Chapitre 3.
Test Structurel (Test Boite Blanche)

a.hettab@centre-univ-mila.dz

Introduction

- **Le test structurel ou test boîte blanche** se base sur **l'analyse** du **code source** de l'application (ou d'un modèle de celui-ci) pour établir les tests selon des **critères de couverture**. Il est utilisé pour vérifier que les instructions du code source de l'application sont correctement exécutées, les instructions conditionnelles et les boucles sont bien implémentées….etc.

Introduction

- **Les principaux critères de couverture** dans le test structurel sont:
  - Les critères de couverture basés sur **le graphe de flot de contrôle** (**toutes les instructions**, **toutes les branches**, **tous les chemins**, …)
  - Les critères de couverture basés sur **le graphe de flot de données** (**toutes les définitions** de variable, **toutes les utilisations**, …)
  - Les critères de couverture basés sur **les fautes** (**test mutationnel**)

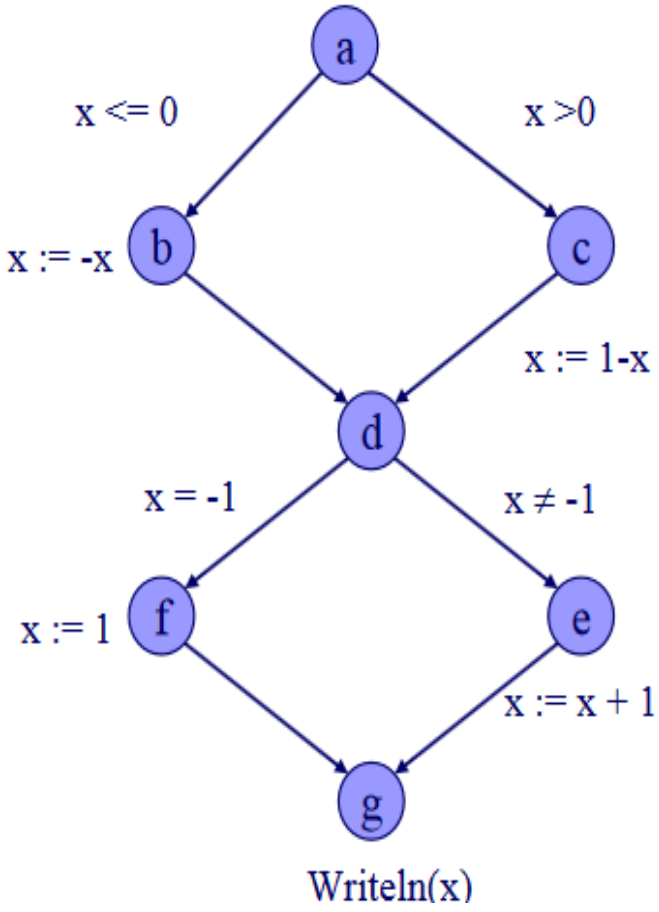
graphe de flot de contrôle

- Un **graphe de flot de contrôle** (**control flow graph** en anglais) est une représentation graphique de tous les chemins pouvant être suivis par un programme lors de son exécution.
- Un **graphe de flot de contrôle se compose** de plusieurs composants comme des nœuds, des arcs, des chemins, des points de décision, des points de jonctions…etc.

graphe de flot de contrôle

- **Exemple**: Soit le programme suivant:

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



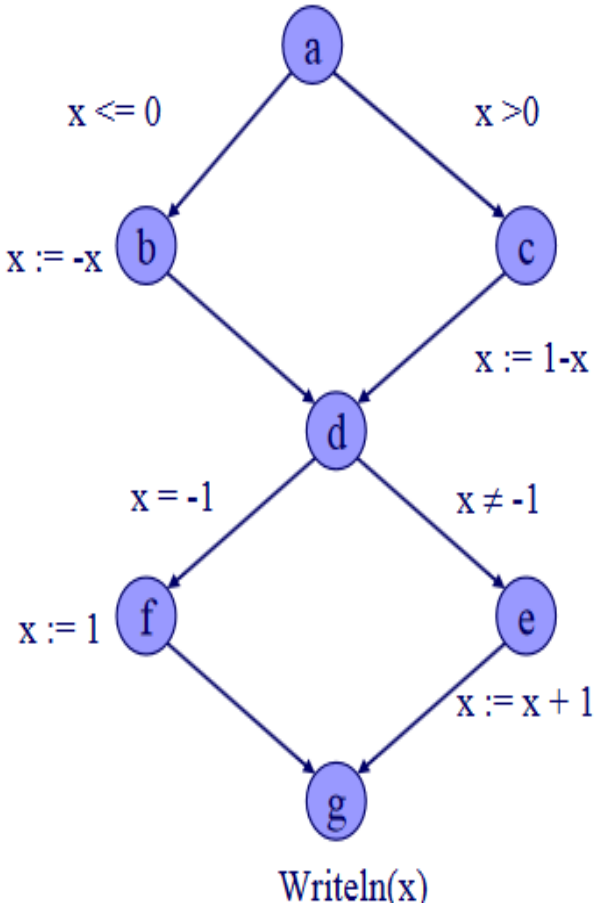
Le graphe de contrôle G1 correspondant

graphe de flot de contrôle

- **Exemple (suite)**:

Le graphe de contrôle est un graphe orienté et connexe **(N,A,e,s)** où:

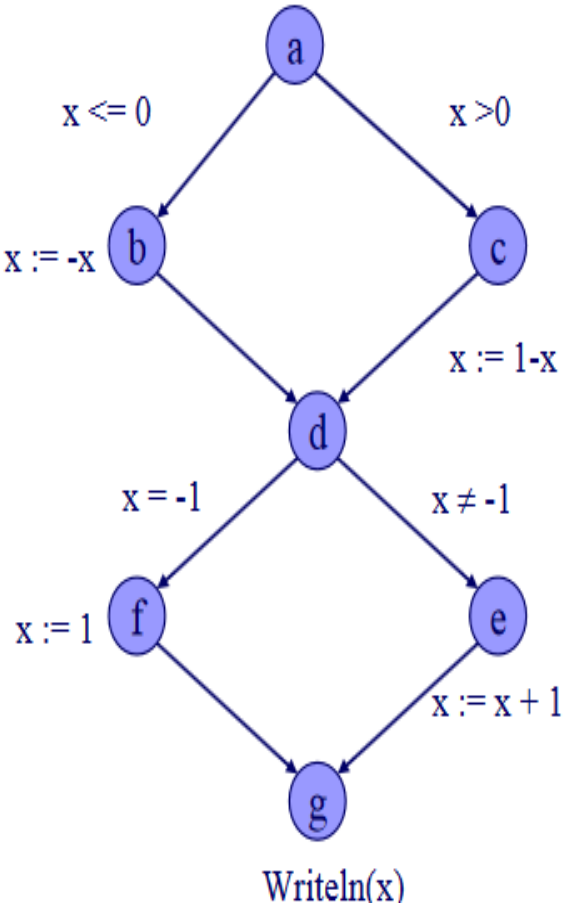
- **N**: un ensemble de nœuds
- **A**: un ensemble d'arcs
- **e**: un nœud entrée (**a**)
- **s**: un nœud sortie (**g**)
- **Un nœud** est un **bloc d'instructions**
- **Un arc** représente la possibilité de **transfert de l'exécution** d'un **nœud** à un **autre.**



G1

graphe de flot de contrôle

- **Un chemin de contrôle** dans le graphe de contrôle est une **exécution possible** qui commence par le **nœud d'entrée** et se termine par le **nœud de sortie**:
- **[a,c,d,e,g] est un chemin de contrôle**
- **[b, d, f, g] n'est pas un chemin de contrôle**
- Ce graphe **G1** comprend **4 chemins** de contrôle :
  - β1 = **[a, b, d, f, g]**
  - β2 = **[a, b, d, e, g]**
  - β3 = **[a, c, d, f, g]**
  - β4 = **[a, c, d, e, g]**



