IUT Nantes
Pôle Sciences et technologie

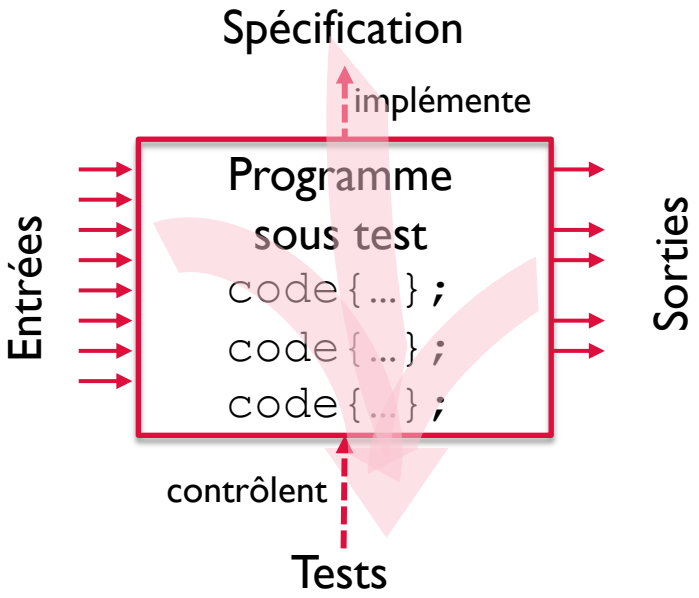
Nantes Université

R4.02 - Qualité de développement 3
CM2 – Test structurel

Jean-Marie Mottu
BUT info 2 – IUT Nantes

Couverture des tests : Création de tests avec différentes approches

- Test structurel (test boîte blanche)
  - Exploite la structure interne du programme



- Critères de test basées sur la couverture
  - Code
    - Instructions
    - Conditions
    - Chemins

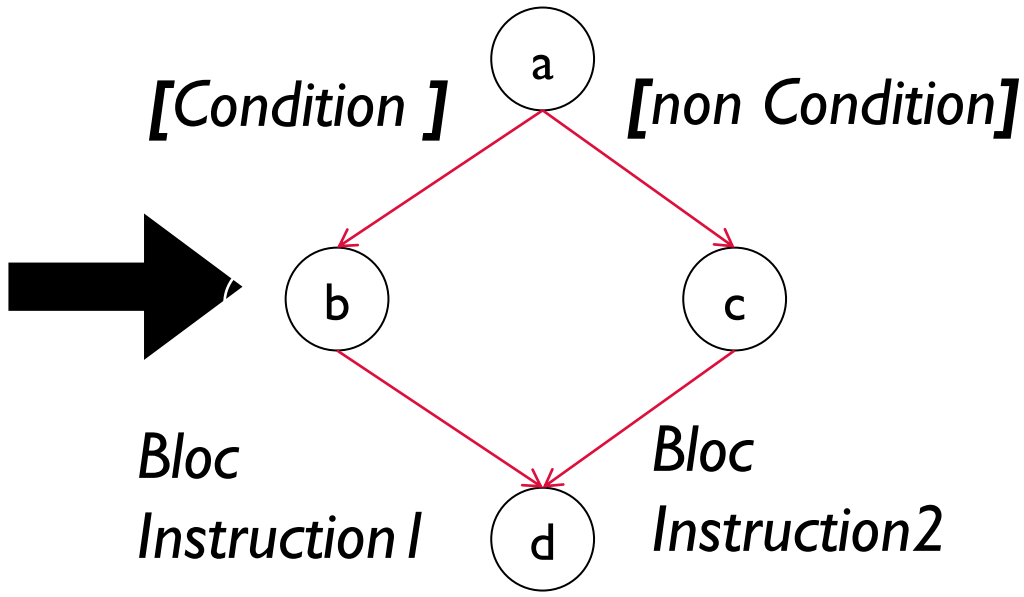
- Modéliser la structure du programme pour pouvoir formaliser les critères

Le test structurel: abstraire le code pour obtenir un critère formel

Code sous test :

si *Condition*
alors
Bloc Instructions1
sinon
Bloc Instructions 2

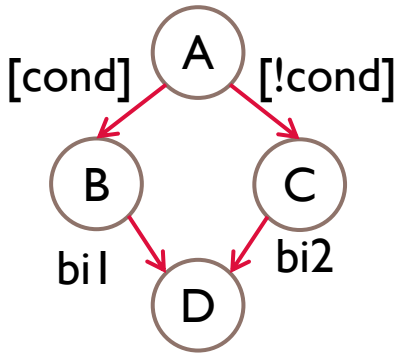
Modélisation
du code sous test :



