Cours de Test Logiciel

Leçon 3 : Sélection de tests boîte blanche

Sébastien Bardin

CEA-LIST, Laboratoire de Sûreté Logicielle

Méthodes de sélection Boite Blanche

Couverture Structurelle

- avec couverture du graphe de flot de contrôle (CFG)
- avec couverture du graphe de flot des données (DFG)

Test par mutation (cf prochain cours)

- sélection des CT par rapport à leur effet au changement du système.

Plan

- Contexte
- Couverture du CFG
- Couverture du DFG
- Mutations
- Discussion

Control-Flow Graph (CFG)

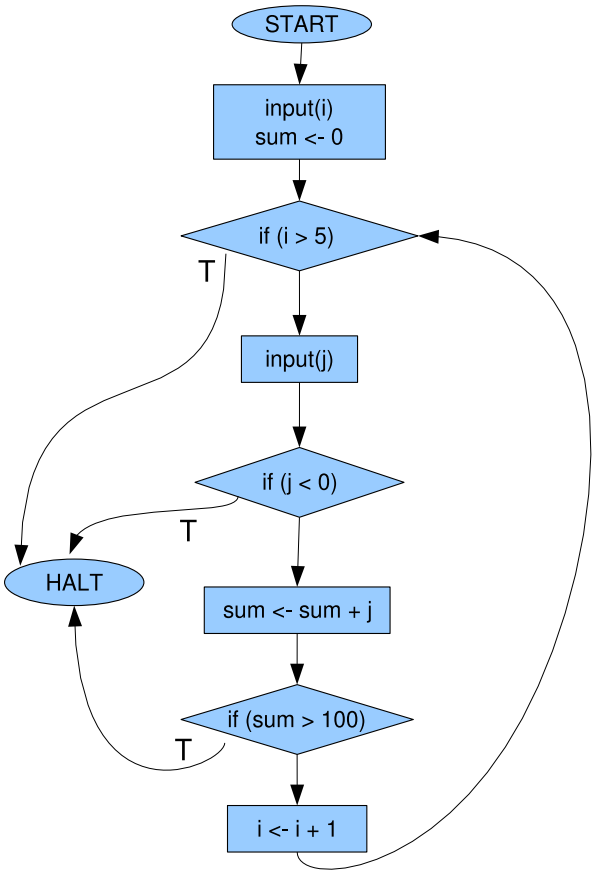
Le graphe de flot de contrôle d'un programme est défini par :

- un noeud pour chaque instruction, plus un noeud final de sortie
- pour chaque instruction du programme, le CFG comporte un arc reliant le noeud de l'instruction au noeud de l'instruction suivante (ou au noeud final si pas de suivant), l'arc pouvant être étiquetté par l'instruction en question

Quelques définitions sur les instructions conditionnelles :
`if (a<3 && b<4) then ... else ...`

- un `if` est une instruction conditionnelle / branchante
- `(a<3 && b<4)` est la condition
- les deux décisions possibles sont
  ((`a<3 && b<4`),*true*) et ((`a<3 && b<4`),*false*)
- les conditions simples sont `a<3` et `b<4`

Control-Flow Graph (CFG)




```
START
input(i)
sum := 0
loop : if (i > 5) goto end
input(j)
if (j < 0) goto end
sum := sum + j
if (sum > 100) goto end
i := i +1
goto loop
end : HALT
```

Couverture du CFG

Quelques critères de couverture sur flot de contrôle

- Tous les noeuds (I) : le plus faible.
- Tous les arcs / décisions (D) : test de chaque décision
- Toutes les conditions simples (C) : peut ne pas couvrir toutes les décisions
- Toutes les conditions/décisions (DC)
- Toutes les combinaisons de conditions (MC) : explosion combinatoire !
- Tous les chemins : le plus fort, impossible à réaliser s'il y a des boucles

Exemple

```
1: input(A,B,C : bool)
2: if (A && B)
3:   then  return OK
4:   else if (C)
5:     then return OK
6:     else KO
```

Couverture :

Exemple

```
1: input(A,B,C : bool)
2: if (A && B)
3:   then  return OK
4:   else if (C)
5:     then return OK
6:     else KO
```

