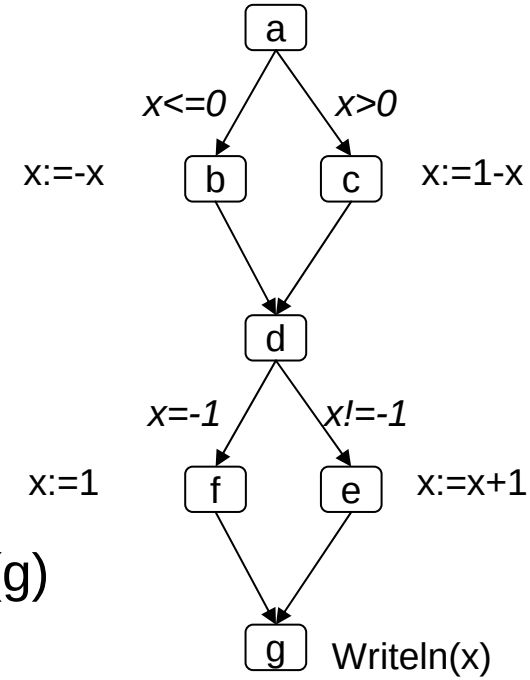
- Analyse **statique** : ne nécessite pas l'exécution du code binaire
  1. Revue de code
  2. Estimation de la complexité
  3. Preuve formelle (prouveur, vérifieur ou model-checking)
  4. Exécution symbolique
  5. Interprétation abstraite

Test structurel dynamique avec technique de couverture du graphe de contrôle

But: produire des DT qui exécuteront un ensemble de comportements du programme

- Utilise : spécification, code source et code exécutable
- Un programme => un graphe de contrôle

```
begin
if (x<=0)then x:=-x
else x:=1-x;
if (x=-1)then x:=1
else x:=x+1;
end
```



- Un sommet entrée (a) et un sommet sortie (g)
- Un sommet = un bloc d'instructions
- Un arc = la possibilité de transfert de l'exécution d'un nœud à un autre

- Une **exécution possible** = un **chemin de contrôle** dans le graphe de contrôle

  [a,c,d,e,g] est un chemin de contrôle

  [b,d,f,g] n'est pas un chemin de contrôle

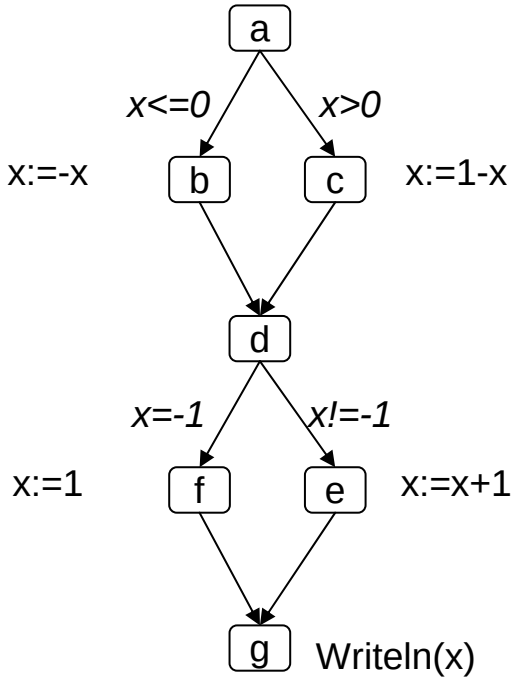
Expression des chemins d'un graphe de contrôle

Soit M l'ensemble des chemins de contrôle du graphe G :

M = abdfg+abdeg+acdfg+acdeg

= a.(bdf+bde+cdf+cde).g

= a.(b+c)d.(e+f).g (expression des chemins de G)