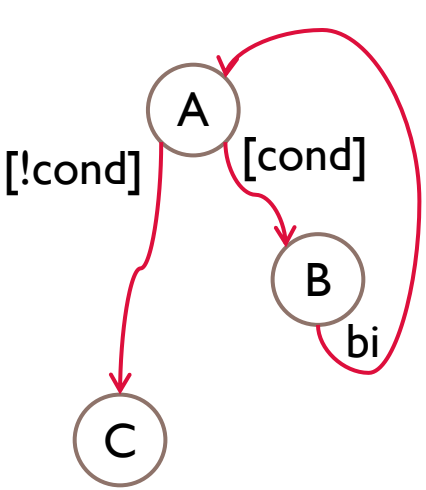
Modélisation des structures de contrôle

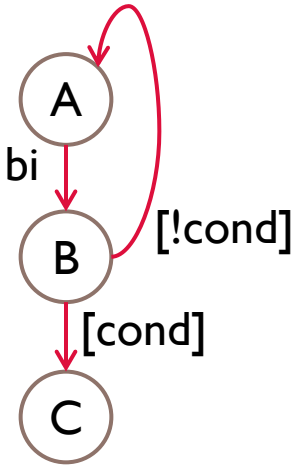
- Bloc d'instructions

- if then else

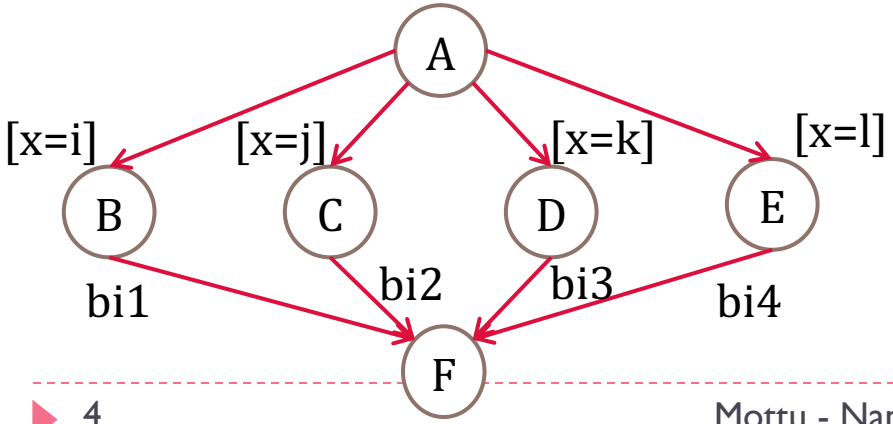


- while(cond) – until(cond)

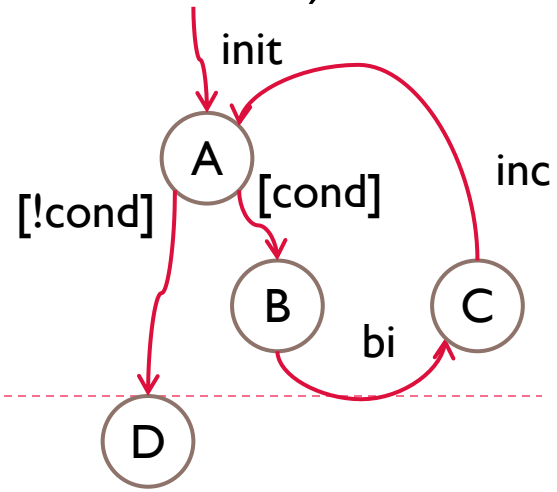




- Case



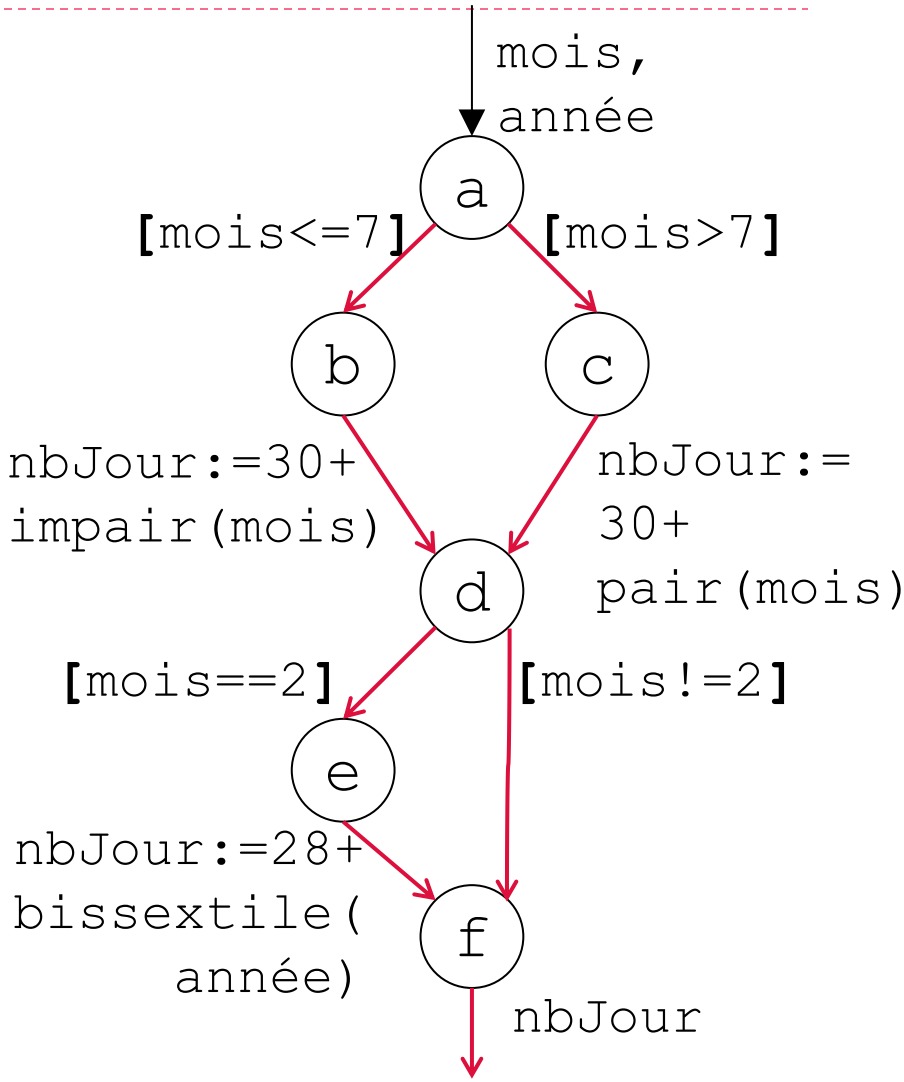
- For (init , cond , inc)



