Tests, graphe de flot de contrôle et couverture de code

Quentin Fortier

August 1, 2022

Preuve formelle par invariant

Pour prouver qu'un programme est correct, on utilise souvent des invariants de boucle. On peut ajouter des annotations au code pour spécifier les conditions et les invariants :

```
bool dichotomie(int e, int* t, int n) {
    // t est un tableau trié
    int i = 0, j = n - 1;
    while (i <= j) {
        // si e est dans t alors e est dans t[i..j]
        int m = (i + j) / 2;
        if (t[m] == e)
            return 1;
        if (t[m] <= e)
            i = m + 1;
        else
            j = m - 1;
    }
    return 1;
}
```

Tests

Malheureusement, il est difficile de trouver des invariants.
À la place, on peut utiliser des tests, qui consistent à vérifier, sur des entrées particulières, que la sortie du programme est conforme à la spécification.

Tests

Malheureusement, il est difficile de trouver des invariants.
À la place, on peut utiliser des tests, qui consistent à vérifier, sur des entrées particulières, que la sortie du programme est conforme à la spécification.

Types de tests :

- **Test unitaire** : Une petite portion de code à la fois (une fonction, par exemple).
- **Test fonctionnel** : Conformité du logiciel développé à la spécification.
- **Test de non-régression** : Teste que l'ajout de code n'a pas introduit d'erreurs sur le code déjà existant.
- **Test de performance** : Rapidité, précision...
- ...

Tests

Pour choisir des valeurs à tester parmi toutes les entrées possibles (possiblement une infinité) :

- On peut partitionner le domaine d'entrée : Choisir des éléments parmi toutes les entrées possibles.
- Choisir des conditions aux limites : Par exemple si n est une entrée entre 0 et 100, on pourra tester, entre autres, sur 0 et 100.

Tests

QCheck est une librairie OCaml pour tester une fonction sur des entrées aléatoires (*fuzz testing*) en donnant une propriété que doit vérifier la fonction (voir mp2i-library):

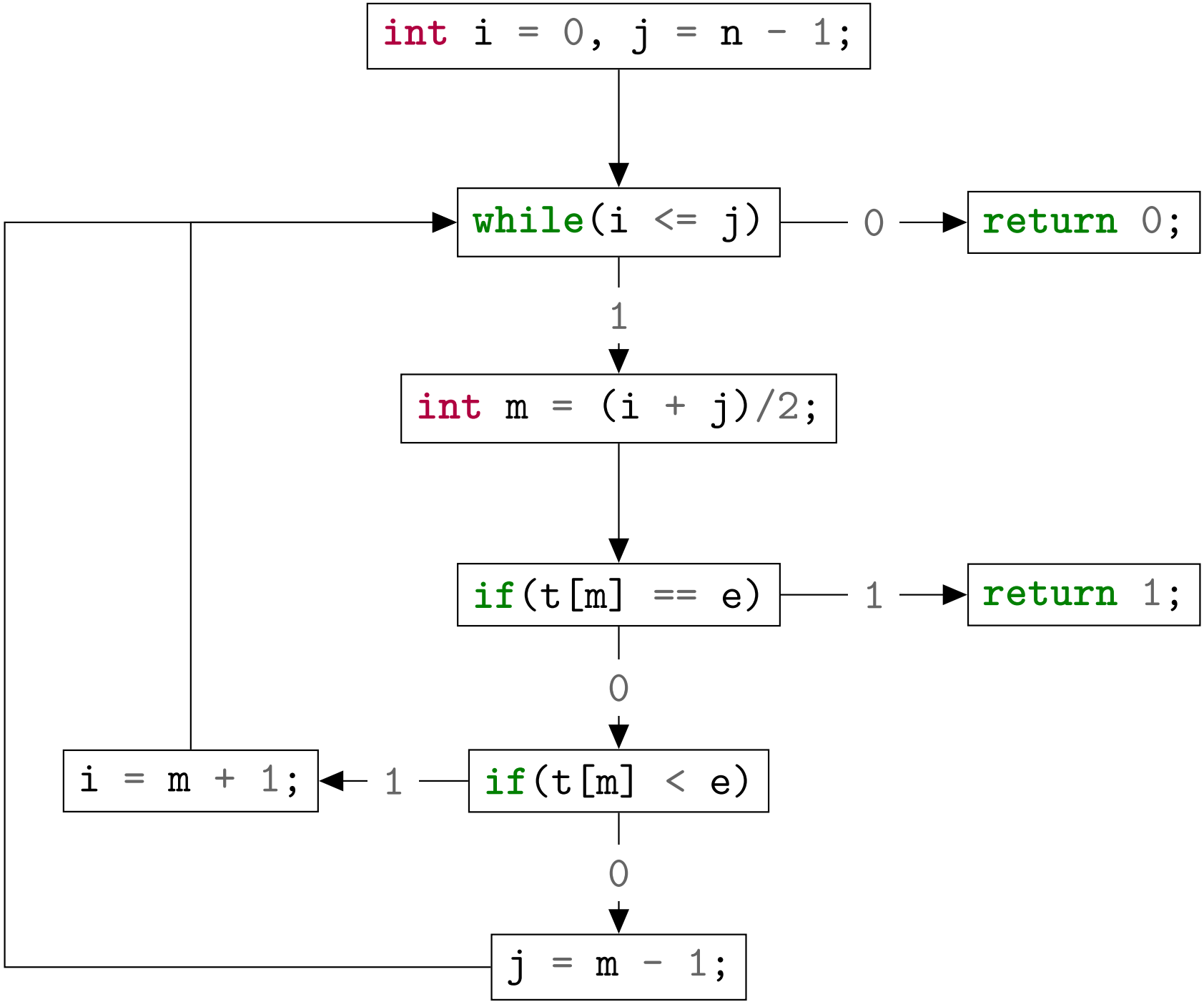
```
let test_mergesort l =
    List.sort Int.compare l = Mp2i.Mergesort.sort l
let t = QCheck.(Test.make ~count:1000 (list int) test_mergesort)
QCheck_runner.run_tests [t]
```

```
--- Failure ------------------------------------------

Test anon_test_1 failed (548 shrink steps):

[-2; 0; -1]
```

