

Test structurel ou boîte blanche

Plan

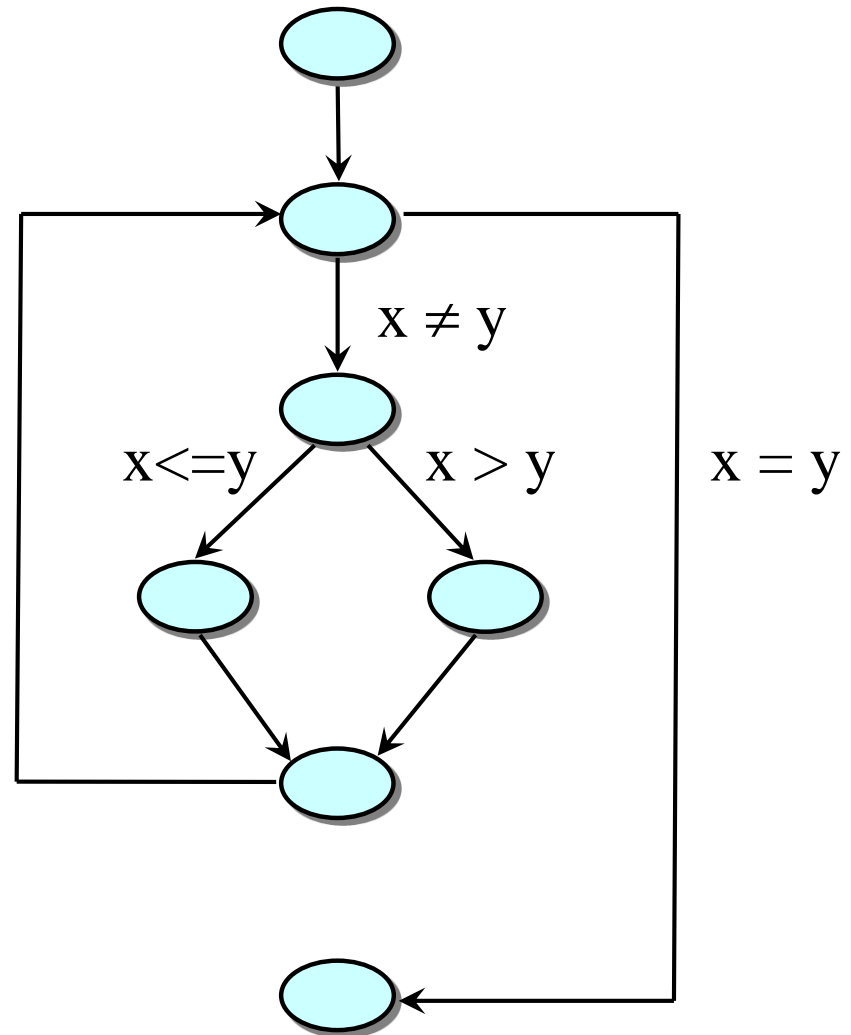
- ❑ Couverture du flot de contrôle :
 - instruction, arête, condition, couverture du chemin.
- ❑ Couverture du flot de données :
 - définitions-usages des données.
- ❑ Analyse de la couverture des données.
- ❑ Test de mutation.

Définitions élémentaires

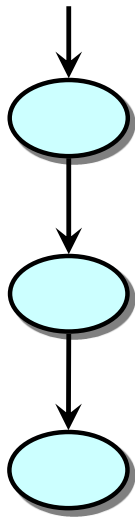
- ❑ Un graphe de flot de contrôle est une représentation sous forme de graphe orienté de tous les chemins possibles dans un programme.
- ❑ Les nœuds sont des blocs d'instructions séquentiels.
- ❑ Les arêtes sont des transferts de contrôle.
- ❑ Les arêtes peuvent être étiquetées avec un attribut représentant la condition du transfert de contrôle.

Graphe de flot de contrôle

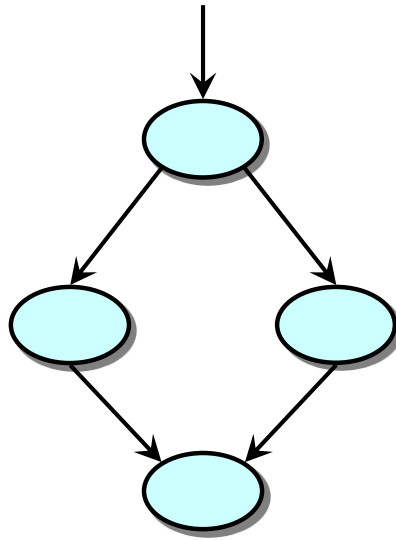
```
read(x);  
read(y);  
while  $x \neq y$  loop  
  if  $x > y$  then  
     $x := x - y;$   
  else  
     $y := y - x;$   
  end if;  
end loop;  
 $\text{gcd} := x;$ 
```