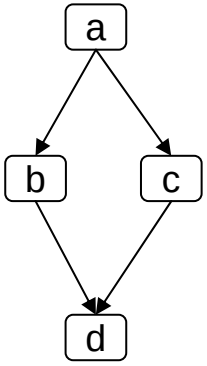
G

Construction de l'expression des chemins :

séquentielle

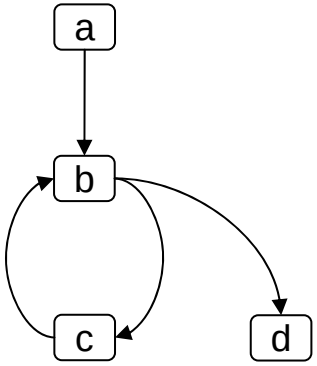
ab

alternative



a.(b+c).d

répétitive

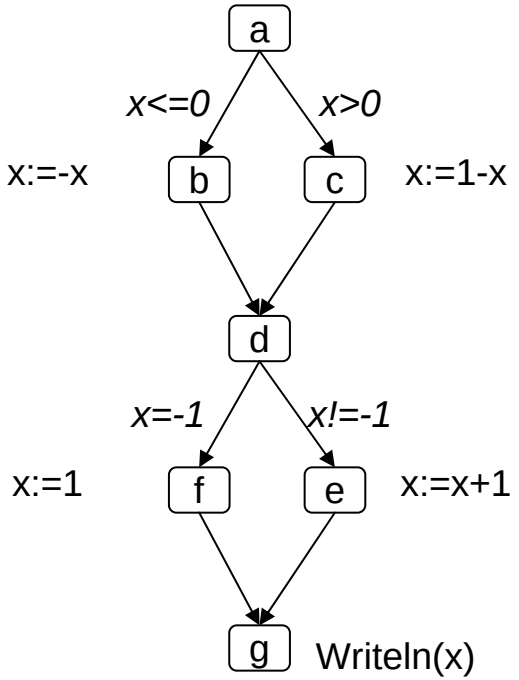


$ab.(cb)^{*}.d$

$ab.(cb)^{4}.d$

Chemin exécutable

```
begin
if (x<=0)then x:=-x
else x:=1-x;
if (x=-1)then x:=1
else x:=x+1;
end
```



DT1={x=2}

DT1 sensibilise le chemin [acdfg] : [acdfg] est un chemin exécutable

[abdgf] est un chemin non exécutable : aucune DT capable de sensibiliser ce chemin

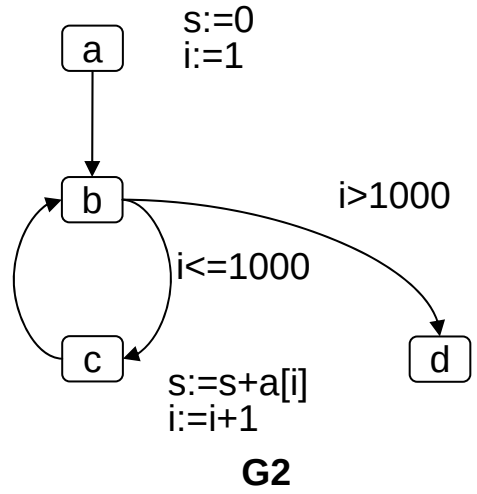
Sensibiliser un chemin peut parfois être difficile : intérêt des outils automatiques (mais attention problème de trouver des DT qui sensibilise un chemin est non décidable)

Existence de chemins non exécutables : signe de mauvais codage ?

