Le test structurel

Frédéric Dadeau

Département Informatique des Systèmes Complexes – FEMTO-ST

Bureau 405C

frederic.dadeau@univ-fcomte.fr

Licence 3 – Année 2023-2024

Les bases du test structurel

Définition

Le test structurel s'appuie sur l'**analyse du code source** de l'application pour établir les tests en fonction de critères de couverture.

Critères de couverture

- basés sur le **graphe de contrôle** : toutes-les-instructions, toutes-les-branches, tous-les-chemins, etc.
- basés sur le **flot de données** : toutes-les-définitions de variables, toutes-les-utilisations, etc.
- basés sur les **fautes** : test par mutation

Graphe de contrôle

- C'est un graphe permettant de représenter n'importe quel algorithme sous la forme de **nœuds** et d'**arcs**.
- Les nœuds du graphe représentent des **blocs d'instructions**.
- Les arcs représentent la possibilité de **transfert de l'exécution** d'un nœud à un autre.
- Il possède une seule **entrée** (nœud à partir duquel on peut visiter tous les autres) et une seule **sortie**.

Graphe de contrôle

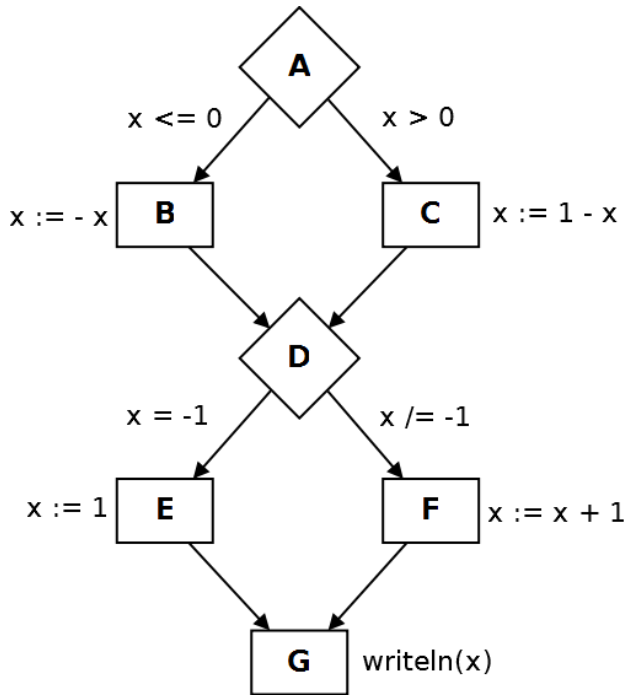
Considérons le programme P_1 suivant :

```
if x <= 0 then
  x := - x
else
  x := 1 - x
end if

if x = -1 then
  x := 1
else
  x := x + 1
end if

writeln(x)
```

Et son graphe de contrôle associé :



Graphe G_1

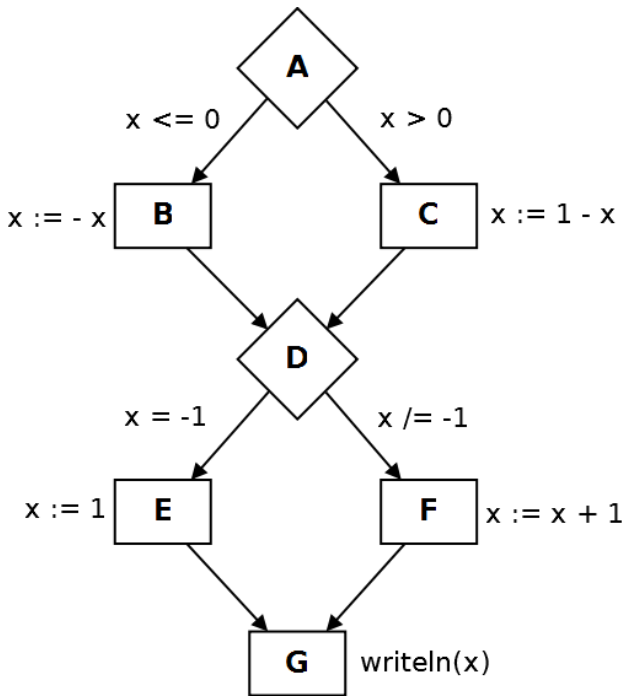
Graphe de contrôle

Le graphe G_1 est un graphe de contrôle qui admet une entrée (le nœud A) et une sortie (le nœud G).

- le chemin [a, b, d, e, g] est un chemin de contrôle
- le chemin [b, d, f, g] n'est pas un chemin de contrôle

Le graphe comporte 4 chemins de contrôle :

- c_1 le chemin [a, b, d, e, g]
- c_2 le chemin [a, b, d, f, g]
- c_3 le chemin [a, c, d, e, g]
- c_4 le chemin [a, c, d, f, g]



Graphe G_1

Expression des chemins de contrôle

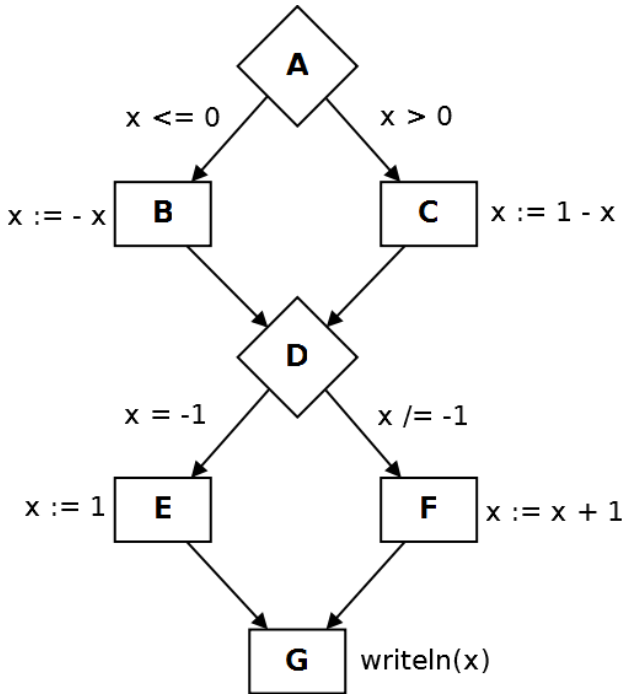
Le graphe G_1 peut être exprimé sous la forme d'une expression algébrique de la manière suivante :

$$abdeg + abdfg + acdeg + acdfg$$

où $+$ représente le ou logique entre chemins.

Simplification de l'expression des chemins :

$$\begin{aligned} & abdeg + abdfg + acdeg + acdfg \\ \Leftrightarrow\ & a(bde + bdf + cde + cdf)g \\ \Leftrightarrow\ & a(b + c)d(e + f)g \end{aligned}$$

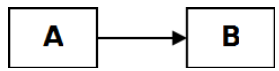


Graphe G_1

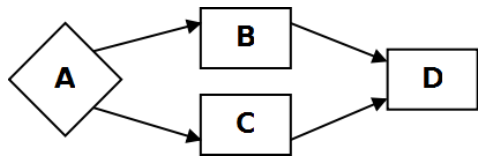
Expression des chemins de contrôle

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

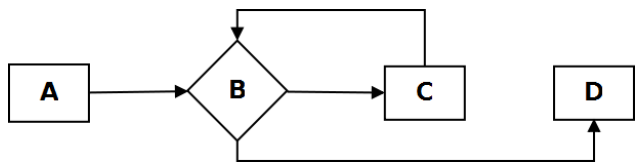
Forme séquencielle : ab



Instruction de décision : $a(b + c)d$



Structures itératives : $ab(cb) * d$



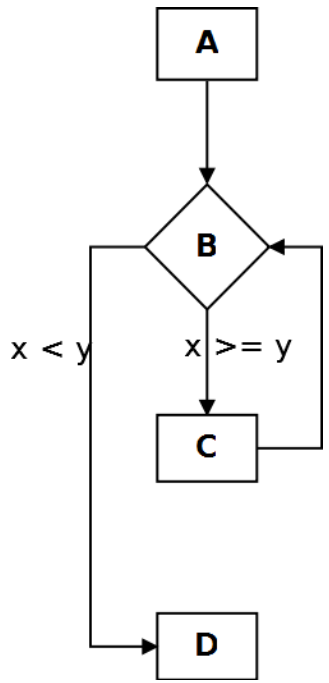
Graphe de contrôle

Soit le programme P_2 suivant :

```
open(fichier1)
read(x, fichier1)
read(y, fichier1)
z := 0

while x >= y do
  x := x - y
  z := z + 1
done

open(fichier2)
print(z, fichier2)
close(fichier1)
close(fichier2)
```

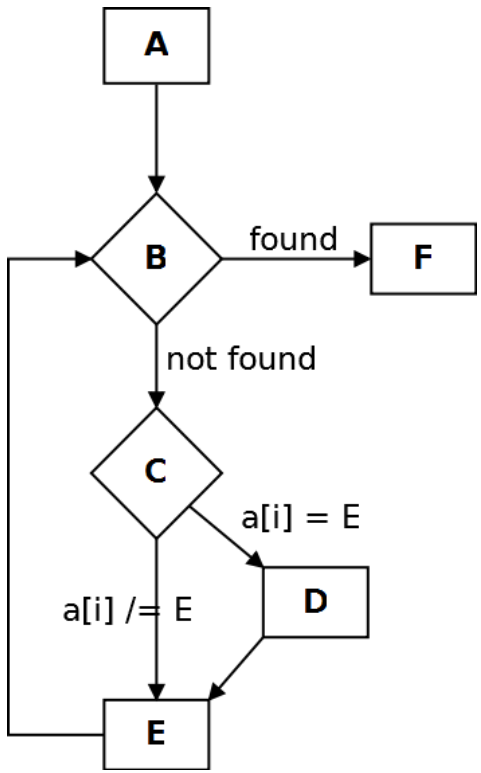


Graphe $G_2 = ab(cb) * d$

Graphe de contrôle

Soit le programme P_3 suivant :

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



Graphe $G_3 = ab(c(1 + d)eb) * f$

Graphe de contrôle - exercice

Soit le programme P_4 suivant :

```
if n <= 0 then
  n := 1 - n
end if

if n mod 2 = 0 then
  n := n / 2
else
  n := 3*n + 1
end if

write (n)
```

1. Donner le graphe de contrôle de P_4
2. Donner l'expression des chemins

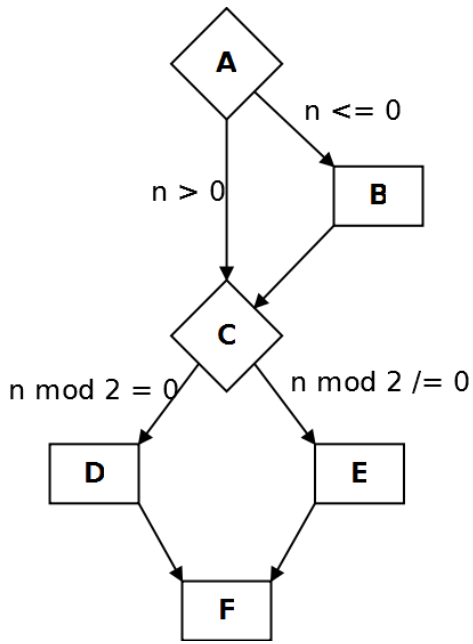
Graphe de contrôle - exercice

Soit le programme P_4 suivant :

```
if n <= 0 then
  n := 1 - n
end if

if n mod 2 = 0 then
  n := n / 2
else
  n := 3*n + 1
end if

write (n)
```



1. Donner le graphe de contrôle de P_4
2. Donner l'expression des chemins

Graphe $G_4 = a(1 + b)c(d + e)f$

Graphe de contrôle - exercice

Soit le programme P_5 suivant :

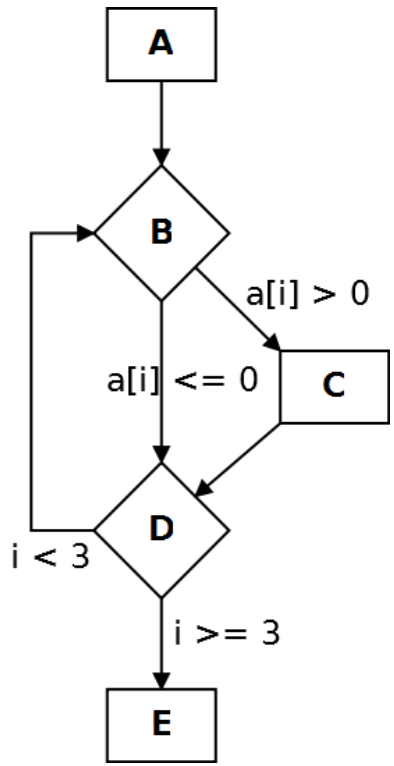
```
read (i)
s := 0
do
  if a[i] > 0 then
    s := s + a[i]
    i := i + 1
  end if
while (i < 3)
```

1. Donner le graphe de contrôle de P_5
2. Donner l'expression des chemins

Graphe de contrôle - exercice

Soit le programme P_5 suivant :

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



1. Donner le graphe de contrôle de P_5
2. Donner l'expression des chemins

Graphe $G_5 = a\,b(1+c)d(b(1+c)d) * e$

Production de données de test

Produire des données de test

Le principe est le suivant :

- On considère les **chemins du graphe de contrôle** (tous ou certains, selon le critère de sélection choisi).
- On produit des **données d'entrée** permettant d'activer le chemin.