Modéliser avec un graphe de flot de contrôle GFC

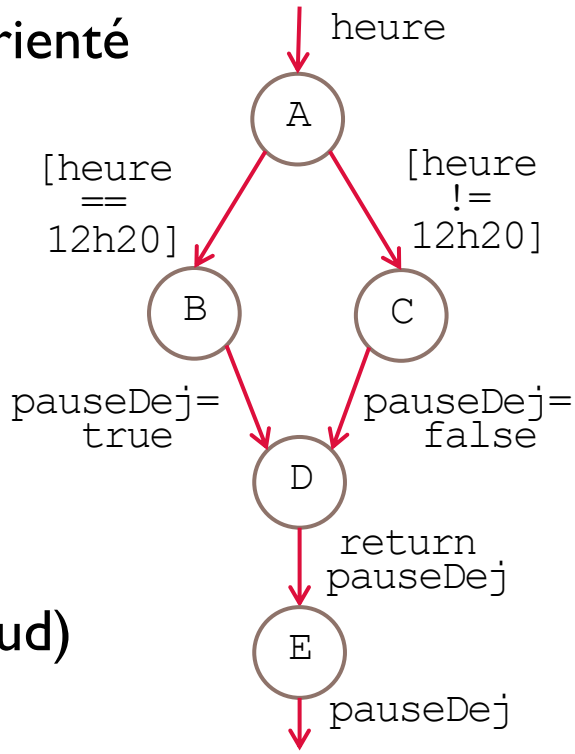
- Représente les possibilités de parcours du code

```
nbJoursDansMois
     (int mois, int année): int{
   si mois <= 7
   alors nbJour:=30+impair(mois)
   sinon nbJour:=30+pair(mois)

   si mois == 2
   alors
     nbJour:=28+
          bissextile(année)

   renvoyer nbJour
}
```



Sémantique et Règles d'écriture

- Le graphe de flot de contrôle GFC est un graphe orienté
- Les arcs sont unidirectionnels, ils supportent
  - les instructions
  - les conditions des prédicats entre crochet [ ]
- Les nœuds sont indexés
  - limites entre blocs élémentaires du programme
  - prédicats des conditionnelles /boucles
  - nœuds de jonction "vide" associé à un noeud prédicat
- Il a un arc d'entrée (seul arc ne venant pas d'un nœud)
  - supportant les variables formant les données de test
- Il a plusieurs arcs de sortie (seuls arcs n'atteignant pas un nœud)
  - supportant les variables ou les levées d'exceptions (tous comportements en général) devant être analysées par les oracles
- Pas d'arc vide
- Tout le code doit apparaitre dans le graphe (éventuellement abrégé)



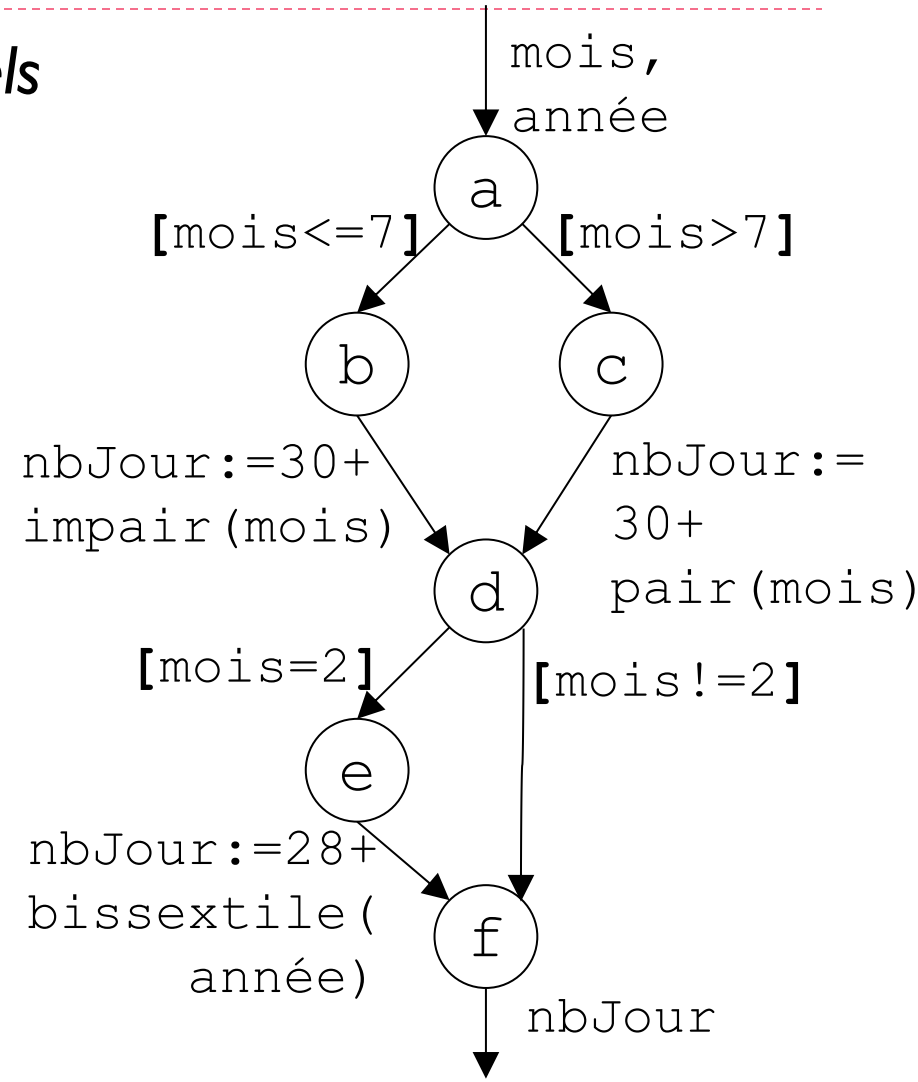
Exploiter le graphe de flot de contrôle pour sélectionner les données de test

