UNIVERSITÉ PARIS SUD

L3 informatique et MIAGE

Année 2016-2017

Génie logiciel avancé

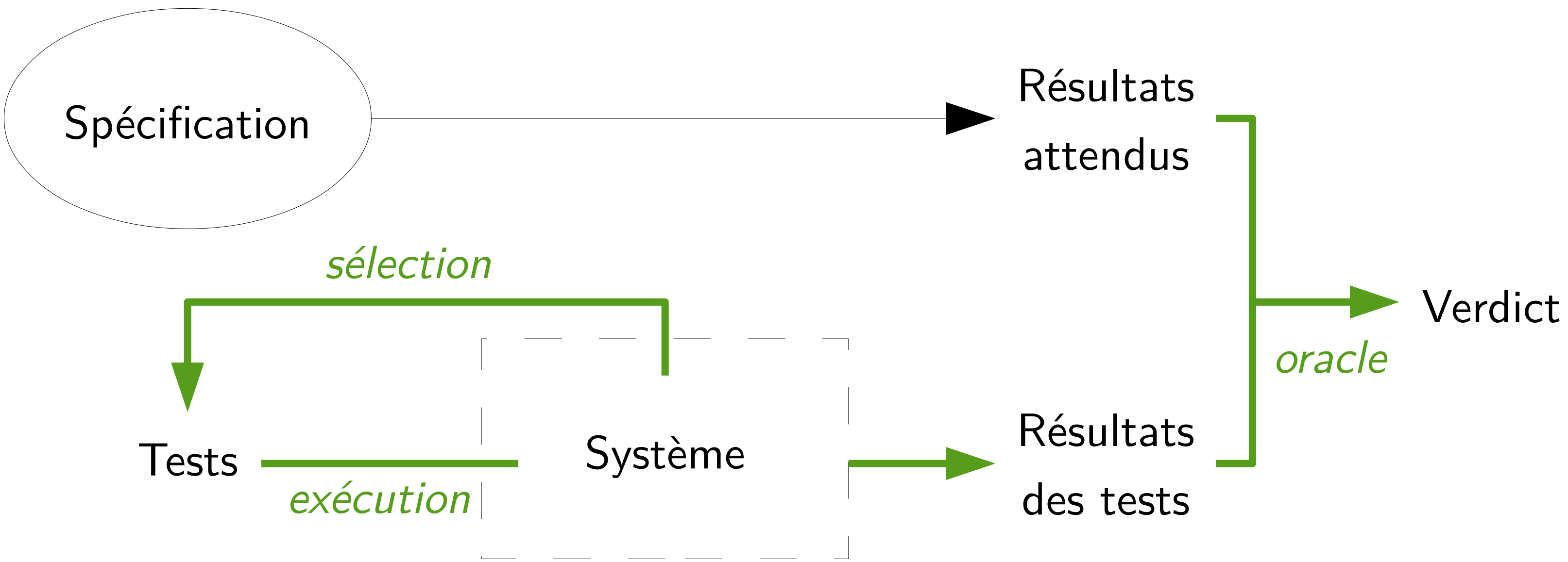
Test structurel à partir du graphe de flot de contrôle

Delphine Longuet
delphine.longuet@lri.fr

http://www.lri.fr/~longuet/Enseignements/16-17/L3-GLA

Test structurel (ou test boîte blanche)

Sélection des tests à partir de l'analyse du code source du système



Construction des tests uniquement pour du code déjà écrit

Test structurel (ou test boîte blanche)

Principe : utiliser la structure du code pour dériver des cas de test

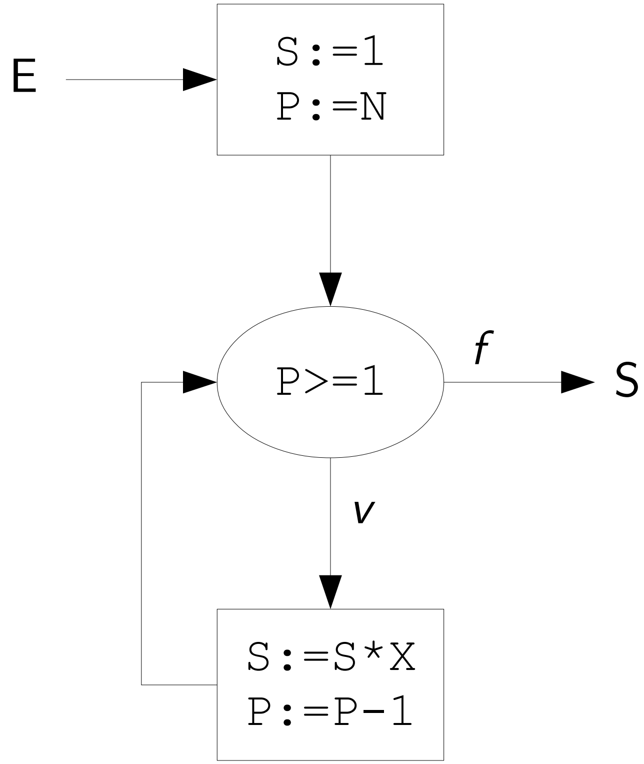
Complémentaire des méthodes de test fonctionnel (ou boîte noire) car étude de la réalisation et pas seulement de la spécification

Deux méthodes :

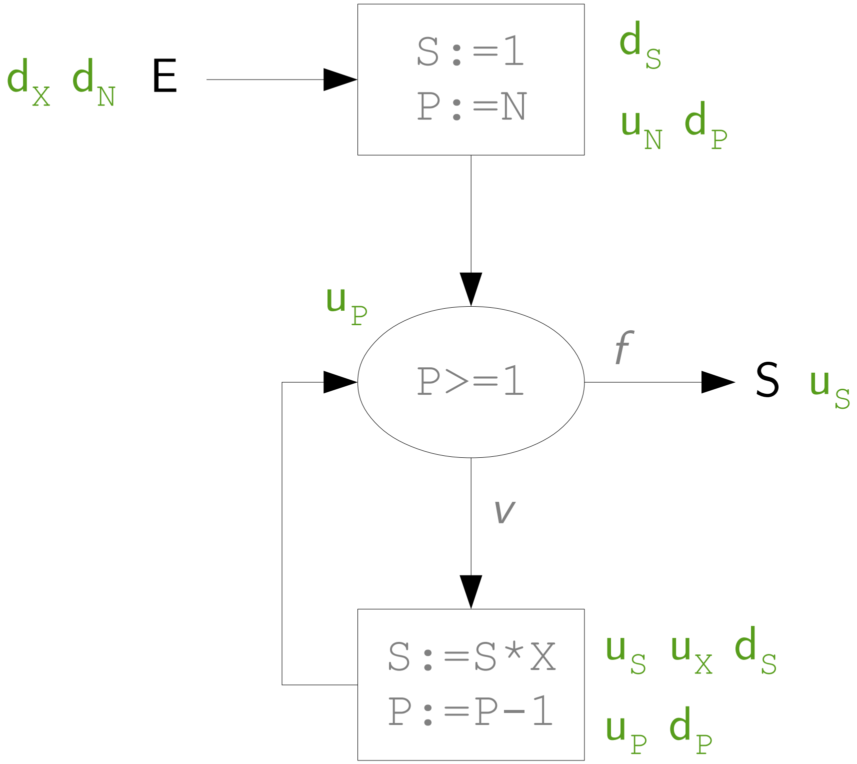
- À partir du graphe de flot de contrôle : couverture de toutes les instructions, tous les branchements...
- À partir du graphe de flot de données : couverture de toutes les utilisations d'une variable, de tous les couples définition-utilisation...

