Cours Qualité et Tests
Chapitre 3 : Tests

INSTITUT SUPERIEUR INFORMATIQUE ISI المعهد العالي للاعلامية

Responsables du cours :
Héla Hachicha

Année Universitaire : 2016 - 2017

Sommaire

3

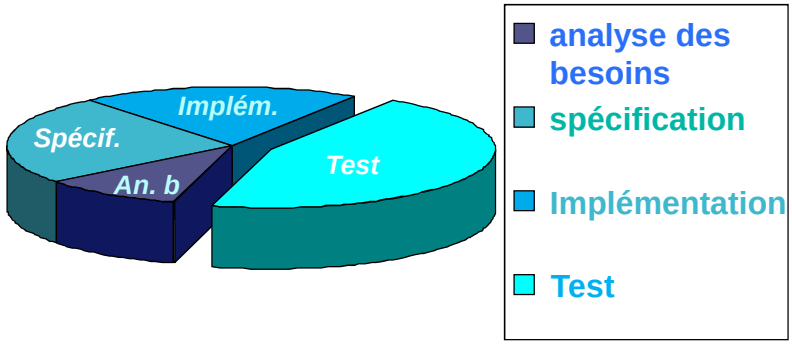
Problématique du test

- On ne peut pas tester tout le temps ni tous les cas possibles
  - Il faut des critères pour choisir les cas intéressants et la bonne échelle pour le test
- Prouver l'absence d'erreurs dans un programme est un problème indécidable
  - il faut des heuristiques réalistes

4

Problématique du test

Le coût du test dans le développement



+ maintenance = 80 % du coût global de développement !!!

5

Problématique du test

- Un jeune diplômé sur trois commence par faire du test
- 50% des start-up échouent à cause du trop grand nombre de bugs
  - mauvaise campagne de test
  - maintenance difficile
  - pas de non régression

6

Terminologie

- Une faute est la cause d'une erreur
- Une erreur (IEEE 729) est Un écart entre une valeur ou condition calculée, observée ou mesurée et la valeur ou condition qui est vraie, spécifiée ou théoriquement correcte.
- Défaut, anomalie (fault, bug) (IEEE 729) est la manifestation d'une erreur dans un logiciel Un défaut peut causer une panne.
- Panne (failure) (IEEE 729) est la fin de la capacité d'un système ou d'un de ses composants d'effectuer la fonction requise, ou de l'effectuer à l'intérieur de limites spécifiées.

7

Terminologie

- SPECIFICATION (ISO 8402)
  Document qui prescrit les exigences auxquelles le produit ou le service doit se conformer.
- SATISFACTION
  Un programme satisfait sa spécification lorsqu'il est en tout point conforme aux exigences de celle-ci.
- VALIDATION ou VERIFICATION (ISO 9000-3)
  Processus d'évaluation du logiciel pour s'assurer qu'il satisfait aux exigences spécifiées. La validation ou la vérification d'un produit cherche à s'assurer qu'on a construit le bon produit (d'un point de vue externe ou interne). Le test est un cas particulier de vérification.

8

Définitions du test

- «Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus » -IEEE (Standard Glossaryof Software Engineering Terminology)
- «Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts »-G. Myers (The Art of Software testing)
- "Testing can reveal the presence of errors but never their absence"–EdsgarW. Dijkstra. *Notes on structured programming. AcademicPress, 1972.*

9

Objectifs

- Le test vise à mettre **en évidence** les erreurs d'un logiciel
- Le test n'a pas pour objectif de **diagnostiquer** la cause des erreurs
- Le test n'a pas pour objectif de **corriger** les fautes
- Le test n'a pas pour objectif de **prouver** la correction d'un programme

10

Qualité du test

- L'efficacité du test (son aptitude à détecter des erreurs) doit être conforme à certains critères de qualité.
- Le niveau de qualité requis dépend du contexte d'utilisation du logiciel: plus le contexte est critique, plus l'effort de tests doit être important.
- La programmation d'un logiciel aérospatial requiert des exigences de qualité supérieures à la programmation d'un éditeur de dessins techniques
- **QUALITE (QUALITY)**, ISO 8402:
  Ensemble des propriétés et caractéristiques d'un produit ou service qui lui confèrent l'aptitude à satisfaire des besoins exprimés ou implicites.

Familles de tests

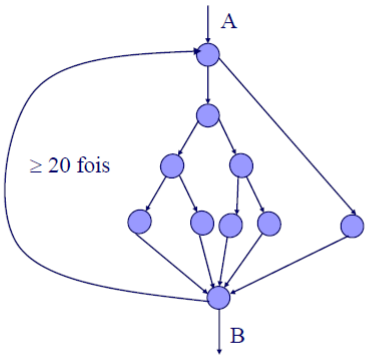
Deux grandes familles de tests

- Test structurel (ou test boîte blanche)
- Test fonctionnel (ou test boîte noire)

Tests de boîte blanche : Test structurel – «White Box Testing»

Les données de test sont produites à partir d'une analyse du code source

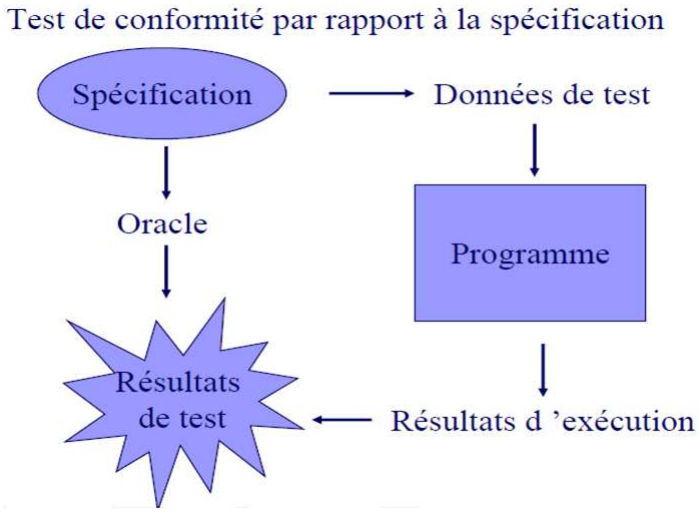
Critères de test :
- tous les chemins,
- toutes les branches,
- toutes les instructions



Flot de contrôle d'un petit programme

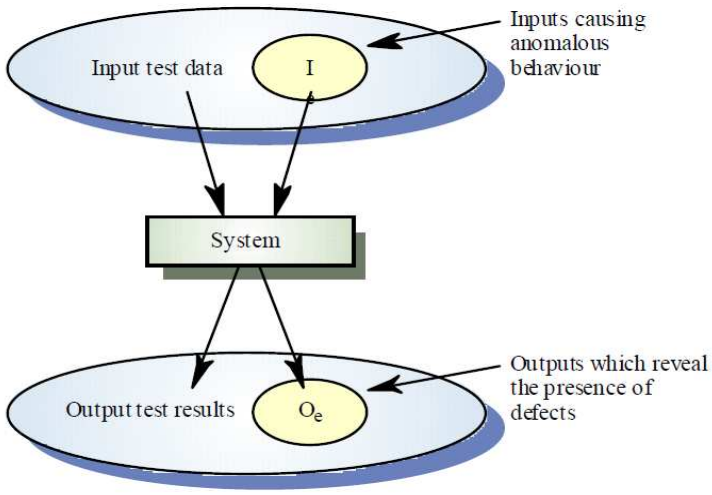
13

Test fonctionnel



14

Tests de boîte noire : Black-box testing



15

Complémentarité test fonctionnel -structurel (1)

- Les techniques fonctionnelles et structurelles sont utilisées de façon complémentaire

Exemple : Soit le programme suivant censé calculer la somme de deux entiers

```
function sum (x,y : integer) : integer;
begin
if (x = 600) and (y = 500) then sum := x-y
else sum := x+y;
end
```

Une approche fonctionnelle détectera difficilement le défaut alors qu 'une approche par analyse de code pourra produire la DT : x = 600, y = 500

16

Complémentarité test fonctionnel -structurel (2)

- En examinant ce qui à été réalisé, on ne prend pas forcément en compte ce qui aurait du être fait :

⇒ Les approches structurelles détectent plus facilement les erreurs commises

⇒ Les approches fonctionnelles détectent plus facilement les erreurs d'omission et de spécification

Une difficulté du test structurel consiste dans la définition de l'Oracle de test.