Chemin exécutable / chemin non exécutable

- Nombre de chemins de contrôle de G :
  - se déduit directement de l'expression des chemins de G
  - a(b+c)d(e+f)g => 1.(1+1).1.(1+1).1 = 4 chemins de contrôle
  - Nb chemins exécutables + Nb chemins non exécutables
  - Parfois le Nb chemins non exécutables peut être important :

```
begin
s:=0;
for i:=1 to 1000 do s:=s+a[i];
end
```



G2

Expression des chemins de G2 : $a.b.(cb)^{\underline{1000}}.d$

Nombre de chemins :

$1.1.(1.1)^{\underline{1000}}.1 = 1^{\underline{1000}} = 1+1^1+1^2+ \ldots +1^{1000}=1001$

Parmi ces 1001 chemins, un seul est exécutable: $a.b.(cb)^{1000}.d$

Exercice 1

```
lire(b,c,x);
if b<c
then begin
     d :=2*b ;
     f :=3*c
     if x>=0
     then begin
          y := x ;
          e := c ;
          if (y=0)
          then begin
               a :=f-e ;
               if d<a
               then begin
                    writeln(a)
                    end
               else begin
                    writeln (d)
                    end
               end
          end
     end
```

- Donner le graphe de contrôle G(P3) associé au programme P3.
- Donner 3 chemins de contrôle du graphe G(P3).
- Donner l'expression des chemins de contrôle de G(P3).
- Soit DT1={b=1,c=2,x=2}. Donner le chemin sensibilisé par DT1.
- On s'intéresse aux instructions en italique… Donner des DT qui vont couvrir ces instructions.
- Donner un chemin de contrôle non exécutable de G(P3).

Exercice 2

```
Lire(choix)

if choix=1

then x=x+1 ;

if choix=2

then x=x-1 ;

writeln(choix ;
```

1. Donner le graphe de contrôle correspondant au programme P4.
2. Donner l'expression des chemins de contrôle de G(P4). En déduire le nombre de chemins de contrôle.
3. Donner les chemins de contrôle non exécutables. Conclure.
4. Proposer une nouvelle solution pour ce programme. Construisez son graphe de contrôle et donner l'expression des chemins de contrôle ainsi que le nombre de chemins de contrôle.

Exercice 3

1. Écrivez un algorithme de recherche de l'emplacement d'un élément e dans un tableau T (on suppose que T contient l'élément e).

2. Donner le graphe de contrôle associé.

3. Donner l'expression des chemins.

4. Dans le cas où le tableau a une taille de 3, donner le nombre de chemins de contrôle.

Satisfaction d'un test structurel avec couverture

Soit T un test structurel qui nécessite la couverture d'un ensemble de chemins $\{\delta_1, \ldots, \delta_k\}$ du graphe de contrôle.

On notera : T= $\{\delta_1, \ldots, \delta_k\}$

Soit DT une donnée de test qui sensibilise le chemin de contrôle C.

- Définition: DT **satisfait** T ssi C couvre tous les chemins de T.
- Exemple : considérons le graphe de contrôle G5

  Soient δ_1=cdebcde et δ_2=ce et T1= $\{\delta_1, \delta_2\}$.

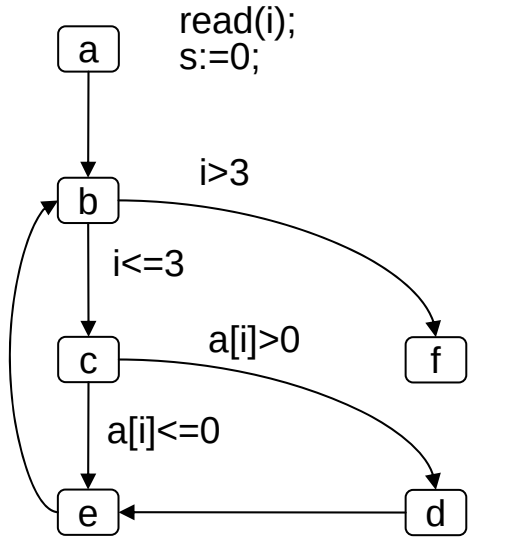
  DT1={a[1]=-2, a[2]=3, a[3]=17,i=1} sensibilise :

  M1=abcebcdebcdebf

  M1= abcebcdebcdebf couvre δ_1=cdebcde

  M1=abcebcdebcdebf couvre δ_2=ce

  Donc DT1 satisfait T1



G5

Hiérarchie des techniques de test structurel

- Exemple : considérons le graphe de contrôle G5

  Soient δ_1=de et δ_2=b et δ_3=cd et T2= {δ_1, δ_2 , δ_3}.