G1

graphe de flot de contrôle

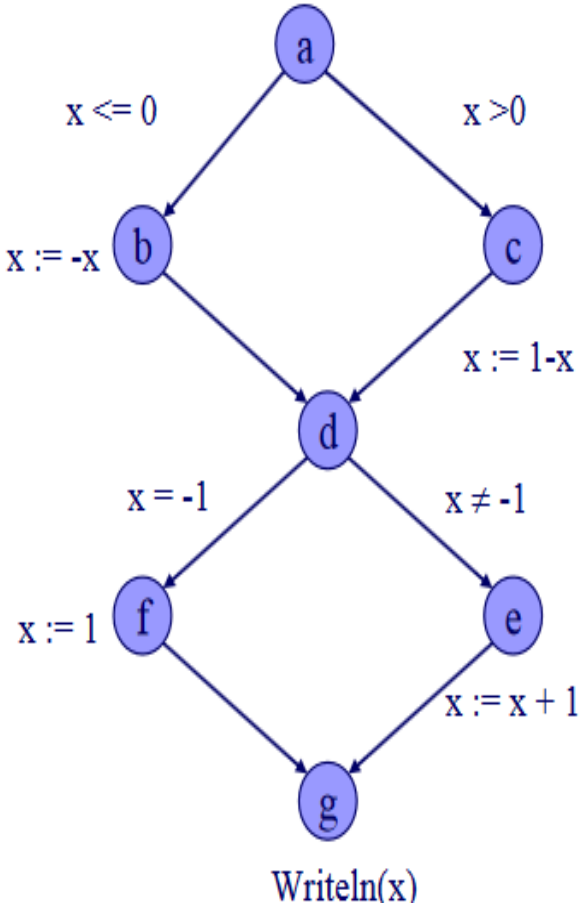
- **Exemple (suite)**:
- Le graphe de flot de contrôle peut-être exprimé sous forme algébrique sous la forme d'une **expression de chemins** :

  G1 = abdfg + abdeg + acdfg + acdeg

  le signe + désigne le « **ou** » **logique** entre chemins.
- La **simplification** de cette **expression de chemins** donne:

  G1 = a (bdf + bde + cdf + cde) g

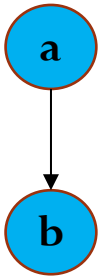
  G1 = a (b + c) d (e + f) g



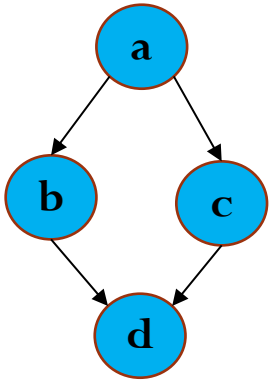
G1

graphe de flot de contrôle

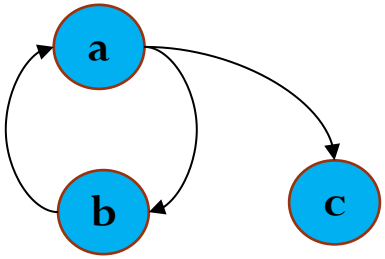
- **Construction de l'expression des chemins**:
- **l'expression des chemins** est la **composition** de toutes les expressions de chemins des **structures primitives** apparaissant dans le graphe de flot de contrôle. Pour chaque **structure primitive** on associe une opération **d'addition** ou de **multiplication** comme suit;



Forme séquentielle : ab



Forme alternative : a (b + c) d

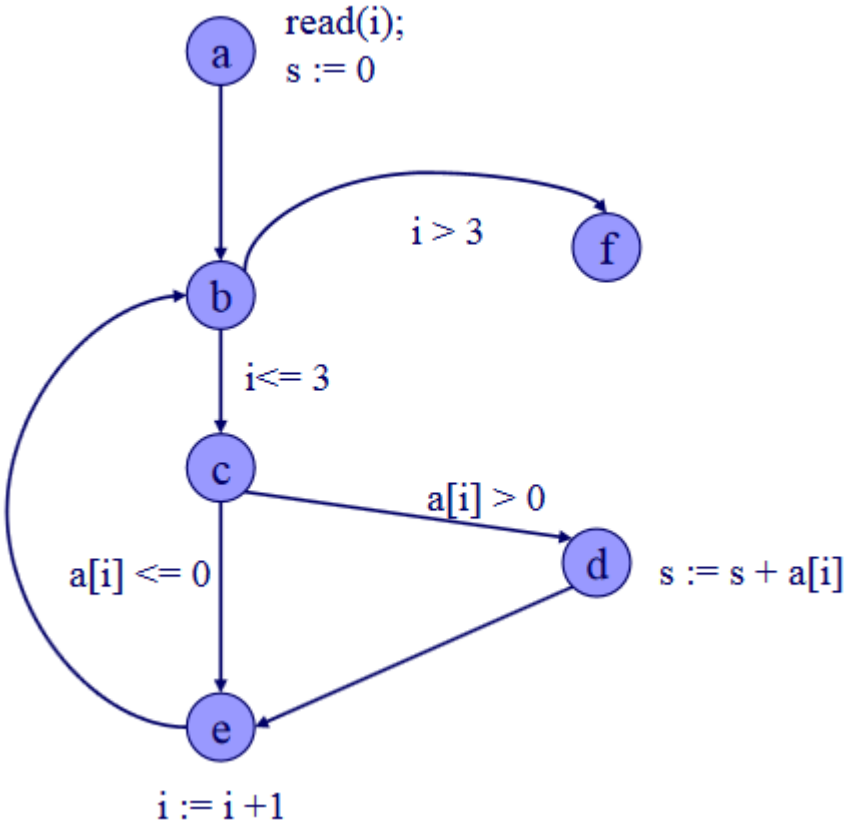


Forme itérative : ab (ab)* c

graphe de flot de contrôle

- **Expression des chemins (exemple)**:
- Soit le programme suivant :