- Créer les cas de test par les données de test
  - On associera un oracle à la DT pour former chaque cas de test
- Chaque donnée de test va parcourir le GFC depuis l'entrée jusqu'à une de ses sorties
  - Elle sensibilise ainsi un chemin (couvrant ses nœuds et ses arcs).
  - Une multitude, voire même une infinité de chemins dans un GFC
- Chemin :
  - suite de prédicat et d'instructions du code
  - suite d'arcs parcourus dans le GFC
    - en partant de l'arc d'entrée et finissant par un des arcs de sortie
  - en général représenté sans perte d'information par une liste de nœuds

Identification des chemins

- Liste finie de chemins *potentiels*
  - ch1 = (a, b, d, f)
  - ch2 = (a, c, d, f)
  - ch3 = (a, b, d, e, f)
  - ch4 = (a, c, d, e, f)

- Expression de l'ensemble des chemins potentiels :
  - Exp = (a, b, d, f)+(a, c, d, f)
    +(a, b, d, e, f)+(a, c, d, e, f)
- factorisable sous forme d'expression régulière
  - Exp = a(b | c) d (e)? f



Prédicat de chemins

- Prédicat de chemin :

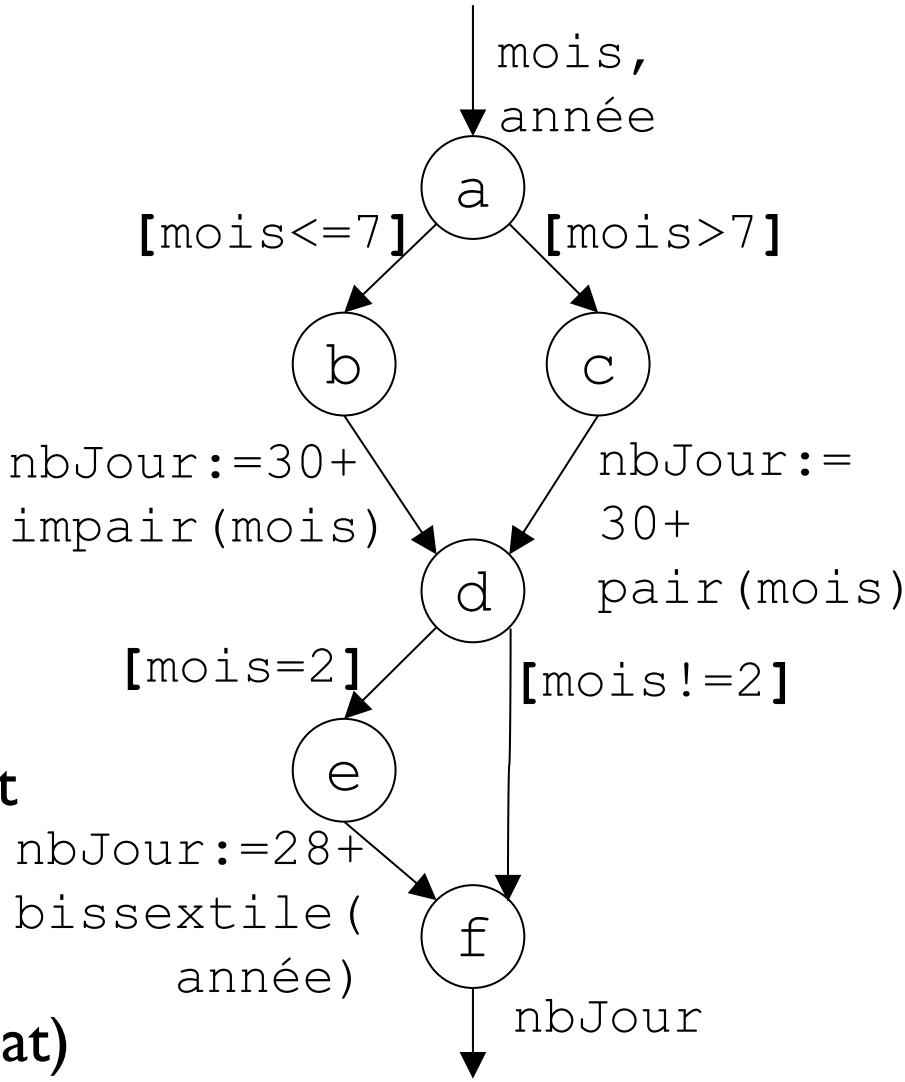
conjonction des conditions (ou de leur négation) rencontrées le long du chemin.