Difficultés du test (1)

Le test exhaustif est en général impossible à réaliser

- En test fonctionnel, l'ensemble des données d'entrée est en général infini ou très grande taille

Exemple : un logiciel avec 5 entrées analogiques sur 8 bits admet 240valeurs différentes en entrée

- En test structurel, le parcours du graphe de flot de contrôle conduit à une forte explosion combinatoire

Exemple : le nombre de chemin logique dans le graphe de la figure 1 est supérieur à1014≈520+ 519+ …+ 51

=> le test est une méthode de vérification partielle de logiciels

=> la qualité du test dépend de la pertinence du choix des données de test

Difficultés du test (2)

Difficultés d'ordre psychologique ou «culturel»

- Le test est un processus destructif : un bon test est un test qui trouve une erreur

alors que l'activité de programmation est un processus constructif -on cherche à établir des résultats corrects

- Les erreurs peuvent être dues à des incompréhensions de spécifications ou de mauvais choix d'implantation

=> L'activité de test s'inscrit dans le contrôle qualité, indépendant du développement

19

Terminologie : alpha- et bêta-test

- **Alpha-test** (alpha testing)
  - test effectué en phase de développement, avant la distribution du produit (→ alpha-versions du produit)
- **Bêta-test** (beta testing)
  - test effectué après l'alpha-test, en distribuant le produit (→ des bêta-versions) à un groupe limité d'utilisateurs avertis

20

Organisation de l'activité de test

- Activité coûteuse → optimiser l'investissement
  - effort minimum / probabilité max. de détection d'erreur
  - Incrémentalité
- Construction des tests
  - aussi organisée que celle d'un produit **(!)**

(☛ il y a des sociétés qui vendent des suites de test)

- Gestion projet
  - planification suffisamment tôt (difficile d'accroître les ressources en fin de développement)

Tâches

- Définition des tests
- Implémentation des jeux de tests
- Soumission des jeux de tests
- Dépouillement des résultats
- Évaluation de la qualité des tests
- Décision d'arrêter l'écriture de tests
- Rejeu (maintenance, non régression)

Environnements (outils) de test

- Mise en œuvre des jeux de test
  - construction de données et de contextes d'exécution
- Diagnostic
  - définition de critères de réussite / échec
  - automatisable ou non (ex. test d'interface)
- Synthèse des résultats
  - car les sorties des tests sont souvent très grosses

→ ne pas rater une erreur dans une masse de succès

- Diffusion des résultats

Types de Tests (éléments testés et phases)

- **Tests unitaires :**
Test de procédures, de modules, de composants
- **Tests d'intégration :**
Test de bon comportement lors de la composition de procédures et modules
- **Tests de conformité ou test système :**
Validation de l'adéquation aux spécifications
- **Tests de non-régression :**
Vérification que les corrections ou évolution dans le code n'ont pas créées d'anomalies nouvelles

Types de Tests (nature des propriétés testées)

- **Tests nominal ou test de fonctionnel**

Les cas de test correspondent à des données d'entrée valide.
(Réaction à certaines entrées (sorties produites)
⇒Test-to-pass

⇒ **Tests de robustesse :**
Les cas de test correspondent à des données d'entrée invalide
⇒Test-to-fail

Règle : Les tests nominaux sont passés avant les tests de robustesse.

- **Test de performance**
vitesse, charge
–Load testing (test avec montée en charge)
–Stress testing (soumis à des demandes de ressources anormales)
- **Tests de fiabilité**
résistance aux pannes
- **Tests de sécurité,**
- ...

☛ Tous types et étapes de test pas nécessairement présents : dépend de la criticité du logiciel

25

Types de Tests (selon les informations accédées)

- **Tests de boîte noire** [black box testing]

Le test porte sur le fonctionnement externe du système. La façon dont le système réalise les traitements n'entre pas dans le champ du test.

– évaluation de l'extérieur (sans regarder le code), uniquement en fonction des entrées et des sorties
– sur le logiciel ou un de ses composants

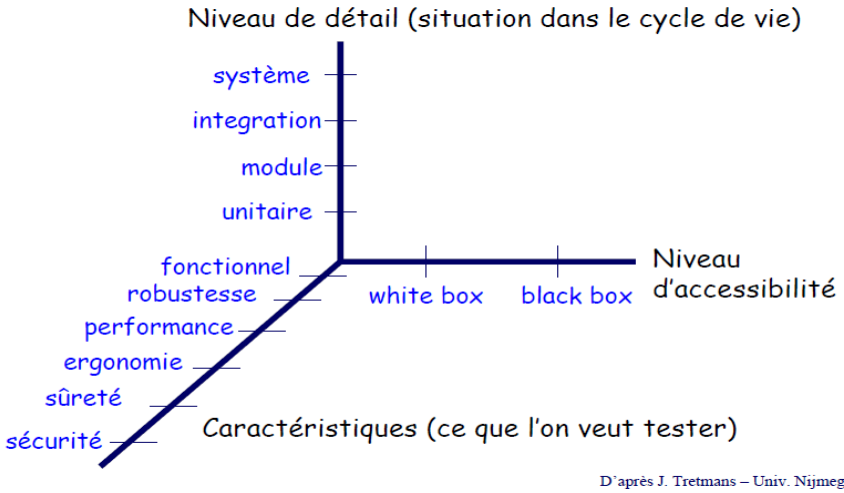
- **Tests de boîte blanche** [white/glass box testing]

Le test vérifie les détails d'implémentation, c'est à dire le comportement interne du logiciel..

– exploite le code (→ besoin du source/de l'architecture)
– tests de portions de code : bloc, branche, etc.