Exécution des tests

- Le programme est exécuté dans ces configurations.
- On recherche des anomalies de fonctionnement qui sont potentiellement détectables sur les chemins considérés.

Critères de couverture

Critères de couverture basés sur le graphe de contrôle :

- couverture de *tous-les-nœuds*
- couverture de *tous-les-arcs*
- couverture de *tous-les-chemins-indépendants*
- couverture de *toutes-les-PLCS* (Portions Linéaires de Code suivies d'un Saut)
- couverture des *chemins limites* et *intérieurs*
- couverture de *tous-les-i-chemins*
- couverture de *tous-les-chemins*

Couverture de tous-les-nœuds

Egalement appelé ...

- toutes-les-instructions
- **statement coverage** (SC) dans la littérature

Définition

Chaque **nœud**, c'est-à-dire chaque bloc d'instructions, est atteint par au moins l'un des chemins parmi les chemins constituant le jeu de tests.

Couverture de tous-les-nœuds

Test Effectiveness Ratio 1

Lorsqu'un jeu de test permet de couvrir tous les nœuds du graphe, on dit qu'il satisfait TER1=1 ou TER1 (Test Effectiveness Ratio 1)

$$\begin{aligned} \text{TER1} = 1 &\Leftrightarrow \text{le critère tous-les-nœuds est satisfait} \\ &\Leftrightarrow \text{tous les nœuds du graphe de contrôle ont été couverts} \\ &\Leftrightarrow \text{toutes les instructions ont été exécutées} \end{aligned}$$

Taux de couverture

Le taux de couverture du critère tous-les-nœuds se calcule par le ratio

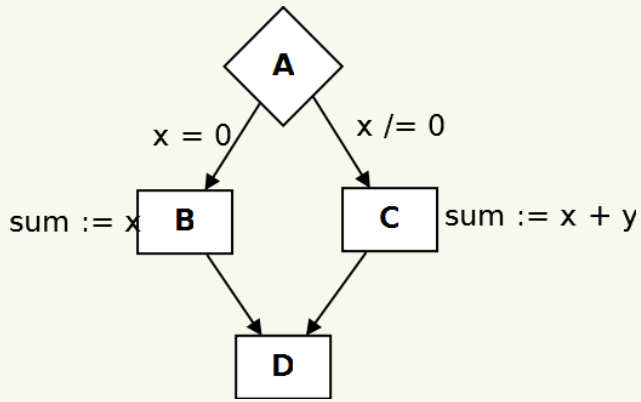
$$TER1 = \frac{\text{nombre de nœuds couverts}}{\text{nombre de nœuds total}}$$

Couverture de tous-les-nœuds - exemple

Application du critère tous-les-nœuds

Soit le programme suivant (somme avec une erreur) :

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



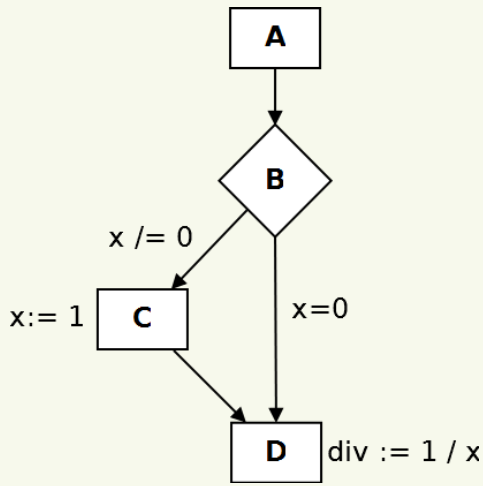
Le chemin *abc* permet d'activer le défaut. Le jeu de test $DT_1 = \{x = 0, y = 42\}$ $DT_2 = \{x = 42, y = 42\}$ permet de satisfaire le critère de couverture de tous-les-nœuds.

Couverture de tous-les-nœuds

Illustration des limites du critère tous-les-nœuds

Soit le programme suivant (avec une erreur) :

```
inv(x : integer) : float
  if (x /= 0) then
    x := 1
  end if
  inv := 1 / x
```



Le chemin *abcd* permet de satisfaire le critère tous-les-nœuds. La donnée de test $DT_1 = \{x = 42\}$ permet de couvrir le chemin *abcd*, couvrant ainsi tous les nœuds du graphe, mais sans faire apparaître l'anomalie.

Couverture de tous-les-arcs

Egalement appelé ...

- tous-les-enchaînements
- toutes-les-branches
- toutes-les-décisions
- **decision coverage** (DC) dans la littérature

Définition

Chaque **arc** est couvert par au moins l'un des chemins parmi les chemins constituant le jeu de tests.

La couverture de tous-les-arcs équivaut à la couverture de toutes les valeurs de vérité pour chaque nœud de décision, c'est-à-dire : leur valeur de vérité a été évaluée au moins une fois vraie et une fois fausse.

Couverture de tous-les-arcs

Test Effectiveness Ratio 2

Lorsqu'un jeu de test permet de couvrir tous les arcs du graphe, on dit qu'il satisfait TER2=1 ou TER2 (Test Effectiveness Ratio 2)

TER2 = 1 $\Leftrightarrow$ le critère tous-les-arcs est satisfait
$\Leftrightarrow$ tous les arcs du graphe de contrôle ont été couverts
$\Leftrightarrow$ toutes les décisions ont été exécutées

Un jeu de test qui satisfait TER2 satisfait aussi TER1 (TER1 $\subset$ TER2)

Taux de couverture

Le taux de couverture du critère tous-les-arcs se calcule par le ratio

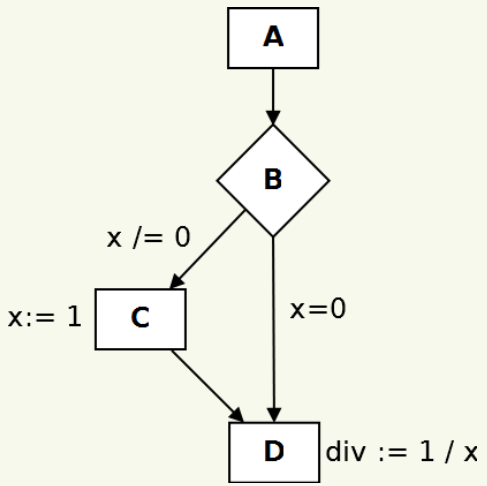
$$TER2 = \frac{\text{nombre d'arcs couverts}}{\text{nombre d'arcs total}}$$

Couverture de tous-les-arcs

Application du critère tous-les-arcs

Reprenons le programme précédent :

```
inv(x : integer) : float
  if (x /= 0) then
    x := 1
  end if
  inv = 1 / x
```



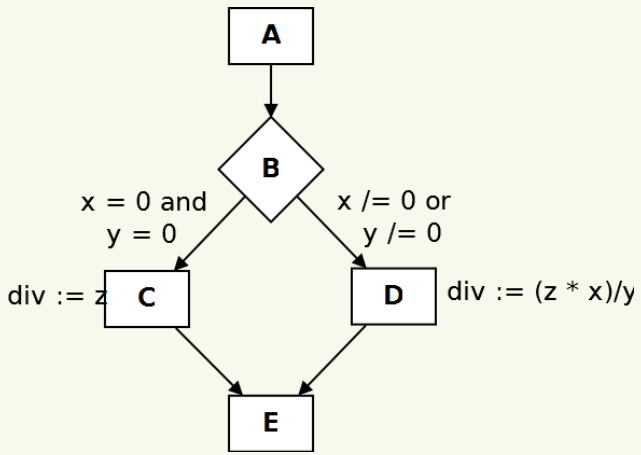
Il y a 4 arcs à couvrir : *ab*, *bc*, *cd* et *bd*. Ces arcs peuvent être couverts par deux chemins *abcd* et *abd*. Les données de test $DT_1 = \{x = 0\}$ et $DT_2 = \{x = 42\}$ permet couvrir ainsi tous les arcs du graphe, faisant ainsi apparaître l'anomalie.

Couverture de tous-les-arcs

Illustration des limites du critère tous-les-arcs

Soit le programme suivant (avec une erreur) :

```
div(x, y : integer) : float
  read(z)
  if x = 0 and y = 0 then
    div := z
  else
    div := (z * x) / y
  end if
```



Les données de test $DT_1 = \{x = 0, y = 0\}$ et $DT_2 = \{x = 42, y = 42\}$ permettent de satisfaire le critère tous-les-arcs. Néanmoins, l'erreur ne peut pas être détectée.

Couverture de toutes-les-conditions

Egalement appelé ...

- **condition coverage** (CC) ou *predicate coverage* dans la littérature

Définition

Le critère *toutes-les-conditions* est satisfait si la suite de tests satisfait le critère tous-les-nœuds et si chaque condition à l'intérieur d'une décision a été évaluée à vrai et à faux au moins une fois.
Attention : ce critère ne demande pas la satisfaction du critère *tous-les-arcs* !

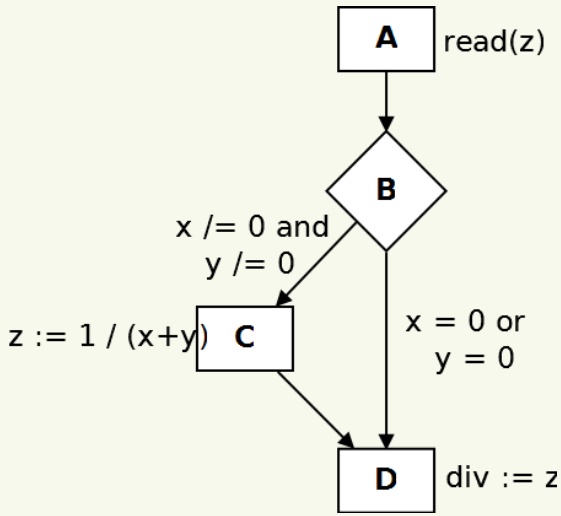
Il n'est pas demandé d'évaluer chaque décision à vrai et à faux, mais, par effet de bord, la couverture des décisions peut également être satisfaite.

Couverture de toutes-les-conditions

Illustration du critère toutes-les-conditions

Prenons le programme suivant :

```
div(x, y : integer) : float
  read(z)
  if x /= 0 and y /= 0 then
    z := 1 / (x + y)
  end if
  div := z
```



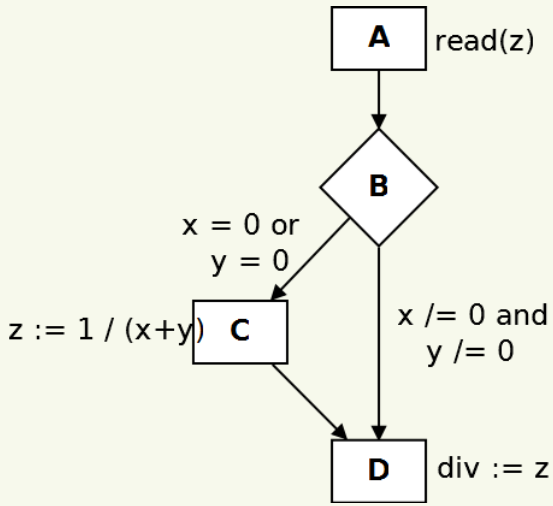
Les données de test $DT_1 = \{x = 42, y = 42\}$ et $DT_2 = \{x = 0, y = 0\}$ permettent de satisfaire le critère tous-les-nœuds et couvrent toutes les valeurs de vérité des deux conditions présentes dans la décision.
Par ailleurs, ces données de test permettent d'activer tous les arcs (mais c'est une coïncidence !).

