

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

Principe : à partir du code source (ou d'un modèle) et spécification, produire des DT qui exécuteront un ensemble de comportements, comparer les résultats avec ceux attendus...

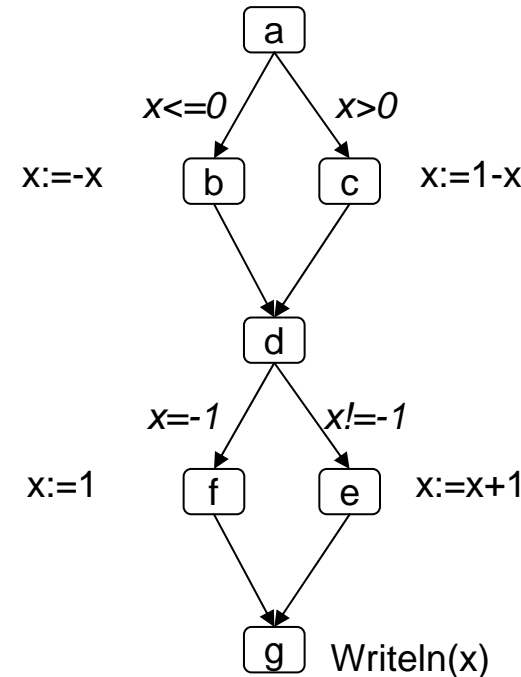
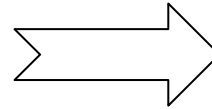
1. Techniques de **couverture du graphe de contrôle**
 - a) Couverture du **flot de contrôle** (toutes les instructions, branches, chemins...)
 - b) Couverture du **flot de données** (toutes les définitions de variables, les utilisations...)
 2. Test mutationnel (test par injection de défaut)
 3. Exécution 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
```



Graphe orienté et connexe (N,A,e,s)

- e: un sommet **entrée** (a)
- s: 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

Le graphe G peut être exprimé sous une forme algébrique : soit M l'ensemble des chemins de contrôle du graphe G :

$$\begin{aligned}
 M &= abdfg+abdeg+acdfg+acdeg \\
 &= a.(bdf+bde+cdf+cde).g \\
 &= a.(b+c)d.(e+f).g \text{ (expression des chemins de G)}
 \end{aligned}$$

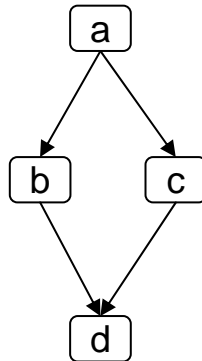
Construction de l'expression des chemins :

séquentielle



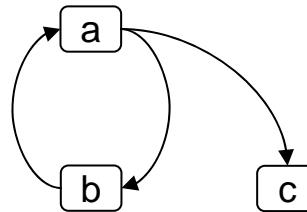
ab

alternative



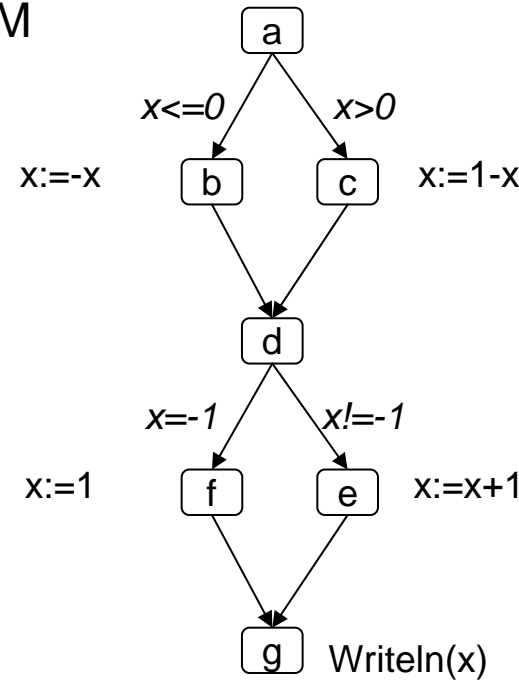
$a.(b+c).d$

itérative



$a.(ba)^*.c$

$a.(ba)^4.c$



G

Notation utilisée (pour rappel...)

Soit $X=\{a,b,c\}$ un alphabet

- ε le mot vide
- $(cb)^2=cbc b$
- $b^+=b+b^2+b^3+b^4+b^5+ \dots$
- $b^*=\varepsilon+b+b^2+b^3+b^4+b^5+ \dots$
- $b^4=\varepsilon+b+b^2+b^3+b^4$

Chemin exécutable