  - Pas toujours calculable
  - Se complique avec les boucles

Prédicat du ch1(a,b,d,f) =

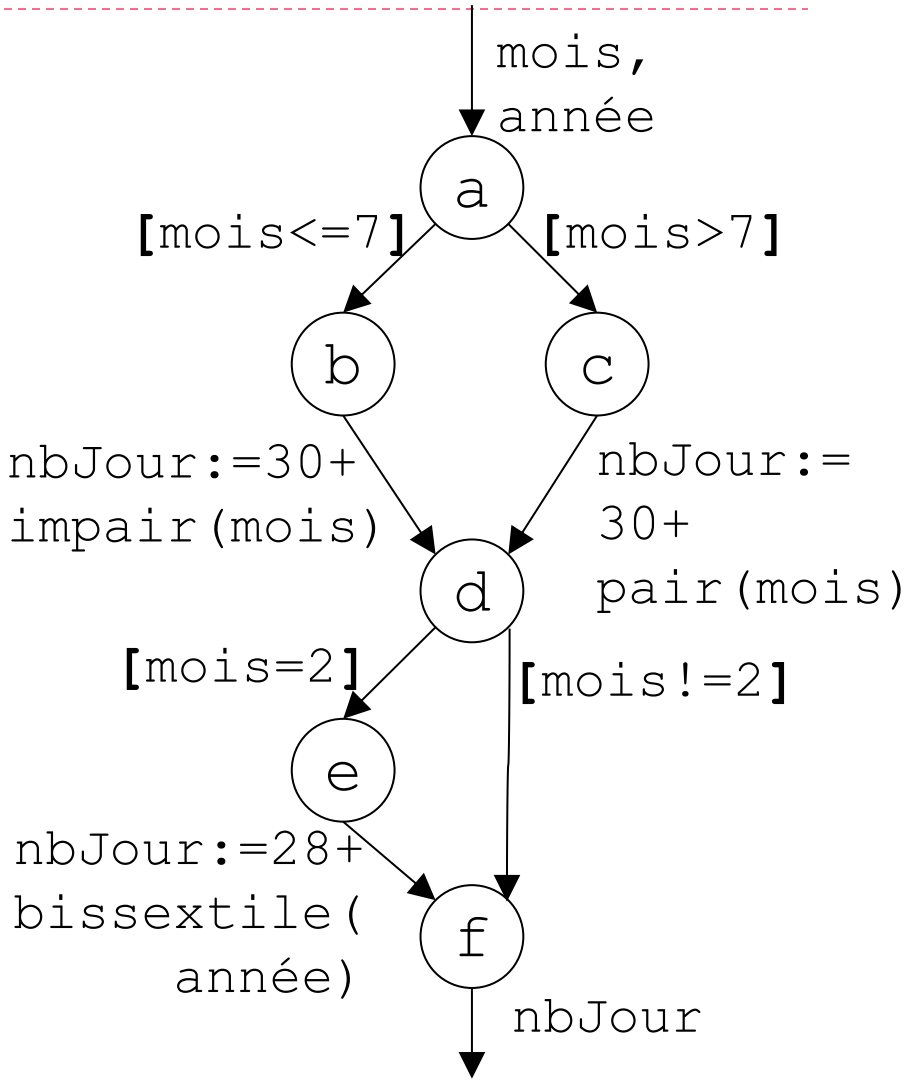
mois<=7 & mois !=2

(attention si une variable changeait plusieurs fois de valeur, il faudrait l'indexer : par exemple x=-x deviendrait $x_1 = -x_0$ dans le prédicat)



DT sensibilisant les chemins

- La DT1 = { mois=1, année=2019}
  sensibilise le ch1 = (a, b, d, f)
  qui est exécutable
- La DT2 = {mois=9, année=2019}
  sensibilise le ch2 = (a, c, d, f)
  qui est exécutable
- La DT3 = {mois=2, année=2019}
  sensibilise le ch3 = (a, b, d, e, f)
  qui est exécutable
- Aucune DT ne peut sensibiliser
  le ch4 = (a, c, d, e, f)
  qui n'est pas exécutable :
  un mois ne peut pas à la fois être
  >7 et =2



Chemins non-exécutables

- Tous les chemins ne sont pas exécutables
  - Prouver qu'un chemin est non-exécutable peut être un problème indécidable
  - Étant donné un chemin, trouver une DT qui exécute ce chemin ?
    - problème difficile corrélé à la testabilité de la méthode sous test
- Chemins non-exécutables réclament la vigilance de testeur
  - une erreur de codage ?
  - du code mort ?
  - un code pas optimisé ?

Quels chemins tester : Critère de couverture

- Plusieurs critères de test pour sélectionner différent sous-ensembles des chemins
  - Couvrant plus ou moins précisément le graphe de flot de contrôle
- Les nœuds, les arcs, les chemins exécutables sont nombreux
  - Comment les couvrir en maitrisant le compromis temps/coûts/objectifs ?
- Trois critères de couvertures pour obtenir des ensembles finis minimaux de tests couvrant :
  - Tous les nœuds
  - Tous les arcs
  - Tous les n-chemins