Couverture de toutes-les-conditions

Illustration du critère toutes-les-conditions

Prenons le programme suivant :

```
div(x, y : integer) : float
  read(z)
  if x = 0 or y = 0 then
    z := 1 / (x + y)
  end if
  div := z
```



Les données de test $DT_1 = \{x = 0, y = 42\}$ et $DT_2 = \{x = 42, y = 0\}$ permettent de satisfaire le critère tous-les-nœuds et couvrent toutes les valeurs de vérité des deux conditions présentes dans la décision.

Néanmoins, d'une part l'arc *bd* n'a pas été activé, et d'autre part, la combinaison $x = 0 \wedge y = 0$ n'a pas été sélectionnée.

Couverture des conditions/décisions

Egalement appelé ...

- **condition/decision coverage** (C/DC) dans la littérature

Définition

Le critère de conditions/décisions est satisfait si et seulement si :

- le critère tous-les-arcs est satisfait, et
- chaque condition est évaluée à toutes les valeurs de vérité possibles.

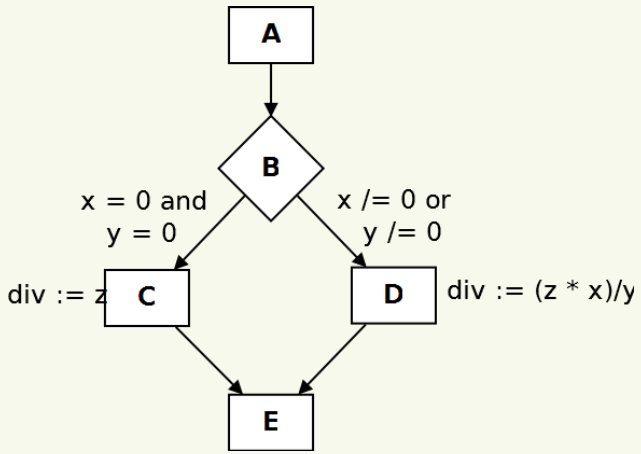
Il s'agit d'une combinaison des deux critères : couverture des décisions (tous-les-arcs) et couverture des conditions.

Couverture de toutes-les-conditions/décisions

Illustration des limites du critère toutes-les-conditions/décisions

Soit le programme suivant (avec une erreur) :

```
div(x, y : integer) : float
  read(z)
  if x = 0 and y = 0 then
    div := z
  else
    div := (z * x) / y
  end if
```



Les données de test $DT_1 = \{x = 0, y = 0\}$ et $DT_2 = \{x = 42, y = 42\}$ permettent de satisfaire le critère tous-les-arcs, et chaque condition est évaluée une fois à vrai et une fois à faux. Néanmoins, toutes les valeurs de vérité des sous-expressions ne sont pas couvertes, et l'erreur ne peut pas être détectée.

Couverture conditions/décisions modifiées

Egalement appelé ...

- **modified condition/decision coverage** (MC/DC) dans la littérature

Définition

Le critère de conditions/décisions modifiées a été initialement proposé par la norme DO178B. Il renforce le critère C/DC en assurant la présence de données de test dans lesquelles, pour chacune des conditions, sa valeur est corrélée à la valeur de la décision correspondante : la décision change lorsque la condition change (on dit que la condition contrôle la décision).

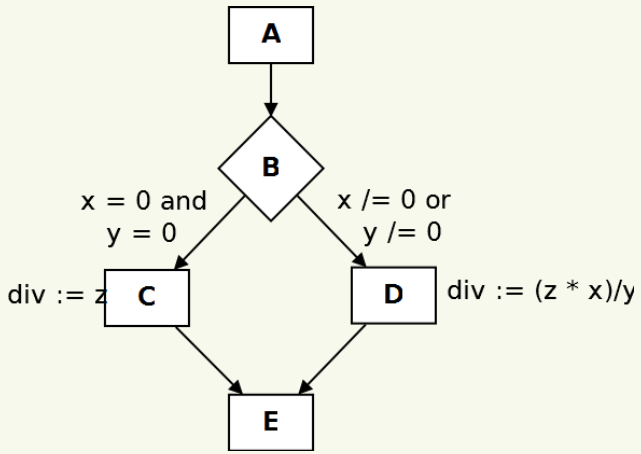
- le critère tous-les-arcs est satisfait, et
- chaque condition est évaluée à toutes les valeurs de vérité possibles, et
- chaque condition doit montrer qu'elle change la décision (si la valeur de vérité de la condition change, alors la décision change également)

Couverture conditions/décisions modifiées

Illustration du critère toutes-les-conditions/décisions modifiées

On reprend le programme :

```
div(x, y : integer) : float
  read(z)
  if x = 0 and y = 0 then
    div := z
  else
    div := (z * x) / y
  end if
```



Les données de test $DT_1 = \{x = 0, y = 0\}$, $DT_2 = \{x = 0, y = 42\}$ et $DT_3 = \{x = 42, y = 0\}$ permettent de satisfaire le critère tous-les-arcs, chaque condition est évaluée au moins une fois à vrai et au moins une fois à faux. De plus, en changeant les valeurs de vérité des conditions, la décision change également.

Couverture des conditions multiples

Egalement appelé ...

- **multiple condition coverage** (MCC) dans la littérature

Définition

Le critère de conditions multiples est satisfait si et seulement si :

- le critère tous-les-arcs est satisfait, et
- chaque sous-expression dans les conditions prend toutes les combinaisons de valeurs possibles.

Couverture des condition multiples

Valeurs de vérité des sous-expressions

Pour satisfaire le critère condition multiples, la couverture de

$$\textbf{if } P \wedge Q \textbf{ then}...$$

nécessite de couvrir :

- $P = Q =$ vrai
- $P = Q =$ faux
- $P =$ vrai et $Q =$ faux
- $P =$ faux et $Q =$ vrai

Attention : ce critère peut-être très combinatoire sur certaines expressions un peu complexes, comme par exemple $(a \vee b) \wedge (c \vee d)$.

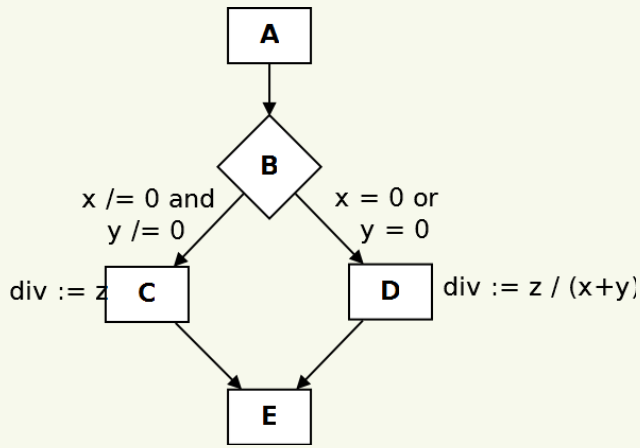
Pour N conditions, il faut considérer 2^N cas de tests.

Couverture de toutes les conditions multiples

Illustration du critère toutes les conditions multiples

Considérons le programme :

```
div(x, y : integer) : float
  read(z)
  if x /= 0 and y /= 0 then
    div := z
  else
    div := z / (x + y)
  end if
```

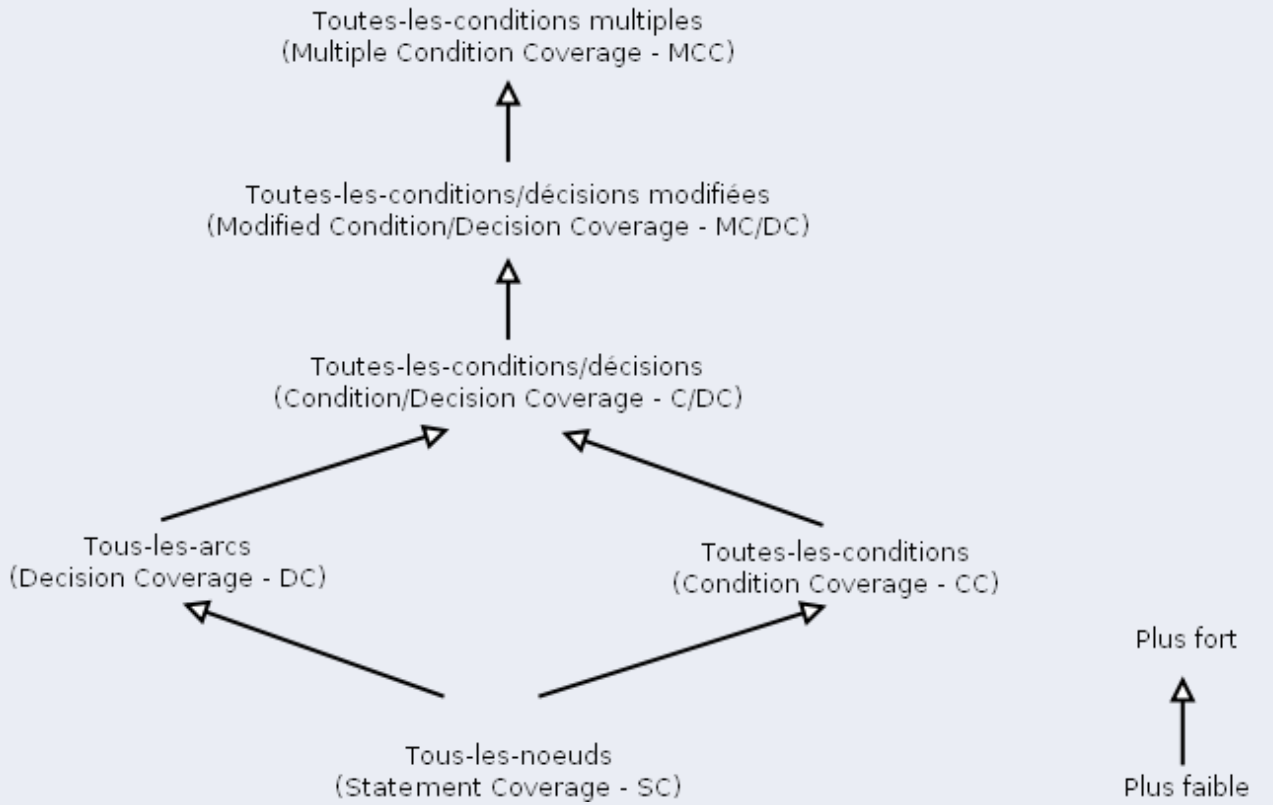


Les données de test $DT_1 = \{x = 42, y = 42\}$, $DT_2 = \{x = 0, y = 42\}$, $DT_3 = \{x = 42, y = 0\}$ et $DT_4 = \{x = 0, y = 0\}$ permettent de satisfaire le critère toutes les conditions multiples.

Toutefois, les valeurs choisies ne permettent pas de détecter l'erreur.