```

Read(x)
if (x<=0)then x:=-x
else x:=1-x;
if (x=-1)then x:=1
else x:=x+1;
Writeln(x)

```

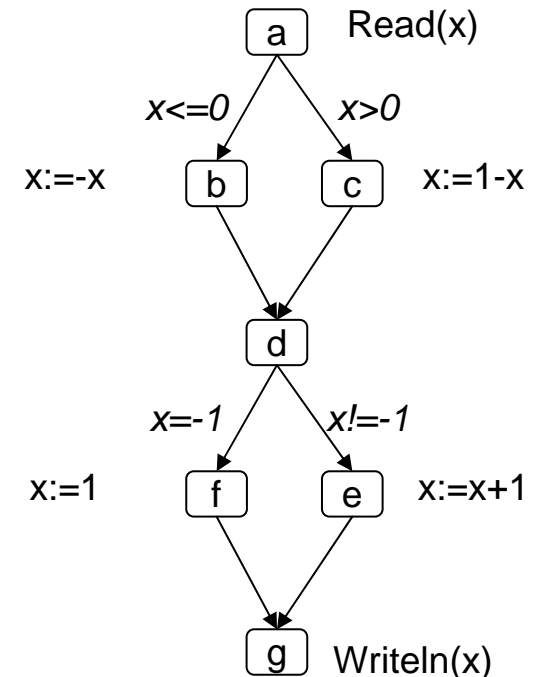
DT1={x=2}

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)

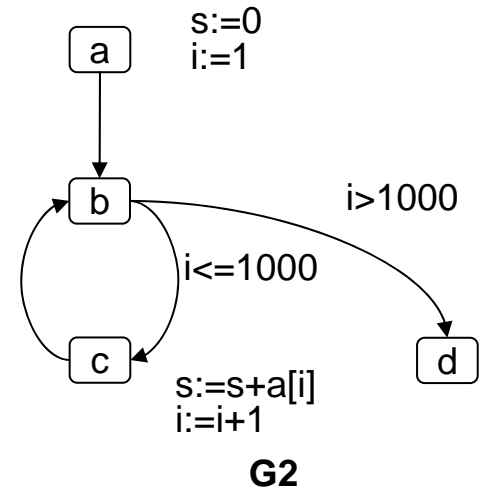
Existence de chemins non exécutables : signe de mauvais codage ?



Chemin exécutable / chemin non exécutable

- Nombre de chemins de contrôle de G :
 - se déduit directement de l'expression des chemins de G
 - $a(b+c)d(e+f)g \Rightarrow 1.(1+1).1.(1+1).1 = 4$ 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
```



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

Nombre de chemins :

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

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 $G(P3)$.

Problème 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'exécutabilité 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 !

Il existe des outils (plus ou moins automatiques) de sensibilisation de chemins (basés sur l'interprétation abstraite ou sur des techniques de vérification de programmes)

Exercice 2

```
Lire(choix)
if choix=1
then x=x+1 ;
if choix=2
then x=x-1 ;
writeln(x) ;
```

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 .
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 structural avec couverture

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

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

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 : Soient le graphe de contrôle $G5$, $\delta_1 = cdebcde$, $\delta_2 = ce$ et $T1 = \{\delta_1, \delta_2\}$.

$DT1 = \{a[1] = -2, a[2] = 3, a[3] = 17, i = 1\}$ satisfait-il $T1$?

