

Test structurel

Gerson Sunyé — gerson.sunye@univ-nantes.fr

Mottu - Lanoix – Le Traon – Baudry - Sunyé

TODO

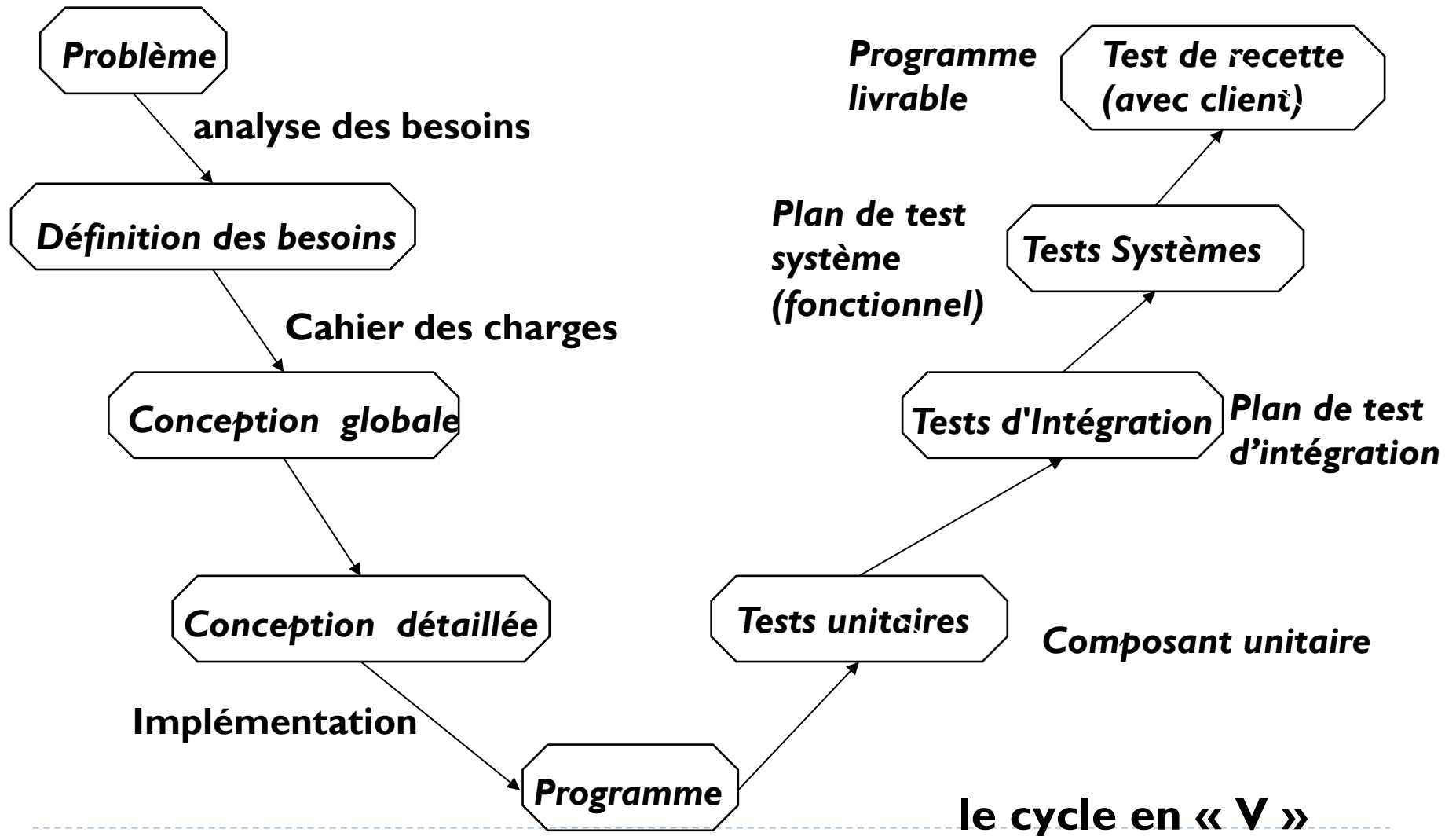
- ▶ Introduire la complexité cyclomatique
- ▶ Introduire quelques métriques d'analyse structurelle: couplage, nombre de méthodes/classe, etc.
- ▶ Graphe d'un programme Java : comment représenter les exceptions ?