Couverture : D

- ligne 2 : ((A && B),true), ((A && B),false)
- ligne 4 : (C,true), (C,false)

Exemple

```
1: input(A,B,C : bool)
2: if (A && B)
3:   then  return OK
4:   else if (C)
5:     then return OK
6:     else KO
```

Couverture : C

- ligne 2 : (A,true), (A,false), (B,true), (B,false),
- ligne 4 : (C,true), (C,false)

Exemple

```
1: input(A,B,C : bool)
2: if (A && B)
3:    then  return OK
4:    else if (C)
5:       then return OK
6:       else KO
```

Couverture : MC

- ligne 2 : ((A,true),(B,true)), ((A,true),(B,false)), ((A,false),(B,true)), ((A,false),(B,false))
- ligne 4 : (C,true), (C,false)

Limites du critère “Instructions”

```
read(x);
if x != 0 then x:=1;
y := a/x
```

Le CT $\{x = 1\}$ permet de couvrir tous les noeuds, mais manque le bug dans le cas où $x == 0$. Ce bug est trouvé par un TS couvrant D ou C (ou plus).

Note : chaque critère a des limitations similaires

MCDC

Utilisé en avionique (DO-178B). But :

- puissance entre DC et MC
- ET garde un nombre raisonnable de tests

Définition

- critère DC
- ET les tests doivent montrer que chaque condition atomique peut influencer la décision :

  par exemple, pour une condition $C = a \wedge b$, les deux DT $a = 1, b = 1$ et $a = 1, b = 0$ prouvent que b seul peut influencer la décision globale C

Remarques :

- problème si conditions atomiques liées
- que faire quand une expression booléenne complexe est "cachée" dans une expression ou dans un appel de fonction ?