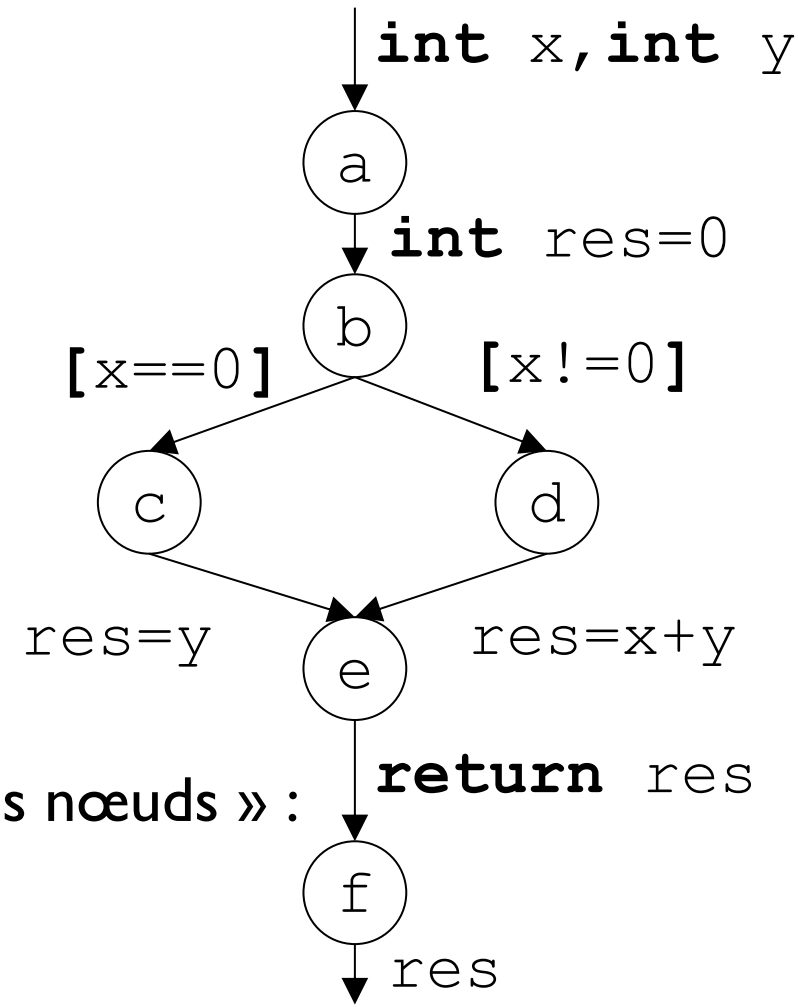
Critère « tous les nœuds »

- Critère le plus faible, aussi appelé TER1 (Test effectiveness ratio 1).
- Couverture de l'ensemble des nœuds du graphe.
- 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 pour « tous les nœuds »

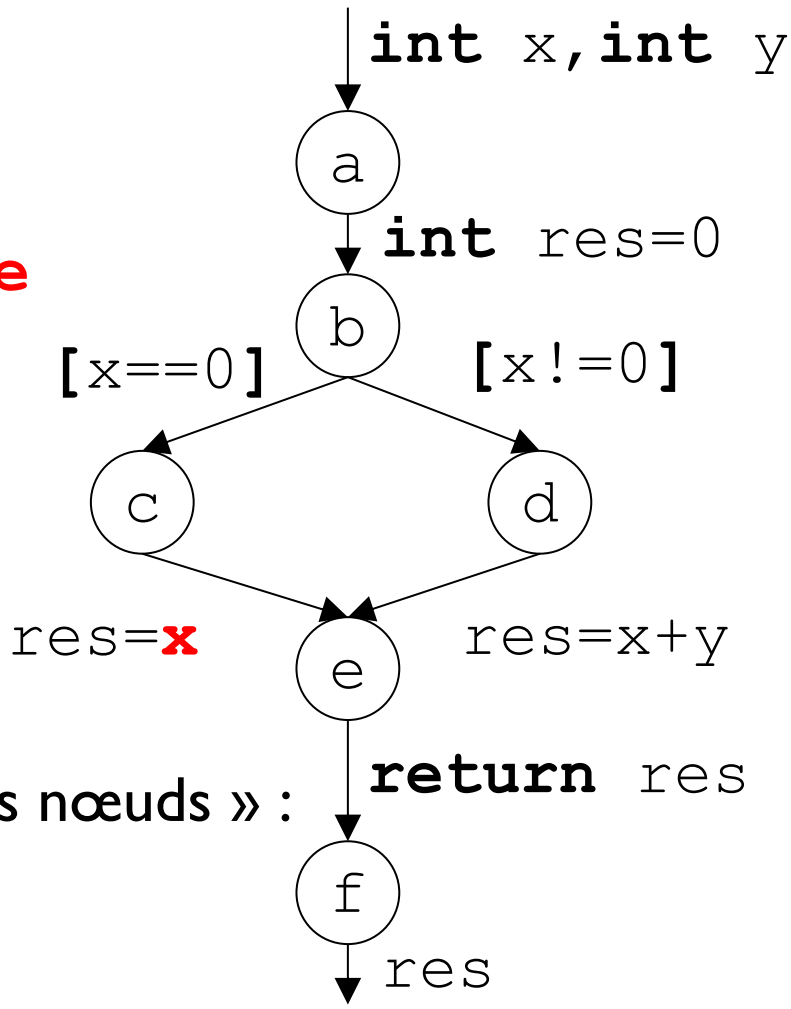
```
int sum(int x, int y){
      int res = 0;
      if (x == 0){
            res = y;
      } else {
            res = x + y;
      }
      return res;
}
```



- Expression des chemins potentiels :
- Chemins satisfaisant le critère « tous les nœuds » :
- Données de test sensibilisant chemins :
- Chemins exécutables :
- Cas de Test :

Exemple de l'efficacité de « tous les nœuds »