Méthodes d'analyse du code

Fonction calculant X^N



Graphe de flot de contrôle



Graphe de flot de données

Sélection de tests à partir du code

Idée : Couvrir les chemins d'exécution du programme

Problème : Nombre de chemins infini

→ Définir des critères de couverture

Critère de couverture : Éléments du code à couvrir par les tests

- nœuds, arcs
- décisions
- couples de définition-utilisation d'une variable...

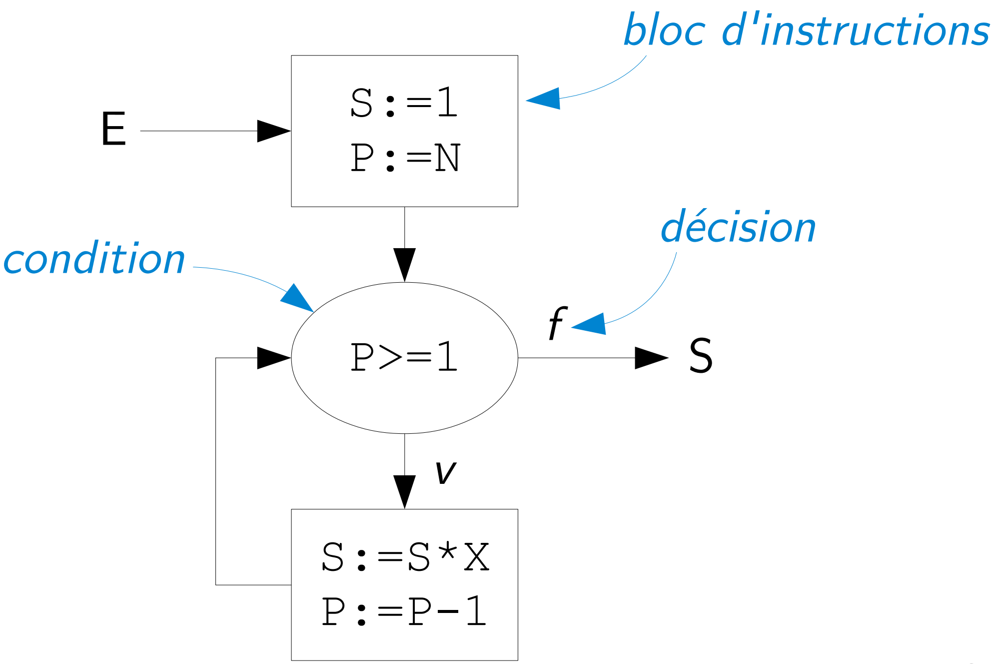
Objectifs de test définis sur la structure du code

Graphe de flot de contrôle

Graphe orienté avec un nœud initial E et un nœud final S tel que :

- un nœud interne est
  - soit un bloc d'instructions élémentaires
  - soit un nœud de décision étiqueté par une condition
- un arc relie les nœuds correspondant à des instructions ou conditions successives (flot de contrôle)

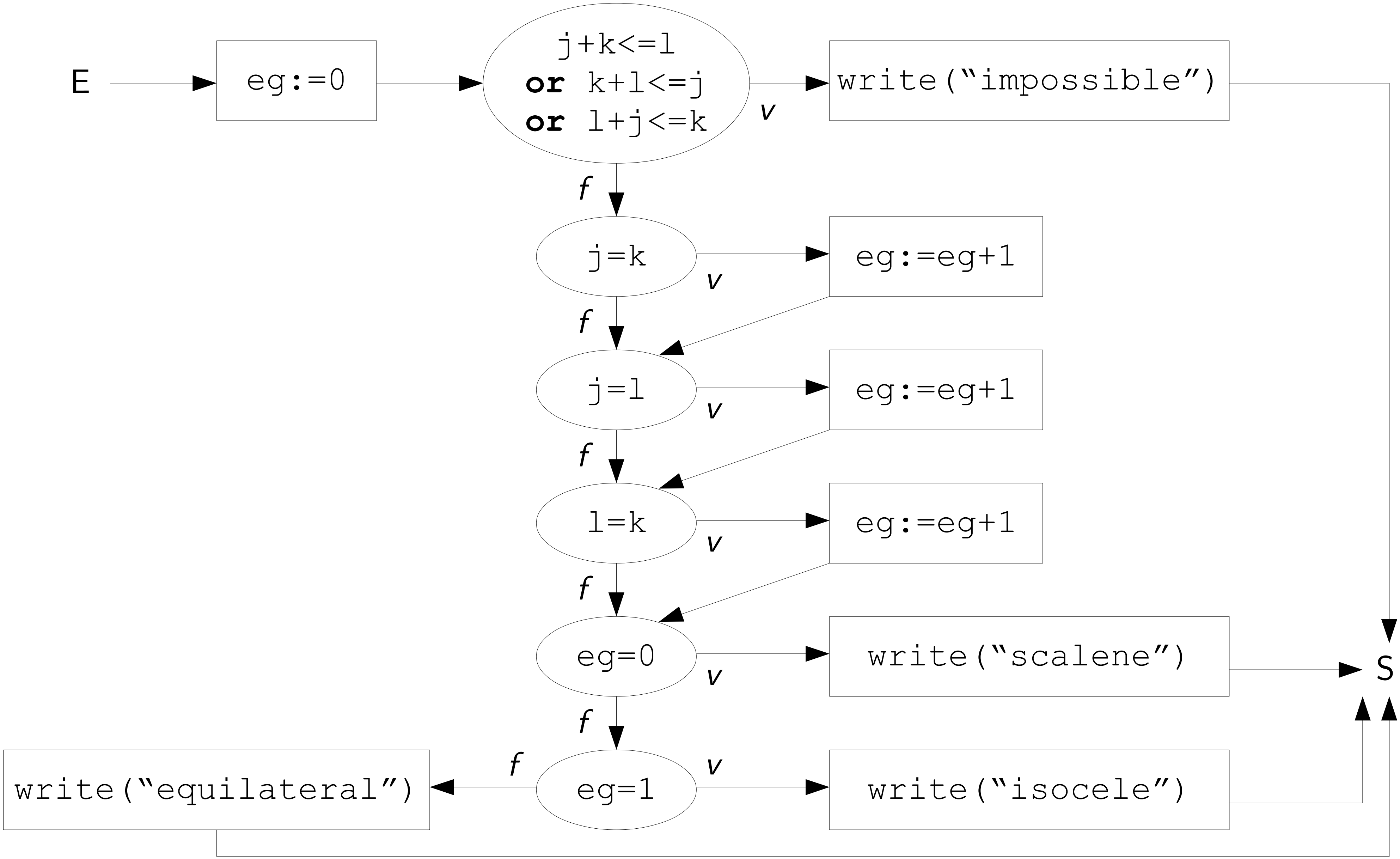
```
S:=1;
P:=N;

while P>=1 do

   S:=S*X;
   P:=P-1;

endwhile;
```