Graphe de flot de contrôle (CFG)

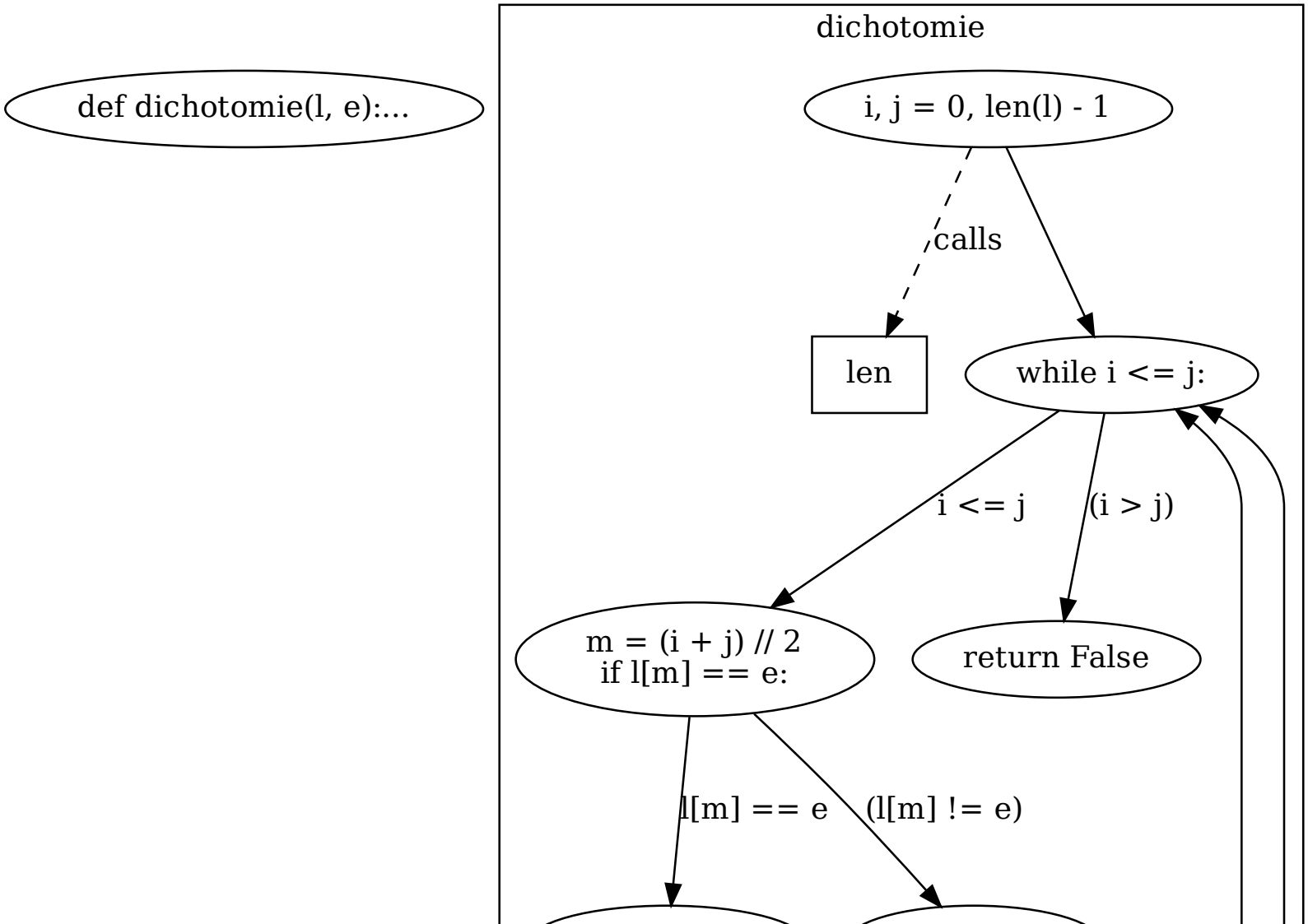
Le **graphe de flot de contrôle** (Control Graph Flow) d'un programme est un graphe orienté avec :