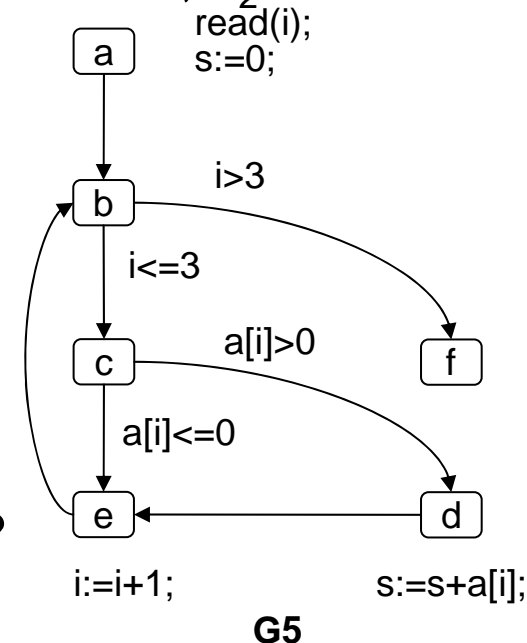
$DT1$ sensibilise $M1 = abcebcdebcdeb$

$M1 = abcebcdebcdeb$ couvre $\delta_1 = cdebcde$

$M1 = abcebcdebcdeb$ couvre $\delta_2 = ce$

Donc $DT1$ satisfait $T1$

$DT2 = \{a[1] = -2, a[2] = 3, a[3] = -17, i = 1\}$ satisfait-il $T1$?



Hiérarchie des techniques de test structurel

- Exemple : considérons le graphe de contrôle G5 et les 2 tests structurels avec couverture T1 et T2 définis par :

$\delta_1 = cdebcde$, $\delta_2 = ce$ et $T1 = \{\delta_1, \delta_2\}$

$\delta_3 = de$, $\delta_4 = b$, $\delta_5 = cd$ et $T2 = \{\delta_3, \delta_4, \delta_5\}$.

Lorsque T1 est satisfait, T2 l'est aussi : pourquoi ?

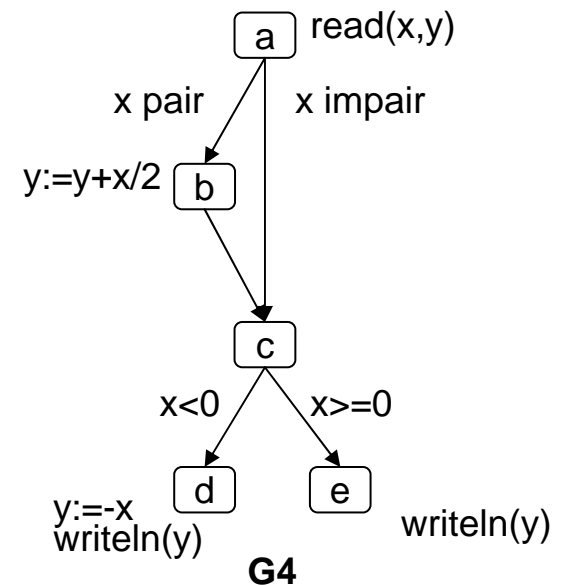
T1 est un test plus **fiable** (ie. 'fort') que T2 et on notera :

$T1 \Rightarrow T2$

- \Rightarrow est une relation d'ordre partielle (réflexive, antisymétrique, transitive)
- \Rightarrow permet de définir une **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={ $x=2, y=0$ }



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)

$$\text{TER1} = |\{\text{nœuds couverts}\}| / |\{\text{nœuds}\}|$$

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

Si on cherche 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.

$$\text{TER2} = |\{\text{arcs couverts}\}| / |\{\text{arcs}\}|$$

- Hiérarchie des tests

« tous-les-arcs » \Rightarrow « tous-les-nœuds »

Exercices 4

1. Donner un graphe de contrôle G et des données de test DT_i montrant que le critère « tous les noeuds » est insuffisant pour détecter une erreur.
 - Calculer le taux de couverture TER1 et TER2 pour votre DT.
2. Complétez le programme suivant qui calcule l'inverse de la somme des éléments, d'indice entre inf et sup , d'un tableau a contenant des entiers strictement positifs :

```
lire (inf, sup);  
i:=inf;  
sum:=0;  
while (i<= sup)  
do begin  
  sum:=sum+a[i];  
  .../...
```

- Tester le programme avec $DT1=\{a[1]=1; a[2]=2; a[3]=3; inf=1; sup=3\}$. Que se passe-t-il ?
- Calculer TER1 et TER2.