  $a := (b \wedge c) \vee (d \wedge \neq c); if\,(a)\,then...else...$

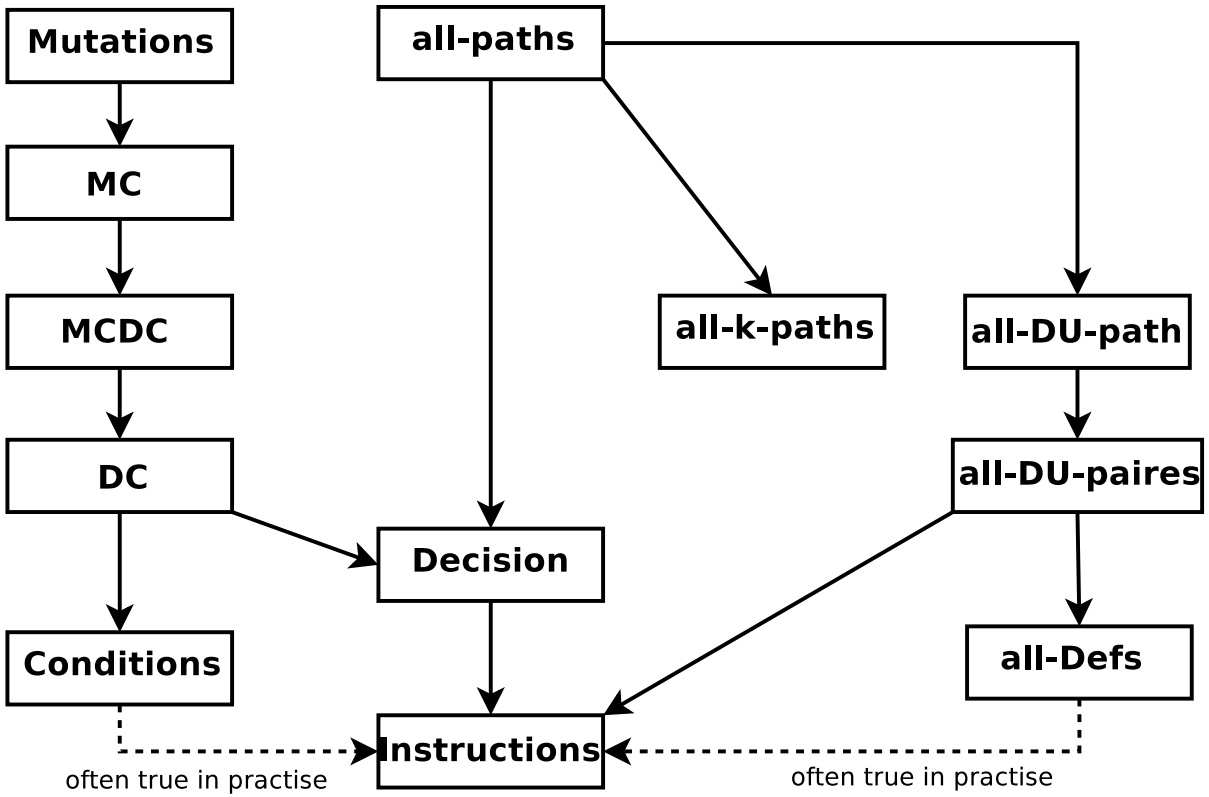
Hiérarchie des critères

Notion de hiérarchie entre ces différents critères de couverture

Le critère CT1 est plus fort que le critère CT2 (CT1 *subsumes* CT2, noté CT1 $\succeq$ CT2) si pour tout programme P et toute suite de tests TS pour P, si TS couvre CT1 (sur P) alors TS couvre CT2 (sur P).

Exercice : supposons que TS2 couvre CT2 et trouve un bug sur P, et TS1 couvre un critère CT1 te CT1 $\succeq$ CT2.
Question : TS1 trouve-t-il forcément le même bug que TS2 ?

Hiérarchie des critères (2)



Remarque : éléments non couvrables

Certains chemins du CFG sont infaisables

De même certaines instructions / branches peuvent ne pas être couvrables

- exemple : `if (debug = true) then ... else ()` et la variable `debug` est initialisé à `false`
- exemple : tests de défaillance matérielle : `x := 0 ; if (x!=0) then problem() else ();`
- etc.

Cela gêne considérablement l'effort de test :

- perte de temps pour couvrir un élément non couvrable
- diminue artificiellement le taux de couverture atteint

Évidemment, savoir si un élément du CFG est couvrable est indécidable

- à faire à la main

Plan

- Contexte
- Couverture du CFG
- Couverture du DFG
- Mutations
- Discussion

Data-Flow Graph (DFG)

Les critères basés sur le flot de données sélectionnent les données de test en fonction des définitions et des utilisations des variables

Définitions sur les occurrences de variables :

- une variable est définie lors d'une instruction si la valeur de la variable est modifiée (affectations, déclaration)
- une variable est dite référencée si la valeur de la variable est utilisée.
- Si la variable référencée est utilisée dans le prédicat d'une instruction de décision, il s'agit d'une p-utilisation, sinon il s'agit d'une c-utilisation (calcul)