- (Un sommet d'entrée, un sommet de sortie.)
- Un sommet pour chaque bloc d'instructions consécutives.
- Un sommet pour chaque condition ou boucle.
- Des arcs pour relier des blocs consécutifs. Les arcs sortants d'un condition ou une boucle sont étiquetées par la condition.


int i = 0, j = n - 1;
while(i <= j)
0
return 0;
1
int m = (i + j)/2;
if(t[m] == e)
1
return 1;
0
i = m + 1;
1
if(t[m] < e)
0
j = m - 1;

Graphe de flot de contrôle (CFG)

Génération automatique de CFG en Python avec staticfg :



Graphe de flot de contrôle (CFG)

Une exécution du programme (sur une entrée donnée) correspond à un **chemin** dans le CFG.

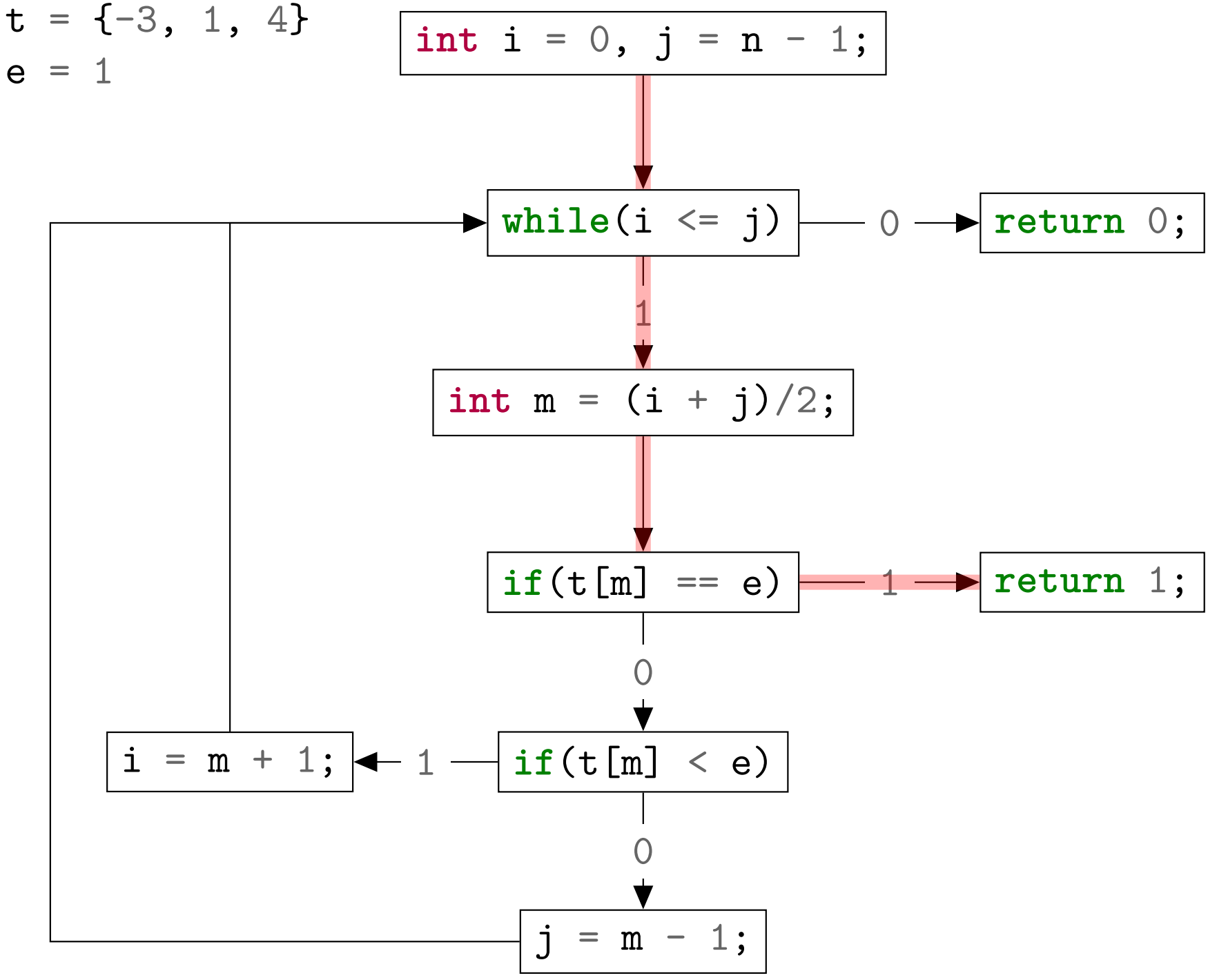
Graphe de flot de contrôle (CFG)