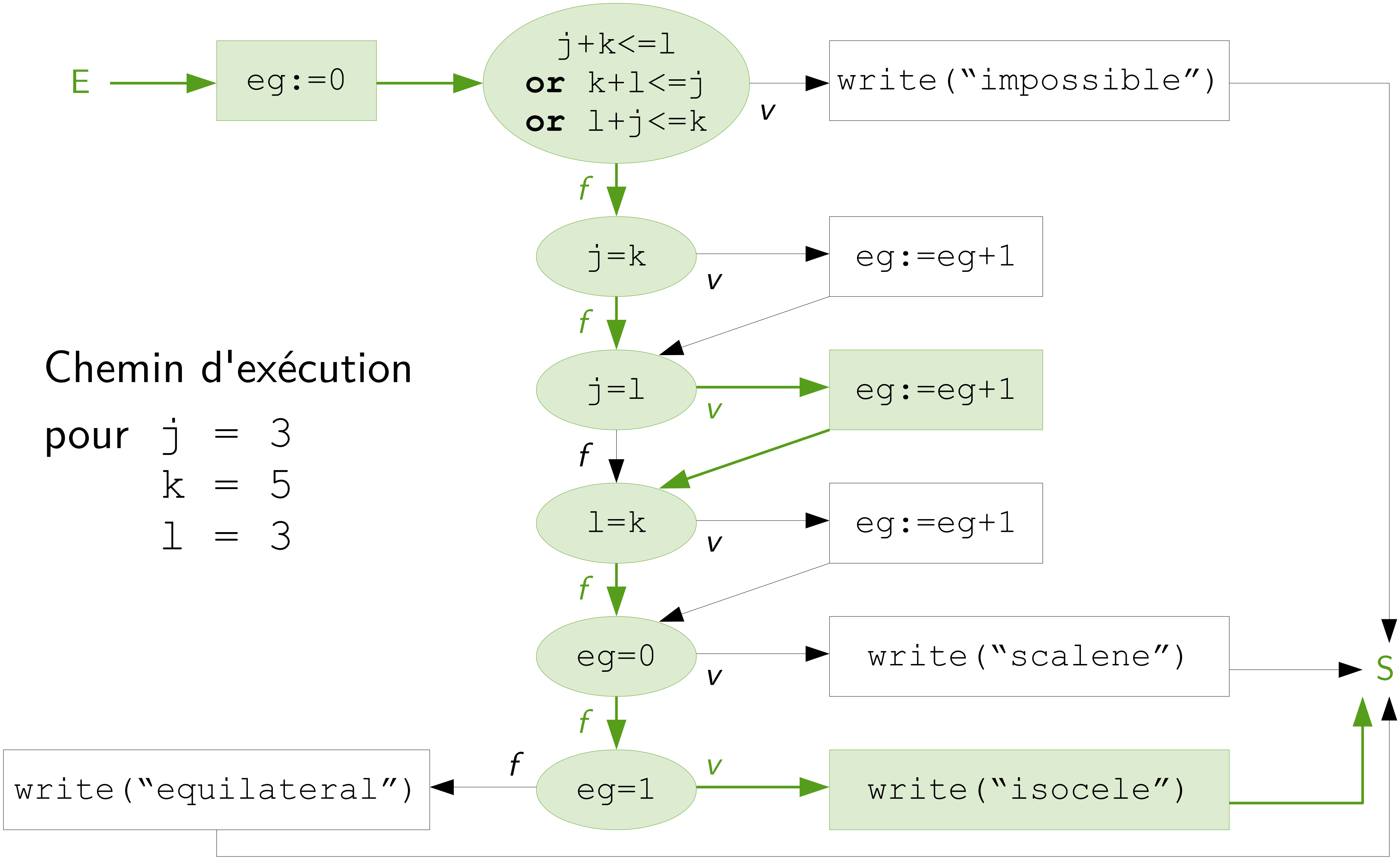
Encore les triangles

```
triangle(j,k,l : positive):void
eg := 0;
if j + k <= l or k + l <= j or l + j <= k
then write("impossible");
else if j = k then eg := eg + 1; endif;
     if j = l then eg := eg + 1; endif;
     if l = k then eg := eg + 1; endif;
     if eg = 0 then write("scalene");
     elsif eg = 1 then write("isocele");
     else write("equilateral"); endif;
endif;
```

Graphe de flot de contrôle associé



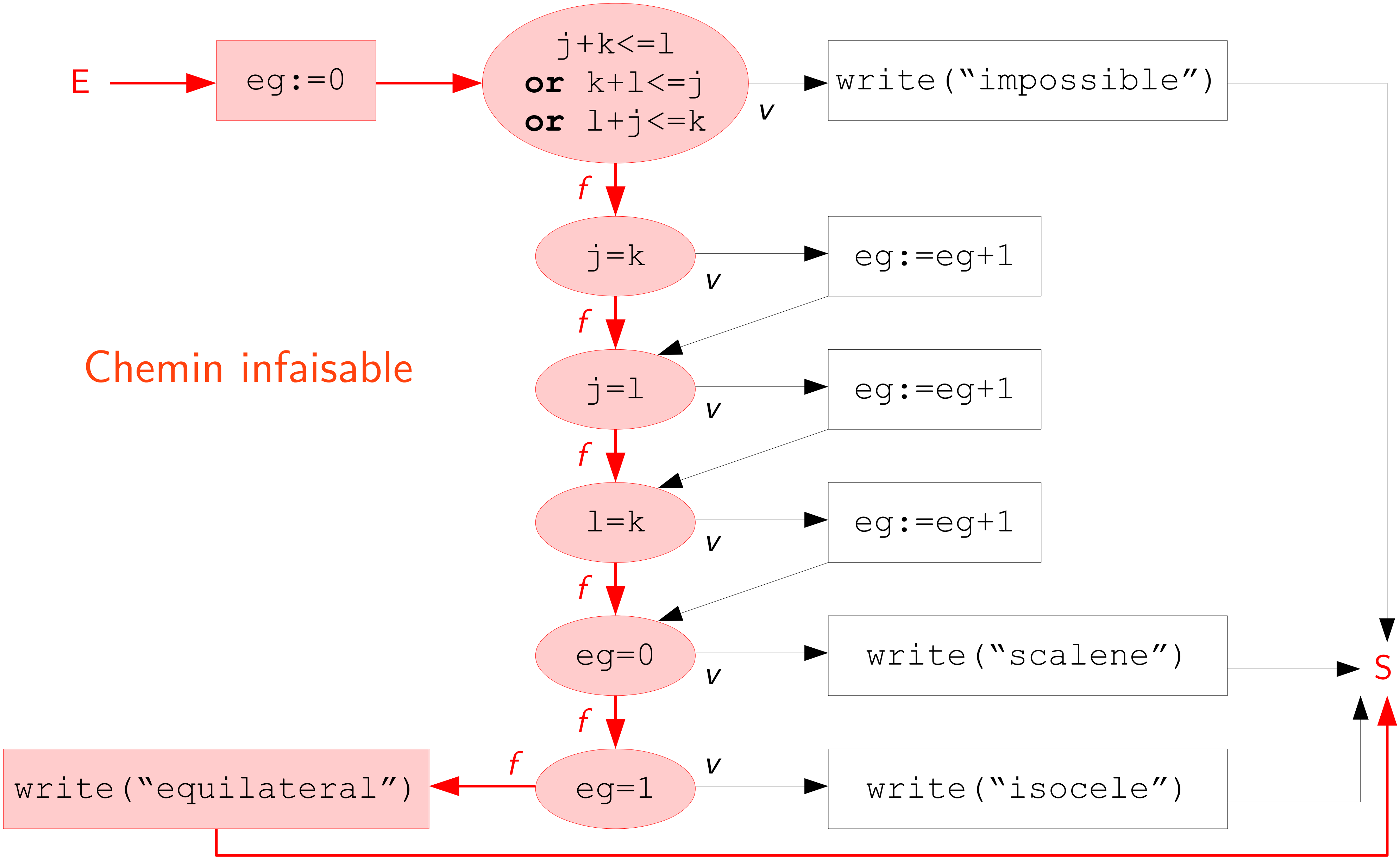
Graphe de flot de contrôle associé



Chemin d'exécution

pour `j = 3`
`k = 5`
`l = 3`

Graphe de flot de contrôle associé



Chemins du graphe de flot de contrôle

Exécution : Chemin allant de E à S dans le graphe de flot de contrôle, associé à l'ensemble des conditions sur les variables d'entrée qui permet d'exécuter précisément ce chemin