Couverture sur le flot de contrôle (suite)

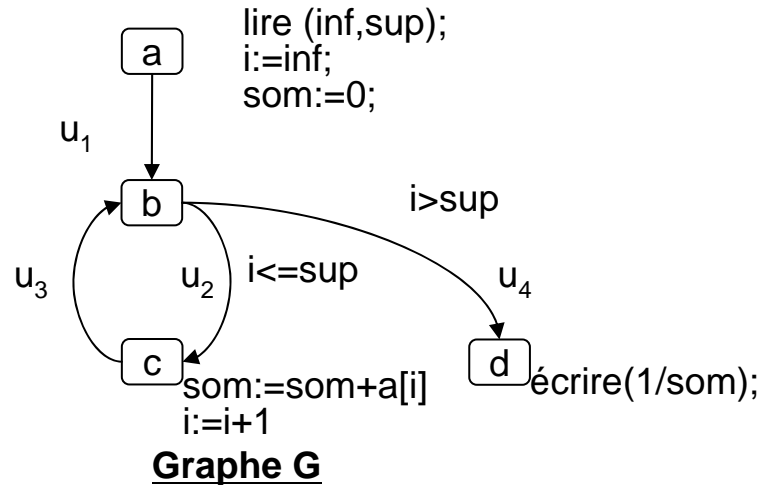
- Critère de couverture «**tous-les-chemins-indépendants** »
 $V(G)$ (le nombre de Mc Cabe ou nombre cyclomatique) donne le nombre de chemins indépendants.
 $V(G) = \#arcs - \#noeuds + 2$ [Si que des décisions binaires :
 $V(G) = \text{Nombre de nœuds de décision} + 1$
(Ce nombre est aussi le nombre de régions du graphe)
Taux de couverture :
 $|\{\text{chemins indépendants couverts}\}| / V(G)$
- Hiérarchie des tests
« tous-les-chemins-indépendants » \Rightarrow « tous-les-arcs »

Exercice 5: Donner le nombre Mc Cabe du graphe associé au programme suivant :

```
if C1 then while (C2) do X1;  
else X2;  
X3;
```


Exercices 6

1. Donner le nombre de chemins indépendants du graphe G.
2. Donner une DT1 qui sensibilise $M1=[abcbcbcbcd]$.
3. Donner une DT2 qui sensibilise $M2=[abd]$.
4. $(M1, M2)$ constitue une base : donner une relation qui lie $M1$, $M2$ et $M3=[abcbcd]$.
5. Calculer le taux de couverture du critère tous-les-chemins-indépendants associé à $DT1 \cup DT2$.



Couverture sur le flot de contrôle (suite)

- Couverture des **PLCS** (**P**ortion **L**inéaire de **C**ode **S**uivie d'un **S**aut)

PLCS : « séquence d'instructions entre deux branchements »

Principes : 2 types de nœuds dans le graphe de flot de contrôle

Type a (ou nœud 'saut') : l'entrée, la sortie et les nœuds qui constituent l'arrivée d'un branchement

Type b : les autres nœuds

Définition : On appelle PLCS un chemin partant d'un nœud 'saut' et aboutissant à un nœud 'saut' ; l'avant dernier et le dernier nœud doivent constituer le seul saut du chemin.

Couverture du critère PLCS : Exemple

Soit le programme :