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

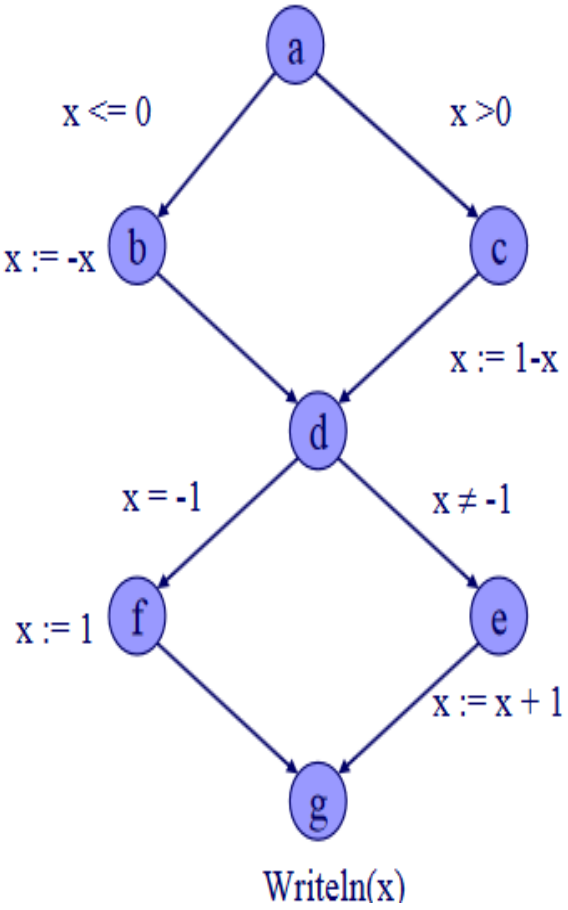


Le graphe de contrôle G2 correspondant

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

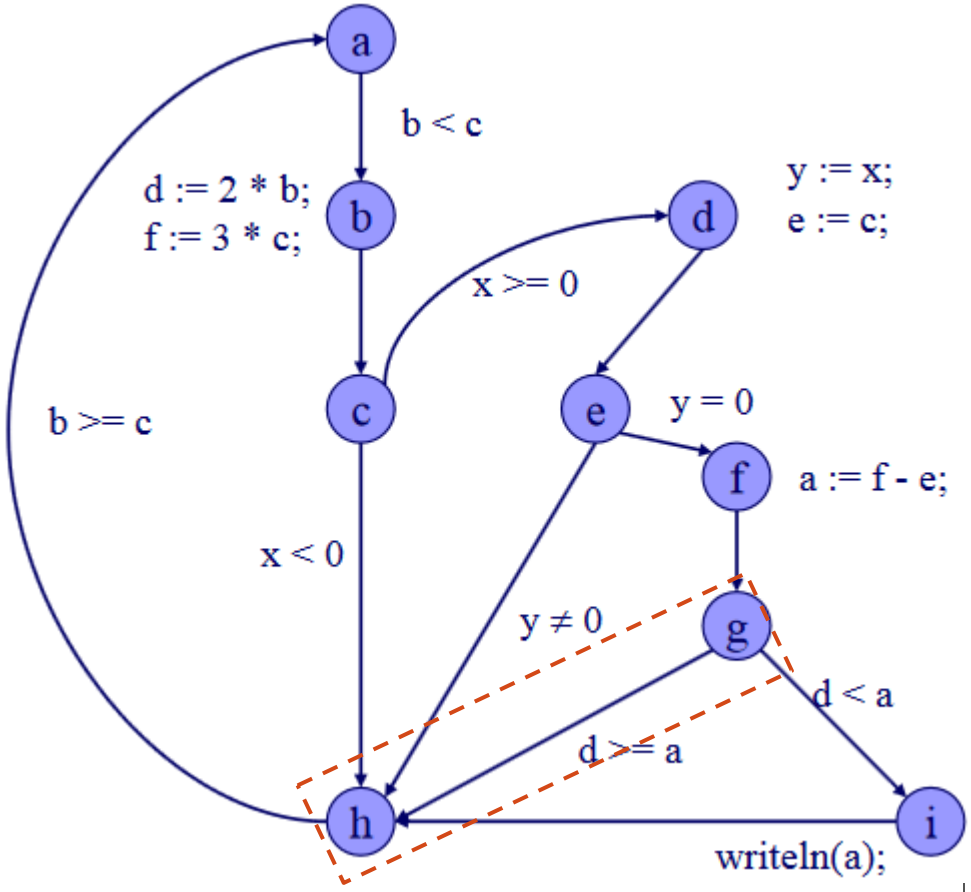
graphe de flot de contrôle

- **Chemin exécutable:**
- Un chemin exécutable est un chemin de contrôle **sensibilisé** par des **données de test (DT)**
- **DT1={x=2}** **sensibilise** le chemin **[acdfg]** alors **[acdfg]** est un **chemin exécutable**
- **[abdfg]** est un **chemin non exécutable** car **aucune DT** capable **de sensibiliser ce chemin.**



graphe de flot de contrôle

- **Problème des chemins non exécutables**
- **Sensibiliser** un chemin peut parfois être **difficile et voire impossible** car le problème de trouver des DT qui sensibilisent un chemin est **indécidable**. **L'existence de chemins non exécutables** est un signe de **mauvais codage**
- L 'arc **g-h n'est pas activable** :car on a toujours **2b < 2c** donc **d < a**



graphe de flot de contrôle

- **Satisfaction d'un test structurel avec couverture**
- Soit **T= {δ1 , … , δk}** un ensemble de chemins à couvrir.
- Soit **DT** un **ensemble de données de test**.
- **DT satisfait le test T si et seulement si :**

∀ δi ∈ T, ∃ d ∈ DT tel que l'exécution de **d** couvre **δi**.

Autrement dit: Un ensemble de données de test satisfait un test structurel si chaque chemin exigé par ce test est exécuté par au moins une donnée de test.

graphe de flot de contrôle

- **Satisfaction d'un test structurel avec couverture (suite)**
- **Exemple :** Soient le graphe de contrôle G3, **δ1 =cdebcde**, **δ2 =ce** et **T= {δ1 , δ2 }**
- **DT1 ={a[1]=-2, a[2]=3, a[3]=17,i=1} satisfait T car:**

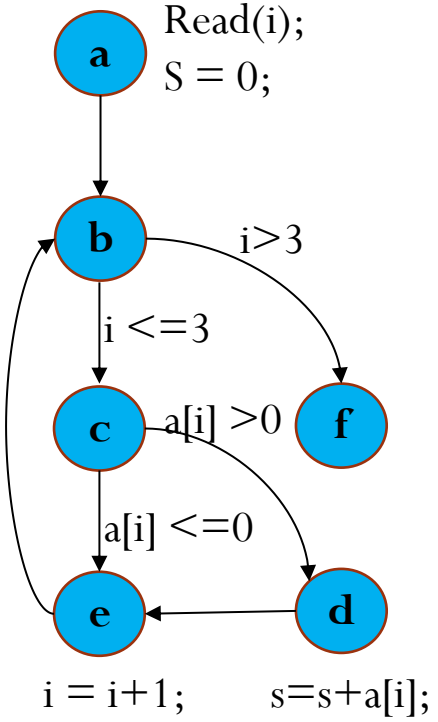
DT1 sensibilise M1= abcebcdebcdebf et

M1= abcebcdebcdebf couvre δ1 =cdebcde

M1=abcebcdebcdebf couvre δ2 =ce

- **DT2 ={a[1]=-2, a[2]= 3, a[3]=-17,i=1} ne satisfait pas T car**

DT2 sensibilise M2= abcebcdebcebf et M2 ne couvre pas δ1 = cdebcde

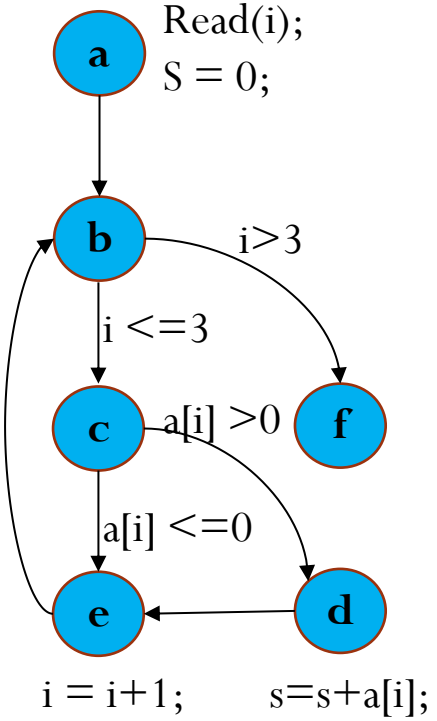


G3

graphe de flot de contrôle

- **Hiérarchie des techniques de test structurel**
- Il y a une **relation d'ordre partielle** (réflexive, antisymétrique, transitive) entre les test structurels s'appelle **relation de la Hiérarchie** et elle notée « ⇒ ».
- **T1 ⇒ T2** signifié que **T1 est un test plus fort que T2** (Tous les données de test qui satisfont **T2** doivent satisfaire **T1**), ou **T2 est plus fiable que T1**
- **Exemple:** Soient **δ1 =cdebcde, δ2 =ce δ3 =de, δ4 =b, δ5 =cd** sont des chemins de contrôle.
- **Et T1= {δ1 , δ2 } et T2 = {δ3 , δ4 , δ5 }** sont des tests

on dit que: **T1 ⇒T2**



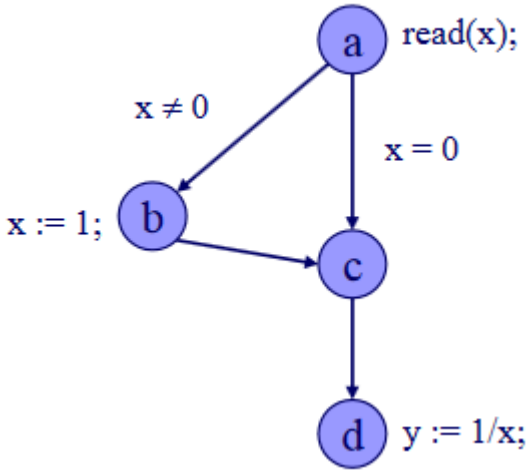
G3

Critères de couverture

- Il existe plusieurs types de critères de couverture dans le test structurel:
  - **Couverture du flot de contrôle** (toutes les instructions, branches, chemins…)
  - **Couverture du flot de données** (toutes les définitions de variables, les utilisations…)
  - **Test mutationnel** (test par injection de défaut)

Critères de couverture du flot de contrôle

- **1) couverture de tous-les-nœuds:** Ce critère de couverture assure que chaque nœud dans le graphe de contrôle est visité au moins une fois.
- **Taux de couverture : = nb de nœuds couverts/ nb total de nœuds**
- **Exemple:**
- Soit le graphe de contrôle **G4.** le chemin **[abcd]** satisfait le critère de couverture de tous-les-nœuds.