Pour $j = 3$, $k = 5$ et $l = 3$, conditions du chemin :

$$\boxed{\begin{array}{c} \neg(j + k \leq l \vee k + l \leq j \vee l + j \leq k) \\ \wedge\, j \neq k \wedge j = l \wedge l \neq k \end{array}}$$

Conditions satisfiables par tout triplet (j, k, l) qui forme un triangle et tel que $j = l$ et $j \neq k$

Chemins du graphe de flot de contrôle

Exécution : Chemin allant de E à S dans le graphe de flot de contrôle, associé à l'ensemble des conditions sur les variables d'entrée qui permet d'exécuter précisément ce chemin

Conditions pour le deuxième chemin :

$$\boxed{\begin{array}{c}\neg(j+k \leq l \vee k+l \leq j \vee l+j \leq k) \\ \wedge\, j \neq k \wedge j \neq l \wedge l \neq k \\ \wedge\, 0 \neq 0 \wedge 0 \neq 1\end{array}}$$

Conditions insatisfiables : il n'existe aucun triplet (j, k, l) tel que ces conditions soient remplies pendant l'exécution

Chemins du graphe de flot de contrôle

Exécution : Chemin allant de E à S dans le graphe de flot de contrôle, associé à l'ensemble des conditions sur les variables d'entrée qui permet d'exécuter précisément ce chemin

Exécution réelle : Chemin dont les conditions peuvent être satisfaites

→ Si conditions impossibles à satisfaire, chemin infaisable (donc pas de test possible pour couvrir ce chemin)

Problème : Comment détecter les chemins infaisables ?

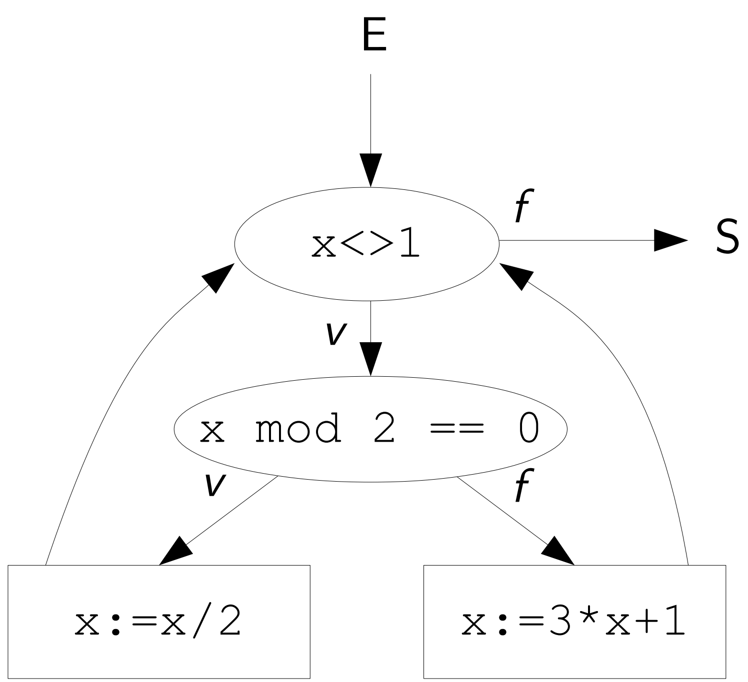
Chemins infaisables

Existence de chemins infaisables : Problème indécidable

Suite de Syracuse

On ne sait pas si ce programme termine toujours, donc s'il existe des chemins infaisables de E à S

```
while x<>1 do
  if x mod 2 == 0
  then x:=x/2;
  else x:=3*x+1;
endif;
endwhile;
```



Conditions de chemin

Existence de chemins infaisables : Problème indécidable

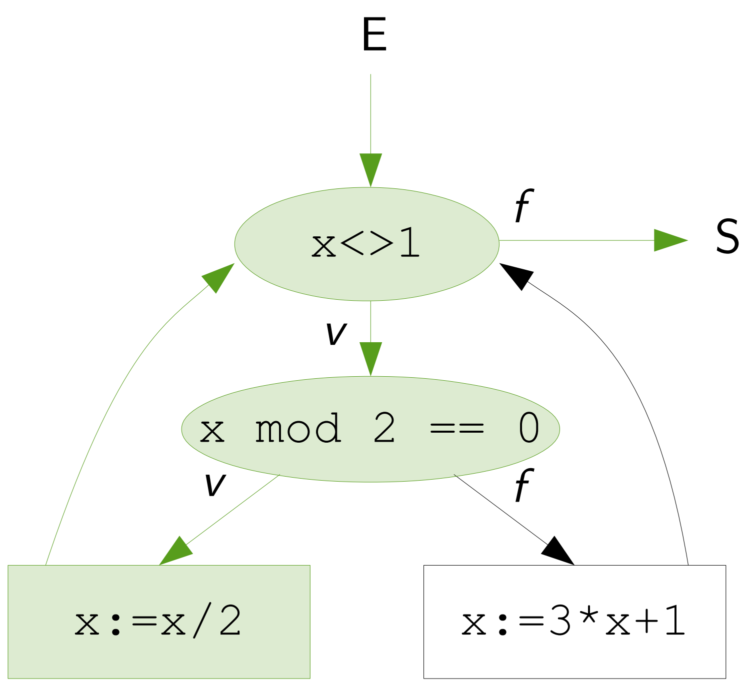
Mais : On sait calculer les conditions pour qu'un chemin donné soit faisable (conditions de chemin)

On a $x = x_0$ en entrée

Conditions pour passer une seule fois par la boucle dans le *then* :

$$x_0 \neq 1$$
$$\text{et } x_0 \bmod 2 = 0$$
$$\text{et } x_0/2 = 1$$

Donc il faut que $x_0 = 2$



Exécution symbolique

Soit P un chemin de E à S dans le graphe de flot de contrôle

1. On donne des valeurs symboliques aux variables x_0, y_0, z_0...
2. On initialise la condition de chemin à *true*
3. On suit le chemin P nœud par nœud depuis E :
   - si le nœud est un bloc d'instructions, on exécute les instructions sur les valeurs symboliques
   - si le nœud est un bloc de décision étiqueté par une condition C :
     - si on suit la branche où C est vraie, on ajoute C à la condition de chemin, en substituant les variables par leur valeur symbolique
     - si on suit la branche où C est fausse, on ajoute la négation de C à la condition de chemin, en substituant les variables par leur valeur symbolique

Exécution symbolique

État : application associant à chaque variable x du programme une valeur d'un ensemble D_x

$$\textit{etat} : V \rightarrow (D_x)_{x \in V}$$
$$x \mapsto d \in D_x$$

État symbolique : application associant à chaque variable x du programme un ensemble de valeurs possibles