26

Types de Tests

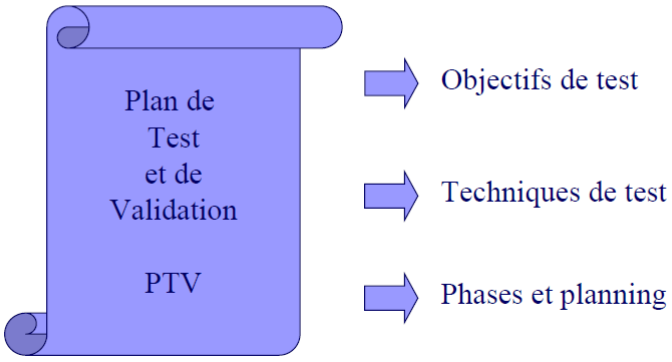


D'après J. Tretmans – Univ. Nijmegen

27

Test et démarche d'assurance qualité

- En début de projet, définition d'un Plan de Test et Validation - PTV



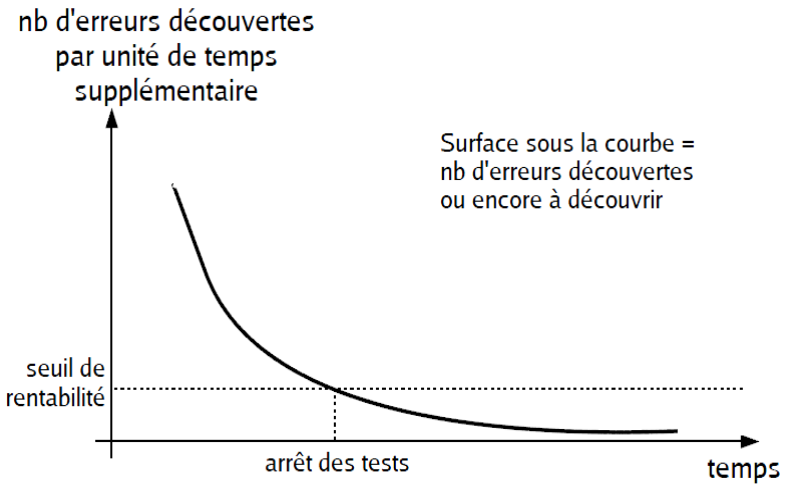
28

Critères d'arrêt des développements de tests

- Taux de couverture atteint (☛ critère a priori)
  - suffisamment d'aspects testés
- Nombre ou taux d'erreurs découvertes (☛ critère a posteriori)
  - courbe du nb d'erreurs en fonction de la durée
  - arrêt sous un certain seuil (→)
  - séparation des erreurs par catégorie
- Épuisement des ressources dédiées au test (☹)
  - effort humain et/ou durée

29

Taux de découverte de bogues et arrêt des tests



30

Rapport qualité prix

Nombre de cas de test arbitrairement grand
→ Nécessité d'un compromis

- précision, bon degré de couverture, bonnes informations pour les développeurs-testeurs (reproductibilité, debug)
- coût (définition, réalisation, passage, dépouillement)
- temps d'exécution de tests
- nb de ressources de calcul (machines) mobilisées

1. Méthodes de test structurel

- Le test structurel s'appuie sur l'analyse du code source de l'application (ou d'un modèle de celui-ci) pour établir les tests en fonction de critères de couverture

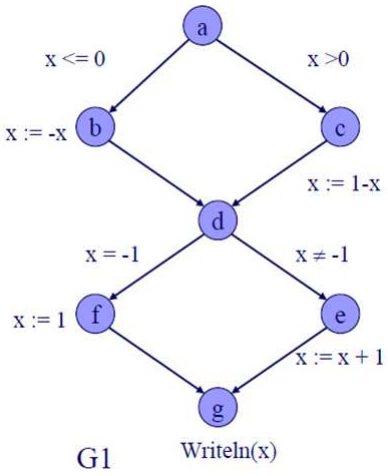
➔ Basés sur le graphe de flot de contrôle (toutes les instructions, toutes les branches, tous les chemins, ...)
➔ Basés sur la couverture du flot de données (toutes les définitions de variable, toutes les utilisations, ...)
➔ Basés sur les fautes (test par mutants)

1-1 Graphe de flot de contrôle

- Soit le programme P1 suivant :

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

Ce programme admet le graphe de contrôle G1.



G1

1-1 Graphe de flot de contrôle

- Graphe orienté et connexe (N,A,e,s)
  - e: un sommet entrée (a)
  - s: un sommet sortie (g)
  - Un sommet/ Un nœud = 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

Chemins dans le graphe de contrôle

- Le graphe G1 est un graphe de contrôle qui admet une entrée -le nœud a -, une sortie -le nœud g.
  - le chemin *[a, c, d, e, g]* ***est un chemin de contrôle,***
  - le chemin *[b, d, f, g]* ***n'est pas un chemin de contrôle.***

- Le graphe G1 comprend 4 chemins de contrôle :
  - $\beta 1 = [a, b, d, f, g]$
  - $\beta 2 = [a, b, d, e, g]$
  - $\beta 3 = [a, c, d, f, g]$
  - $\beta 4 = [a, c, d, e, g]$

35

Expression des chemins de contrôle

- Le graphe G1 peut-être exprimé sous forme algébrique sous la forme suivante :

G1 = abdfg+ abdeg+ acdfg+ acdeg
le signe + désigne le «ou» logique entre chemins.

- Simplification de l'expression de chemins

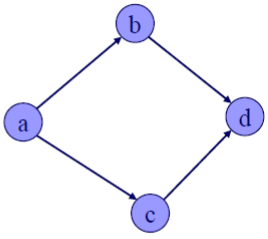
G1 = a (bdf+ bde+ cdf+ cde) g
G1 = a (b + c) d (e + f) g

36

Calcul de l'expression des chemins de contrôle

- On associe une opération d'addition ou de multiplication à toutes les structures primitives apparaissant dans le graphe de flot de contrôle

Forme séquentielle : ab



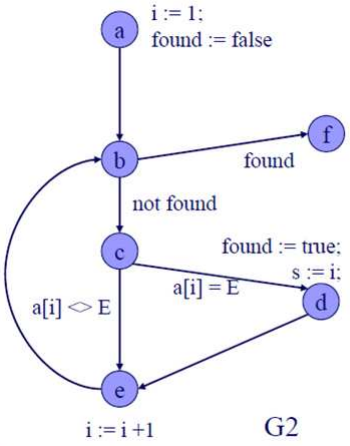
Forme alternative :
a (b + c) d

Forme itérative : ab (cb)* d

Chemins de contrôle avec boucles

- Soit le programme P2 suivant :

```
i := 1;
found := false;
while (not found) do
    begin
    if (a[i] = E) then
       begin
       found := true;
       s := i;
       end;
    i := i + 1;
    end;
```



G2 = ab [c (1 + d) eb]* f

Expressions de chemins -Exercice

- Soit le programme P3 suivant :
  - Soit le programme P3 suivant :

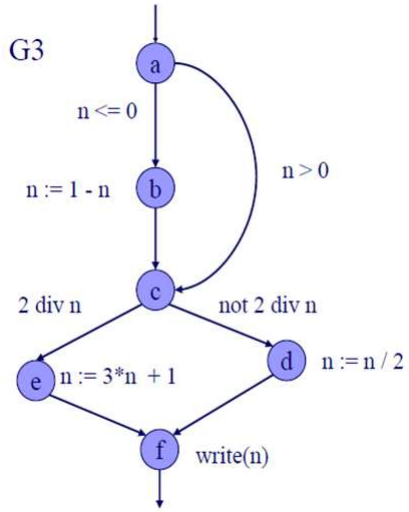
```
if n <= 0 then n := 1-n
   end;
if 2 div n
      then n := n / 2
      else n := 3*n + 1
      end ;
write(n);
```

Question :

- Etablir **le graphe de flot de contrôle** de ce programme
- Fournir **l'expression des chemins**

39

Graphe de flot de contrôle du programme P3 – Solution



G3 = a (1 + b) c (e + d) f

40

Expressions de chemins -Exercice

- Soit le programme P4 suivant :