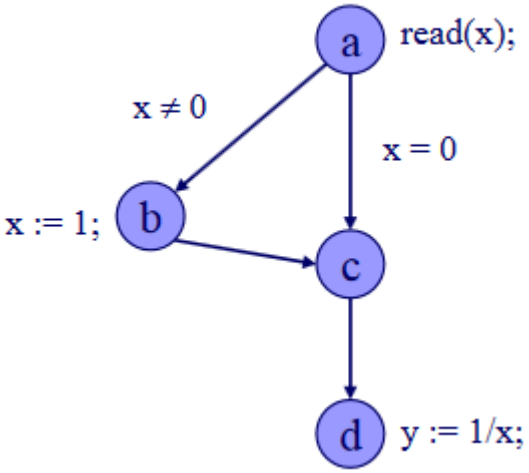


G4

Limites: Ce critère ne garantit pas la couverture de tous les arcs. Ainsi, certaines erreurs liées à des arcs spécifiques (par exemple une division par 0 sur l'arc $a \rightarrow c$) peuvent ne pas être détectées, même si tous les nœuds sont couverts.

Critères de couverture du flot de contrôle

- **2) couverture de tous-les-arcs:** Ce critère de couverture assure que chaque arc dans le graphe de contrôle est visité au moins une fois.
- **Taux de couverture : = nb d'arcs couverts/ nb total d' arcs**
- **Exemple:**
- Soit le graphe de contrôle **G4.** le chemin **[abcd+acd]** satisfait le critère de couverture de tous-les-arcs.

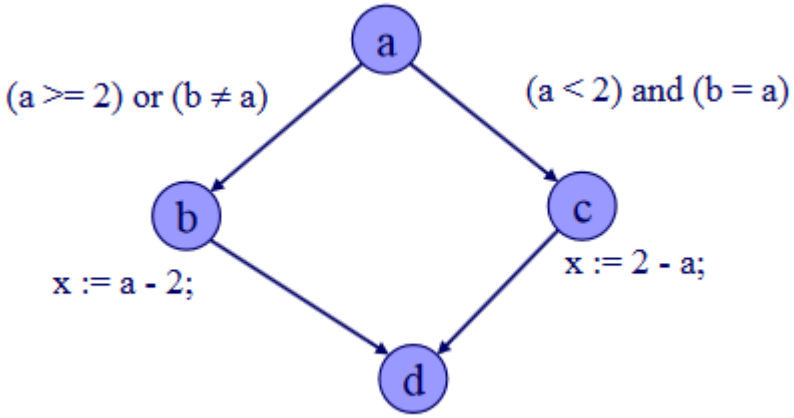


G4

Remarque: Lorsque **le critère tous-les-arcs** est totalement réalisé, cela implique que **le critère tous-les-nœuds est Satisfait** et on écrit : **critère tous-les-arcs ⇒ critère tous-les-nœuds** .

Critères de couverture du flot de contrôle

- **2) couverture de tous-les-arcs (suite):**
- **Limites:** Le critère de couverture de tous-les-arcs ne couvre pas toutes les **combinaisons de conditions possibles**.
- **Exemple:** Soit le graphe de contrôle **G5.**
- Le jeu de test **DT1 = {a=b=1}**



G5

sensibilise la chemin M1= acd, et **DT2 = {a=b=3} sensibilise le chemin M2 = abd. Alors DT1 + DT2 satisfont le critère de tous-les-arcs**

mais ils **ne couvrirent pas** toutes les **combinaisons de conditions possibles (le cas de b≠a)** qui est satisfait par **DT3 = {a=3, b=2}**.

Critères de couverture du flot de contrôle

- **3) couverture de toutes les condition-décision multiple:** Le critère de couverture de **toutes les condition-décision multiple** assure que chaque **sous-expression** dans les conditions prend **toutes les combinaisons de valeurs possibles**.
- **Exemple:** pour l'expression « **A ou B** » il faut tester toutes les 4 combinaisons suivantes: **A = B = vrai**, **A = B = faux**, **A=vrai, b=faux ,** et **A=faux et B=vrai.**

Critère de couverture de toutes les condition-décision multiple ⇒ critère tous-les-arcs.

limites: Problème de l'explosion combinatoire avec les expressions logiques complexes.

Critères de couverture du flot de contrôle

- **4) couverture de tous-les-chemins-indépendants:**
- La couverture de tous-les-chemins-indépendants exige que **tous les chemins indépendants du graphe de contrôle** soient exécutés au moins une fois.
- Un chemin est **indépendant** s'il introduit au moins un nouvel arc ou une nouvelle combinaison de nœuds par rapport aux chemins déjà testés.
- Pour les boucles, cela implique de tester : le corps de la boucle 0 fois, 1 fois, et plusieurs fois jusqu'à la sortie de la boucle.

Critères de couverture du flot de contrôle

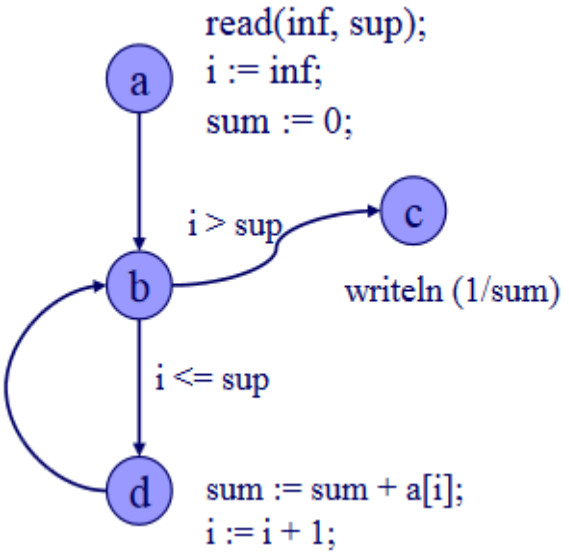
- **4) couverture de tous-les-chemins-indépendants (suite):**
- **Exemple**: Soit le graphe de contrôle **G6**.
- **DT1** = {a[1]=50, a[2]=60, a[3]=80, inf=1, sup=3}

→ couvre le chemin **M1 = abdbdbdbc** (boucle exécutée plusieurs fois)

- **DT2** = {i=3, sup=2}

→ couvre le chemin **M2 = abc** (boucle non exécutée)

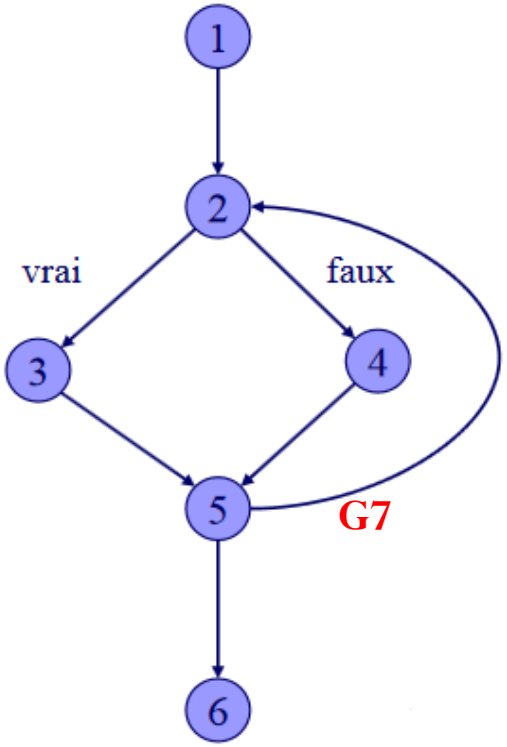
- Ensemble, **DT1 + DT2 satisfont le critère de tous-les-chemins-indépendants**.