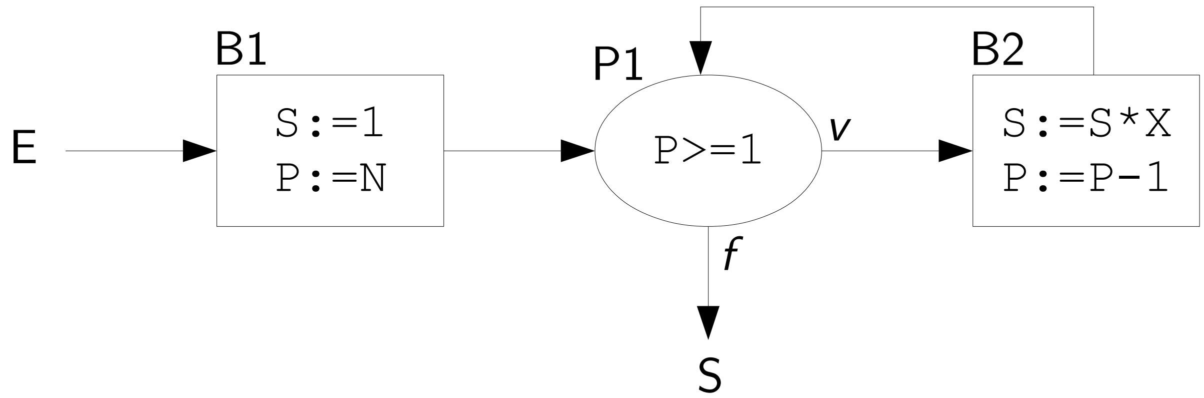
$$\textit{etatsym} : V \rightarrow \mathscr{P}(D_x)_{x \in V}$$
$$x \mapsto E \subseteq D_x$$

Représentation d'un état symbolique :

substitution des variables : $\mathtt{S} \mapsto x_0$, $\mathtt{P} \mapsto n_0 - 1$, $\mathtt{X} \mapsto x_0$, $\mathtt{N} \mapsto n_0$

$+$ ensemble de conditions (condition de chemin) : $n_0 \geq 1 \wedge n_0 - 1 < 1$

Exécution symbolique (formellement)



$$E \longrightarrow B1 \longrightarrow P1 \xrightarrow{v} B2 \longrightarrow P1 \xrightarrow{f} S$$

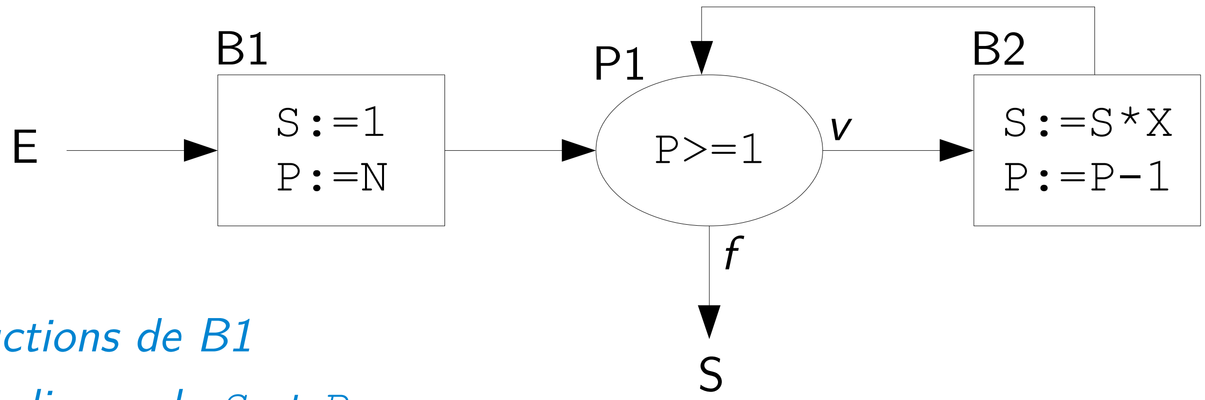
$$\boxed{\begin{array}{l} \mathtt{S} \mapsto s_0 \\ \mathtt{P} \mapsto p_0 \\ \mathtt{X} \mapsto x_0 \\ \mathtt{N} \mapsto n_0 \\ \textit{true} \end{array}}$$

substitution initiale

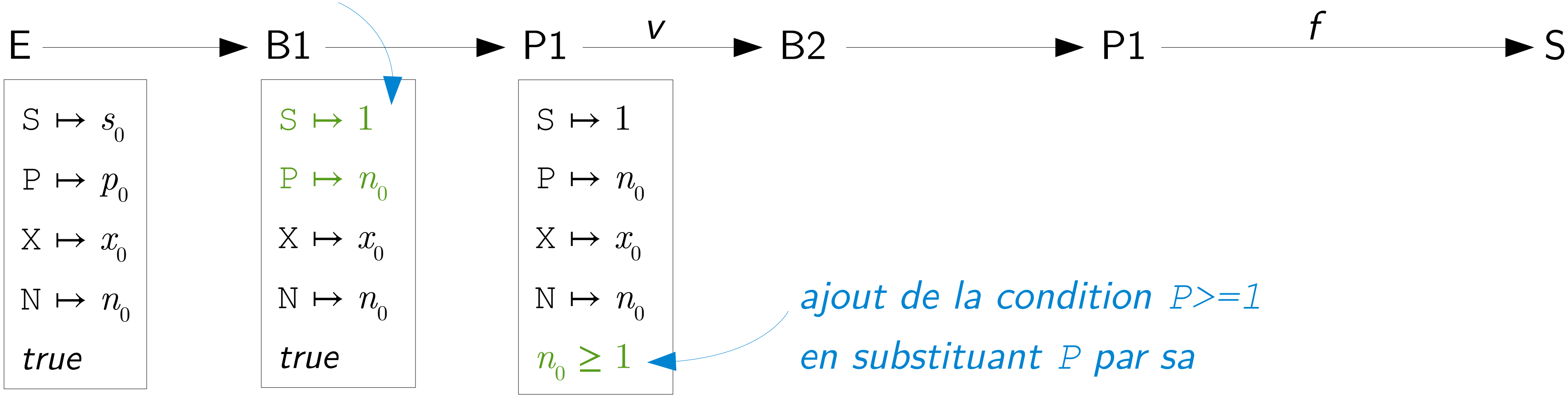
des variables et

condition de chemin initiale

Exécution symbolique (formellement)