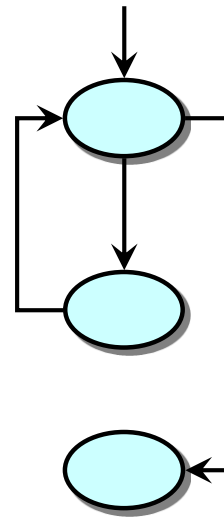
Principes du CFG



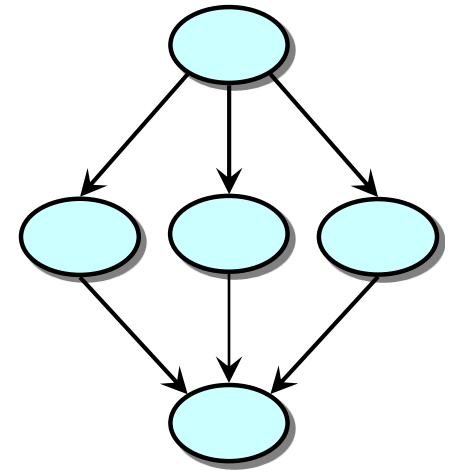
Séquence



Si-Alors-Sinon
(If-Then-Else)

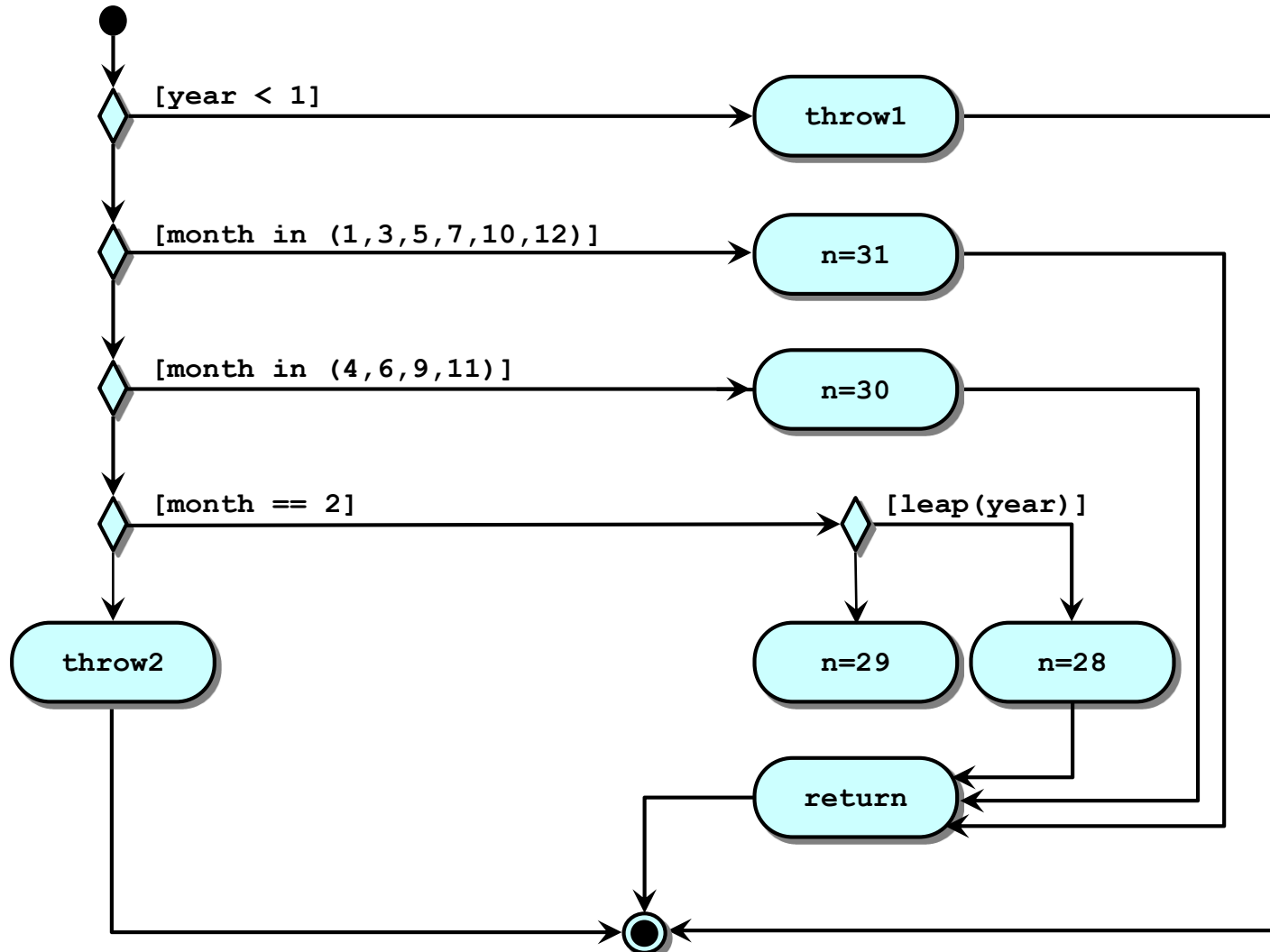


Boucle tant que
(While loop)



Commutateur
(Switch)

Alternative: Diagramme d'activité UML



Instruction/Couverture des nœuds

❑ **Couverture d'instruction :**

les anomalies ne peuvent pas être découvertes si les parties les contenant ne sont pas exécutées.

❑ Équivaut à couvrir tous les nœuds dans le CFG.

❑ En général, plusieurs entrées exécutent les mêmes instructions → une question importante en pratique:

Pouvons-nous minimiser les cas de test ?