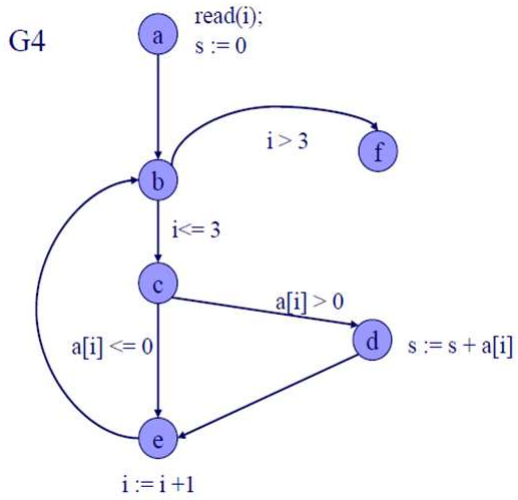
```
read(i);
s := 0;
while (i <= 3) do
      begin
      if a[i] > 0 then s := s + a[i];
      i := i + 1;
      end
end;
```

Question :

- Etablir **le graphe de flot** de contrôle de ce programme
- Fournir **l'expression des chemins**

41

Graphe de flot de contrôle du programme P4 – Solution



G4 = ab [c (1 + d) eb]* f

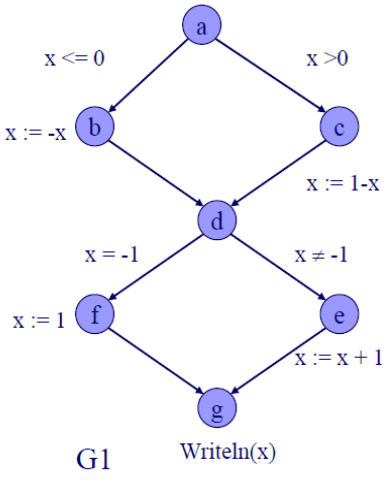
42

Chemins Exécutables

- Soit le programme P1 suivant :

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

Ce programme admet le graphe de contrôle G1.



G1

43

Chemins Exécutables

- DT1={x=2}
- DT1 sensibilise le chemin [acdfg] : [acdfg] est un chemin exécutable
- [abdfg] 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 le problème de trouver des DT qui sensibilise un chemin est non décidable)
- Existence de chemins non exécutables : signe de mauvais codage !!!

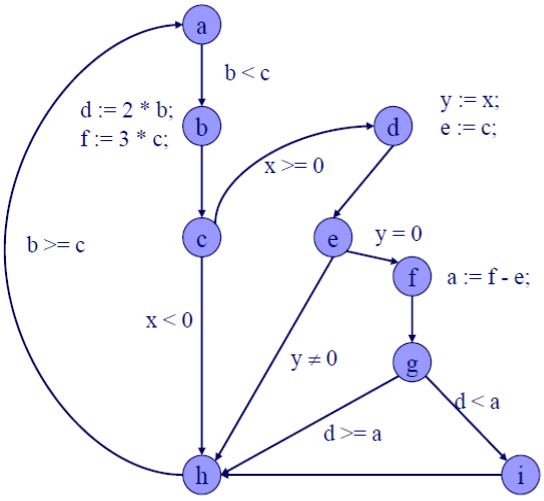
44

Chemins Exécutables

- 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

45

Chemins non exécutables



- L 'arc **g-h** n'est pas activable :
  2b < 2c donc d < a
- Le problème de la détection de chemin non exécutable est un problème difficile, indécidable dans le cas général

46

Problèmes des chemins non Exécutables

- Étant donné un chemin qu'on a envie de sensibiliser, comment trouver une DT qui exécute ce chemin ? Problème très difficile:
  - 1. décider si le chemin est exécutable ou pas;
  - 2. s'il l'est trouver une DT.
- Le problème 1 est indécidable.
- [indécidable = formellement impossible de construire un algorithme général qui décide de l'exécutablilité ou de la non exécutabilité de n'importe quel chemin]
- La présence de chemins non-exécutables est souvent signe de code mal écrit, voire erroné !

47

Couverture sur le flot de contrôle

Critère de couverture « tous-les-nœuds »

But : sensibiliser tous les chemins de contrôle qui nous permettent de visiter tous les Nœuds du graphe de contrôle.

Taux de couverture :

TER1 (Test Effectiveness Ratio 1 ou C1) TER1 = |{nœuds couverts}| / |{nœuds}|

Critère de couverture « tous-les-arcs »

- Si on chercher à couvrir tous les nœuds sans couvrir tous les arcs, on risque de ne pas détecter certains défauts sur les arcs non couverts…

But : sensibiliser tous les chemins de contrôle qui nous permettent de visiter tous les arcs du graphe de contrôle.

TER2 = |{arcs couverts}| / |{arcs}|

Hiérarchie des tests

« tous-les-arcs » ⇒ « tous-les-noeuds »

48

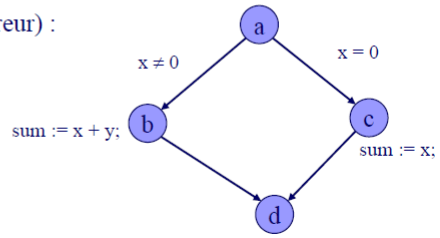
Critère de couverture « *tous-les-nœuds* »

- Taux de couverture :

$$\frac{\text{nb de nœuds couverts}}{\text{nb total de nœuds}}$$

Soit le programme P5 (somme avec erreur) :

```
function sum (x,y : integer) : integer;
begin
if (x = 0) then sum := x
else sum := x + y
end;
```



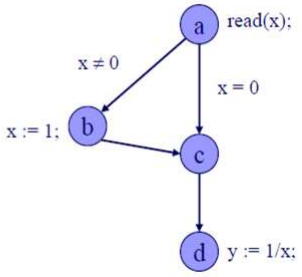
⇒ L 'erreur est détectée par l'exécution du chemin [acd]

49

Limites du critère *tous-les-noeuds*

Soit le programme P6 (avec erreur) :

```
read(x);
…
if (x <> 0) then x := 1;
…
y := 1/x;
```



⇒ Le critère tous-les-nœuds est satisfait par le chemin [abcd] sans que l 'erreur ne soit détectée.

50

Critère de couverture »*tous-les-arcs* »

- Taux de couverture :

$$\frac{\text{nb des arcs couverts}}{\text{nb total des arcs}}$$

- La couverture de tous les arcs équivaut à la couverture de toutes les valeurs de vérité pour chaque nœud de décision.

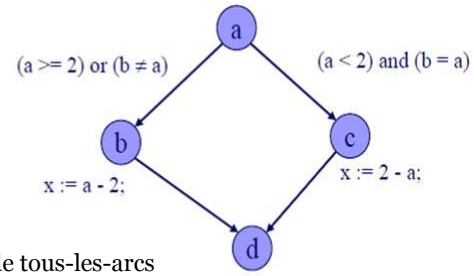
⇒ Lorsque le critère *tous-les-arcs* est totalement réalisé, cela implique que le critère *tous-les-nœuds* est satisfait

51

Cas des conditionnelles composées (1)

- Exemple :

```
if ((a < 2) and (b = a))
then x := 2 - a
else x := a - 2
```



Donner le DT qui satisfait le critère de tous-les-arcs

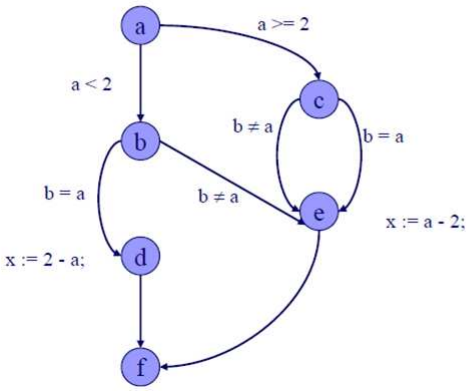
- Le jeu de test DT1 = {a=b=1}, DT2 = {a=b=3}

satisfait le critère *tous-les-arcs* sans couvrir toutes les décisions possibles - ex. DT3 = {a=3, b=2}.