Résumé intermédiaire

Hiérarchie des critères vus précédemment



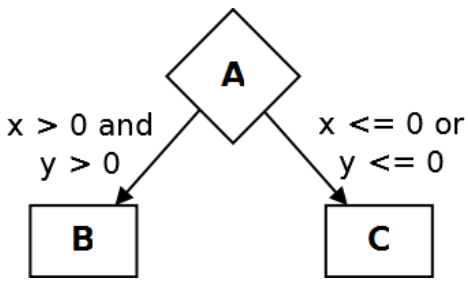
Résumé intermédiaire

Valeurs de vérité à considérer pour tester une décision en fonction du critère de couverture souhaité :

Critère de couverture	if $P \wedge Q$ then ...	if $P \vee Q$ then ...
tous-les-nœuds (SC)	$P \wedge Q$	$P \wedge \neg Q$
tous-les-arcs (DC)	$P \wedge Q$, $\neg P \wedge Q$	$P \wedge \neg Q$, $\neg P \wedge \neg Q$
toutes-les-conditions (CC)	$P \wedge Q$, $\neg P \wedge \neg Q$	$P \wedge \neg Q$, $\neg P \wedge Q$
toutes-les-conditions/décisions (C/DC)	$P \wedge Q$, $\neg P \wedge \neg Q$	
toutes-les-conditions/décisions modifiées (MC/DC)	$P \wedge Q$, $\neg P \wedge Q$, $P \wedge \neg Q$	$P \wedge \neg Q$, $\neg P \wedge Q$, $\neg P \wedge \neg Q$
toutes-les-conditions multiples (MCC)	$P \wedge Q$, $P \wedge \neg Q$, $\neg P \wedge Q$, $\neg P \wedge \neg Q$	

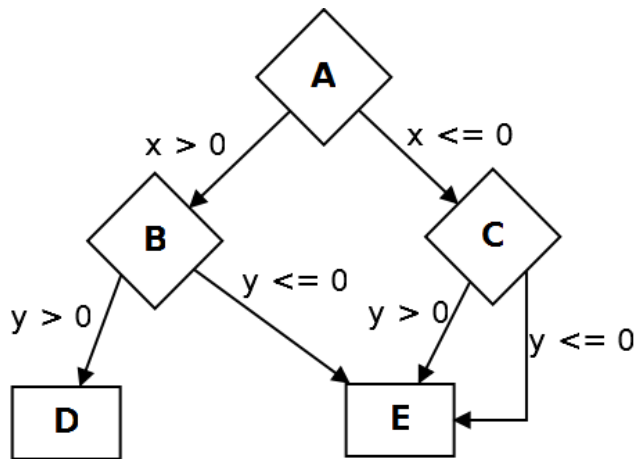
Couverture de tous les arcs et ses variantes

En pratique, il est intéressant d'intégrer le critère de couverture choisi dans le graphe, en restructurant ce dernier.



$\Longrightarrow$

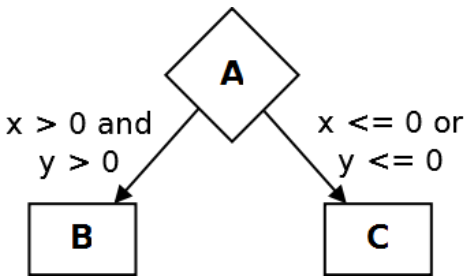
Couverture des conditions multiples



La couverture de tous-les-arcs permettra ensuite de satisfaire le critère voulu.

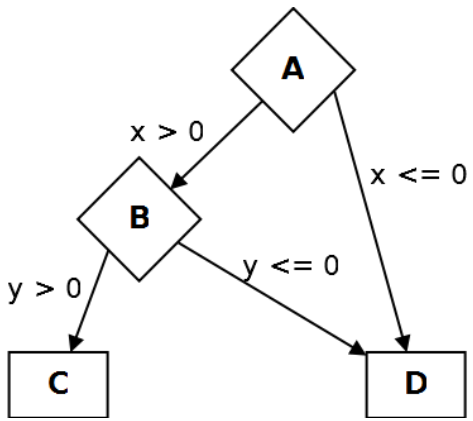
Couverture de tous les arcs et ses variantes

Attention cependant, la construction du graphe doit être sensible aux éventuelles optimisations du compilateur.



$\Longrightarrow$

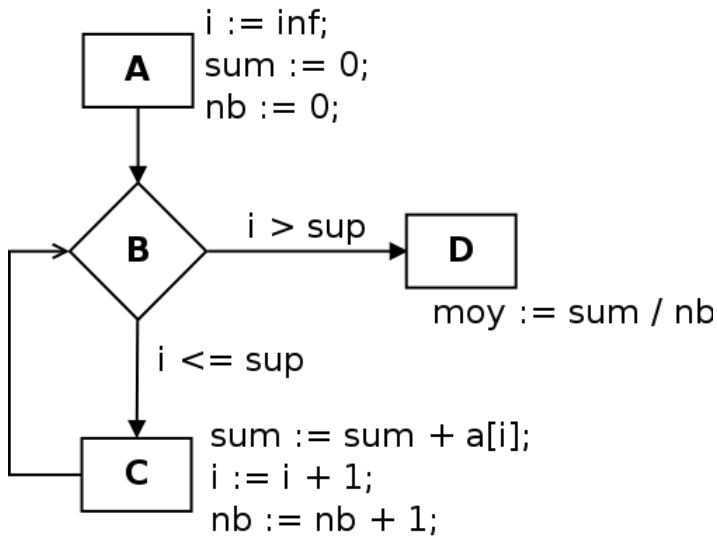
Couverture des conditions multiples avec évaluation paresseuse



Couverture de tous les arcs et ses variantes

Considérons le programme :

```
moy(a : integer[], inf, sup :
integer) : float
  i := inf
  sum := 0
  nb := 0
  while i <= sup do
    sum := sum + a[i]
    nb := nb + 1
    i := i + 1
  done
  moy := sum / nb
  end if
```



Le critère tous-les-arcs est satisfait par le chemin *abcbd*. La donnée de test $DT = \{a = [4, 8, 15, 16, 23, 42], inf = 0, sup = 5\}$ permet de satisfaire ce critère. Néanmoins une erreur pourrait être révélée si on évitait la boucle (inf > sup).

Les critères sur les chemins

Quelques critères basés sur les chemins (reprise de la diapo 13)

- couverture de *tous-les-chemins-indépendants*
- couverture de *toutes-les-PLCS* (Portions Linéaires de Code suivies d'un Saut)
- couverture des *chemins limites* et *intérieurs*
- couverture de *tous-les-i-chemins*
- couverture de *tous-les-chemins*

Les critères tous-les-chemins-indépendants

Définition

Le critère **tous-les-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)

Calcul du nombre de chemins indépendants

Le nombre de chemins indépendants d'un graphe G est donné par le nombre de Mc-Cabe, également appelé nombre de cyclomatique, noté $V(G)$

$$V(G) = \textit{Nb d'arcs} - \textit{Nb de nœuds} + \textit{Nb de nœuds d'entrée} + \textit{Nb de nœuds de sortie}$$

ou

$$V(G) = \textit{Nb de nœuds de décision} + 1$$

Les critères tous-les-chemins-indépendants

Procédure de calcul des chemins indépendants

1. Evaluer V(G)
2. Produire une donnée de test au hasard couvrant le maximum de nœuds de décisions du graphe
3. Produire une donnée de test qui modifie le flot de la première instruction de décision du graphe
4. Vérifier l'indépendance du chemin calculé par rapport aux autres chemins déjà calculés (2 chemins sont indépendants si et seulement si il existe un arc qui est couvert par l'un et pas par l'autre)
5. Recommencer les étapes 3 et 4 jusqu'à la couverture de toutes les décisions
6. S'il n'y a plus de décisions à étudier sur le chemin initial et que le nombre de chemins trouvés est inférieur à V(G), considérer les chemins secondaires déjà trouvés

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants

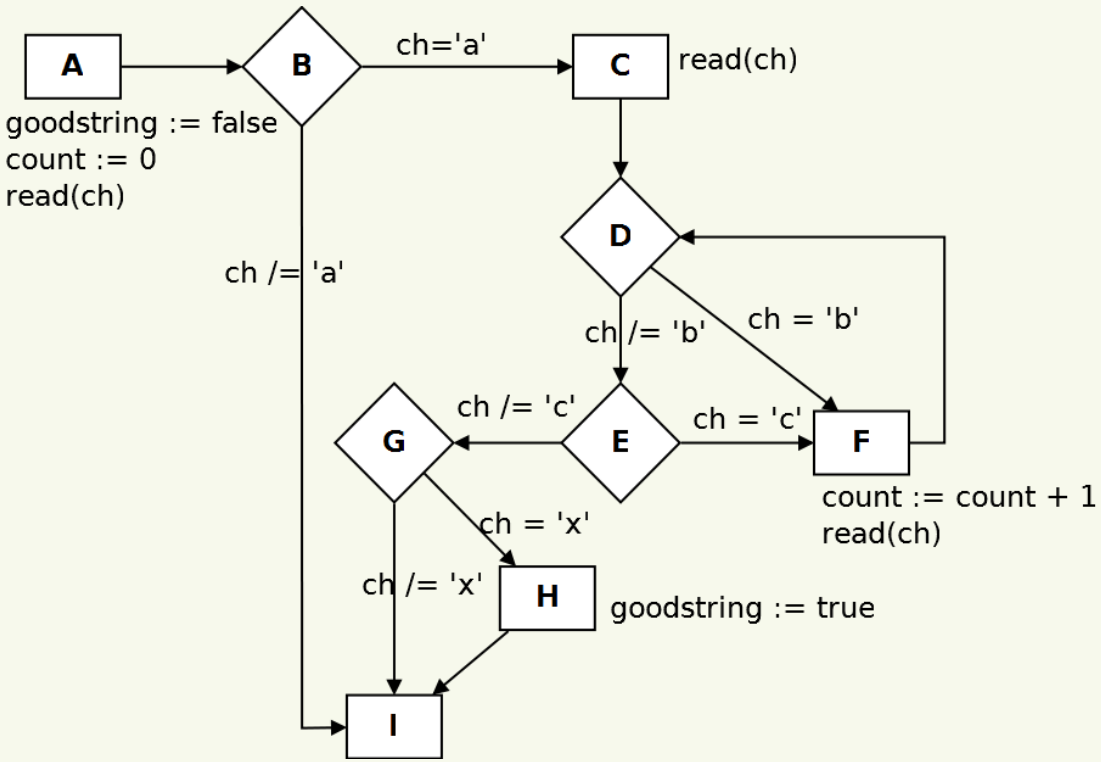
La fonction `goodstring` ci-dessous fait saisir une chaîne et reconnait si celle-ci correspond à l'expression régulière $a[bc] * x$

```
goodstring() : boolean
  ch : char
  goodstring := false
  count := 0
  read(ch)
  if ch = 'a' then
    read(ch)
    while (ch='b') or (ch='c') do
      count := count + 1
      read(ch)
    done
    if ch = 'x' then
      goodstring := true
    end if
  end if
```

- construire le graphe de contrôle en décomposant les conditions multiples
- appliquer la démarche

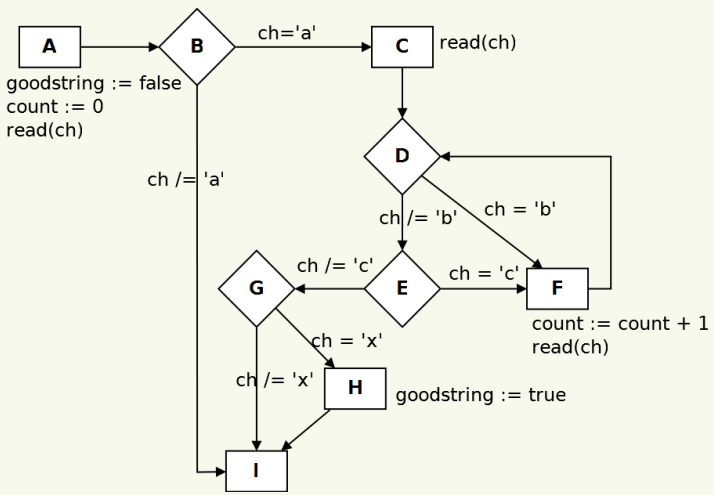
Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants



Les critères tous-les-chemins-indépendants

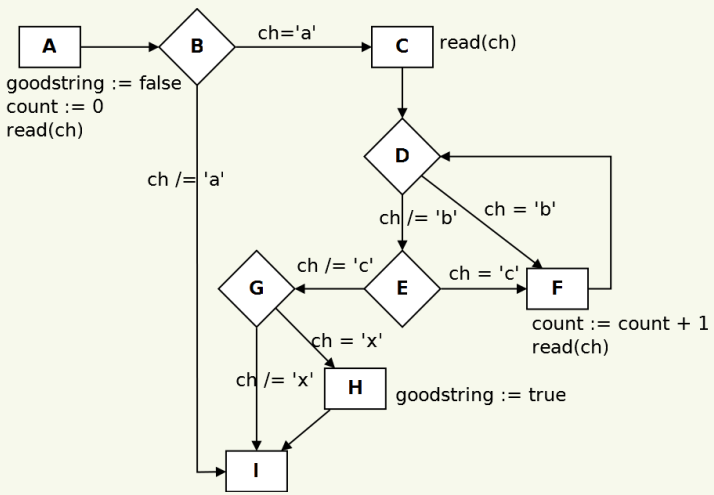
Calcul des chemins indépendants



V(G) = 12 arcs - 9 nœuds + 2 = 5

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants

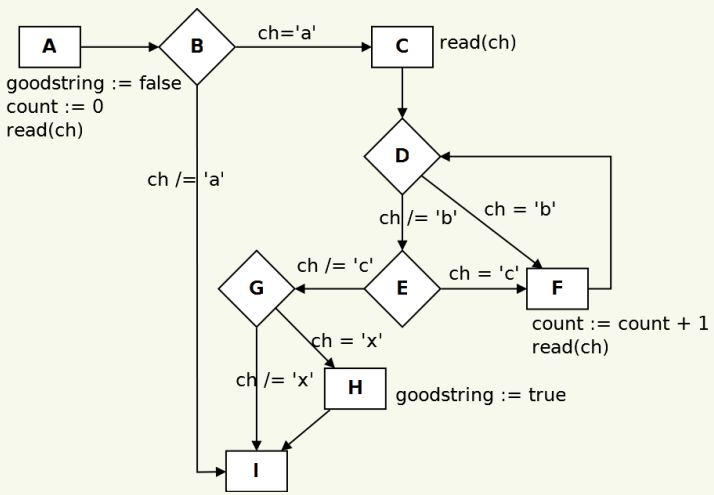


- Choix de la DT “abcx” qui couvre
  a b c d f d e f d e g h i

V(G) = 12 arcs - 9 nœuds + 2 = 5

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants

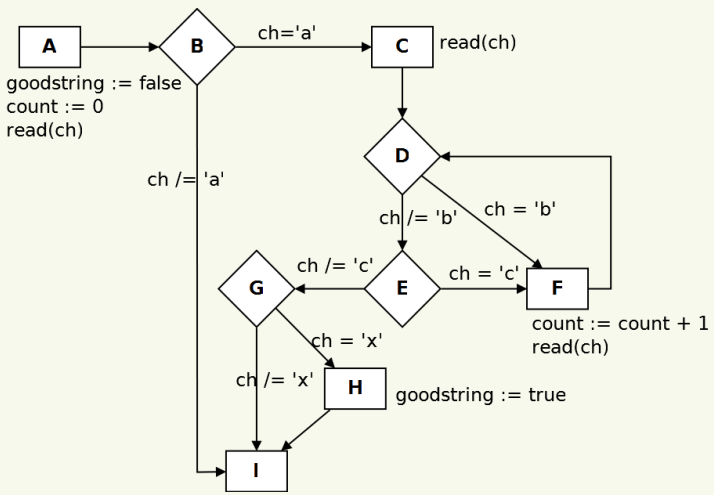


- Choix de la DT “abcx” qui couvre
  a b c d f d e f d e g h i
- Modifier la première décision (*b*)
  Couverture de *abi* (indépendant du précédent)
  avec la DT “b”

V(G) = 12 arcs - 9 nœuds + 2 = 5

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants

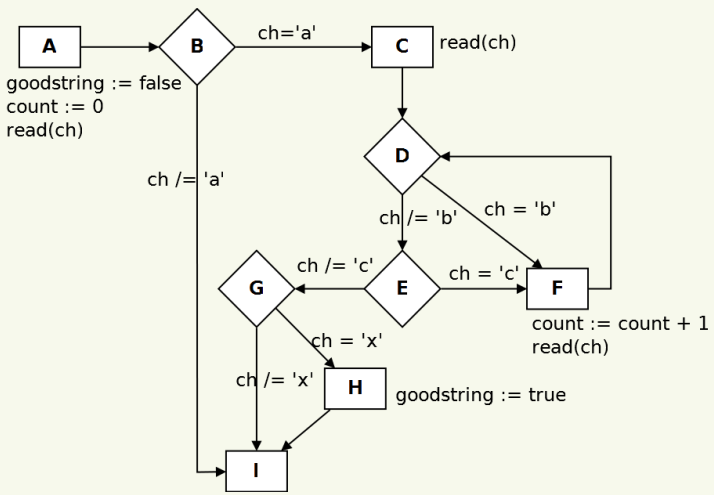


- Choix de la DT “abcx” qui couvre
  a b c d f d e f d e g h i
- Modifier la première décision (*b*)
  Couverture de *abi* (indépendant du précédent) avec la DT “b”
- Modifier la deuxième décision (*d*)
  Couverture de *abcdeghi* (indépendant des 2 précédents) avec la DT “ax”

V(G) = 12 arcs - 9 nœuds + 2 = 5

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants

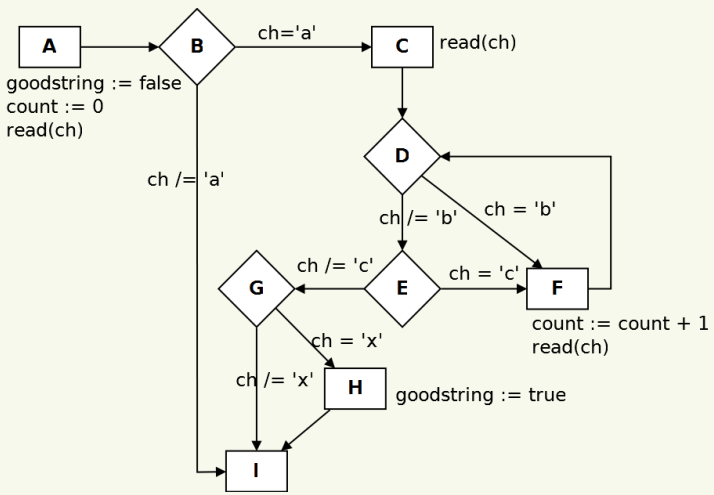


- Choix de la DT “abcx” qui couvre
  a b c d f d e f d e g h i
- Modifier la première décision (*b*)
  Couverture de *abi* (indépendant du précédent) avec la DT “b”
- Modifier la deuxième décision (*d*)
  Couverture de *abcdeghi* (indépendant des 2 précédents) avec la DT “ax”
- Modifier la troisième décision (*d*)
  Couverture de *abcdfdfdeghi* (indépendant des 3 précédents) avec la DT “abbx”

V(G) = 12 arcs - 9 nœuds + 2 = 5

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants

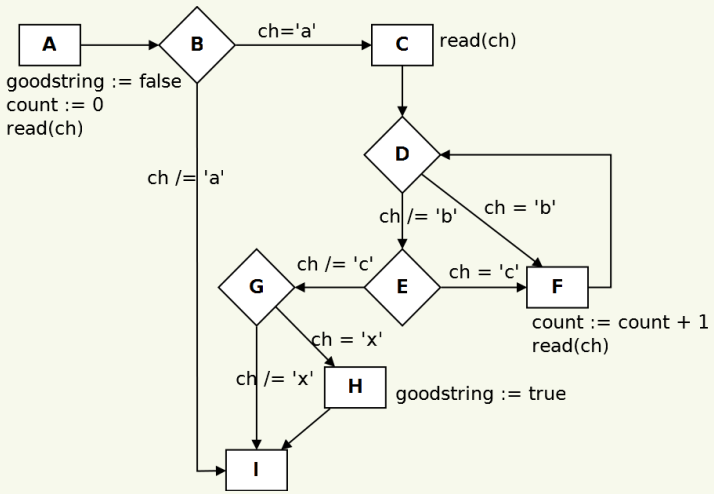


V(G) = 12 arcs - 9 nœuds + 2 = 5

- Choix de la DT “abcx” qui couvre
  a b c d f d e f d e g h i
- Modifier la première décision (*b*)
  Couverture de *abi* (indépendant du précédent) avec la DT “b”
- Modifier la deuxième décision (*d*)
  Couverture de *abcdeghi* (indépendant des 2 précédents) avec la DT “ax”
- Modifier la troisième décision (*d*)
  Couverture de *abcdfdfdeghi* (indépendant des 3 précédents) avec la DT “abbx”
- Modifier la quatrième décision (*e*)
  Couverture de *abcdfdeghi* (pas indépendant avec *abcdfdfdeghi*)

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants



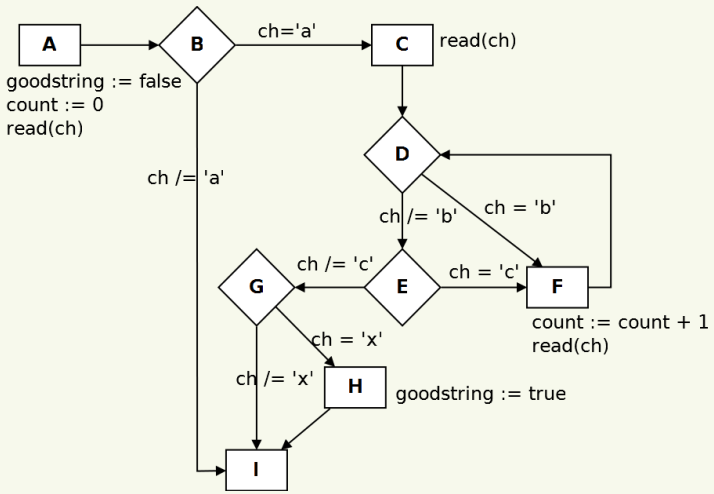
Chemins déjà trouvés : *a b c d f d e f d e g h i*, *abi*, *abcdeghi*, *abcdfdfdeghi*

- ...
- Modifier la cinquième décision (*d*)
  Couverture de *abcdfdefdfdeghi* : pas indépendant avec *abcdfdefdeghi*

V(G) = 12 arcs - 9 nœuds + 2 = 5

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants



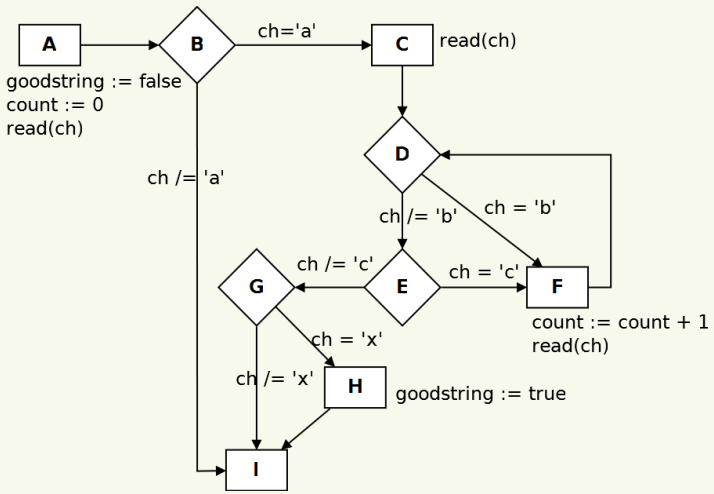
Chemins déjà trouvés : *a b c d f d e f d e g h i*, *abi*, *abcdeghi*, *abcdfdfdeghi*

- ...
- Modifier la cinquième décision (*d*)
  Couverture de *abcdfdefdfdeghi* : pas indépendant avec *abcdfdefdeghi*
- Modifier la sixième décision (*e*)
  Couverture de *abcdfdefdefdeghi* : pas indépendant avec *abcdfdefdeghi*

V(G) = 12 arcs - 9 nœuds + 2 = 5

Les critères tous-les-chemins-indépendants

Calcul des chemins indépendants



V(G) = 12 arcs - 9 nœuds + 2 = 5

Chemins déjà trouvés : *a b c d f d e f d e g h i*, *abi*, *abcdeghi*, *abcdfdfdeghi*

- ...
- Modifier la cinquième décision (*d*)
  Couverture de *abcdfdefdfdeghi* : pas indépendant avec *abcdfdefdeghi*
- Modifier la sixième décision (*e*)
  Couverture de *abcdfdefdefdeghi* : pas indépendant avec *abcdfdefdeghi*
- Modifier la septième décision (*g*)
  Couverture de *abcdfdefdegi* (indépendant des 4 premiers) avec la DT “abca”

Les critères de couverture des PLCS

Définition

Une PLCS est une **Portion Linéaire de Code suivie d'un Saut** (également appelée *Linear Code Sequence And Jump – LCSAJ* dans la littérature). Il s'agit d'une séquence d'instructions entre deux branchements.