Incomplétude

- ❑ La couverture des instructions peut mener à l'incomplétude.

```
if x < 0 then
    x := -x;
end if
z := 1/x;

-----


if x < 0 then
    x := -x;
else
    null;
end if
z := 1/x;
```


Un $x < 0$ couvre toutes les instructions.
Mais le cas $x \geq 0$ n'est pas pris en compte.
Le code implicite (en italique et rouge) n'est pas couvert.
Ne rien faire pour le cas $x \geq 0$ peut s'avérer faux, voire dangereux et devrait être testé.

Couverture des arêtes

- ❑ Utiliser la structure de programme, le graphe de flot de contrôle (CFG).
- ❑ **Critère de la couverture des arêtes :**
sélectionner un ensemble de tests T de telle façon qu'en exécutant P pour chaque cas de test présent dans T , chaque arête du graphe du flot de contrôle P ait été traversée au moins une fois.
- ❑ Exercer toutes les conditions qui gouvernent le flot de contrôle du programme avec des valeurs vraies et fausses.

Limite du Méthode: Exemple de code

```
counter:= 0;
found := false;
if number_of_items  $\neq$  0 then counter :=1;
    while (not found) and counter  number_of_items loop
        if table(counter) = desired_element then
            found := true;
        end if;
        counter := counter + 1;
    end loop;
end if;
if found then write ("the desired element exists in
                        the table");
else write ("the desired element does not exists in
            the table");
end if;
```



Jeu de tests

- ❑ Nous choisissons un jeu de tests avec une table de 0 item et une table de 3 items, le second étant celui désiré ($|T| = 2$).
- ❑ Pour le second cas de tests, le corps de la boucle est exécuté deux fois, quand la branche **then** est exécutée.
- ❑ Le critère de la couverture des arêtes est vérifié et l'erreur n'a pas été découverte par le jeu de tests.
- ❑ **Toutes** les valeurs possibles des *éléments de la condition* de la **boucle while loop** n'ont pas été exercées.

Couverture des conditions

- ❑ Plus ample renforcement de la couverture des arêtes.
- ❑ **Critère de la couverture des conditions :**
sélectionner un jeu de tests T de telle façon que, en exécutant P pour chaque élément de T, chaque arête du graphe du flot de contrôle P est traversé, et ***toutes les valeurs des éléments des conditions combinées sont traversés au moins une fois (all possible values!).***
- ❑ **Conditions combinées :**
C1 **et** C2 **ou** C3 ... où les Ci sont des expressions relationnelles ou des variables booléennes (conditions atomiques).
- ❑ **Couverture modifiée des conditions :**
seulement les combinaisons de valeurs de telle façon que Ci conduit aux deux valeurs de la condition générale (vrai et faux).

Non couverture des arêtes masquées

```
if c1 and c2 then
  st;
else
  sf;
end if;
```

```
if c1 then
  if c2 then
    st;
  else
    sf;
  end if;
else
  sf;
end if;
```

- ❑ Deux programmes équivalents
 - quoique vous écrieriez celui de gauche.
- ❑ Couverture des arêtes
 - ne couvrirait pas obligatoirement les arêtes “masquées”,
 - ex.: $C2 = \text{false}$ peut ne pas être couvert.
- ❑ La couverture des conditions peut le faire.

Couverture des conditions

❑ **Couverture des conditions :**

sélectionner un jeu de tests T de telle façon qu'en exécutant P pour chaque élément de T, chaque condition atomique ait été évaluée à Vrai et Faux.

❑ **Couverture des conditions / décisions :**

sélectionner un jeu de tests T de telle façon qu'en exécutant P pour chaque élément de T, chaque arête du graphe du flot de contrôle P ait été traversée, et chaque condition atomique ait été évaluée à Vrai et Faux.

❑ **Couverture modifiée des conditions/décisions (MC/DC)**

Renforce la couverture des conditions décisions. Requiert uniquement les combinaisons de valeurs telles que la condition à tester affecte de manière indépendante la décision finale.

* MC/DC Cause Unique = RACC, MC/DC Masquant = CACC

Exemple

- ❑ Le standard international DO-178B pour la certification des systèmes aéroportés (1992).
- ❑ Exemple : $A \wedge (B \vee C)$

	ABC	Res.	Corr. cas faux
1	TTT	T	A (5)
2	TTF	T	A (6), B (4)
3	TFT	T	A (7), C (4)
4	TFF	F	B (2), C (3)
5	FTT	F	A (1)
6	FTF	F	A (2)
7	FFT	F	A (3)
8	FFF	F	-

MC/DC Cause Unique

Prendre une paire pour chaque élément :

- A : (1,5), or (2,6), or (3,7)
- B : (2,4)
- C : (3,4)

Deux ensembles minimum pour couvrir le critère de condition modifié :

- $\{2,3,4\} \cup x \in \{6, 7\}$

Cela fait 4 cas de tests (au lieu de 8 pour chaque combinaison possible).

Couverture des chemins

- ❑ **Critère de couverture des chemins :**
sélectionner un test T tel qu'en exécutant P pour chaque élément de T, tous les chemins conduisant du nœud initial au nœud final du graphe de flot de contrôle de P soient traversés.
- ❑ En pratique, le nombre de chemins est trop grand, sinon infini.
- ❑ Certains chemins sont infaisables.
- ❑ C'est l'une des clés pour déterminer les "chemins critiques".