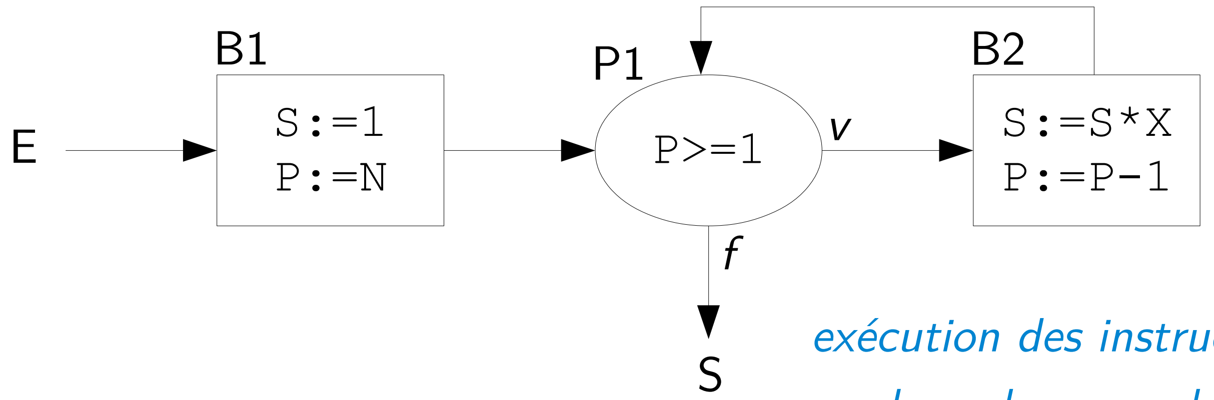


exécution des instructions de B1
sur les valeurs symboliques de S et P

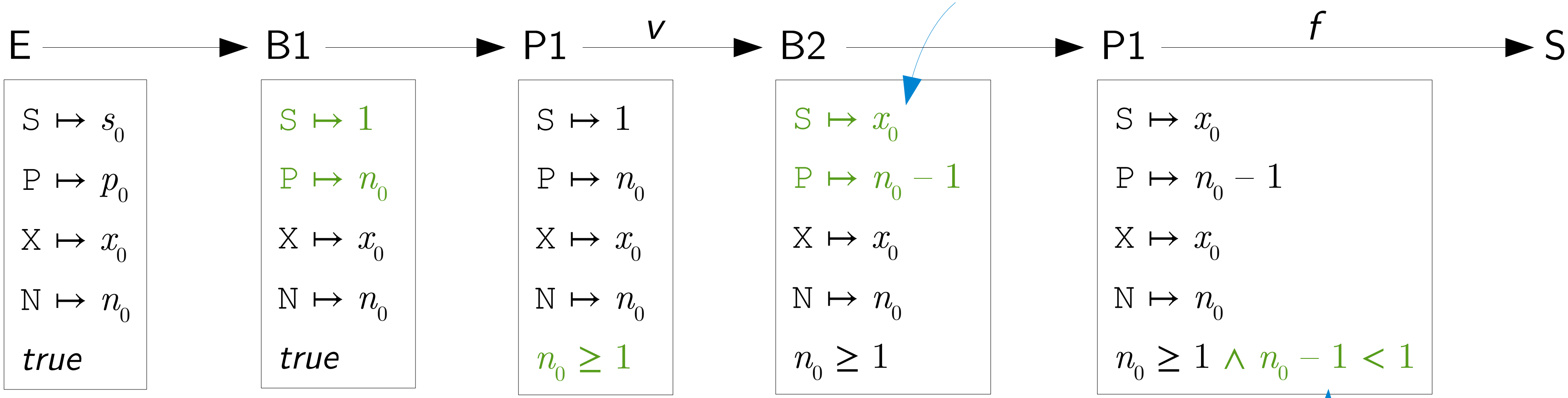


ajout de la condition P>=1
en substituant P par sa
valeur symbolique n_0

Exécution symbolique (formellement)

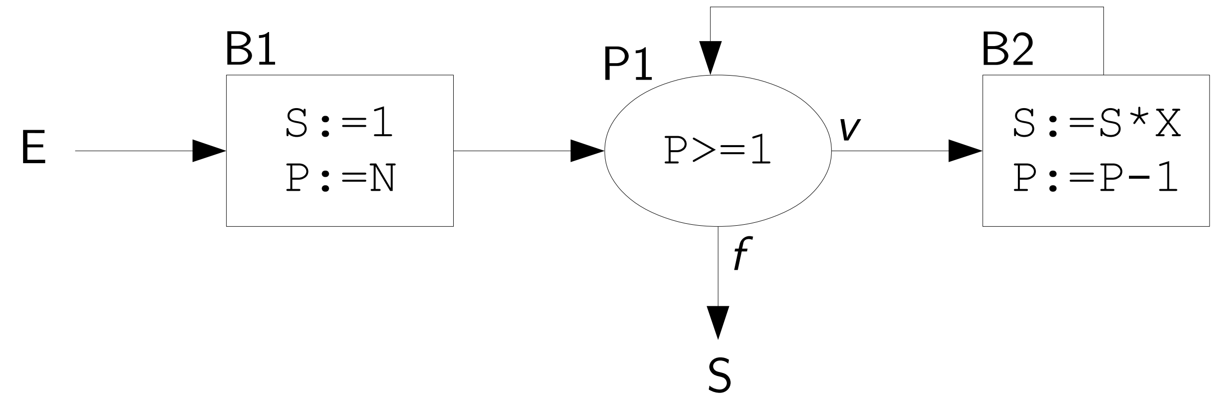


exécution des instructions de B2
sur les valeurs symboliques de `S` *et* `P`



ajout de la négation de la condition `P>=1`
en substituant `P` *par sa valeur symbolique* $n_0 - 1$

Exécution symbolique (formellement)

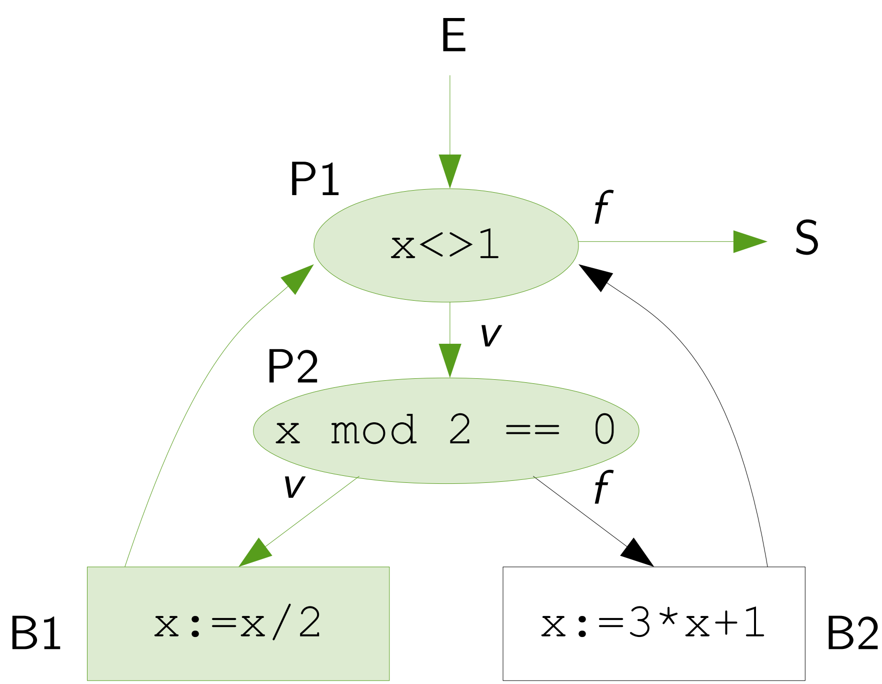


$E \longrightarrow B1 \longrightarrow P1 \xrightarrow{v} B2 \longrightarrow P1 \xrightarrow{f} S$

E	B1	P1	B2	P1
$\mathtt{S} \mapsto s_0$	$\mathtt{S} \mapsto 1$	$\mathtt{S} \mapsto 1$	$\mathtt{S} \mapsto x_0$	$\mathtt{S} \mapsto x_0$
$\mathtt{P} \mapsto p_0$	$\mathtt{P} \mapsto n_0$	$\mathtt{P} \mapsto n_0$	$\mathtt{P} \mapsto n_0 - 1$	$\mathtt{P} \mapsto n_0 - 1$
$\mathtt{X} \mapsto x_0$	$\mathtt{X} \mapsto x_0$	$\mathtt{X} \mapsto x_0$	$\mathtt{X} \mapsto x_0$	$\mathtt{X} \mapsto x_0$
$\mathtt{N} \mapsto n_0$	$\mathtt{N} \mapsto n_0$	$\mathtt{N} \mapsto n_0$	$\mathtt{N} \mapsto n_0$	$\mathtt{N} \mapsto n_0$
true	*true*	$n_0 \geq 1$	$n_0 \geq 1$	$n_0 \geq 1 \wedge n_0 - 1 < 1$