005 INPUT A, C

010 B= 2*A

020 A=A+1

030 IF A<0 THEN GOTO 60

040 B= -A

050 PRINT A+B

060 IF B=2*C THEN GOTO 80

070 A=1:GOTO 90

080 A=-2:GOTO 20

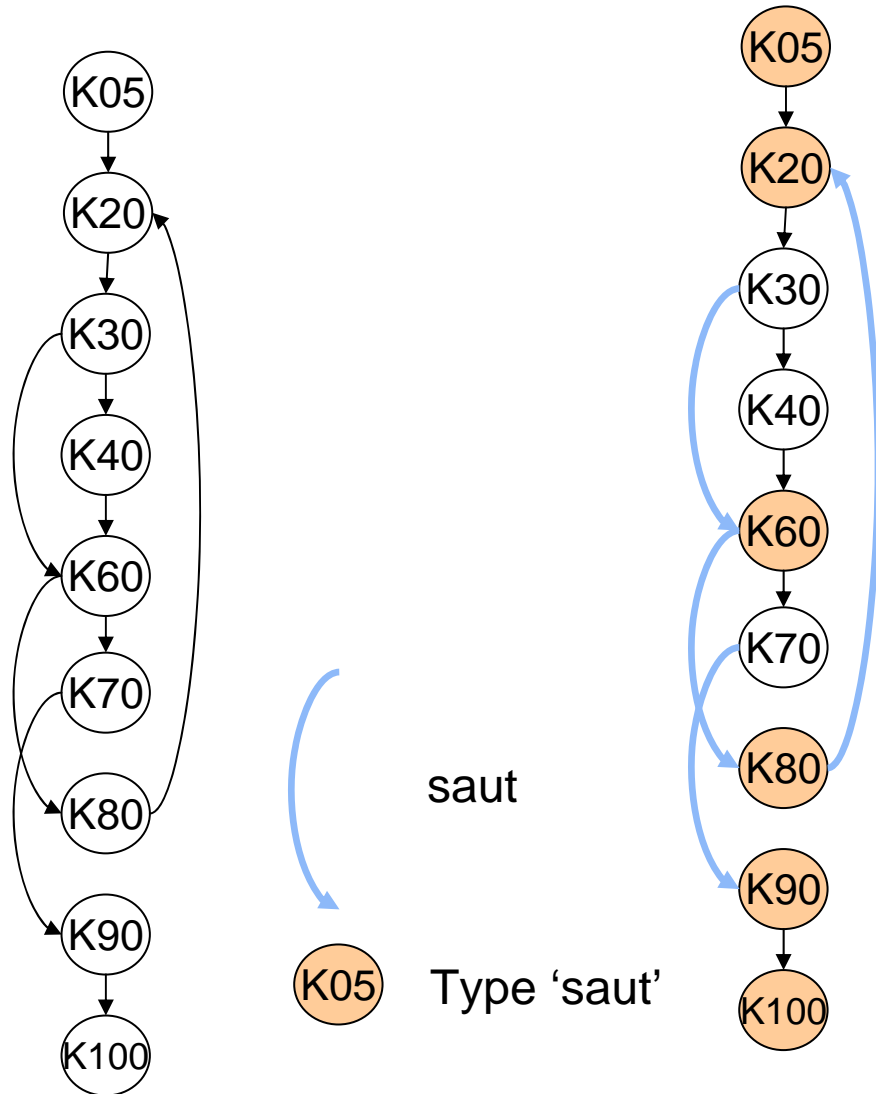
090 PRINT A

100 END

Donner le graphe de contrôle.

Repérer les nœuds de type 'saut'.

Donner les PLCS.



Couverture du critère PLCS : Exemple

Le programme contient 9 PLCS :

[5, 20, 30, 60]

[5, 20, 30, 40, 60, 80]

[5, 20, 30, 40, 60, 70, 90]

[20, 30, 60]

[20, 30, 40, 60, 80]

[20, 30, 40, 60, 70, 90]

[60, 80]

[60, 70, 90]

[80, 20]

Incluses dans les 3 premières...

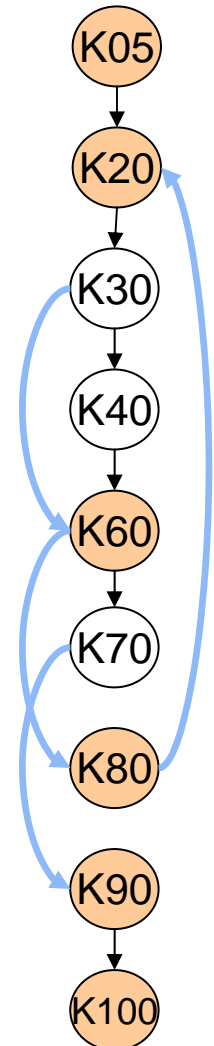
Si on exécute les 3 chemins :

B1 = [5, 20, 30, 60, 70, 90, 100]

B2 = [5, 20, 30, 40, 60, 80, 20, 30, 60, 70, 90, 100]