Exemple: Toutes les arêtes vs. tous les chemins...

```
if x ≠ 0 then  
    y := 5;  
else  
    z := z - x;  
end if;  
if z > 1 then  
    z := z / x;  
else  
    z := 0;  
end if;
```

$T1 = \{ \langle x=0, z=1 \rangle, \langle x=1, z=3 \rangle \}$
exécute toutes les arêtes mais ne
montre pas un risque de division par 0.

$T2 = \{ \langle x=0, z=3 \rangle, \langle x=1, z=1 \rangle \}$
trouverait le problème en exerçant les
flot de contrôle restants possibles à
travers le fragment du programme.

$T1 \cup T2 \rightarrow$ tous les chemins couverts.

Cas des boucles

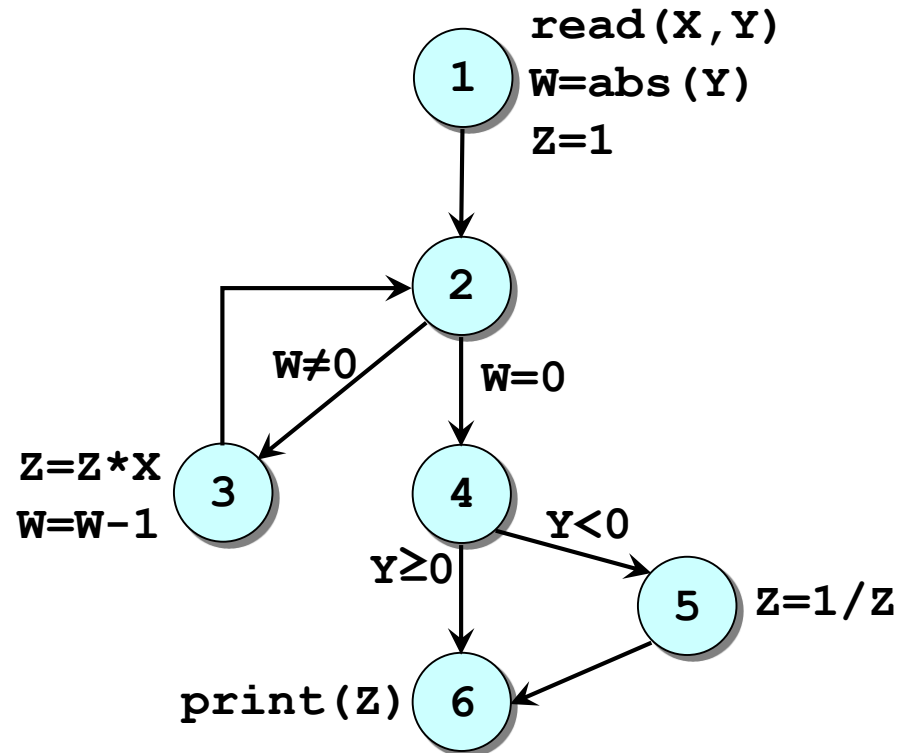
- ❑ Chercher les conditions qui exécutent les boucles :
 - zéro fois,
 - un nombre maximum de fois,
 - un nombre moyen de fois (critère statistique).

- ❑ Par exemple, dans un algorithme de recherche d'un élément dans une table, on peut :
 - sauter une boucle (la table est vide),
 - exécuter la boucle une ou deux fois et par la suite trouver l'élément,
 - chercher dans toute la table sans trouver l'élément désiré.

Autre exemple : la fonction puissance

Programme estimé $Z=X^Y$

```
BEGIN
  read (X, Y) ;
  W = abs(Y) ;
  Z = 1 ;
  WHILE (W <> 0) DO
    Z = Z * X ;
    W = W - 1 ;
  END
  IF (Y < 0) THEN
    Z = 1 / Z ;
  END
  print (Z) ;
END
```



Exemple : Test du flot de contrôle

□ Tous les chemins.

➤ Chemin infaisable :

✓ $1 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6$

➤ Nombre infini de chemins :

✓ Autant de chemins à réitérer :

$2 \rightarrow (3 \rightarrow 2)^*$ que la valeur de $\text{Abs}(Y)$ $[W]$

□ Toutes les branches.

➤ Deux cas de tests suffisent :

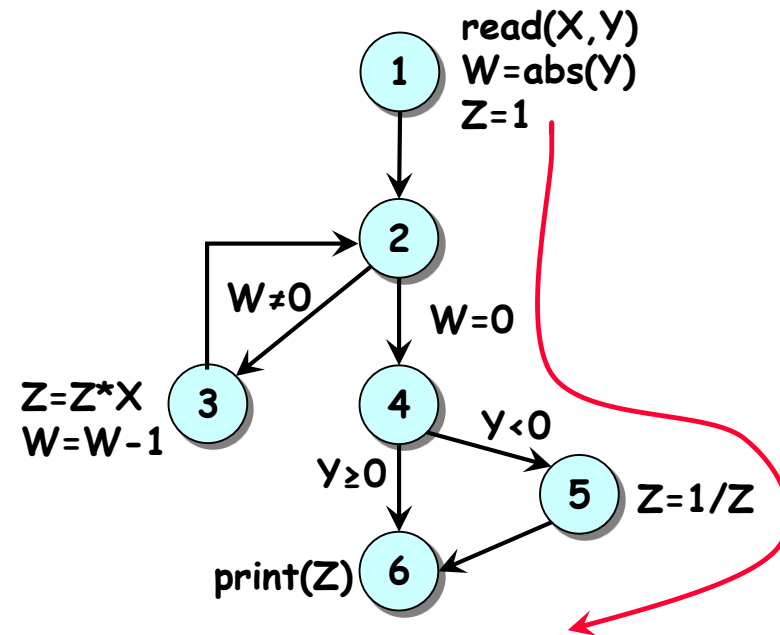
✓ $Y < 0$: $1 \rightarrow 2 \rightarrow (3 \rightarrow 2)^+ \rightarrow 4 \rightarrow 5 \rightarrow 6$