Test boîte blanche / structurel

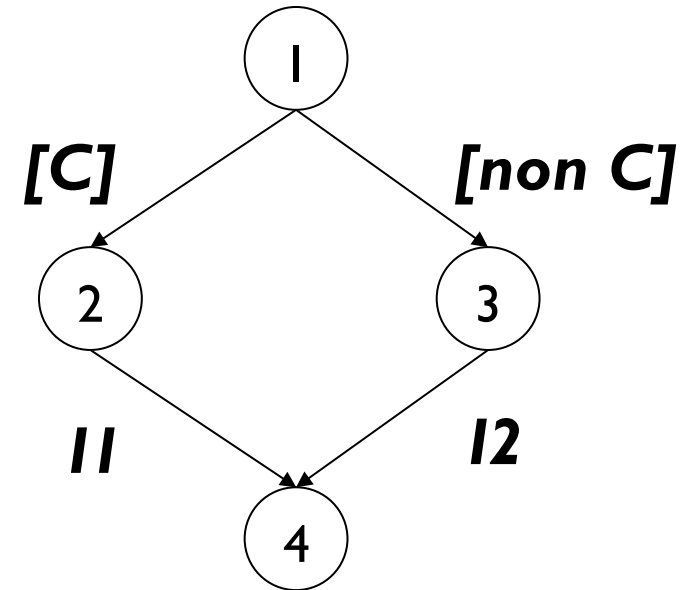
- Critères basées sur la couverture
 - Code (instruction ...)
 - Chemin
- Abstraction de la structure du programme pour obtenir un critère formel

Echelle des tests



Le test structurel: abstraire pour obtenir un critère formel

***si C alors I1
sinon I2***



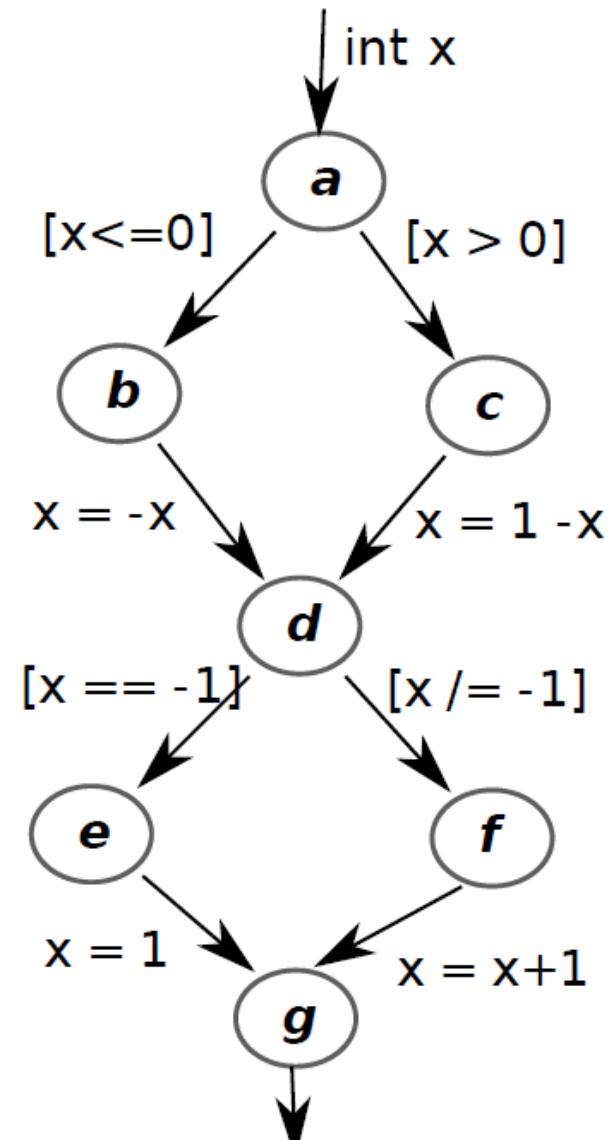
Modéliser avec un graphe

- Graphe de Flot de Contrôle
 - Représente tous les chemins d'exécution potentiels
 - Graphe orienté : avec un arc d'entrée et un arc de sortie
 - Noeuds :
 - Limites entre blocs élémentaires du programme
 - prédicats des conditionnelles /boucles
 - nœud de jonction “vide” associé à un noeud prédicat
 - Arcs :
 - Instructions
 - Conditions d'un prédicat
 - Bloc élémentaire : ensemble d'instructions séquentielles