52

Cas des conditionnelles composées (2)

- Le graphe de flot de contrôle doit décomposer les conditionnelles :



Données de test :
- DT1 = {a=b=1}
- DT2 = {a=1, b=0}
- DT3 = {a=3, b=2}
- DT4 = {a=b=3}

53

Critère de couverture de *condition-décision multiple*

- Le critère de condition-décision multiple est satisfait si :
  - Le critère *tous-les-arcs* est satisfait
  - En plus, chaque sous-expression dans les conditions prend toutes les combinaisons de valeurs possibles
  - Si A & B Then …….. Nécessite :
    - A = B = vrai
    - A = B = faux
    - A = vrai, B = faux
    - A = faux, B = vrai
  - Problème de la combinatoire lors d'expression logique complexe

54

Limites des critères *tous-les-arcs et condition-décision multiple*

- Il n 'y a pas détection d 'erreurs en cas de non-exécution d 'une boucle

Soit le programme P7 (avec erreur) :

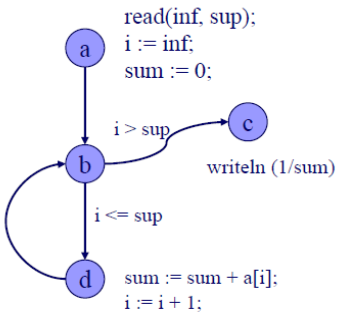
```
read(inf, sup);
i := inf;
sum := 0;

while (i <= sup) do
begin
sum := sum + a[i];
i := i + 1;
end;
writeln (1/sum);
```

55

Limites des critères *tous-les-arcs et condition-décision multiple*

- Il n 'y a pas détection d 'erreurs en cas de non-exécution d 'une boucle



La donnée de test DT1 :
DT1= {a[1]=50, a[2]=60, a[3]=80, inf=1, sup=3}
couvre le *critère tous-les-arcs*

Problème non détecté par le critère *tous-les-arcs* : si inf >sup erreur sur 1/sum

56

Critère de couverture des « chemins indépendants »

- Le critère des *« chemins-indépendants » vise à parcourir tous les arcs dans chaque configuration possible (et non pas au moins une fois comme dans le critère tous-les-arcs)*

57

Critère de couverture des « chemins indépendants »

« Sélectionner un jeu de test T tel que, lorsqu'on exécute P sur les d ∈ DT, tous les 1-chemins du graphe de flot de P sont parcourus au plus une fois. »

- Chemin = Séquence de nœuds et d'arcs dans le graphe de flot de contrôle, initiée depuis le nœud de départ jusqu'à un nœud terminal. (Il peut y avoir plusieurs nœuds terminaux dans un programme.)
  - 1-chemin : Chemin parcourant les boucles 0 ou 1 fois.
  - Chemin indépendant : (1-)chemin du graphe de flot de contrôle qui parcourt au moins un nouvel arc par rapport aux autres chemins définis dans une base B (i.e. ce chemin introduit au moins une nouvelle instruction non parcourue).

58

Critère de couverture des « chemins indépendants »

- Méthode de sélection des jeux de test
  1. Construire le graphe de flot de contrôle de P
  2. Déterminer la complexité cyclomatique ***V(G)*** du GFC
     - Constitue une borne supérieure sur le nombre de chemins nécessaires pour couvrir tous les chemins indépendants du graphe de flot d'un programme.
  3. Définir un ensemble de base B de chemins indépendants dans le graphe.
  4. Construire un jeu de test qui permettra l'exécution de chaque chemin de l'ensemble B.

59

Critère de couverture des « chemins indépendants »

Critère de couverture des « chemins- indépendants »

- V(G) (le nombre de Mc Cabe ou nombre cyclomatique) donne le nombre de chemins indépendants.
- V(G)=#arcs - #noeuds + 2
- Si que des décisions binaires : V(G)=Nombre de noeuds de décision + 1

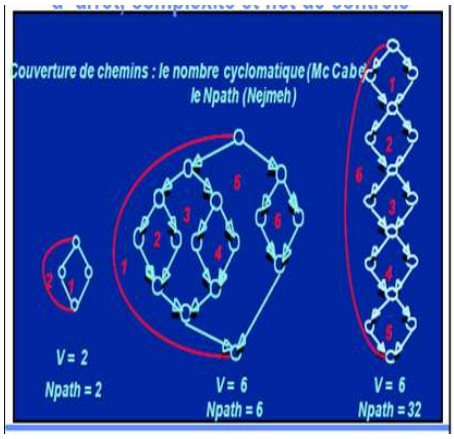
→ Ce nombre est aussi le nombre de régions du graphe

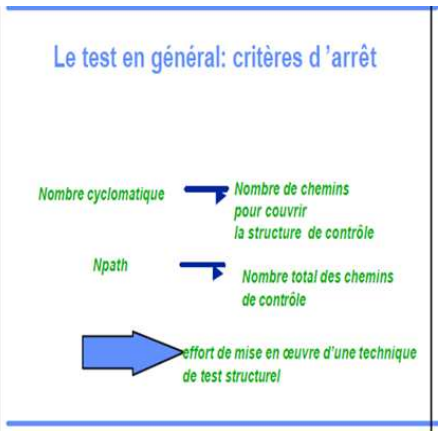
Taux de couverture :

- |{chemins indépendants couverts}| / V(G)
- **Hiérarchie des tests**
- «tous-les-chemins-indépendants »⇒«tous-les-arcs»

60

Critère de couverture des « chemins indépendants »





61

Critère de couverture des « chemins indépendants »

- Méthode de sélection des jeux de test

Définir un ensemble de base B de chemins indépendants dans le graphe

- *Chemin indépendant* : chemin du graphe de flot de contrôle qui parcourt au moins un nouvel arc par rapport aux autres chemins définis (introduit au moins une nouvelle instruction non parcourue).
- *Une base comportant V(G) chemins nous assure de couvrir tous chemins indépendants du graphe de flot G.*
- *Mais on ne couvre pas nécessairement tous les 1-chemins du graphe…*

62

Critère de couverture des « chemins indépendants »

- Méthode de sélection des jeux de test

Sélection des jeux de test

- Pour chaque chemin indépendant de la base, on doit trouver un jeu de test qui permette de le traverser (en itérant possiblement sur certains segments du chemin).
- Cette sélection peut être ardue dans le cas de gros programmes!
- En effet, ceci équivaut à
  - Résoudre un système de contraintes composé des nœuds prédicats qui se trouvent sur le chemin à parcourir.
  - Attention: Tous les chemins ne sont pas nécessairement satisfiables!!! (Problème indécidable)

Critère de couverture des « chemins indépendants »

- Lorsque le critère des *« chemins-indépendants » est satisfait, cela implique :*
  - le critère *tous-les-arcs est satisfait,*
  - le critère *tous-les-nœuds est satisfait.*

Critère de couverture des « chemins indépendants »

Exercice 1 : Algorithme d'Euclide

```
begin
    read(x); read(y)
    while x <> y do
        If x>y then x:= x-y;
        else y:= y-x;
        end if
    end while
    pgcd := x;
end
```

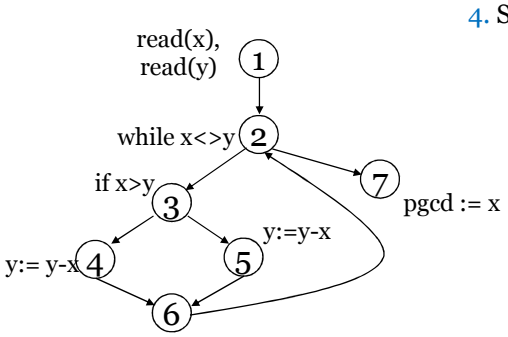
Questions :
1- déterminer le graphe de contrôle
2-Détermine le nombre cyclomatique
3- trouver les DTs qui satisfait le critère de tous les chemins indépendants