B3 = [5, 20, 30, 40, 60, 70, 90, 100]

=> Couverture de la totalité des PLCS



Taux de couverture

TER3 (Test Effectiveness Ratio 3 ou C3)

$$\text{TER3} = |\{\text{PLCS couvertes}\}| / |\{\text{PLCS}\}|$$

Hiérarchie des tests

- Hiérarchie : $\text{TER3}=1 \Rightarrow \text{TER2}=1$

Autres critères de type PLCS

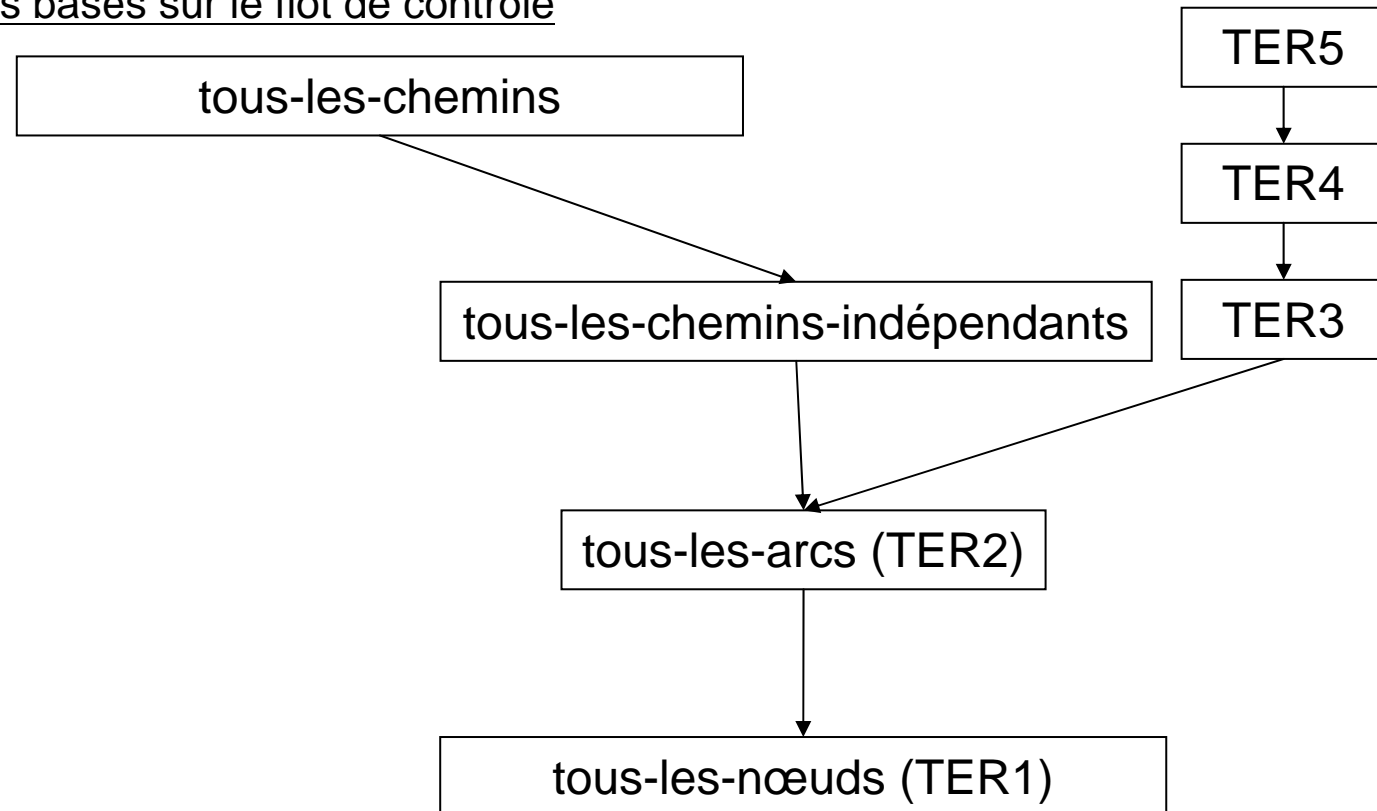
- Chemins composés de 2 PLCS

$$\text{TER4} = |\{\text{chemins composés de 2 PLCS couverts}\}| / |\{\text{chemins composés de 2 PLCS}\}|$$