G6

Critères de couverture du flot de contrôle

- **4) couverture de tous-les-chemins-indépendants (suite):**
- **Chemins limites:** permettent de tester les scénarios où la boucle **ne s'exécute pas**.
- **Chemins intérieurs:** permettent de tester les scénarios où la boucle **s'exécute plusieurs fois.**
- **Exemple:** soit le graphe de contrôle G7.
- **Chemins limites : [1,2,3,5,6], [1,2,4,5,6]**
- **Chemins intérieurs : [1,2,3,5,2,3,5,6],**
- **[1,2,3,5,2,4,5,6], [1,2,4,5,2,3,5,6], [1,2,4,5,2,4,5,6]**



Critères de couverture du flot de contrôle

- **4) couverture de tous-les-chemins-indépendants (suite):**
- **Critère tous-les-chemins-indépendants ⇒ critère tous-les-arcs ⇒ critère tous-les-nœuds.**

•Limites:

•Ce critère **ne garantit pas la couverture de toutes les itérations possibles d'une boucle**, surtout si une boucle peut s'exécuter un nombre important ou infini de fois.

•Pour les boucles, il est souvent nécessaire d'utiliser **le critère tous-les-i-chemins** pour limiter le nombre d'itérations testées.

Critères de couverture du flot de contrôle

- **5) couverture de tous-les-i-chemins:**
- La couverture de tous-les-i-chemins consiste à tester **tous les chemins possibles dans un graphe de contrôle en limitant le nombre d'itérations dans les boucles à i passages maximum**.
- Cela rend le test des chemins réalisable, alors que tester tous les chemins réels serait impossible en raison de la présence de boucles (nombre infini de chemins).
- **Remarque:** en pratique, **i** est souvent choisi entre **0 et 2** pour couvrir la boucle non exécutée, une seule fois, et plusieurs fois.

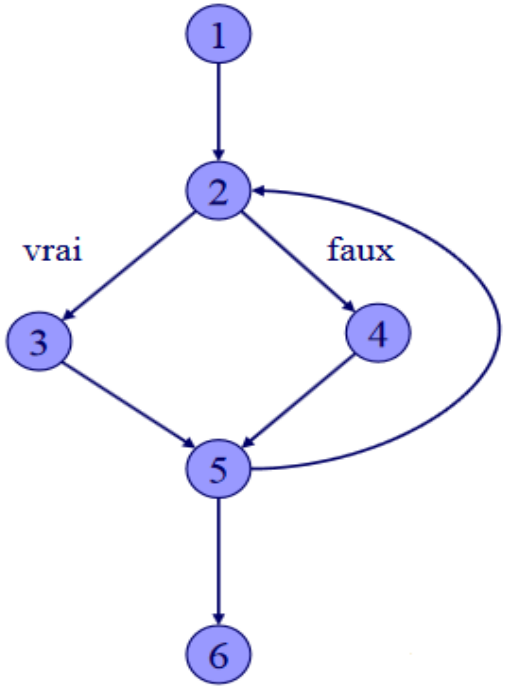
Critères de couverture du flot de contrôle

- **5) couverture de tous-les-i-chemins(suite):**
- **Exemple**: soit le graphe de contrôle G7.
- **tous-les-0-chemins :** [1,2,3,5,6], [1,2,4,5,6]
- **tous-les-1-chemins :**[1,2,3,5,6], [1,2,4,5,6],

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

- **tous-les-2-chemins :**

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



G7

Critères de couverture du flot de contrôle

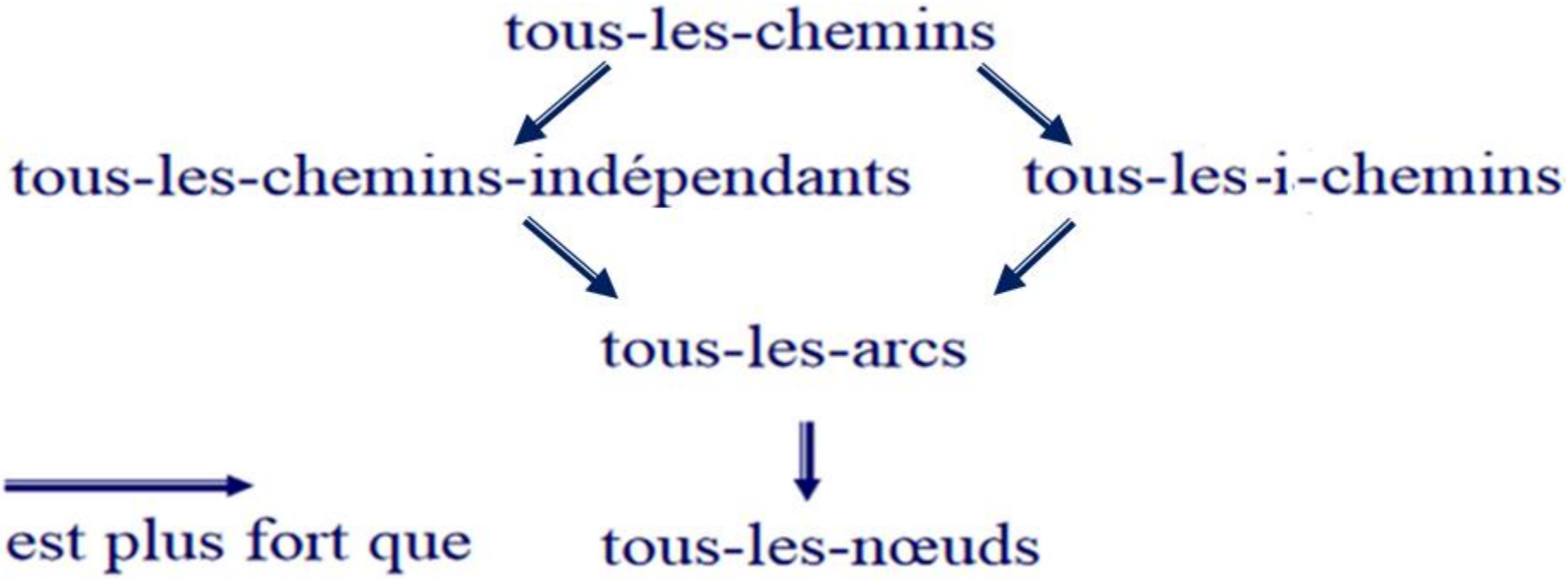
- **6) couverture de tous les chemins:**
- Ce critère exige **l'exécution de tous les chemins possibles** dans le graphe de contrôle.
- Inclut **toutes les séquences possibles de nœuds et d'arcs**, y compris **toutes les itérations de boucles** et toutes les combinaisons de décisions.

Limite principale :

- **Impraticable pour les programmes avec des boucles**, car une boucle peut générer un **nombre infini de chemins**.
- Même pour des programmes sans boucle, le nombre de chemins peut croître **exponentiellement** avec le nombre de décisions imbriquées.