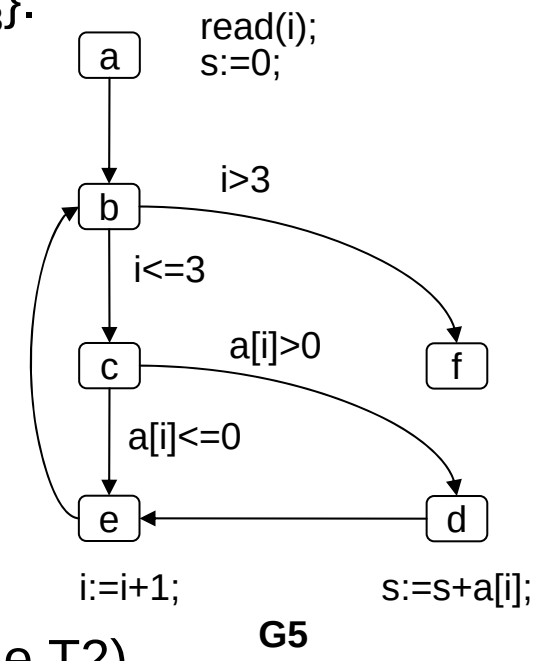
  DT1={a[1]=-2, a[2]=3, a[3]=17,i=1} sensibilise :
  M1=abcebcdebcdebf

  M1= abcebcdebcdebf couvre δ_1=de

  M1=abcebcdebcdebf couvre δ_2=b

  M1=abcebcdebcdebf couvre δ_3=cd

  Donc DT1 satisfait T2



G5

- Lorsque T1 est satisfait, T2 l'est aussi :
  T1 ⇒ T2 (T1 est un test plus **fiable** (ie. 'fort') que T2)

- T1 ⇒ T2 et T2 ⇒ T3 alors T1 ⇒ T3 (Transitivité)

- Hiérarchie entre les différentes techniques structurelles de test (relation d'ordre partielle)

Deux Catégories de critère de couverture

- Approche '**Flot de contrôle**' avec couverture de tous les arcs :

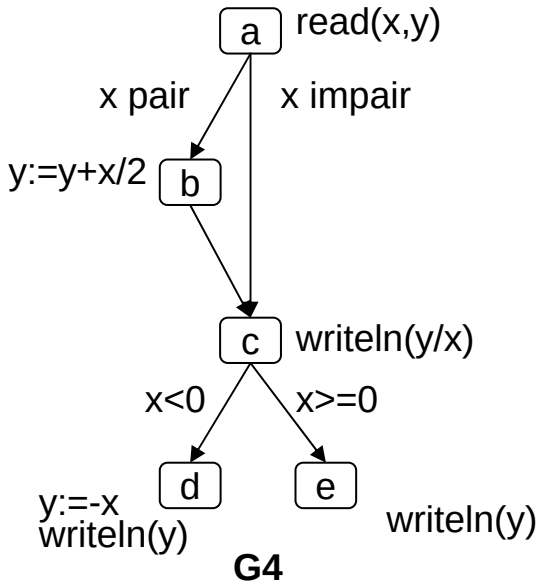
  DT1={x=-2, y=0} sensibilise le chemin M1=abcd

  DT2={x=1, y=0} sensibilise le chemin M2=ace

  - Si l'affectation du nœud b est erronée, cette erreur ne sera pas détectée par DT1 et DT2.

- Approche '**Flot de données**'

  L'affectation de y au nœud b n'est pas utilisée par DT1 et DT2 : il faudrait tester le chemin abce sensibilisé par la DT3={x=2, y=0}



G4