✓ $Y \geq 0$: $1 \rightarrow 2 \rightarrow (3 \rightarrow 2)^* \rightarrow 4 \rightarrow 6$

□ Toutes les instructions.

➤ Un cas de test suffit :

✓ $Y < 0$: $1 \rightarrow 2 \rightarrow (3 \rightarrow 2)^+ \rightarrow 4 \rightarrow 5 \rightarrow 6$



Dériver des valeurs d'entrée

- ❑ Il arrive, dans les programmes du monde réel, que certaines instructions ne soient pas atteignables
- ❑ Ce n'est pas toujours possible de décider automatiquement si une instruction est atteignable, ni même le pourcentage des instructions atteignables.
- ❑ Quand on n'atteint pas une couverture de 100%, il est parfois très difficile d'en déterminer la raison.
- ❑ Des outils sont nécessaires pour supporter cette activité et la recherche est très active quant à la proposition d'algorithmes efficaces pour dériver automatiquement des jeux de tests présentant des bons niveaux de couverture.
- ❑ Le test de flot de contrôle est, en général, plus adapté pour tester les petits cas.

Plan

- ❑ Couverture du flot de contrôle :
 - instruction, arête, condition, couverture du chemin.
- ❑ Couverture du flot de données :
 - définitions-usages des données.
- ❑ Analyse de la couverture des données.
- ❑ Test de mutation.
- ❑ Test d'intégration :
 - Stratégies,
 - critères.
- ❑ Conclusions :
 - génération des données de test, outils, recommandations de Marick.

Analyse du flot de données

- ❑ S'intéresse aux chemins du CFG qui sont significatifs pour le flot de données dans le programme.
- ❑ Se concentre sur l'affectation des valeurs aux variables et à leurs usages.
- ❑ Analyse des occurrences des variables.
 - L'occurrence de définition : la valeur est liée à la variable.
 - Les occurrences d'utilisation : la valeur de la variable est référée.
 - ✓ L'utilisation comme prédicat : la variable est utilisée pour décider si le prédicat est vrai.
 - ✓ L'utilisation de calcul : la variable est utilisée pour définir d'autres variables ou calculer une valeur (possiblement de sortie).

Exemple factoriel

```
1. public int factorial(int n) {  
2.     int i, result = 1;  
3.     for (i=2; i<=n; i++) {  
4.         result = result * i;  
5.     }  
6.     return result;  
7. }
```

Variable	Ligne de définition	Ligne d'utilisation
n	1	3
résultat	2	4
résultat	2	6
résultat	4	4
résultat	4	6