Critères de couverture du flot de contrôle

- **Hiérarchie des critères basés sur le flot de contrôle**



Critères de couverture du flot de données

- Les critères de couverture basés sur le **flot de données** analyse les **relations entre les instructions** du programme en tenant compte **les données de test en fonction des définitions et des utilisations des variables**.
  - **Variable définie:** Dans une instruction, si la valeur d'une variable est **modifiée (affectations),** on dit que cette **variable est définie**.
  - **Variable utilisée:** Dans une instruction, si la valeur d'une variable **est utilisée**, on dit que cette **variable est utilisée ou référencée**.
    - Si la valeur de la variable utilisée est utilisée pour **un calcul**, on parle de **c-utilisation**
    - Si la valeur de la variable utilisée est utilisée dans un prédicat d'une instruction de **décision** (if, while, …), on parle de **p-utilisation**

Critères de couverture du flot de données

- **Exemple**: soit le programme suivant:

```
while (i < N)  do        i et N : p-utilisées
  begin
    s := s + i;          s et i : c-utilisées     s: définie
    i := i + 1;          i : c-utilisées          i: définie
  end;
  writeln (s);           s : c-utilisées
```

Critères de couverture du flot de données

- **L'instruction utilisatrice:** Une instruction **I** est **utilisatrice** d'une **variable x** par rapport à une instruction **J** si **x** a été définie en **I** et directement référencé en **J (x** n'est pas redéfinie entre **I** et **J).**

- **Exemple:** soit le programme suivant:

(1) x := 7;

(2) a := x+2; **(2)** est **c-utilisatrice** de **(1)** pour la variable **x**

(3) b := a*a; **(3)** est **c-utilisatrice** de **(2)** pour la variable **a**

(4) a := a+1; **(4)** est **c-utilisatrice** de **(2)** pour la variable **a**

(5) if (b > a) **(5)** est **p-utilisatrice** de **(3) et (4)** pour les variables **b** et **a**

(6) y := x + a; **(6)** est **c-utilisatrice** de **(1) et (4)** pour les variables **x** et **a**

Critères de couverture du flot de données

- **Un chemin d'utilisation:** on dit chemin d'utilisation (**c-utilisation ou p-utilisation**) le chemin qui relie l'instruction de définition et les instructions utilisatrices d'une variable donnée.

- **Exemple**: reprend l'exemple précèdent:

(1) x := 7; **[1,2,6]** est un **chemin c-utilisation** pour la variable **x**

(2) a := x+2; **[2,3,4,5,6]** est un **chemin p/c-utilisation** pour la variable **a**

(3) b := a*a; **[3,5]** est un **chemin p-utilisation** pour la variable **b**

(4) a := a+1;

(5) if (b > a) **[4,5]** sont des **chemins p-utilisation** pour **a**

(6) y := x + a;

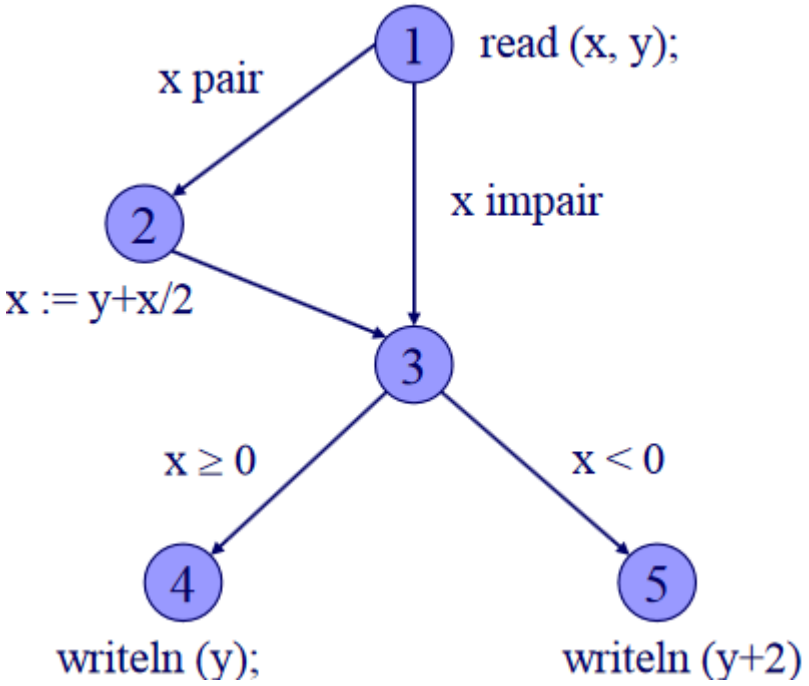
Critères de couverture du flot de données

- **1) toutes-les-définitions:** Le critère de couverture de **toutes-les-définitions** assure au moins la couverture d'un chemin d'utilisation pour chaque définition du graphe.
- **Exemple**: soit le graphe de contrôle **G8**

Le chemin **[1,2,3,5] couvre**

Le critère de couverture de

toutes-les-définitions



G8

Critères de couverture du flot de données

- **2) tous-les-utilisateurs:** Le critère de couverture de **tous-les-utilisateurs** assure au moins la couverture de tous les utilisateurs (nœuds c-utilisateurs ou arcs p-utilisateurs) pour chaque définition

et pour chaque référence accessible à partir

de cette définition. [tous-les-p-utilisateurs

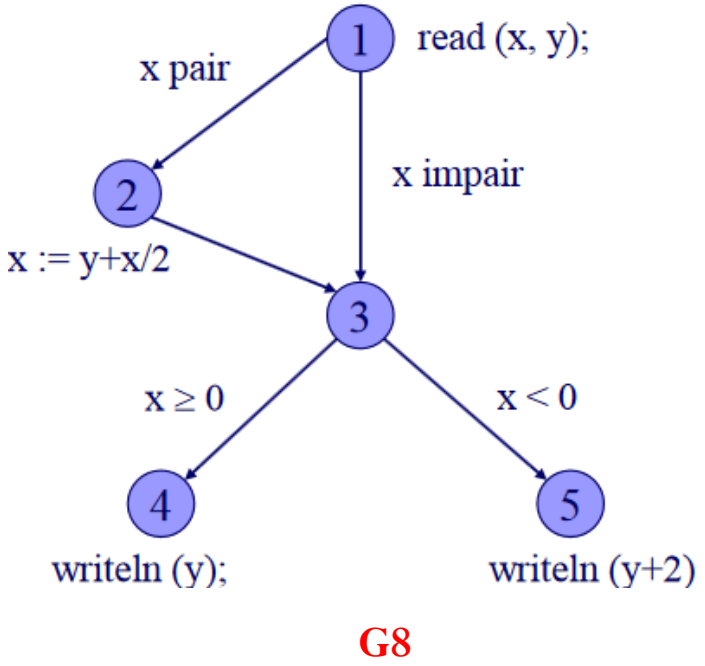
et tous-les-c-utilisateurs].

- **Exemple**: soit le graphe de contrôle **G8**

Les chemins **[1,2,3,4]** et **[1,2,3,5], [1,3,4]**

et [1,3,5] couvre le critère de couverture

de tous-les-utilisateurs



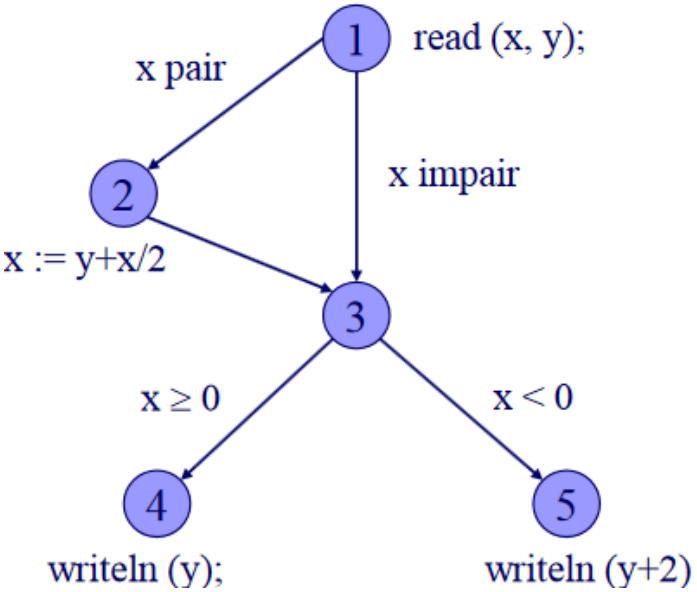
G8

Critères de couverture du flot de données

- **2) tous-les-utilisateurs (suite):** Le critère de couverture de **tous-les-utilisateurs** se compose de deux critères suivants **tous-les-c-utilisateurs et tous-les-p-utilisateurs**.
- **Exemple**: soit le graphe de contrôle **G8**
- Les chemins **[1,2,3,4], [1,2,3,5], [1,3,4]**

et **[1,3,5] couvre le critère de couverture de tous-les-p-utilisateurs**

- Les chemins **[1,2,3,4]** et **[1,2,3,5] couvre le critère de couverture de tous-les-c-utilisateurs**