Condition de chemin : $n_0 \geq 1 \wedge n_0 - 1 < 1 \Leftrightarrow n_0 = 1$

Exécution symbolique (notations allégées)

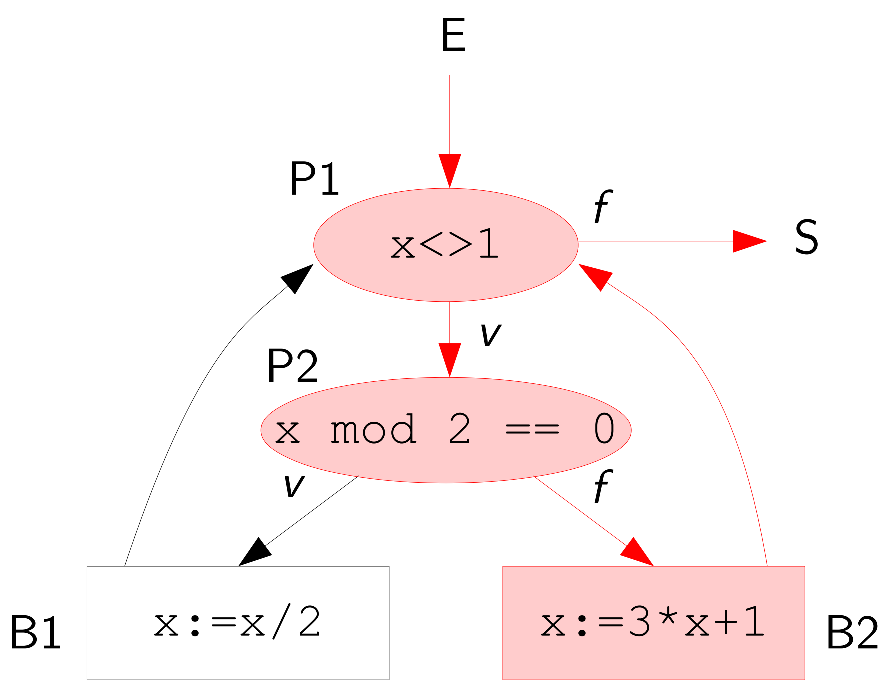


	x	condition de chemin
E	x_0	true
P1	x_0	$x_0 \neq 1$
P2	x_0	x_0 mod $2 = 0$
B1	$x_0/2$	
P1	$x_0/2$	$x_0/2 = 1$
S	$x_0/2$	

Condition de chemin : $x_0 \neq 1$ et x_0 est pair et $x_0/2 = 1$

Satisfaite par $x_0 = 2$

Exécution symbolique (notations allégées)



	x	condition de chemin
E	x_0	true
P1	x_0	$x_0 \neq 1$
P2	x_0	x_0 mod $2 \neq 0$
B2	$3x_0+1$	
P1	$3x_0+1$	$3x_0+1 = 1$
S	$3x_0+1$	

Condition de chemin : $x_0 \neq 1$ et x_0 est impair et $3x_0+1 = 1$

Impossible à satisfaire : chemin infaisable

Tests pour un critère de couverture

Critère de couverture : Condition caractérisant un ensemble de chemins du graphe de flot de contrôle

Génération de tests grâce à un critère de couverture :

1. Sélectionner un ensemble (minimal) de chemins satisfaisant le critère
2. Éliminer les chemins infaisables. Chaque chemin faisable définit un objectif de test (ensemble de conditions sur les entrées)
3. Générer un cas de test pour chaque chemin (choisir des valeurs satisfaisant l'ensemble de conditions associé au chemin)

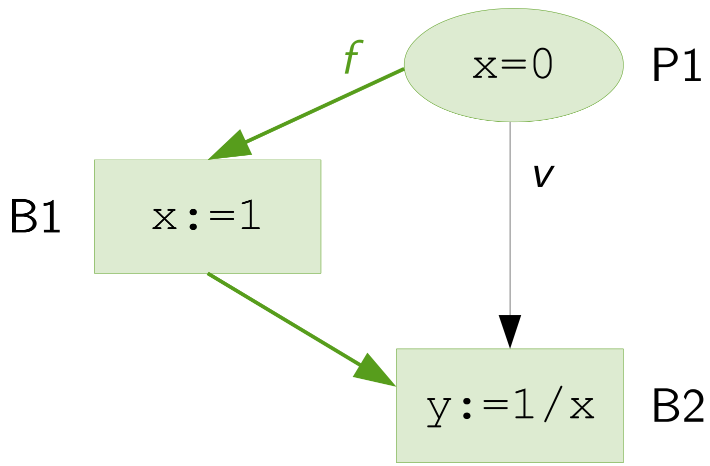
Exemple fil rouge : contains

Renvoie `true` si et seulement si l'élément `v` apparaît dans le tableau `t` de longueur `n`

```
boolean contains(int t[], int n, int v) {
    int i = 0;
    boolean res = false;
    while(i < n && !res) {
        if(t[i] == v) {
            res = true;
        }
        i++;
    }
    return(res);
}
```

Critère « toutes les instructions »

Satisfait par un ensemble de chemins T si pour chaque nœud du graphe, il existe un chemin dans T passant par ce nœud



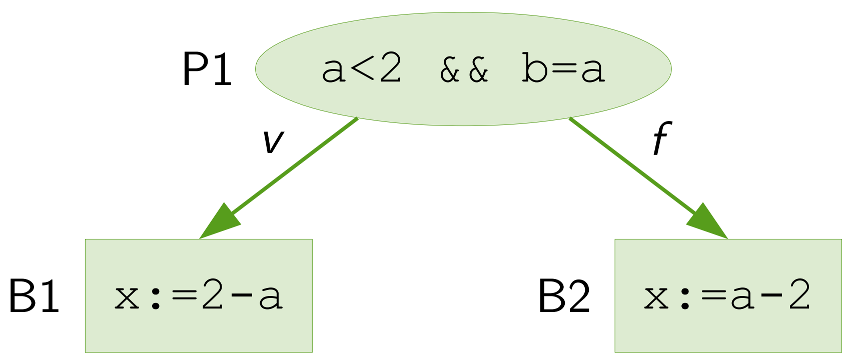
Critère satisfait par le chemin
P1 B1 B2

Limite du critère : Ne couvre pas nécessairement tous les arcs

→ Ici, division par 0 non couverte

Critère « toutes les décisions »

Satisfait par un ensemble de chemins T si pour chaque nœud de décision du graphe, il existe un chemin dans T rendant la décision vraie et un autre chemin rendant la décision fausse



Critère satisfait par les chemins $\underset{v}{\underline{\text{P1}}}$ B1 et $\underset{f}{\underline{\text{P1}}}$ B2

Chemin P1 B1 : $\{a = 1,\ b = 1\}$

Chemin P1 B2 : $\{a = 3,\ b = 3\}$

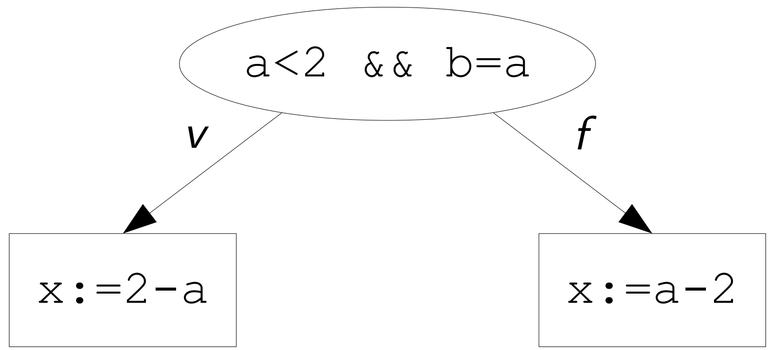
- Également appelé critère « tous les arcs »
- Plus fort que critère « toutes les instructions » : couverture de toutes les décisions implique couverture de toutes les instructions