65

Critère de couverture des « chemins indépendants »

Exemple

- Méthode de sélection des jeux de test



4. Sélection des jeux de test

Partitionnement

Chemin 1-2-7
D1= {(x,y) | x=y}
Cas de test: <x=3,y=3>

Chemin 1-2-3-4-6-2-7
D2 = {(x,y)| x>y, x-y=y}
Cas de test: <x=8,y=4>

Chemin 1-2-3-5-6-2-7
D3 = {(x,y)| x<y, x=y-x}
Cas de test: <x=3,y=6>

Jeu de test: {<x=3 y=3>, <x=8, y=4>, <x=3, y=6>}

66

Critère de couverture des « chemins indépendants »

- Méthode de sélection des jeux de test

Sélection des jeux de test

Limites

```
Found:= false; counter:=1;
While (not found) and counter< numberItems do    /* erreur <= */
      If table(counter) = desiredElem then
       found := true;
      endif
      counter:= counter+1;
End while;
If found then
      write(« Elément existe. »);
Else
      write(« Élément n'existe pas. »);
Endif;
```

Limite…

Ex: Soit le jeu de test suivant qui parcourt tous les chemins indépendants du graphe (il y en a 4, mais l'un d'eux est impossible à parcourir) :

- table vide
- table avec 1 élément ne contenant pas celui désiré.
- table avec 3 éléments dont le premier est celui cherché.

Malheureusement, on n'a pas découvert l'erreur !

67

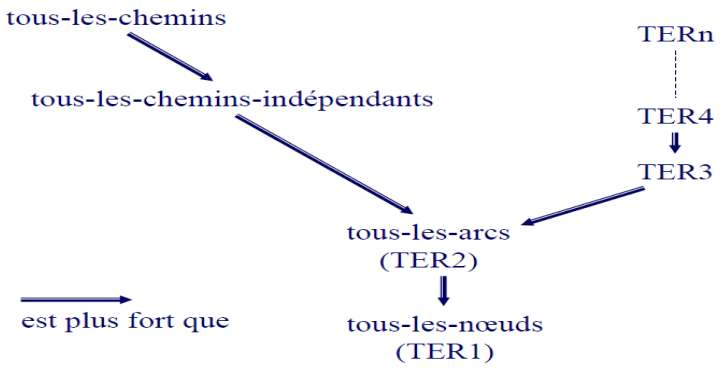
Hiérarchie des critères basés sur le flot de contrôle

Quelques critères de couverture sur flot de contrôle:

- Tous les nœuds : le plus faible.
- Tous les arcs / décisions : test de chaque décision
- Toutes les conditions simples : peut ne pas couvrir toutes les décisions
- Toutes les conditions/décisions
- Toutes les combinaisons de conditions : explosion combinatoire !
- Tous les chemins : le plus fort, impossible à réaliser s'il y a des boucles

68

Hiérarchie des critères basés sur le flot de contrôle



1-2 Critères de couverture basés sur le flot de données

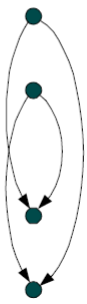
- Toute variable a des
  - points de définition, c.-à-d. affectations (x = 0;)
  - points d'utilisation dans des conditions logiques (x > 0)
  - points d'utilisation dans des instructions de calcul (x+3)
- Couverture
  - de toutes les définitions
  - de toutes les utilisations dans des conditions logiques
  - de toutes les utilisations dans des calculs
  - définitions exécutées au moins une fois pour toutes les utilisations qu'elle atteint, ...

1-2 Critères de couverture basés sur le flot de données

Ex. définitions exécutées au moins une fois pour toutes les utilisations qu'elle atteint → 4 cas

```
if (cond1)
  x = exp1;   // définition 1 de x
else
  x = exp2;   // définition 2 de x

if (cond2)
  y = ... x ...;   // utilisation 1 de x
else
  y = ... x ...;   // utilisation 2 de x
```



71

1-2 Critères de couverture basés sur le flot de données

- Les critères basés sur le flot de données sélectionnent les **données de test** en fonction des **définitions** et des **utilisations** des variables du programme
- Définitions sur les occurrences de variables :
  - une variable est *définie lors d'une instruction si la valeur de la variable est modifiée (affectations),*
  - Une variable est dite *référencée si la valeur de la variable est utilisée.*
- Si la variable référencée est utilisée dans le **prédicat** d'une instruction de décision (if, while, …), il s'agit d'une ***p-utilisation***, *dans les autres cas (par exemple dans un calcul), il s'agit d'une* ***c-utilisation***.

72

Critères basés sur le flot de données

- Notion d'instruction utilisatrice :
  - Une instruction J2 est *utilisatrice* d'une variable x par rapport à une instruction J1, si la variable x qui a été définie en J1 est peut être directement référencée dans J2, c'est-à-dire sans redéfinition de x entre J1 et J2

Exemple :

```
(1)   x := 7;
(2)   a := x+2;
(3)   b := a*a;
(4)   a := a+1;
(5)   y := x + a;
```

Considérons l'instruction (5) :

(5) est c-utilisatrice de (1) pour la variable x
(5) est c-utilisatrice de (4) pour la variable a

Critères basés sur le flot de données

- Notion de chemin d'utilisation
  - Un chemin d'utilisation relie l'instruction de définition d'une variable à une instruction utilisatrice; un tel chemin est appelé chemin dr-strict

Exemple :

```
(1)   x := 7;
(2)   a := x+2;
(3)   b := a*a;
(4)   a := a+1;
(5)   y := x + a;
```

Le chemin [1,2,3,4,5] est un chemin dr-strict pour la variable x (mais pas pour la variable a)

73

Critères basés sur le flot de données

d	variable définie
r	variable référencée (utilisée)
p-utilisation	dans le prédicat d 'une instruction de décision
c-utilisation	dans un calcul

```
while (i < N) do          i et N sont p-utilisées et à la dernière exécution,
                          i est c-utilisée et ensuite définie
  begin
  s := s + i;             s et i sont c-utilisées, puis s est définie
  i := i + 1;
  end;
writeln (s);              s est utilisée
```

75

Critères basés sur le flot de données

x := a + b ;	x est définie et a et b référencées
read(x)	x est définie
write (x)	x est référencée
if (x=1) then x := 7	x est référencée, puis définie
a [i] := x	a est définie, i et x référencées
x := x + 1	x est référencée, puis définie

76

Critères basés sur le flot de données

dr-chaîne

```
program p (input, output) ;
var x, y, z, a, b, c : integer ;
begin
read (c);
x := 7;
y := x + a;
b := x+y+a;
c := y + 2*x + z;
write (x, c)
end.
```

Variable	dr-chaîne
x	drrrr
y	drr
z	r
a	rr
b	d
c	ddr

z et a : référencées et non définies
c définies 2 fois de suite
définition de b inutile