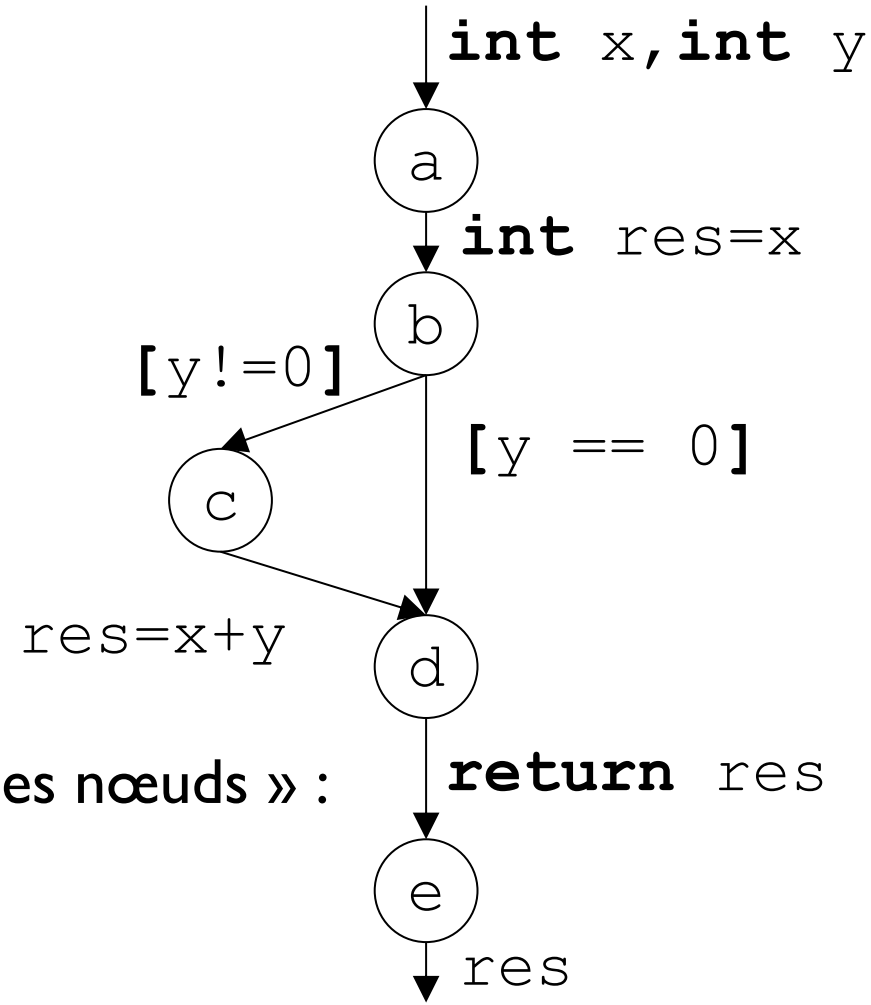
```
int sum(int x, int y){
        int res = 0;
        if (x == 0){
              res = x; // faute
        } else {
              res = x + y;
        }
        return res;
}
```



- Expression des chemins potentiels :
- Chemins satisfaisant le critère « tous les nœuds » :
- Données de test sensibilisant chemins :
- Chemins exécutables :
- Cas de Test :

Contre-exemple de l'efficacité de « tous les nœuds »

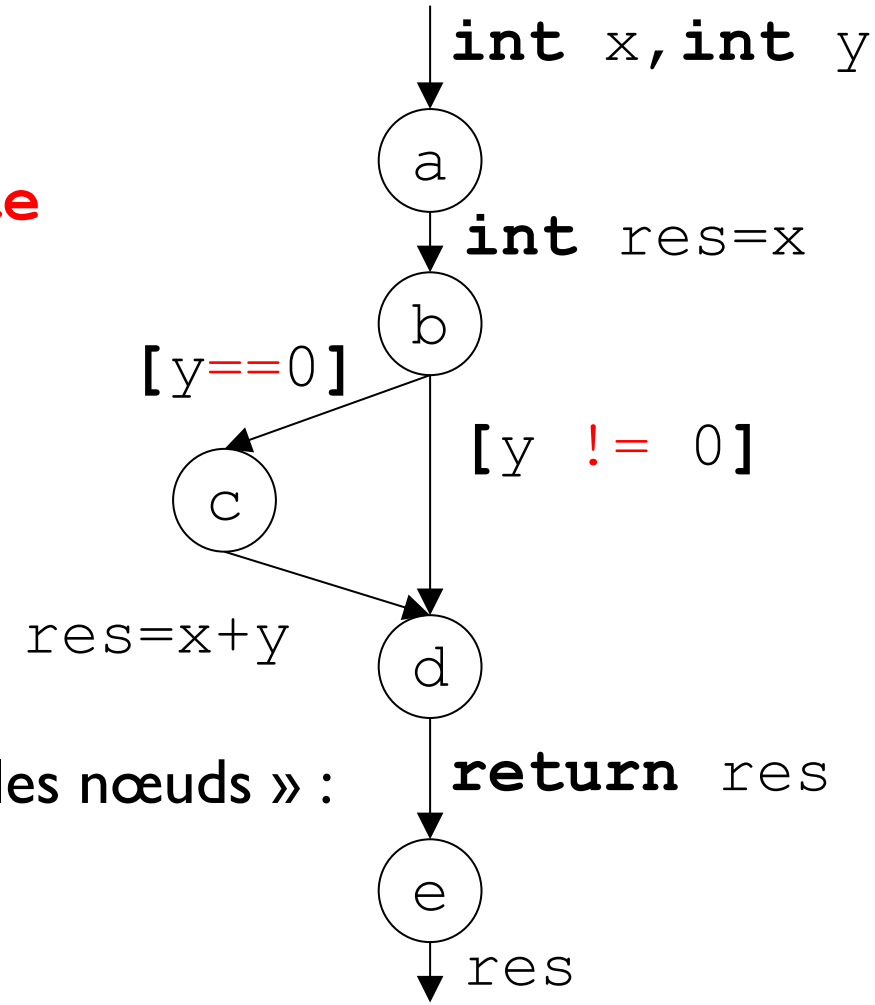
```
int sum(int x, int y){
      int res = x;
      if (y != 0){
            res = x + y;
      }
      return res;
}
```



- Expression des chemins potentiels :
- Chemins satisfaisant critère « tous les nœuds » :
- DT sensibilisant chemins :
- Chemins exécutables :
- Cas de Test :

Contre-exemple de l'efficacité de « tous les nœuds »

```
int sum(int x, int y){
      int res = x;
      if (y == 0){// faute
            res = x + y;
      }
      return res;
}
```