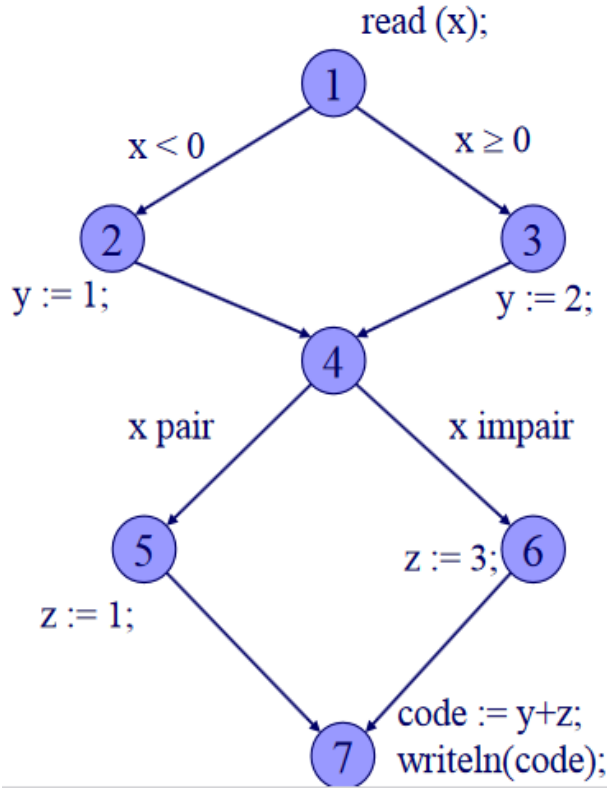
G8

Critères de couverture du flot de données

- **3) tous-les-du-chemins (tous-les-du-utilisateurs):** assure au moins la couverture de tous les chemins sans cycle possibles entre la définition et la référence (l'utilisation).

- **Exemple**: soit le graphe de contrôle **G9**
- Les chemins **[1,2,4,5,7], [1,2,4,6,7], [1,3,4,5,7]** et **[1,3,4,6,7] couvre le critère de couverture de tous-les-du-chemins (tous-les-du-utilisateurs)**
- Les chemins **[1,2,4,5,7]** et **[1,3,4,6,7] couvre le critère de couverture de tous-les-utilisateurs.**

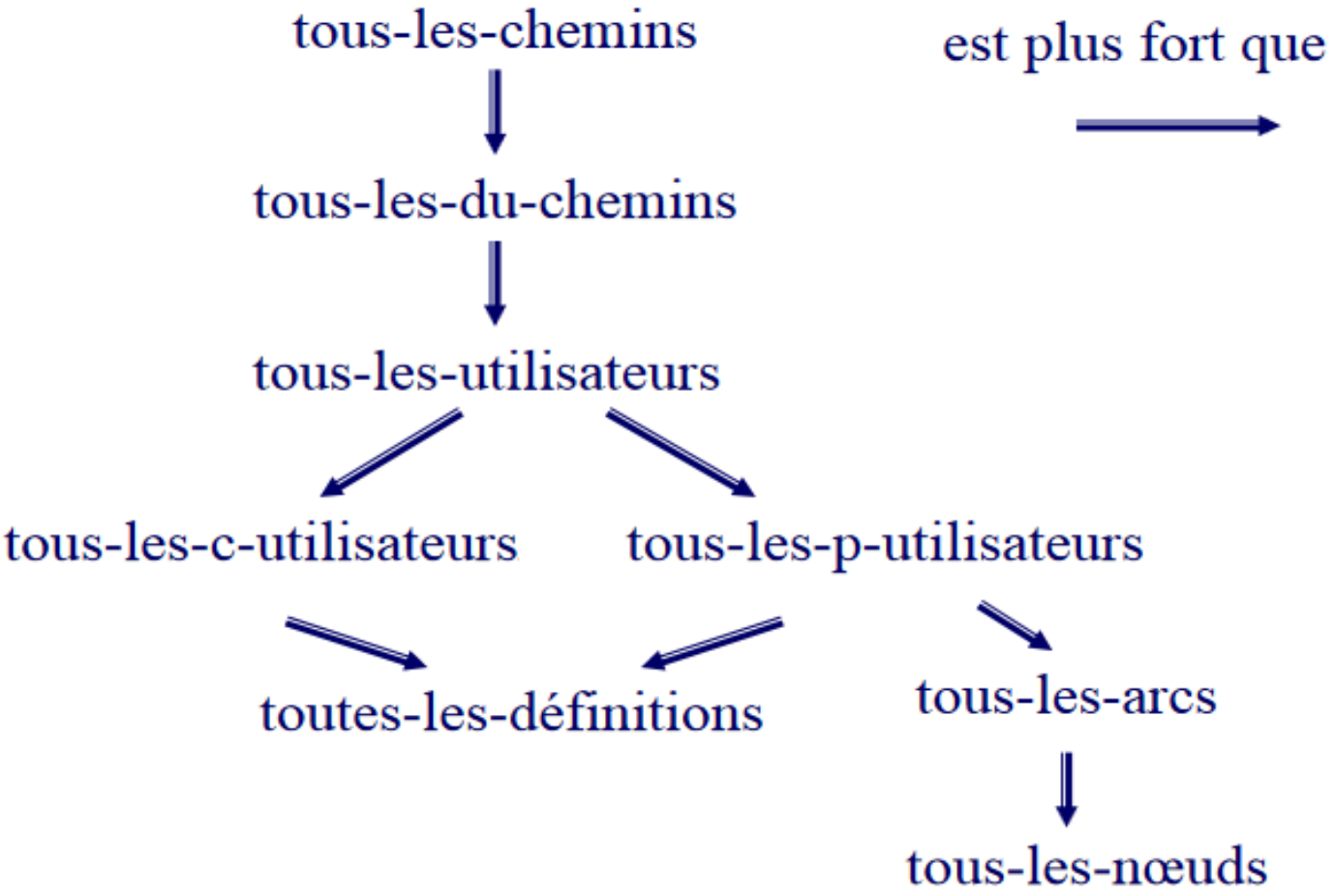
Remarque: Ces deux tests **ne garantissent pas** la **couverture de tous les chemins d'utilisation**, car **(7)** peut **c-utilisatrice** à la fois de **(2)** pour la variable **y** et de **(6)** pour la variable **z**.