- Chemins composés de 3 PLCS, $\text{TER5} =$

Couverture sur le flot de contrôle (suite)

Hiérarchie des tests basés sur le flot de contrôle



Exercice 7

Donnez les PLCS du programme suivant:

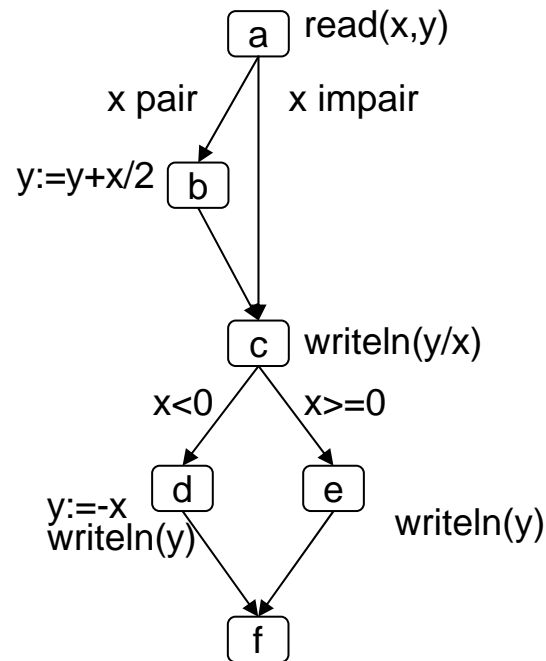
```
main( )
{
int i, factoriel;
cin>>n;factoriel=1;
for(i=1;i<=n;i++)
    factoriel=factoriel*i;
printf( "%d\n" , factoriel);
}
```

Autre définition...

Définition. Un **saut** est une arête (s,s') du graphe de contrôle tq :

- s est le sommet initial du graphe ou bien
 - s' est le sommet terminal du graphe ou bien
 - s est une condition et s' est le sommet atteint dans le cas où s est évalué à faux ou bien
 - s' est la condition d'une boucle et s le sommet terminal du corps de cette boucle
- Définition. Une **PLCS** est un couple $[c,s]$ où c est un chemin $c=s_1s_2\dots s_k$ tel que :
 - s_1 est le sommet d'arrivée d'un saut
 - $s_1s_2\dots s_k$ est sans saut
 - (s_k,s) est un saut

Couverture sur le flot de contrôle : pas assez fine !



G4

Analyse des relations entre instructions en tenant compte des variables qu'elles utilisent/définissent : on cherchera à couvrir les différentes façons de définir et d'utiliser les variables...