77

Critères basés sur le flot de données

dr-chaîne

r.. Variable a une valeur indéfinie lors de sa 1e utilisation

...dd... 2 définitions consécutives, la 1e est inutile

...d dernière définition inutile

78

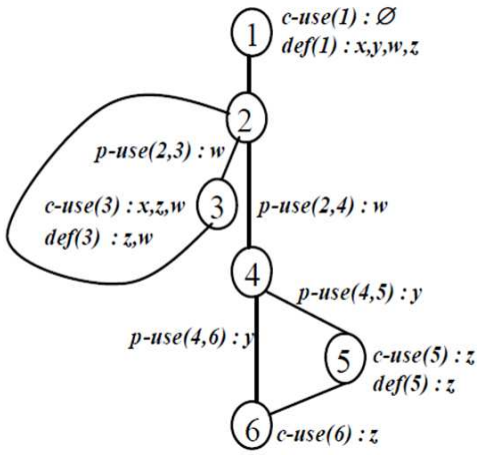
Critères basés sur le flot de données

Graphe Def/Use : exemple

```
double P(int x, int y) {
    w = abs(y) ;
    z  = 1.0 ;

    while ( w != 0 )
       {
         z  = z*x ;
         w = w -1 ;
       }

    if ( y<0 )
        z = 1.0 / z ;
  return(z) ;
  }
```



79

Critères toutes-les-définitions et tous-les-utilisateurs

- *toutes-les-définitions: pour chaque définition, il y a au moins un chemin dr-strict dans un test*

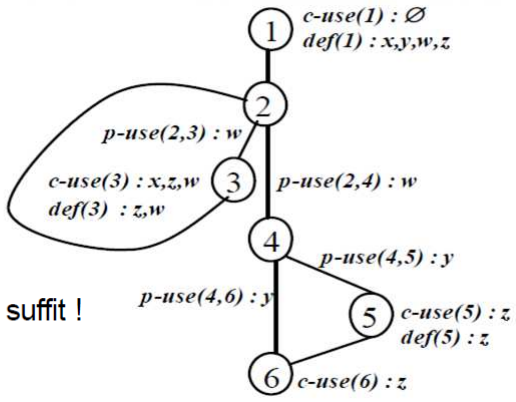
Critères « toutes les définitions » : Satisfait par un ensemble de chemins T si pour toute variable x, pour toute définition $d_B(x)$, il existe au moins une utilisation $u_{B'}(x)$ telle qu'il existe un chemin qui contient BCB' dans T, où C est sans redéfinition de x

$\forall\ x, \forall\ d_B(x), \exists\ u_{B'}(x), \exists$ BCB'

80

Critères toutes-les-définitions et tous-les-utilisateurs -exemple 1

Couverture de *toutes_les_définitions*



Ici, un seul chemin suffit !

1-2-3-2-4-5-6

81

Critères toutes-les-définitions et tous-les-utilisateurs

- *tous-les-utilisateurs: pour chaque définition et pour chaque référence accessible à partir de cette définition, couverture de tous les utilisateurs (noeuds c-utilisateurs ou arcs p-utilisateurs)*

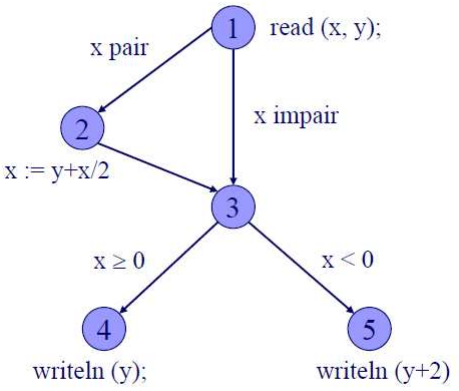
Critères « toutes les utilisations » : Satisfait par un ensemble de chemins T si pour toute variable x, pour toute définition $d_B(x)$, pour toute utilisation $u_{B'}(x)$ atteinte par $d_B(x)$, pour tout successeur B'' de B', il existe un chemin qui contient BCB'B'' dans T, où C est sans redéfinition de x

$$\forall\ x,\ \forall\ d_B(x),\ \forall\ u_{B'}(x),\ \forall\ B'',\ \exists\ BCB'B''$$

tous-les-utilisateurs ➔ *toutes-les-définitions*

82

Critères toutes-les-définitions et tous-les-utilisateurs -exemple 3



Couverture du critère toutes-les-définitions :

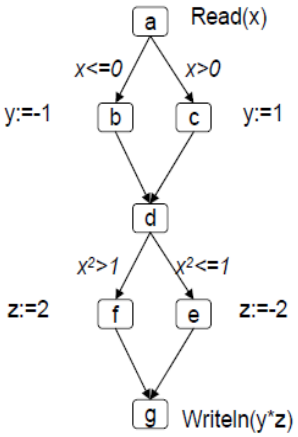
[1,3,5]
[1,2,3,5]

Couverture du critère tous-les-utilisateurs :

[1,3,4]
[1,2,3,4]
[1,3,5]
[1,2,3,5]

83

Critères toutes-les-définitions et tous-les-utilisateurs -exemple



- Couverture du critère toutes-les-définitions :

  [abdfg] [acdeg]

- Couverture du critère tous-les-utilisateurs :

  [abdfg] [acdeg]

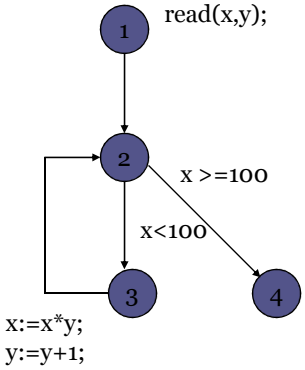
84

Autres critères basés sur le flot de données

- Le critère ***tous-les-utilisateurs*** *nécessite souvent la sensibilisation d'un grand nombre de chemin, deux autres critères, intermédiaires entre tous-les-utilisateurs et toutes-les-définitions sont proposés :*
  - ***tous-les-p-utilisateurs/quelques-c-utilisateurs****: pour chaque définition, et pour chaque p-utilisation accessible à partir de cette définition et pour chaque branche issue de la condition associée, il y a un chemin dr-strict prolongé par le premier segment de cette branche ; s'il n'y a aucune p-utilisation, il suffit d'avoir un chemin dr-strict entre la définition et l'une des c-utilisation,*
  - ***tous-les-c-utilisateurs/quelques-p-utilisateurs****: pour chaque définition, et pour chaque c-utilisation accessible à partir de cette définition, il y a un chemin dr-strict ; s'il n'y a aucune c-utilisation, il suffit d'avoir un chemin dr-strict entre la définition et l'une des p-utilisation.*

C-utilisation et p-utilisation

Chemin d ’utilisation (c-utilisation ou p-utilisation) :
chemin reliant l ’instruction de définition d ’une variable à une instruction utilisatrice.



[1,2,3,2,4] couvre tous les arcs

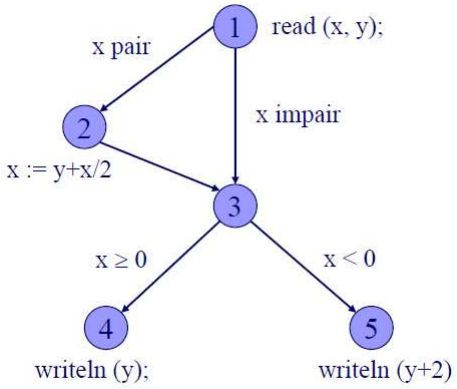
L ’arc (2, 4) est p-utilisateur de x par rapport au nœud 1
Or le chemin de p-utilisation [1,2,4] qui relie la définition de x au nœud 1 avec son arc p-utilisateur (2,4) n ’est pas inclus dans le chemin de test initial.

P-utilisation et c-utilisation

Le critère tous-les-p-utilisateurs nécessite que tous les arcs p-utilisateurs correspondant à toutes les définitions du graphe (affectation, read, etc.) soient couvertes, par un chemin de p-utilisation.