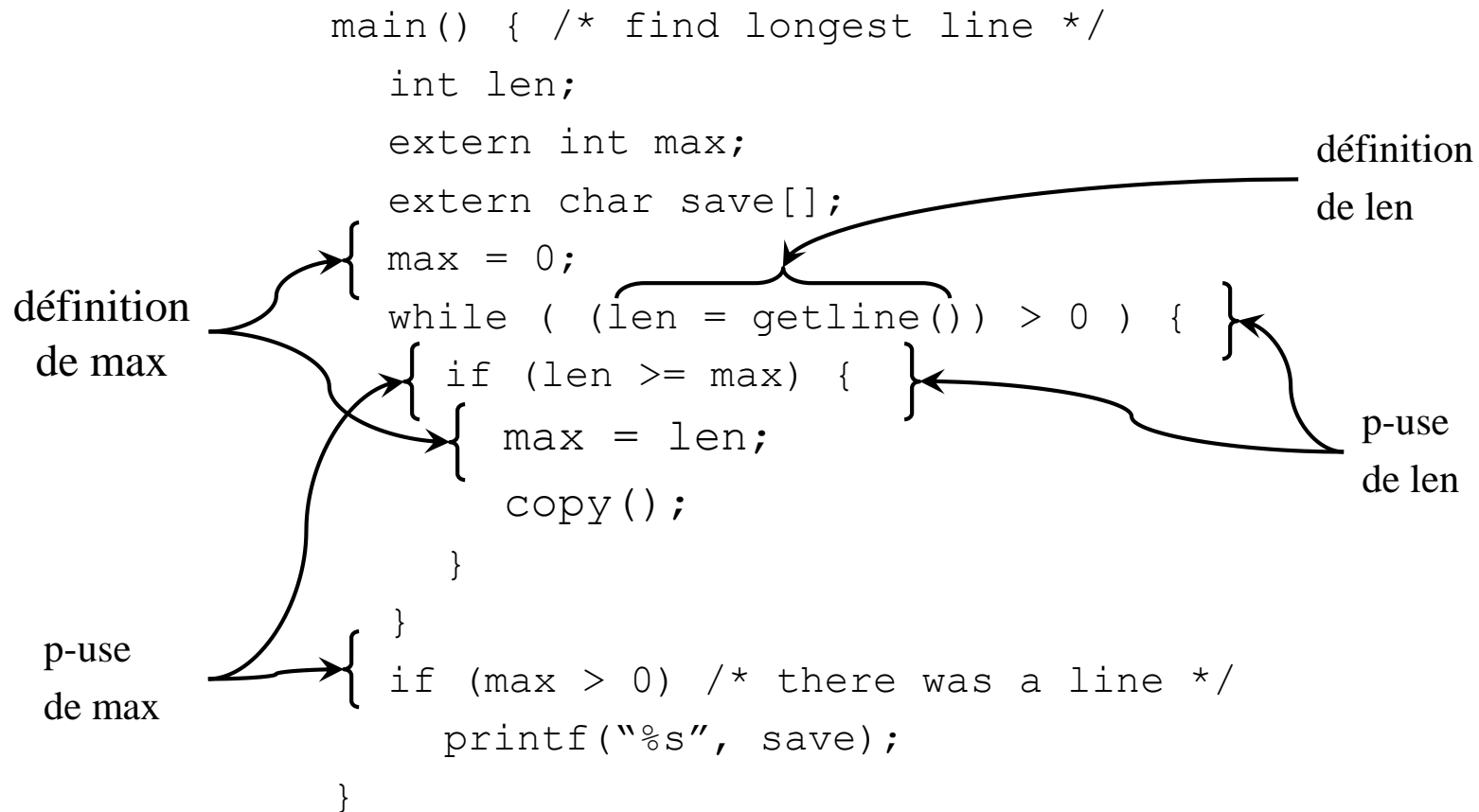
Définitions élémentaires

- ❑ Le nœud $n \in \text{CFG}(P)$ est un **nœud de définition** de la variable $v \in V$, écrit comme $\text{DEF}(v,n)$, ssi la valeur de la variable v est définie dans l'instruction correspondant au nœud n .
- ❑ Le nœud $n \in \text{CFG}(P)$ est un **nœud d'utilisation** de la variable $v \in V$, écrit comme $\text{USE}(v,n)$, ssi la valeur de la variable v est utilisée dans l'instruction correspondant au nœud n .
- ❑ Un nœud utilisé $\text{USE}(v,n)$ est une **utilisation comme prédicat** (représenté comme P-Use) ssi l'instruction n est une instruction prédicat, sinon $\text{USE}(v,n)$ est une **utilisation de calcul** (représentée comme C-use).

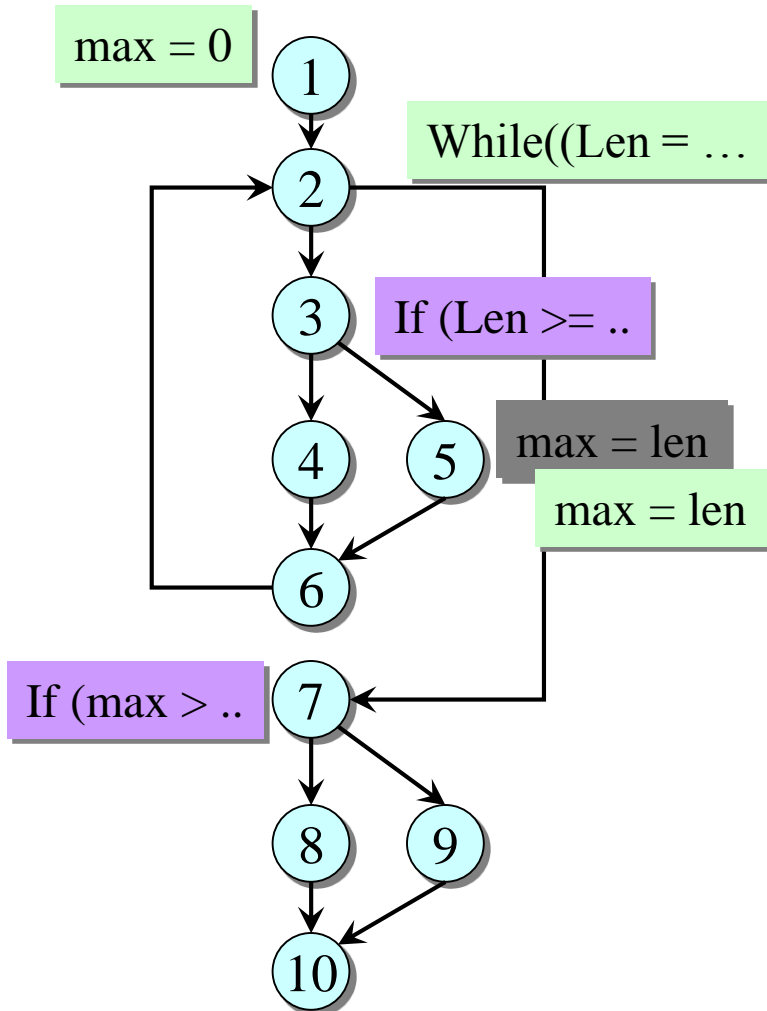
Définitions élémentaires II

- Un **(sous)chemin definition-use** d'une variable v (notée du-path) est un (sous)chemin dans $\text{PATHS}(P)$ tel que, pour $v \in V$, il y a un nœud de définition $\text{DEF}(v, m)$ et un nœud d'utilisation $\text{USE}(v, n)$ avec m et n sont les nœuds initial et final du (sous) chemin.
- Un **(sous)chemin definition-clear** d'une variable v (notée dc-path) est un chemin definition-use de $\text{PATHS}(P)$ avec un nœud initial $\text{DEF}(v, m)$ et un nœud final $\text{USE}(v, n)$ de telle manière qu'aucun autre nœud du chemin ne soit un nœud de définition de v .