Principe

On distingue dans le graphe de contrôle deux types de nœuds.

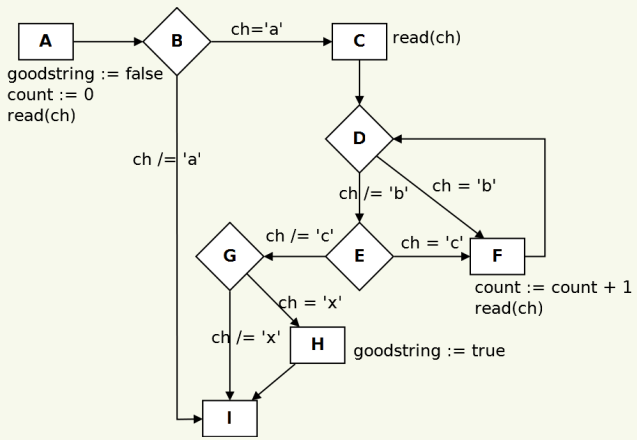
- les nœuds de "saut" : le nœud d'entrée, de sortie et tous les nœuds qui constituent l'arrivée d'un branchement.
- les autres nœuds

Une PLCS est un chemin partant d'un nœud "saut" et arrivant à un nœud "saut" ; les deux derniers nœuds doivent constituer le seul saut du chemin.

Les critères de couverture des PLCS

Repérage des PLCS

Reprenons l'exemple précédent :



Les nœuds "saut" sont les suivants : A (nœud entrée), I (nœud sortie), C, E, F, G, H
Les PLCS sont : ABI, ABC, CDE, CDF, FDE, FDF, EF, EG, GI, GH, HI
On peut tous les couvrir par des DT sensibilisant les chemins : ABI (par ex. "b"), ABCDEFDFDEGHI (par ex. "acbx"), ABCDFDEGI (par ex. "aba")

Les critères de couverture des PLCS

Test Effectiveness Ratio 3

Lorsqu'un jeu de test permet de couvrir **tous les chemins composés d'au moins une PLCS**, on dit qu'il satisfait TER3=1 ou TER3 (Test Effectiveness Ratio 3)

$$TER3 = \frac{\text{nombre de PLCS couvertes}}{\text{nombre de PLCS total}}$$

TER3 = 1 $\Leftrightarrow$ toutes les PLCS sont couvertes

Les critères de couverture des PLCS

Test Effectiveness Ratio 4, 5, ...

En considérant les **chemins composés de 2 PLCS**, on peut si ces chemins sont tous couverts, satisfaire le TER4

$$TER4 = \frac{\text{nombre de chemins composés de 2 PLCS couvertes}}{\text{nombre de chemins composés de 2 PLCS}}$$

Et on peut continuer ...

$$TER5 = \frac{\text{nombre de chemins composés de 3 PLCS couvertes}}{\text{nombre de chemins composés de 3 PLCS}}$$

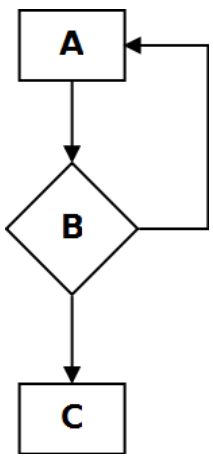
... très loin !

$$TERN = \frac{\text{nombre de chemins composés de (N-2) PLCS couvertes}}{\text{nombre de chemins composés de (N-2) PLCS}}$$

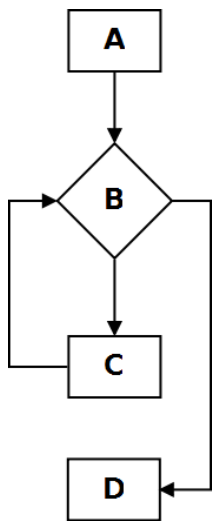
Couverture des chemins limites et chemins intérieurs

Chemins limites et chemins intérieurs

Les **chemins limites** traversent la boucle, mais ne l'itèrent pas.
Les **chemins intérieurs** itèrent la boucle une seule fois.



Chemin limite : *abc*
Chemin intérieur : *ababc*



Chemin limite : *abd*
Chemin intérieur : *abcbd*

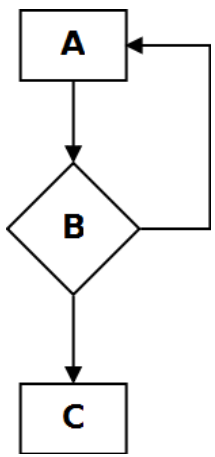
Couverture de tous-les-i-chemins

Définition

Couverture de tous les chemins possibles du graphe passant de 0 à *i* fois dans les boucles du graphe de contrôle.

Place dans la hiérarchie

La couverture de **tous-les-*i*-chemins** pour i > 0 satisfait les critères *tous-les-nœuds* (TER1), *tous-les-arcs* (TER2), et le critère de couverture des chemins limite et intérieurs.



Le jeu de test composée de données de test permettant de couvrir les chemins *abc*, *ababc* et *abababc* satisfait le critère *tous-les-2-chemins*.

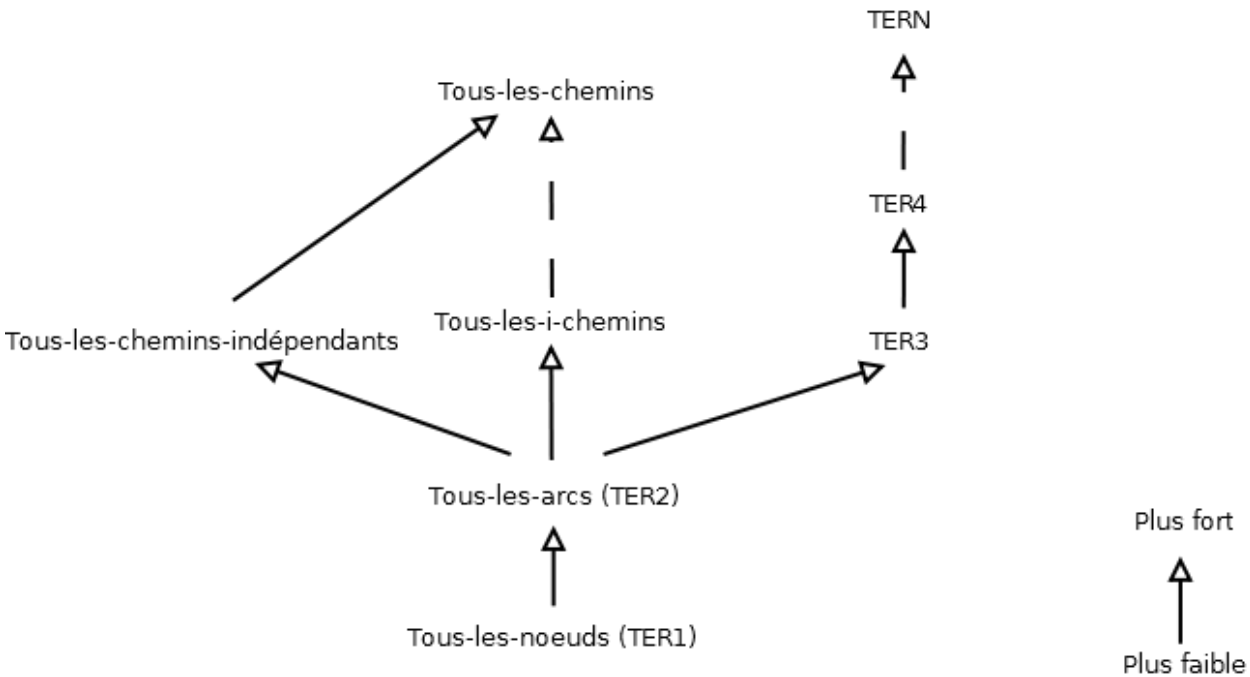
Couverture de tous les chemins

Définition

Choisir $i = n$, avec n = nombre maximal d'itérations possibles, pour le critère *tous-les-i-chemins*, revient à exécuter tous les chemins possible du graphe de flot de contrôle. Ceci permet de satisfaire le critère **tous-les-chemins**.

En pratique, ce critère est rarement envisageable, même sur des applications de taille modeste, en raison de l'explosion du nombre de données de test nécessaire. De même, la valeur de n n'est pas toujours calculable.

Critères de couverture sur le flot de contrôle



Ces critères de couverture sont des indicateurs. Ils fixent les objectifs à atteindre et donnent le critère d'arrêt du test, permettant de savoir si le système a suffisamment été exercé pour être digne de confiance.

Graphe de contrôle - exercice

Soit le programme P_4 suivant :

```
if n <= 0 then
  n := 1 - n

if n mod 2 = 0 then
  n := n / 2
else
  n := 3*n + 1
end if

write (n)
```

Calculer des données de test suivant les critères :

1. tous-les-nœuds
2. tous-les-arcs
3. tous-les-chemins-indépendants

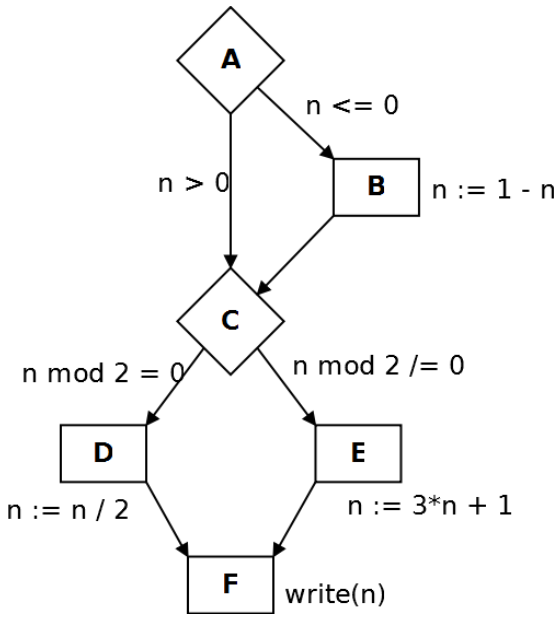
Graphe de contrôle - exercice

Soit le programme P_4 suivant :

```
if n <= 0 then
  n := 1 - n

if n mod 2 = 0 then
  n := n / 2
else
  n := 3*n + 1
end if

write (n)
```



Calculer des données de test suivant les critères :

1. tous-les-nœuds
2. tous-les-arcs
3. tous-les-chemins-indépendants

Tous-les nœuds : $\{n = 0\}$, $\{n = -1\}$
Tous-les arcs : $\{n = 2\}$, $\{n = -2\}$
Tous-les-chemins-indépendants : $\{n = -1\}$, $\{n = -2\}$, $\{n = 2\}$

Le flot de données

Représentation

Le flot de données se représente en annotant le graphe de contrôle par certaines informations sur les manipulations des variables du programme :

- les **définitions** de variables : la valeur de la variable est modifiée (exemple : membre gauche d'une affectation, instruction de lecture, etc.
- les **utilisations** de variables : accès à la valeur de la variable (exemple : membre droit d'une affectation, paramètre d'une instruction d'écriture, indice de tableau, utilisée dans une condition d'instruction conditionnelle, dans une boucle, etc.