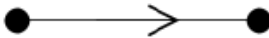
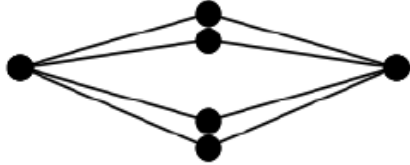
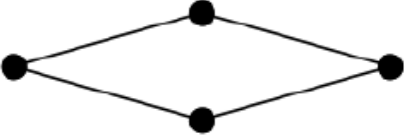
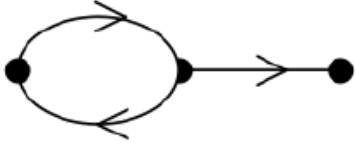

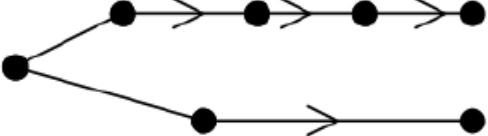
Exemple de graphe de flot de contrôle

```
void function(int x)
if (x<=0)
    x = -x;
else
    x= 1 - x;
fi
if (x == -1)
    x = 1;
else
    x = x + 1;
fi
```

- **Entrée :** a / **Sortie :** g
- **Conditions :** $[x == -1]$
- **Instructions :** $x = x + 1$



Modélisation en graphe de flot de contrôle

➤ Suite linéaire :		➤ Case :	
➤ If :		➤ Until :	
➤ While (for) :		➤ Else :	

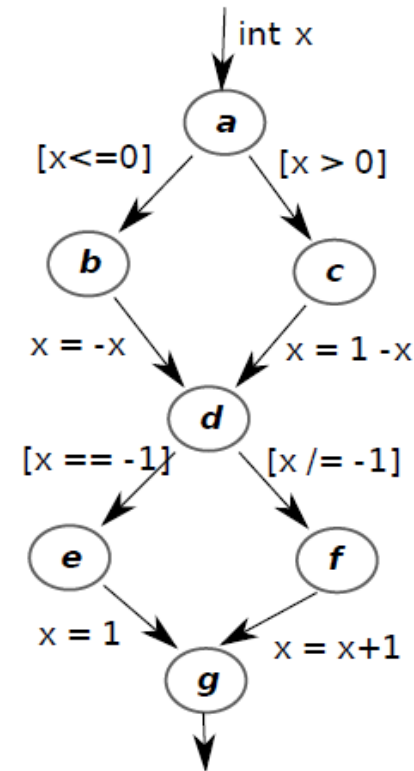
Critères basé sur les chemins

- **Chemins** : suite d'arcs rencontrés dans le graphe, en partant de E et finissant en S
 - en général représenté sans perte d'information par une liste de sommets

Exemples de chemins

Chemins “possible”

- $\text{ch1} = (a, b, d, e, g)$
- $\text{ch2} = (a, c, d, e, g)$
- $\text{ch3} = (a, b, d, f, g)$
- $\text{ch4} = (a, c, d, f, g)$



Sous forme algébrique,

- $(a, b, d, f, g) + (a, b, d, e, g) + (a, c, d, f, g) + (a, c, d, e, g)$
- $(a, [b + c], d, [f + e], g)$ (*expression factorisée*)

Critère de couverture

- ▶ Sur ce modèle imaginer un critère de couverture
 - ▶ Minimal
 - ▶ Maximal
- ▶ De nouveau comment choisir le meilleur critère ?



Critères basés sur les chemins

- ▶ **Prédicat de chemin** : conjonction des prédicats (ou de leur négation) rencontrés le long du chemin.
 - Pas toujours calculable

a b d e g : $x_0 \leq 0 \ \& \ x_i = -1$ (x_i = ième valeur de x)

a b d f g : $x_0 \leq 0 \ \& \ x_i \neq -1$ (x_i = ième valeur de x)

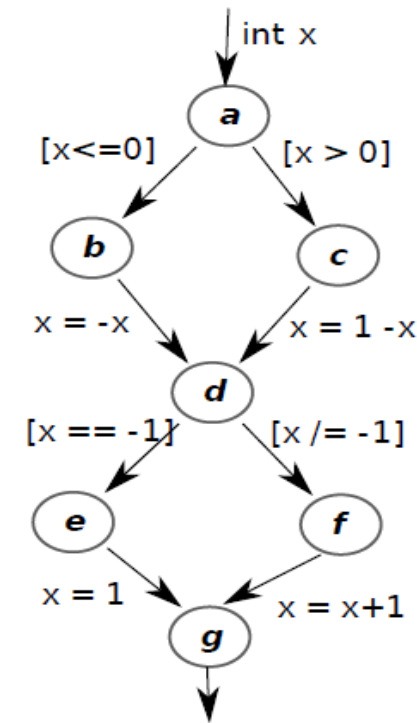
Se complique avec les boucles

Le test unitaire structurel

- Sélection des tests fondés sur le flot de contrôle
 - Couverture des instructions : chaque bloc doit être exécuté au moins une fois
 - Couverture des arcs ou enchaînements
 - tous les chemins élémentaires ou l-chemins
 - tous les i-chemins : de 0 à i passages dans les boucles
 - tous les chemins : si boucles, potentiellement infinis