Exemple simple



Exemple simple (CFG)



<code>v = ...</code>	Définition DEF(v, n)
<code>v >= ...</code>	P-use USE(v, n)
<code>... = ...v ...</code>	C-use USE(v, n)

- DEF(max, 1)
- DEF(len, 2)
- DEF(max, 5)
- C-USE(len, 5)
- P-USE(len, 2)
- P-USE(len, 3)
- P-USE(max, 3)
- P-USE(max, 7)

Définitions formelles de critères I

- ❑ L'ensemble T satisfait le critère **all-Definitions** pour le programme P ssi, pour chaque variable $v \in V$, T contient au moins un chemin definition-clear à partir de chaque nœud de définition de v à un nœud d'utilisation de v .
- ❑ L'ensemble T satisfait le critère **all-Uses** pour le programme P ssi, pour chaque variable $v \in V$, T contient au moins un chemin definition-clear partant de chaque nœud de définition de v à chaque nœud d'utilisation atteignable de v .
- ❑ L'ensemble T satisfait le critère **all-P-Uses/Some C-Uses** pour le programme P , ssi pour chaque variable $v \in V$, T contient au moins un chemin definition-clear partant de chaque nœud de définition de v à chaque utilisation prédicat de v , et si une définition de v n'a pas de P-Uses, il y a un chemin definition-clear à au moins une utilisation calcul.

Définitions formelles de critères II

- ❑ L'ensemble T satisfait le critère **all-C-Uses/Some P-Uses** pour le programme P ssi, pour chaque variable $v \in V$, T contient au moins un chemin definition-clear partant de chaque nœud définition de v à chaque nœud d'utilisation de calcul de v , et si une définition de v n'a pas de C-Uses, il y a un chemin definition-clear à au moins une utilisation prédicat de v .
- ❑ L'ensemble T satisfait le critère **all-DU-Paths** pour un programme P ssi, pour chaque variable $v \in V$, T contient tous les chemins definition-clear partant de chaque nœud de v à chaque nœud d'utilisation de v atteignable, et que ces chemins sont soit des traverses simples de boucles, soit ils ne contiennent pas de cycles.

Petit programme

Programme estimé $Z=X^Y$

```
BEGIN
  read (X, Y) ;
  W = abs(Y) ;
  Z = 1 ;
  WHILE (W <> 0) DO
    Z = Z * X ;
    W = W - 1 ;
  END
  IF (Y < 0) THEN
    Z = 1 / Z ;
  END
  print (Z) ;
END
```

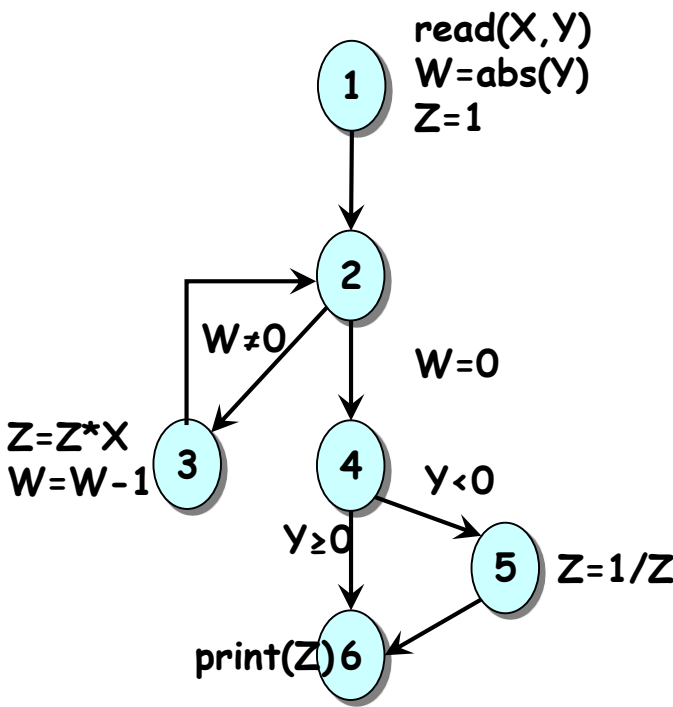
Exemple de puissance (w,1-2,1-3)

Dcu = Definition c-use (clear)

Dpu = Definition p-use (clear)

nœud i	def(i)	c-use(i)	p-use(i,j)
1	X, Y, W, Z	Y	
2			W
3	W, Z	X, W, Z	
4			Y
5	Z	Z	
6		Z	