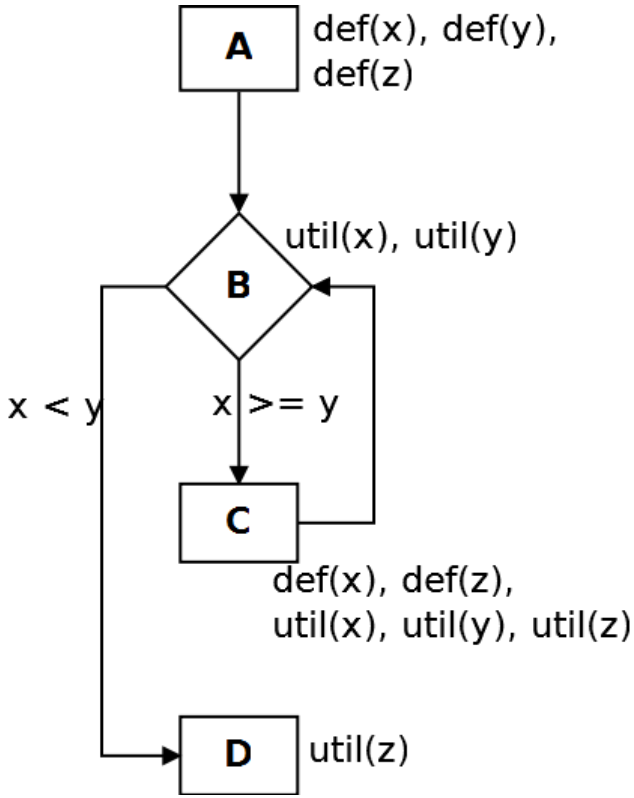
Deux cas d'utilisations

- si une variable est utilisée dans un prédicat d'une instruction de décision (if, while, etc.), on parlera d'une **p-utilisation**
- dans les autres cas (par exemple un calcul), on parlera d'une **c-utilisation**

Définition et utilisation de variables

Soit le programme suivant :

`open(fichier1)`	Définitions : x, y, z
`read(x, fichier1)`	
`read(y, fichier1)`	
`z := 0`	
`while` `x >= y` **`do`**	Utilisations : x, y
`x := x - y`	Définitions : x, z
`z := z + 1`	Utilisations : x, y, z
`done`	
`open(fichier2)`	Utilisations : z
`print(z, fichier2)`	
`close(fichier1)`	
`close(fichier2)`	



On se focalise sur les variables x, y, et z.

Instruction utilisatrice

Notion d'instruction utilisatrice

Une instruction J_2 est **utilisatrice** d'une variable x par rapport à une instruction J_1, si la variable x qui a été définie en J_1 est directement référencée en J_2, c'est-à-dire sans redéfinition de x entre J_1 et J_2.

Exemple d'instruction utilisatrice

Soit le programme suivant :

```
(1)   x := 42
(2)   a := x + 2
(3)   b := a * a
(4)   a := a + 1
(5)   z := x + a
```

- L'instruction (5) est utilisatrice de `x` par rapport à l'instruction (1)
- L'instruction (5) est utilisatrice de `a` par rapport à l'instruction (4)

Instruction utilisatrice - exercice

Exemple d'instruction utilisatrice

Pour chaque instruction utilisatrice, dire si celle-ci est p-utilisatrice ou c-utilisatrice, pour quelle variable et par rapport à quelle autre instruction

```
(1)   read(i)
(2)   s := 0
(3)   while (i <= 3) do
(4)     if a[i] > 0 then
(5)       s := s + a[i]
(6)     end if
(7)     i := i + 1
(8)   done
```

Instruction utilisatrice - exercice

Exemple d'instruction utilisatrice

Pour chaque instruction utilisatrice, dire si celle-ci est p-utilisatrice ou c-utilisatrice, pour quelle variable et par rapport à quelle autre instruction

```
(1)   read(i)
(2)   s := 0
(3)   while (i <= 3) do
(4)     if a[i] > 0 then
(5)       s := s + a[i]
(6)     end if
(7)     i := i + 1
(8)   done
```

Instr.	p-/c-util.	var.	par rapport à
(3)	p-utilisation	i	(1) ou (7)
(4)	c-utilisation	i	(1) ou (7)
(5)	c-utilisation	i	(1) ou (7)
(5)	c-utilisation	s	(2) ou (5)
(7)	c-utilisation	i	(1) ou (7)

Chemin d'utilisation

Définition

Un chemin d'utilisation relie l'instruction de définition d'une variable à une instruction utilisatrice. Ce chemin est appelé **dr-strict**.

Exemple de chemin d'utilisation

Soit le programme suivant :

```
(1)   x := 42
(2)   a := x + 2
(3)   b := a * a
(4)   a := a + 1
(5)   z := x + a
```

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

Critères de couverture du flot de données

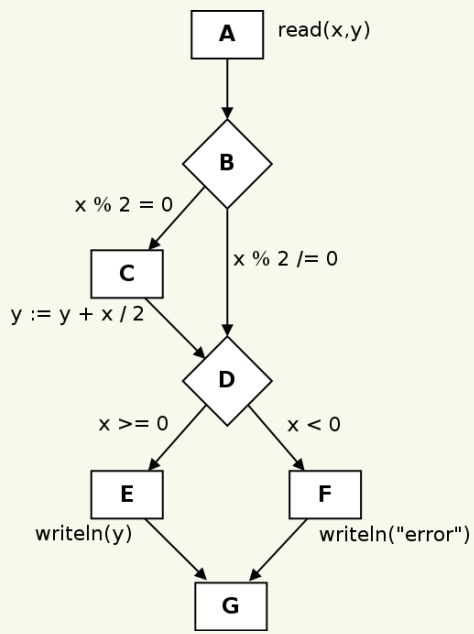
- toutes-les-définitions
- toutes-les-utilisations
- toutes-les-p-utilisations/quelques-c-utilisations
- toutes-les-c-utilisations/quelques-p-utilisations
- tous-les-DU-chemins

Couverture de toutes-les-définitions

Définition

Pour chaque définition, il y a au moins un chemin dr-strict pour un test.

Couverture toutes-les-définitions



Chemins dr-stricts :

- pour x (défini en *a*) : *ab* (p-utilisation en *b*), *abc* (c-utilisation en *c*), *abd* ou *abcd* (p-utilisation en *d*)
- pour y (défini en *a*) : *abc* (c-utilisation en *c*), *abde* (c-utilisation en *e*)
- pour y (défini en *c*) : *cde* (c-utilisation en *e*)

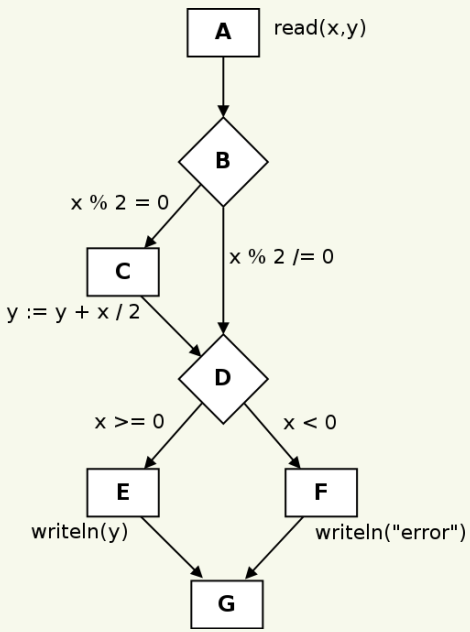
Couverture du critère toutes-les-définitions : *abcdeg*

Couverture de toutes-les-utilisations

Définition

Pour chaque définition, et pour chaque utilisation accessible depuis cette définition, les tests produits couvrent toutes les utilisations (nœuds c-utilisateurs ou arcs p-utilisateurs).

Couverture toutes-les-utilisations



Définitions/utilisations :

- pour x, défini en *a* : p-utilisation en *bc* et *bd*, c-utilisation en *c*, p-utilisation en *de* et *df*
- pour y, défini en *a* : c-utilisation en *c*, c-utilisation en *e*
- pour y, défini en *c* : c-utilisation en *e*

Couverture du critère toutes-les-utilisations : *abcdeg*, *abdfg*, *abdeg*

Autres critères basés sur le flot de données

Le critère toutes-les-utilisations nécessite souvent la sensibilisation d'un grand nombre de chemins. Pour pallier ce problème, deux critères intermédiaires entre toutes-les-utilisations et toutes-les-définitions sont proposés.

Critère toutes-les-p-utilisations/quelques-c-utilisations

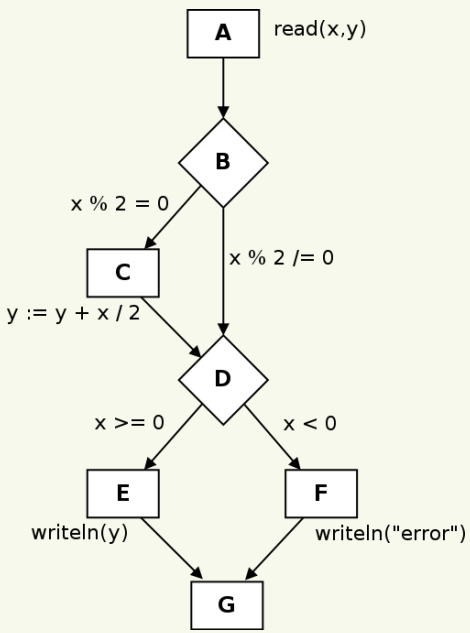
Pour chaque définition, et pour chaque p-utilisation accessible à partir de cette définition, et pour chaque branche issue de la condition associée, il y a un chemin dr-strict prolongé par le premier segment de cette branche.
S'il n'y a aucune p-utilisation, il suffit d'avoir un chemin dr-strict entre la définition et l'une des c-utilisations.

Critère toutes-les-c-utilisations/quelques-p-utilisations

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-utilisations.

Couverture de toutes-les-utilisations

Couverture toutes-les-p-utilisations/quelques-c-utilisations



Définitions/utilisations :

- pour x, défini en *a* : p-utilisation en *bc* et *bd*, c-utilisation en *c*, p-utilisation en *de* et *df*
- pour y, défini en *a* : c-utilisation en *c*, c-utilisation en *e*
- pour y, défini en *c* : c-utilisation en *e*

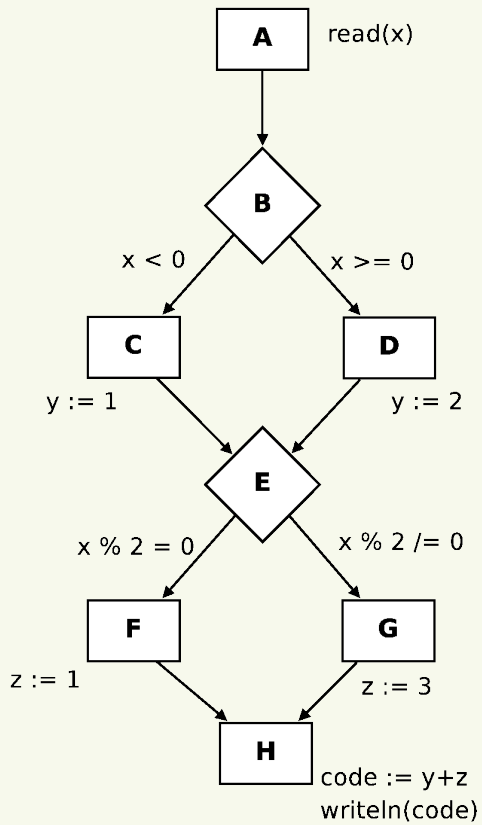
Chemins dr-stricts vers p-utilisations :

- pour x défini en *a* : *ab* (p-utilisation en *b*), *abd* ou *abcd* (p-utilisation en *d*)

Couverture du critère toutes-les-p-utilisations/quelques-c-utilisations :
abdeg, *abcdfg*

Couverture de toutes-les-utilisations

Couverture de toutes-les-utilisations



Définitions/utilisations :

- pour x, défini en *a* : p-utilisation en *bc* et *bd*, p-utilisation en *ef* et *eg*
- pour y, défini en *c* : c-utilisation en *h*
- pour y, défini en *d* : c-utilisation en *h*
- pour z, défini en *f* : c-utilisation en *h*
- pour z, défini en *g* : c-utilisation en *h*

Couverture du critère toutes-les-utilisations : *abcefh*, *abdegh*

Or, *h* est utilisatrice de *c* pour la variable y de deux manières
$\Rightarrow$ critère tous-les-DU-chemins

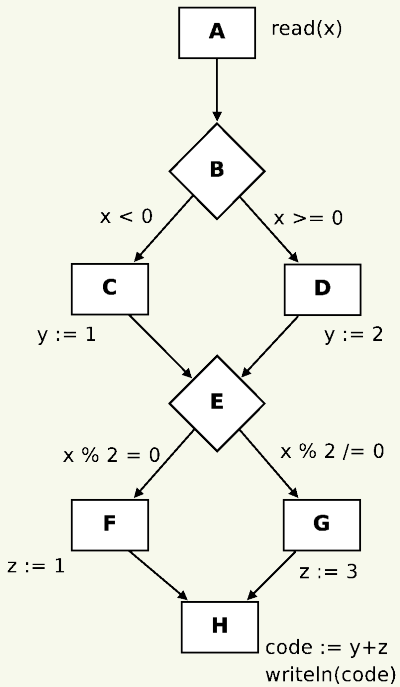
Couverture de tous-les-DU-chemins

Définition

Le critère **tous-les-DU-chemins** rajoute au critère toutes-les-utilisations le fait de devoir couvrir tous les chemins possibles entre la définition et la référence, en se limitant aux chemins sans cycles.