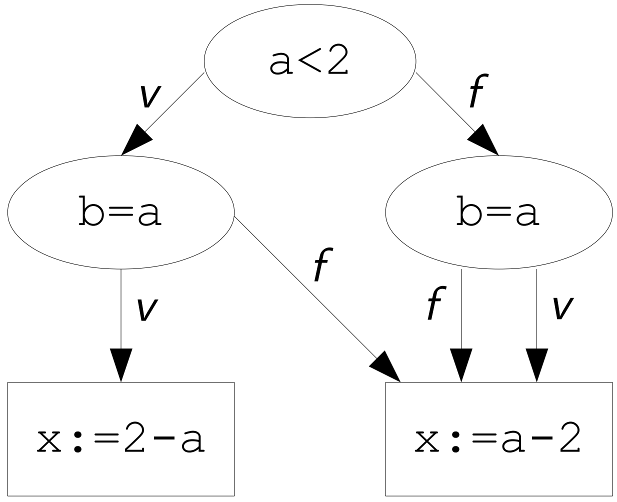
Critère « toutes les décisions »

Limite du critère : Ne couvre pas nécessairement toutes les valeurs possibles des sous-conditions

→ Ici, cas où $b \neq a$ non couvert

Solution : Décomposer les conditions multiples

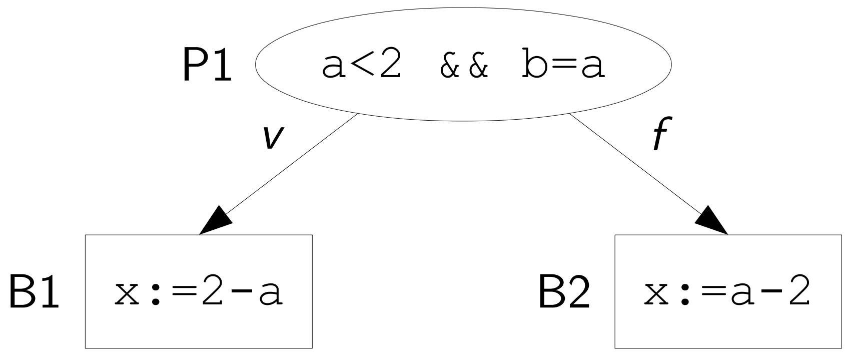




Critère « toutes les conditions multiples »

Satisfait par un ensemble de chemins T si :

- critère « toutes les décisions » satisfait par T
- toutes les combinaisons de toutes les valeurs de toutes les sous-conditions couvertes



Critère satisfait par les chemins :

P1 B1 avec $a < 2$ et $b = a$

P1 B2 avec $a \geq 2$ et $b = a$

P1 B2 avec $a \geq 2$ et $b \neq a$

P1 B2 avec $a < 2$ et $b \neq a$

Équivalent à « toutes les décisions » avec conditions décomposées

Problème : nombre de chemins exponentiel en fonction du nombre de sous-conditions

Critère « toutes les conditions multiples »

```
if (A && (B || C))
```

Toutes les décisions :

A	B	C	décision
0	1	1	0
1	1	1	1

Critère « toutes les conditions multiples »

```
if (A && (B || C))
```

Toutes les conditions multiples :

A	B	C	décision
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Comment limiter
la combinatoire ?

Critère MC/DC

Objectif : améliorer le critère de couverture toutes les décisions en contrôlant la combinatoire

Principe : on ne considère une combinaison de valeurs faisant varier une sous-condition que si cette sous-condition influence la décision

```
if (A && (B || C))
```

Pour A :

A	B	C	décision
0	1	1	**0**
1	1	1	**1**

Critère MC/DC

MC/DC : *modified condition/decision coverage*

Critère « toutes les conditions/décisions modifié » : Satisfait par un ensemble de chemins T si :

- critère « toutes les décisions » satisfait par T
- toutes les valeurs de toutes les sous-conditions couvertes
- chaque valeur d'une sous-condition influence la décision lorsque les autres sous-conditions sont fixées

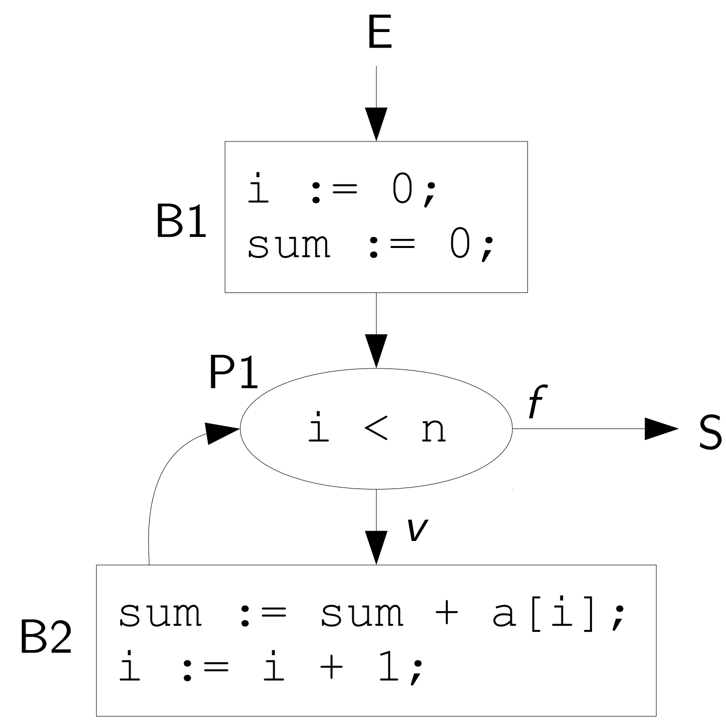
Critère MC/DC

```
if (A && (B || C))
```

A	B	C	décision	
~~0~~	~~0~~	~~0~~	~~0~~	
~~0~~	~~0~~	~~1~~	~~0~~	
~~0~~	~~1~~	~~0~~	~~0~~	
0	1	1	**0**	$\longrightarrow$ A
1	**0**	**0**	**0**	$\longrightarrow$ B et C
1	0	**1**	**1**	$\longrightarrow$ C
1	**1**	0	**1**	$\longrightarrow$ B
1	1	1	**1**	$\longrightarrow$ A

Critères sur les conditions et décisions

```
invsum(int a[],int n):float
i := 0;
sum := 0;
while (i<n) do
    sum := sum + a[i];
    i := i + 1;
endwhile;
return 1/sum;
```



Critère « toutes les décisions » satisfait par le chemin E B1 $\underset{v}{\underline{\text{P1}}}$ B2 $\underset{f}{\underline{\text{P1}}}$ S

Limite des critères sur les conditions : manque chemins à problèmes

↳ Ici, division par 0 si tableau vide non détectée

Critère « tous les chemins »

Satisfait seulement par l'ensemble des chemins du graphe

Problème : Impossible à satisfaire dès qu'il y a des boucles

Affaiblissements possibles :

- « tous les chemins de longueur au plus k » (en nombre de nœuds)
- « tous les chemins passant entre 0 et k fois dans chaque boucle » (aussi appelés k-chemins)

Hiérarchie des critères de couverture pour le graphe de flot de contrôle

toutes les conditions multiples

MC/DC

tous les chemins

tous les chemins de longueur au plus k

toutes les décisions

tous les noeuds

plus fort que