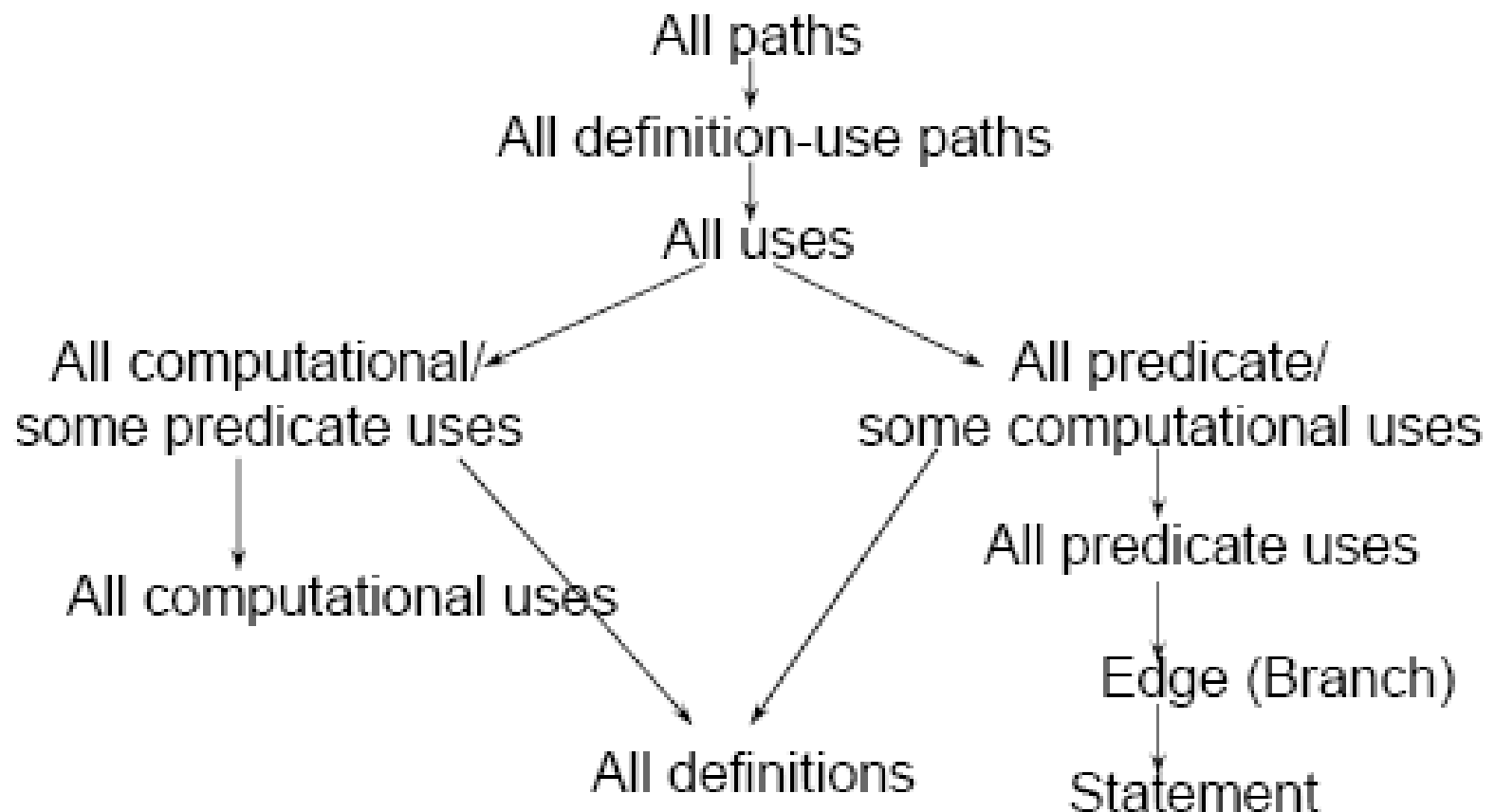
nœud i	dcu(i,v)	dpu(i,v)	du(i,v)
1	dcu(1,X) = {3} dcu(1,Z) = {3,6} dcu(1,W) = {3}	dpu(1,Y) = {4} dpu(1,W) = {2}	du(1,W)={2,3} du(1,X)={3} du(1,Y)={4} du(1,Z)={3,6}
3	dcu(3,W) = {3} dcu(3,Z) = {3,5,6}	dpu(3,W) = {2}	du(3,W)={2,3} du(3,Z)={3,5,6}
5	dcu(5,Z) = {6}		du(5,Z) = {6}



Discussion

- ❑ Aide à générer des données de test en suivant la manière dont la donnée est manipulée dans le programme.
- ❑ Aide à définir les critères intermédiaires entre le test de toutes les arêtes (possiblement trop faible) et le test de tous les chemins (souvent impossible).
- ❑ Mais nécessite le support d'un outil efficace (voir la fin du chapitre).

Hiérarchie des critères de couverture



Mesure de la couverture du code

- ❑ Un avantage des critères structurels est que leur couverture peut être mesurée *automatiquement* :
 - Pour contrôler le progrès des tests,
 - Pour estimer l'achèvement des tests en terme d'anomalies restantes et de fiabilité,
 - Pour aider à fixer des objectifs pour les testeurs.
- ❑ Une haute couverture n'est pas une garantie d'un logiciel sans anomalie, seulement un élément d'information pour augmenter notre confiance → modèles statistiques.