Couverture de tous-les-DU-chemins

Couverture de tous-les-DU-chemins



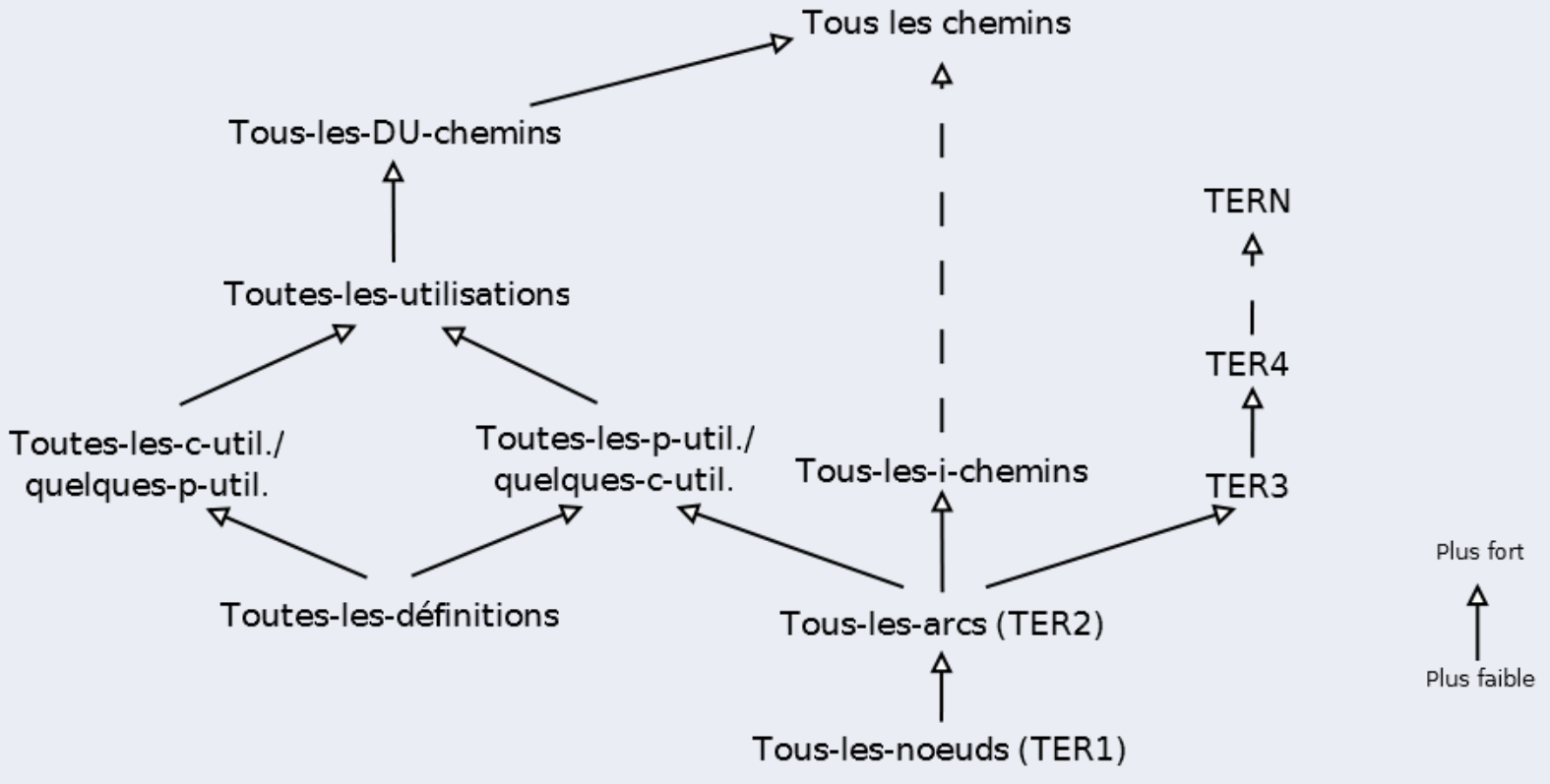
Définitions/utilisations :

- pour x, défini en *a* : p-utilisation en *bc* et *bd*, p-utilisation en *ef* et *eg*
- pour y, défini en *c* : c-utilisation en *h*
- pour y, défini en *d* : c-utilisation en *h*
- pour z, défini en *f* : c-utilisation en *h*
- pour z, défini en *g* : c-utilisation en *h*

Couverture du critère tous-les-DU-chemins : *abcefh*, *abcegh*, *abdefh*, *abdegh*

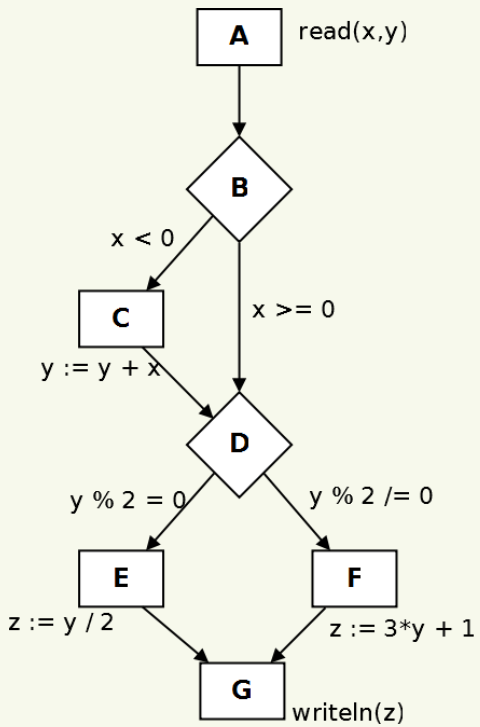
Bilan

Hiérarchie des critères vus précédemments



Couverture du flot de données – Exercice

Couverture de toutes-les-utilisations

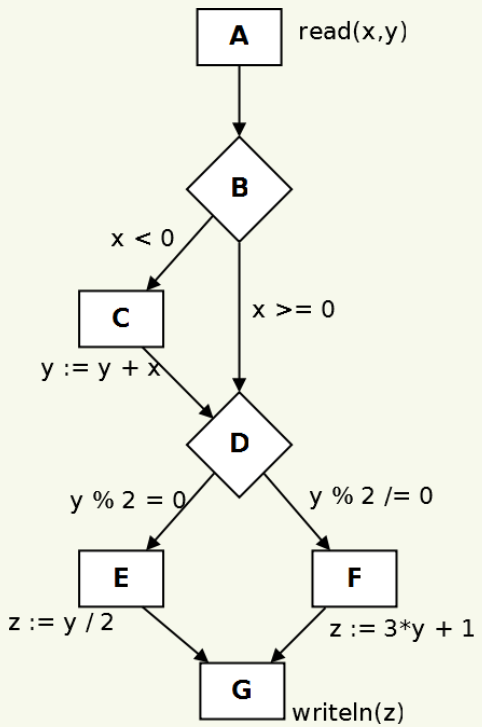


Produire des données de tests pour les critères :

- toutes-les-définitions
- toutes-les-utilisations

Couverture du flot de données – Exercice

Couverture de toutes-les-utilisations



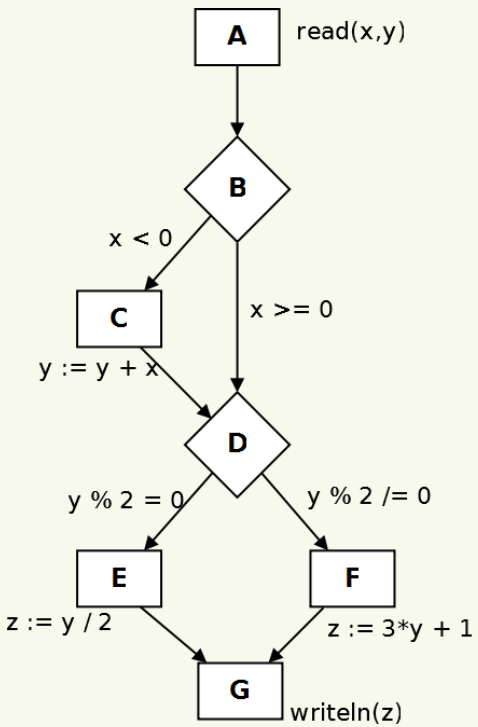
Chemins dr-stricts :

- pour x, défini en *a* : *ab* (p-utilisation en *b*), *abc* (c-utilisation en *c*)
- pour y, défini en *a* : *abc* (c-utilisation en *c*), *abd* (p-utilisation en *d*), *abde* (c-utilisation en *e*), *abdf* (c-utilisation en *f*)
- pour y, défini en *c* : *cd* (p-utilisation en *d*), *cde* (c-utilisation en *e*), *cdf* (c-utilisation en *f*)
- pour z, défini en *e* : *eg* (c-utilisation en *g*)
- pour z, défini en *f* : *fg* (c-utilisation en *g*)

Couverture du critère toutes-les-définitions :
abdeg par $DT_1 = \{x = 1, y = 2\}$
abcdfg par $DT_2 = \{x = -1, y = 2\}$

Couverture du flot de données – Exercice

Couverture de toutes-les-utilisations



Définitions/utilisations :

- pour x, défini en *a* : p-utilisation en *bc* et *bd*, c-utilisation en *c*
- pour y, défini en *a* : c-utilisation en *c*, p-utilisation en *de* et *df*, c-utilisation en *e*, c-utilisation en *f*
- pour y, défini en *c* : p-utilisation en *de* et *df*, c-utilisation en *e*, c-utilisation en *f*
- pour z, défini en *e* : c-utilisation en *g*
- pour z, défini en *f* : c-utilisation en *g*

Couverture du critère toutes-les-utilisations
abcdeg par $DT_1 = \{x = -1, y = 3\}$
abcdfg par $DT_2 = \{x = -1, y = 2\}$
abdeg par $DT_3 = \{x = 1, y = 2\}$
abdfg par $DT_4 = \{x = 1, y = 1\}$

Evaluation par mutation

Principe

L'idée est de considérer des **variantes** du programme d'origine, appellées **mutants**, qui ne diffèrent que par une instruction.

Mutation d'une instruction

Pour l'instruction :

```
if x > 8 then x := y ...
```

on peut considérer les mutants suivants :

- `if x < 8 then x := y ...`
- `if x >= 8 then x := y ...`
- `if x > 10 then x := y ...`
- `if x > 8 then x := y+1 ...`
- `if x > 8 then x := x ...`
- `etc.`

Evaluation par mutation

Création des mutants

Chaque mutant est issu d'une **transformation syntaxique élémentaire** du programme d'origine.

Dans la pratique

Cette technique vise principalement à **évaluer les données de test** par rapport aux fautes les plus probables qui ont été envisagées. (issues d'un **modèle de fautes à définir**). Le modèle de fautes doit être réaliste (*competent programmer hypothesis*).

Evaluation par mutation

Score mutationnel

Pour un programme P, soit $M(P)$ l'ensemble de tous les mutants obtenus par le modèle de faute. Certains mutants peuvent avoir le même comportement que P, notés $E(P)$. On fait exécuter la suite de tests T à chacun des mutants de $M(P)$ et on note $DM(P)$ l'ensemble de ceux qui ne fournissent pas le même résultat que P (ils sont "tués" par la suite de test).

Le score mutationnel $MS(P, T)$ se calcule par le ratio :

$$MS(P, T) = \frac{DM(P)}{M(P) - E(P)}$$

Evaluation par mutation

Evaluation de la qualité d'une suite de tests

Le test mutationnel est plus une approche pour calculer la pertinence d'une suite de tests, en comptant combien de mutants fonctionnellement distinguables du programme d'origine sont "tués".
N.B. l'utilisation de la mutation pour guider la génération des tests en boîte blanche ne présente que peu d'intérêt.

Cette méthode est coûteuse à mettre en œuvre en particulier pour définir des mutants qui ne soient pas équivalents fonctionnellement au programme d'origine.

Et maintenant...

A vous de jouer !