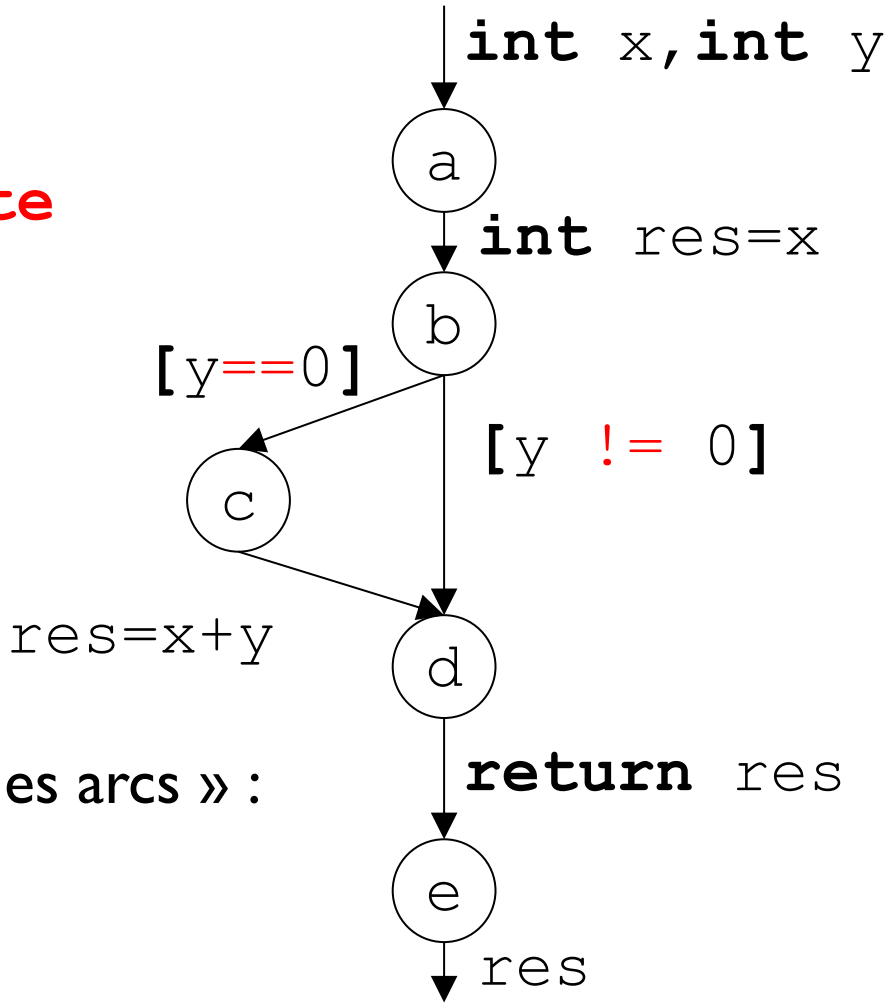
- Expression des chemins potentiels :
- Chemins satisfaisant critère « tous les nœuds » :
- DT sensibilisant chemins :
- Chemins exécutables :
- Cas de Test :

Critère "tous les arcs"

- ou TER2 :
  - tous les arcs du graphe de contrôle sont couverts
  - toutes les branches conditionnelles ont été couvertes
  - taux de couverture = $\frac{\text{nombre des arcs couverts}}{\text{nombre total des arcs}}$
  - chaque condition prend une fois la valeur "vrai" et "faux" ;
  - attention aux prédicats composés.
- Signifie que toutes les instructions ont été couvertes et que les conditions prennent les deux valeurs : vrai et faux
- "Tous les arcs" => "tous les nœuds".
  - L'inverse n'est pas vrai.

Exemple « tous les arcs »

```
int sum(int x, int y){
      int res = x;
      if (y == 0){// faute
            res = x + y;
      }
      return res;
}
```



- Expression des chemins potentiels :
- Chemins satisfaisant critère « tous les arcs » :
- DT sensibilisant chemins :
- Chemins exécutables :
- Cas de Test :

Critère « tous les chemins »

- "Tous les chemins" =>"tous les arcs" => "tous les nœuds"
- Problème des boucles :
  - Chemin "limite" : traversée de la boucle sans itération
  - Chemin "intérieur" : itération de la boucle une seule fois
  - Chemin "élémentaire" : entre 0 et 1 passage dans la boucle

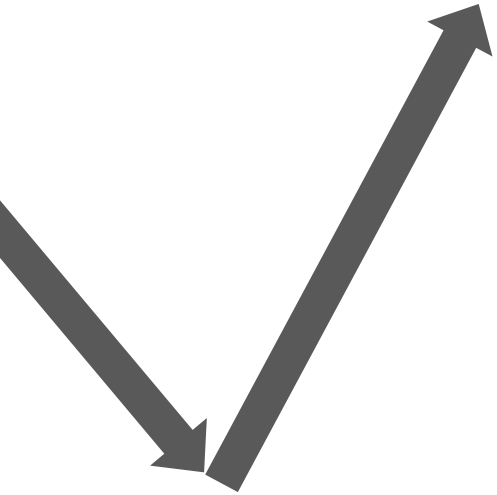
- test exhaustif = « tous les chemins avec toutes les valeurs possibles »
  - Pas de solution Ø car potentiellement infini avec des boucles

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
  - Par exemple des « if then else », dans les « while » ou les « for »

Rappel de BUT1 – R2.03 (QD1)
Etapes et hiérarchisation des tests



Tests de validation	**Test fonctionnel**	
Tests d'intégration		**Test structurel**
Tests unitaires		

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