87

Critère *tous-les-p-utilisateurs/quelques-c-utilisateurs -exemple*

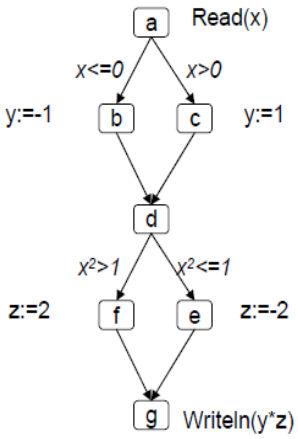


Couverture du critère tous-les-p-utilisateurs/ quelques-c-utilisateurs :

[1,3,4]
[1,2,3,5]

88

Critères toutes-les-définitions et tous-les-utilisateurs -exemple



- Couverture du critère tous-les-utilisateurs :

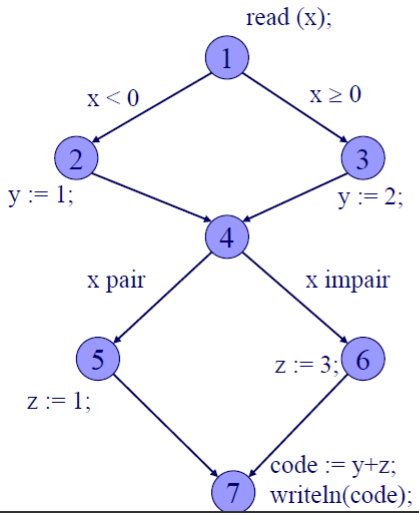
[abdfg] [acdeg]

Remarque : Ces 2 tests ne couvrent pas tous les chemins d'utilisations : si on rajoute au critère tous-les-utilisateurs le fait qu'on doit couvrir tous les chemins possibles entre définition et référence (en se limitant aux chemins sans cycle) on obtient le critère tous-les-du-utilisateurs

- Couverture du critère tous-les-du-utilisateurs :

[abdfg] [abdeg] [acdfg] [acdeg]

Limite du critère *tous-les-utilisateurs*



Couverture du critère tous-les-utilisateurs :

[1,2,4,5,7]
[1,3,4,6,7]

Ces deux tests ne couvrent pas tous les chemins d'utilisation :
(7) peut être utilisatrice de (2) pour la variable y

=> critère *tous-les-du-chemins*

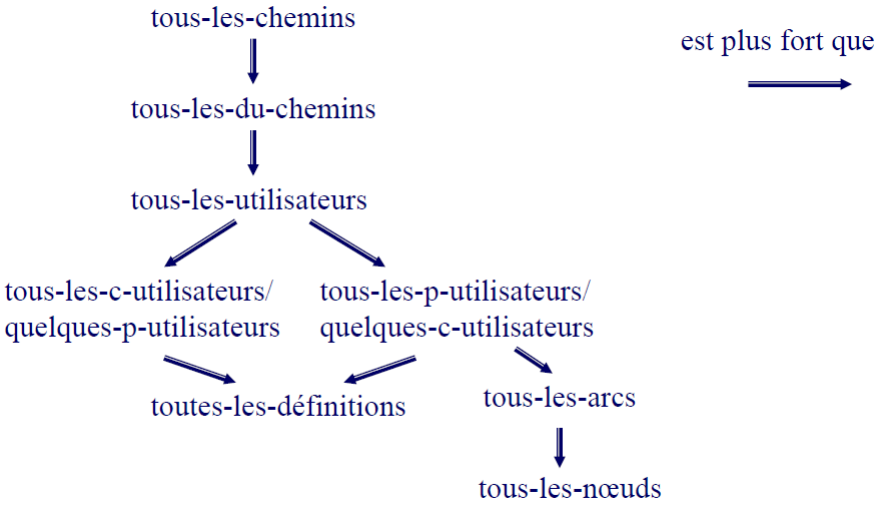
Critère *tous-les-du-chemins*

- Ce critère rajoute au critère *tous-les-utilisateurs* le fait qu'on doit couvrir tous les chemins possibles entre la définition et la référence, en se limitant aux chemins sans cycle.
- Sur l'exemple précédent, ce critère sensibilise :

[1,2,4,5,7]
[1,3,4,6,7]
[1,2,4,6,7]
[1,3,4,5,7]

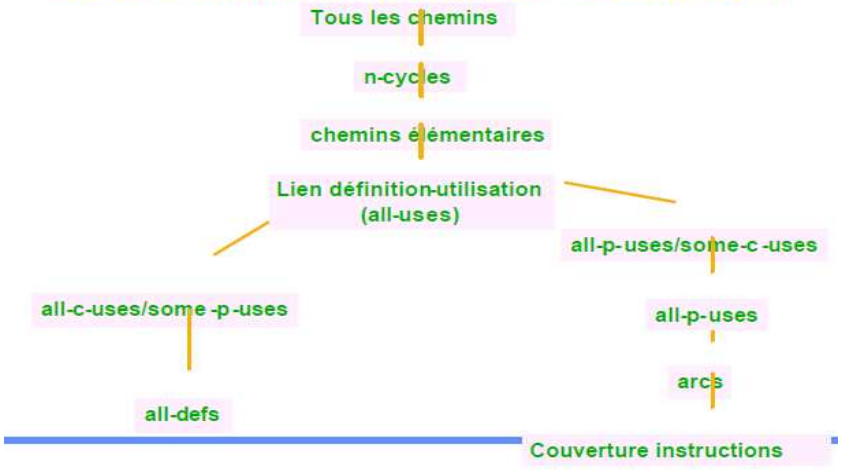
91

Hiérarchie des critères basés sur le flot de données



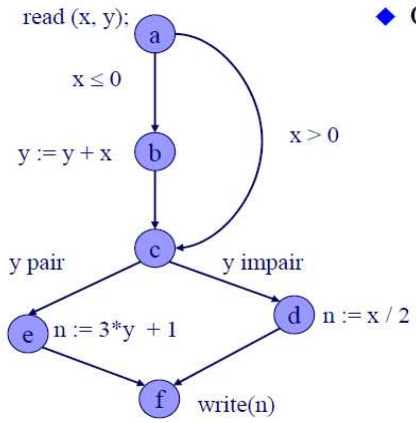
92

Le test unitaire structurel: classement



93

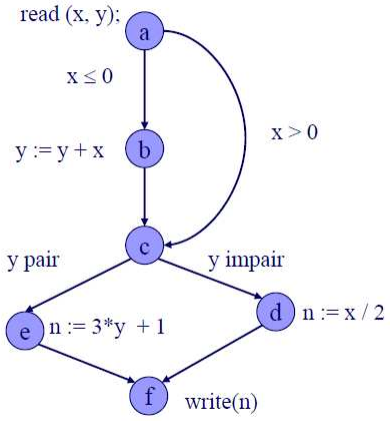
Méthodes de test structurel basés sur la couverture du flot de données –Exercice



- Critères de test :
  - *toutes-les-définitions*
  - *tous-les-utilisateurs*

94

Méthodes de test structurel basés sur la couverture du flot de données -Solution



- Critères de test :
  - *toutes-les-définitions*
    - » [a,c,d,f] : x=1 & y=3
    - » [a,b,c,e,f] : x=-1 & y=3
  - *tous-les-utilisateurs*
    - » [a,c,d,f] : x=1 & y=3
    - » [a,c,e,f] : x=-1 & y=3
    - » [a,b,c,e,f] : x=0 & y=2
    - » [a,b,c,d,f] : x=0 & y=3