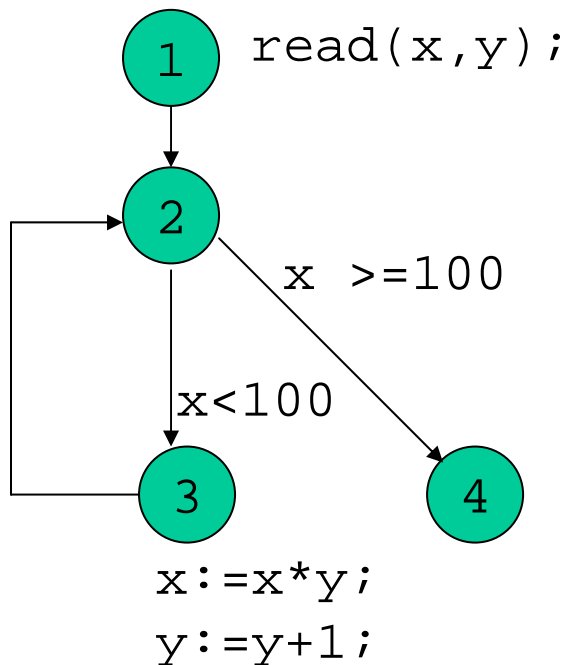
- variable **définie** si sa valeur est modifiée (affectation, lecture)
- variable **utilisée** : si sa valeur est utilisée :
 1. **p-utilisation** : dans le prédicat d'une instruction de décision (if, while,...)
 2. **c-utilisation** : dans les autres cas (utilisation de la valeur d'une variable pour un calcul)

```

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

Une instruction I est **utilisatrice** d'une variable x par rapport à une instruction de définition J si :

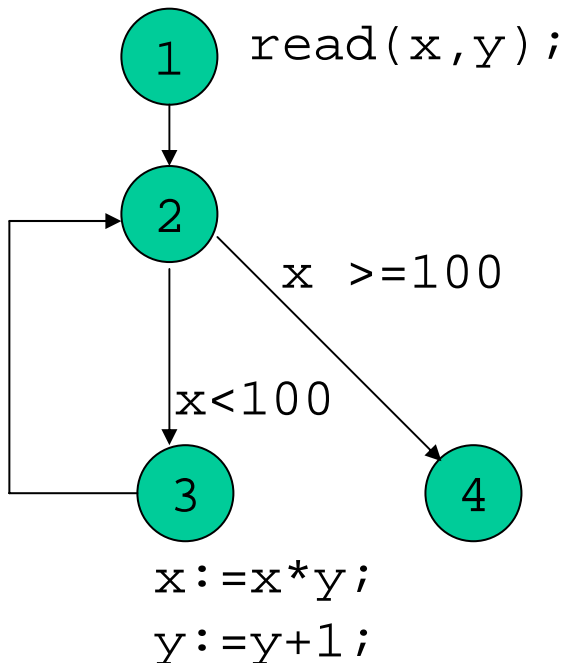
- x est définie en I et référencé en J
- x n'est pas redéfinie entre I et J



Exemple:

- L'arc (2, 4) est p-utilisateur de x par rapport au nœud 1
- L'instruction 'x:=x*y' est c-utilisatrice de y par rapport au nœud 1

Un **chemin d'utilisation** (c-utilisation ou p-utilisation) est un chemin reliant l'instruction de définition d'une variable à une instruction utilisatrice.



Exemple :
[1,2,4] est un chemin p-utilisation
pour la variable x.

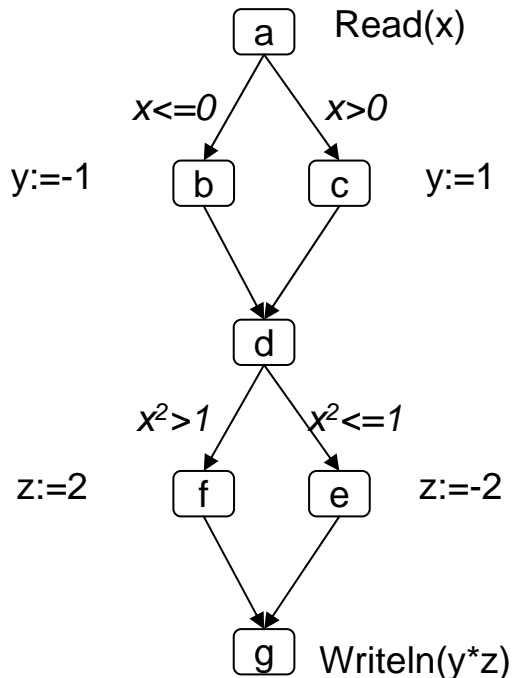
Critères

Le critère **tous-les-utilisateurs** nécessite 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**]

Le critère **toutes-les-définitions** nécessite que l'on couvre au moins un chemin d'utilisation pour chaque définition.

tous-les-utilisateurs \Rightarrow toutes-les-définitions

Critère tous-les-du-utilisateurs



- Couverture du critère **toutes-les-définitions** :

[abdfg] [acdeg]

- 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**

[du : définition / utilisation]

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

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

Exercice 8

- Soit le programme suivant :

```
lire (a,b)
```

```
si a<0 alors b:=b-a
```

```
si b pair
```

```
alors p:=bxb
```

```
sinon p:= 2xa
```

```
écrire(p)
```

- Couverture du critère **tous-les-arcs** ?
- Couverture du critère **toutes-les-définitions** ?
- Couverture du critère **tous-les-du-utilisateurs** ?

Hiérarchie des tests structurels