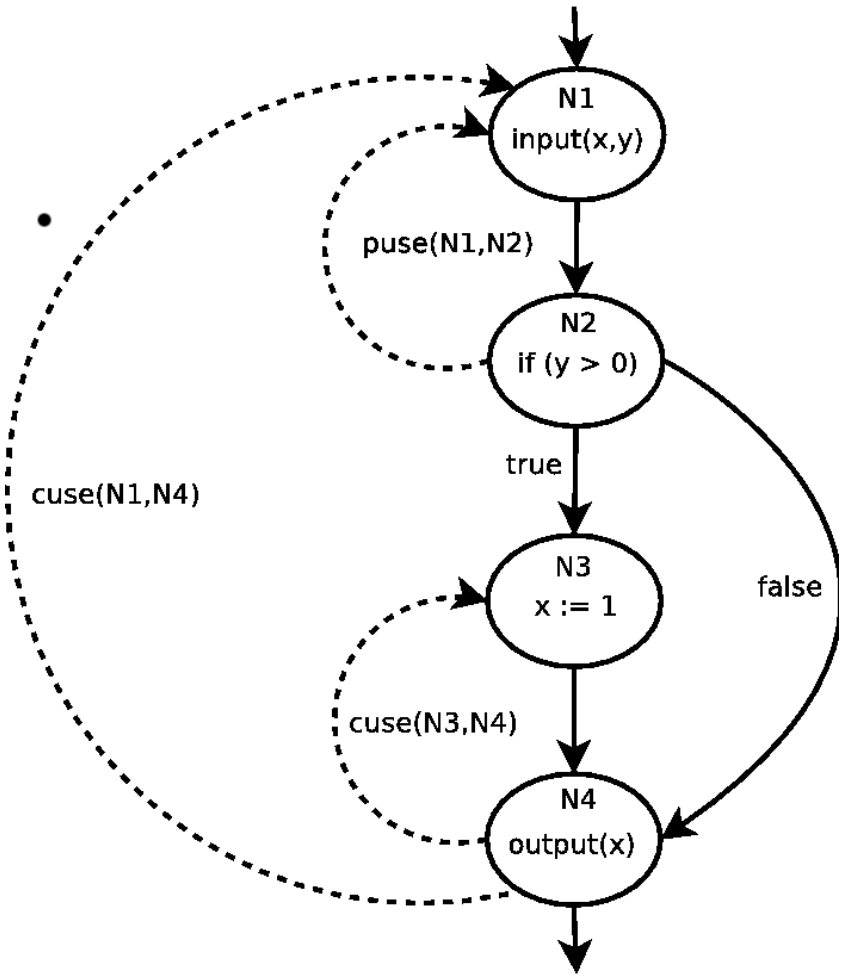
Couverture du DFG

Graphe de dépendance de donnée (data dependance graph) : ajoute au CFG les informations liant définition (def) et utilisation (use) de variables

Définitions :

- def(x) : instruction où x est définie
- use(x) : instruction où x est utilisée
- paire Def-Use pour x (noté DU(x)) : paire d'instructions (Def,Use) pour x, telles que il existe au moins une trace d'exécution passant par Def puis par Use, tq x n'est pas redéfini entre Def et Use
- chemin Def-Use (DU-path) pour x : un chemin passant par une paire (Def,Use) de x comme expliqué avant

Couverture du DFG



Couverture du DFG

Critères :

- $all - defs - one - use$ ($all - def$) : pour chaque définition D, avoir un test qui relie D a une de ses utilisations
- $all - du - one - path$ ($all - uses$) : couvrir au moins un chemin pour chaque paire DU

  variantes $all - c - uses$ ou $all - p - uses$
- $all - du - all - paths$ ($all - du - paths$) : couvrir tous les chemins de chaque paire DU

Plan

- Contexte
- Couverture du CFG
- Couverture du DFG
- Mutations
- Discussion

Critères structurelles : autres applications (1)

En général les critères structurels sont présentés pour la couverture de code au niveau unitaire
On peut adapter à d'autres niveaux de tests / artefact

Critères structurels à partir du code, niveaux intégration / système

difficulté = passage à l'échelle. Aux niveaux système ou intégration, on peut

- couvrir uniquement quelques fonctions principales (`main`)
- utiliser des critères plus lâches sur tout le code
  - couvrir le CallGraph : entrées de fonctions (noeuds du CG), appels de fonctions (transitions du CG)
- cibler les besoins, ex : couvrir toutes les utilisations de `lock`, `unlock`

Quelques repères :

- code critique, test unitaire : 100% du MC/DC
- code normal : facile : 50% I, bien : >85% D

Critères structurelles : autres applications (2)

Critères structurels sur autre chose que le code

Exemple : couverture structurelle

- spécifications : couverture du workflow ou use-case
- conception : couverture du modèle (automate étendu)

Exemple : mutations

- spécifications : mutation du workflow
- conception : mutation du modèle
- test d'intégration : mutation des interfaces

Couverture structurelle et vrais bugs

Seul le test statistique donne formellement une mesure de qualité du logiciel

Expérimentalement :

- mutations : score élevé corrélé à bon pouvoir de détection de bugs
- couverture : corrélation au début, puis stagnation

Remarque : pour la couverture, distinguer programmes orientés contrôle et orientés données

Outils

Couverture : outils de calcul de divers critères pour un jeu de test donné

- `gcover`, `coverlipse`

Mutations : création des mutants (choix d'opérateurs), calcul du score de mutation

- `Proteum`, `Lava`

Génération automatique de DT pour couvrir les branches

- `Pex` pour C#
- prototypes académiques (`PathCrawler`, `JavaPathFinder`, `Cute`, `DART`, `EXE`, etc.)
- Voir fin du cours pour comprendre comment fonctionnent ces outils