Une exécution du programme (sur une entrée donnée) correspond à un **chemin** dans le CFG.

Un chemin est **faisable** s'il correspond à l'exécution d'un programme sur une entrée donnée.

Exercice

Donner un exemple de CFG avec un chemin infaisable.


t = {-3, 1, 4}
e = 1
int i = 0, j = n - 1;
while(i <= j)
0
return 0;
1
int m = (i + j)/2;
if(t[m] == e)
1
return 1;
0
i = m + 1;
1
if(t[m] < e)
0
j = m - 1;

Couverture de code

Pour tester en utilisant le CFG, on peut choisir des tests qui couvrent un certain ensemble de chemins :

- **Couverture de tous les chemins** (*path cover*) : On choisit autant de tests qu'il y a de chemins possible.
  Mais il peut y avoir une infinité de chemins...

Couverture de code

Pour tester en utilisant le CFG, on peut choisir des tests qui couvrent un certain ensemble de chemins :

- **Couverture de tous les chemins** (*path cover*) : On choisit autant de tests qu'il y a de chemins possible.
  Mais il peut y avoir une infinité de chemins...
- **Couverture de toutes les instructions** (*statement cover*) : On écrit des tests de façon à ce que chaque instruction soit exécuté au moins une fois.

Couverture de code

Pour tester en utilisant le CFG, on peut choisir des tests qui couvrent un certain ensemble de chemins :

- **Couverture de tous les chemins** (*path cover*) : On choisit autant de tests qu'il y a de chemins possible.
  Mais il peut y avoir une infinité de chemins...
- **Couverture de toutes les instructions** (*statement cover*) : On écrit des tests de façon à ce que chaque instruction soit exécuté au moins une fois.
- **Couverture de toutes les branches** (*branch cover*) : On écrit des tests de façon à ce que par chaque arc passe au moins un chemin.

Couverture de code

Pour tester en utilisant le CFG, on peut choisir des tests qui couvrent un certain ensemble de chemins :

- **Couverture de tous les chemins** (*path cover*) : On choisit autant de tests qu'il y a de chemins possible.
  Mais il peut y avoir une infinité de chemins...
- **Couverture de toutes les instructions** (*statement cover*) : On écrit des tests de façon à ce que chaque instruction soit exécuté au moins une fois.
- **Couverture de toutes les branches** (*branch cover*) : On écrit des tests de façon à ce que par chaque arc passe au moins un chemin.

Taux de couverture : $\dfrac{\text{nombre sommets/arcs parcourus}}{\text{nombre sommets/arcs}}$

Couverture de code

Exercice

Écrire un ensemble de tests couvrant toutes les branches de `dichotomie(int e, int* t, int n)`.

Couverture de code

Exercice

Écrire un ensemble de tests couvrant toutes les branches de `dichotomie(int e, int* t, int n)`.

bisect_ppx est un outil de couverture de code en OCaml, pour savoir quelles sont les instructions couvertes par les tests.

Exemple de rapport pour mp2i-library