Chemins exécutables

- $ch1 = (a, b, d, e, g)$
- $ch2 = (a, c, d, e, g)$
- $ch3 = (a, b, d, f, g)$
- $ch4 = (a, c, d, f, g)$



- $DT = \{x = -2 \mid x' = 3\}$ sensibilise $ch3$
- $DT = \{x = 3 \mid x' = -2\}$ sensibilise $ch4$
- $DT = \{x = 2 \mid x' = 1\}$ sensibilise $ch2$
- Aucune valeur de x ne sensibilise $ch1 \Rightarrow$ chemin non-exécutable

Chemins non-exécutables

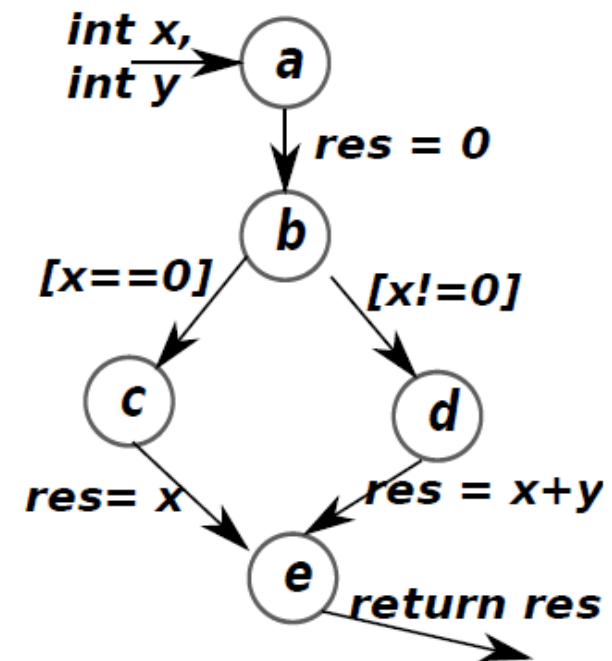
- Tous les chemins d'un graphe de contrôle ne sont pas exécutables
- Détecter les chemins non-exécutables est un problème indécidable
- Étant donné un chemin, trouver une donnée de test qui exécute ce chemin ?
 - problème très difficile
- Chemins non-exécutables : écueil de testeur
 - une erreur de codage ?
 - du code mort ?
 - un code pas optimisé du tout ?

Critère « tous les noeuds »

- Critère le plus faible, aussi appelé TER I (Test effectiveness ratio I).
- Couverture de l'ensemble des nœuds du graphe.
- $\text{taux de couverture} = \frac{\text{nombre de nœuds couverts}}{\text{nombre total de nœuds}}$
- Signifie que toutes les instructions ont été exécutées au moins une fois.

Exemple « tous les nœuds »

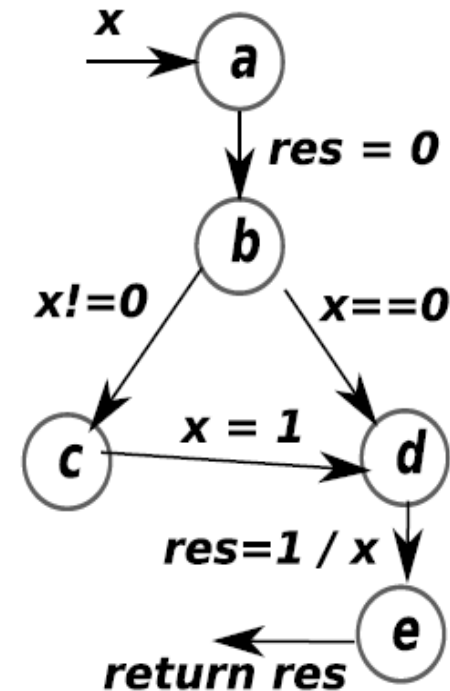
```
int sum(int x, int y) {  
    int res = 0;  
    if (x == 0)  
        res = x // erreur !  
    else  
        res = x+y;  
    fi  
    return res;  
}
```



- DTs = { {x=2,y=5|res=7}, {x=0,y=4|res=4} }
Les chemins (a,b,c,e) et (a,b,d,e) sont parcourus.
Taux de couverture de 100% (= critère TER1 satisfait)
+ erreur détectée (res attendu = 4, res obtenu = 0 ?)