G9

Critères de couverture du flot de données

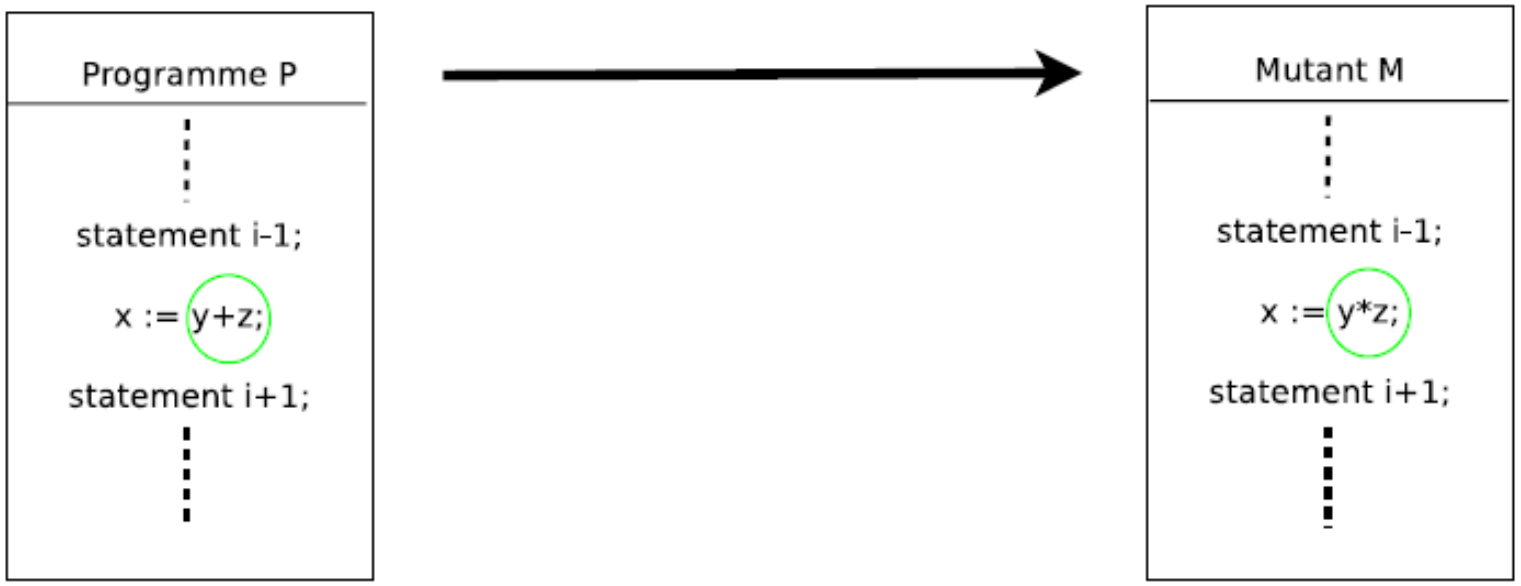
- **Hiérarchie des critères basés sur le flot de données**



Test de mutation

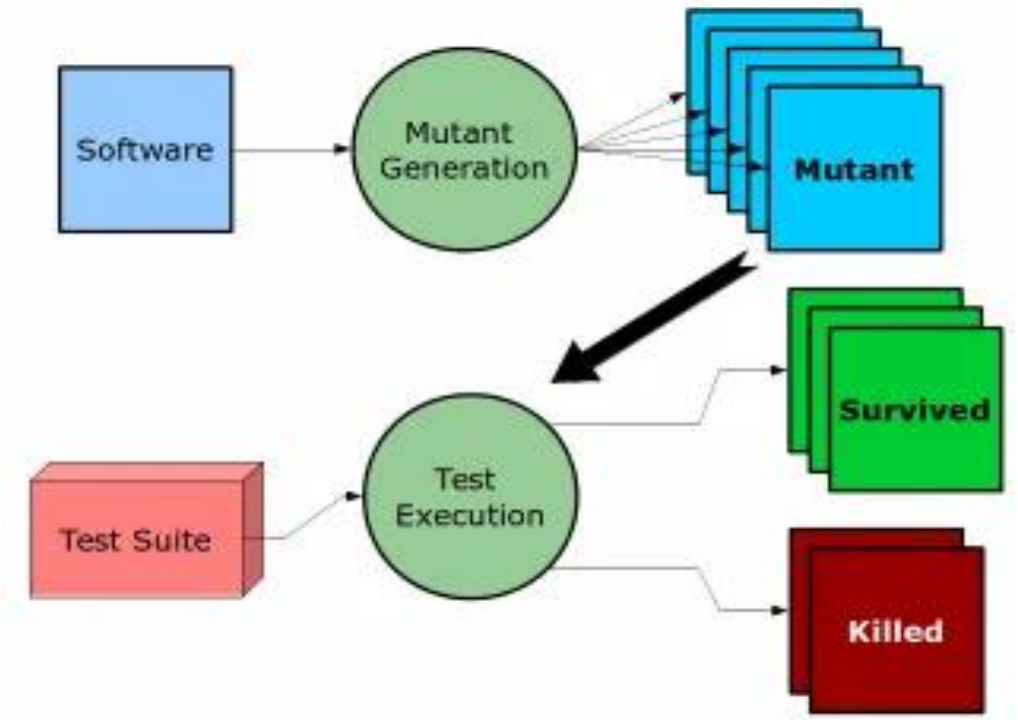
- **Le test de mutation** est un type de test logiciel utilisé pour **évaluer la qualité des tests** des logiciels.
- Pour ce faire, on implique la **modification** d'un programme de **petites manières**, et on vérifie si les données de test sont capables de **détecter** les erreurs.
- Chaque version mutée est appelée « **mutant** », et les tests détectent et rejettent les mutants en faisant en sorte que le comportement de la version originale diffère de celui du mutant, cela s'appelle **tuer le mutant**.

Test de mutation



Test de mutation

- Les suites de tests sont mesurées en calculant **le score de mutation** qui défini comme le **pourcentage de mutants tués avec le nombre total de mutants**.
- **le score de mutation = nombre de mutants tués / nombre total de mutants * 100**



Test de mutation

- **Opérateurs de mutation**
- **Les mutants** sont basés sur **des opérateurs de mutation** bien définis qui imitent soit des **erreurs** de programmation typiques (comme utiliser un opérateur ou un nom de variable incorrect), soit imposent la création de tests utiles (tels que la division de chaque expression par zéro). Les opérateurs de mutation les plus répandus sont les opérateurs de mutation traditionnels qui sont conçus pour les langages impératifs comme:

Test de mutation

- **Opérateurs de mutation (suite)**

- La suppression des expressions ;

- Le changement d'associativité des termes ;

- La duplication ou l'insertion d'instructions ;

- La négation des expressions booléennes ;

- Le remplacement de certaines opérations arithmétiques par d'autres, (+ par *, - par /) ;

- Le remplacement de certaines relations booléennes par d'autres (> par > =, == par <=) ;

- Le remplacement de variables par d'autres variables ;

- Le remplacement de variables par des constantes ;

Test de mutation

- **Outils de mutation**
- Il existe de nombreux **outils de mutation**, et chaque outil est dédié à un langage de programmation spécifique.

- Pour le langage **JAVA** il y a: ***MuJava***, ***PIT***, ***LittleDarwin*** et ***MAJOR***
- Pour le langage **C/C++** il y a: ***Mull*** , ***Mutate CPP***, ***AccMut***, ***Srciror*** et ***MUSIC*** .
- Pour le langage **Python** il y a: ***Mutpy***, ***Mutmut***, ***XMutant*** et ***Cosmic Ray***.

Test de mutation

- **Avantages du test de mutation**
- Les tests de mutation ont plusieurs avantages à savoir:
- Ils sont **puissants** pour atteindre une **couverture élevée** du programme source.
- Ils sont également capables de tester de **manière exhaustive** le programme sous test.
- Les tests de mutation apportent aussi un **bon niveau de détection des erreurs** aux développeurs de logiciels et révèlent les ambiguïtés du code source et a la capacité de détecter tous les défauts du programme.
- Par conséquent, les clients bénéficient de ces tests en obtenant un système **extrêmement fiable et stable**.

Test de mutation

- **Inconvénients du test de mutation**
- **l'inconvénient majeur** des tests de mutation est le **coût élevé et le temps perdu,** car de **nombreux** mutants doivent être générés. En plus, chaque mutant aura le même nombre de cas de test que celui du programme d'origine, ainsi, un grand nombre de programmes mutants devront peut-être testés par rapport à la suite de tests d'origine. Par conséquent, ces tests ne peuvent pas être effectués sans utilisation d'un outil d'automatisation.