Contre-exemple « tous les nœuds »

```
int div0error(x) {  
    int res = 0 ;  
    if (x != 0)  
        x = 1;  
    res = 1 / x;  
    return res;  
}
```



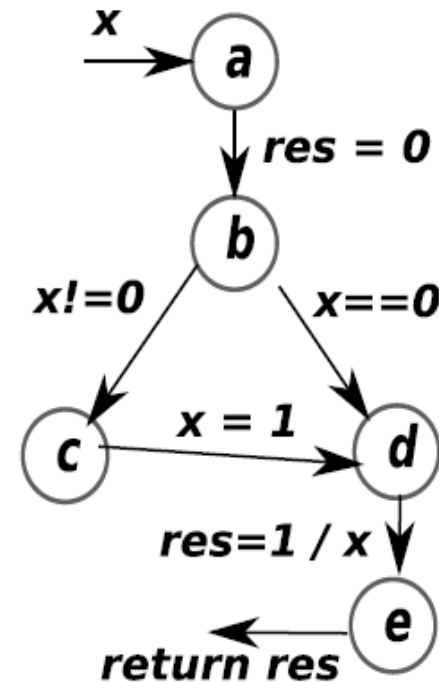
- $DT = \{x=2\}$ satisfait le critère “tous les noeuds” : {abcde}
- Erreur de division par 0 non détectée, chemin révélateur : {abde}
- Il aurait fallu couvrir les deux branches du if : “critère tous les arcs”

Critère "tous les arcs"

- ou TER2 :
 - tous les arcs du graphe de contrôle sont couverts
 - toutes les branches conditionnelles sont couvertes
 - taux de couverture = nombre d'arcs couverts / nombre total d'arcs
 - chaque prédicat prend une fois la valeur "vrai" et la valeur "faux" ;
 - attention aux prédicats composés.
- "Tous les arcs" => "tous les noeuds".
 - L'inverse n'est pas vrai.

Exemple "tous les arcs"

```
int div0error(x) {  
    int res = 0 ;  
    if (x != 0)  
        x = 1;  
    res = 1 / x;  
    return res;  
}
```



- $DT = \{\{x=0\}, \{x=2\}\}$ satisfait le critère "tous les arcs"

Critère « tous les chemins »

- "Tous les chemins" => "tous les arcs" => "tous les noeuds"
- Problème des boucles :
 - Chemin "limite" : traversée de la boucle sans itération
 - Chemin "intérieur" : itération de la boucle une seule fois
 - Ce critère est rarement applicable
- test exhaustif = "tous les chemins avec toutes les valeurs possibles".

Critère « tous les n-chemins »

- ▶ Nombre de passage dans les boucles
- ▶ n-chemins :
 - ▶ De 0 à n passage(s) dans les boucles
- ▶ Attention à la combinatoire quand une boucle a plusieurs branches.



Modéliser avec un graphe

- Graphe de Flot de Données (Weyuker)
 - But : représenter les dépendances entre les données du programme dans le flot d'exécution.
 - Graphe de contrôle **décoré** d'informations sur les données (variables) du programme.
 - Structure : idem GC plus décoration
 - une définition (= affectation) d'une variable v est notée $\text{def}(v)$
 - une utilisation d'une variable est v notée $P_use(v)$ dans un prédicat et $C_use(v)$ dans un calcul/bloc.

Le test unitaire structurel

```
pgcd: integer is
local p,q : integer;
do
  read(p, q)          Def(p), Def(q)
  while p<> q do      Puse(p), Puse(q)
    if p > q          Puse(p), Puse(q)
      then
        p := p-q      Def(p), Cuse(q),Cuse(p)
      else
        q:= q-p      Def(q), Cuse(p),Cuse(p)
      end -- if
    end -- while
    result:=p         Cuse(p)
  end-- pgcd
```


Structurel/fonctionnel: conclusion

- Les critères structurels et fonctionnels sont complémentaires
 - une erreur d'omission ne peut pas être détectée par le test structurel
 - du code mort ne peut pas être détecté par le test fonctionnel
- Au niveau unitaire
 - on commence par le test fonctionnel
 - on complète par du test structurel

Outillage

- Programmes automatisant la vérification de la couverture de la structure de programme sous test
 - i.e. Coverlipse, Emma...