Exemples de méthodes et outils pour la correction des programmes.

Définition (spécification): Ensembles des propriétés qu'un programme est censé respecter. Un programme est correct s'il respecte sa spécification.

Definition (Types de propriétés):
- vivacité: Une propriété de vivacité est une propriété qui indique que le programme passe toujours par un état voulu
- sureté: Une propriété de sureté est une propriété qui indique que le programme ne passe jamais par un état non voulu.

exemple: la fonction f appliquée à un entier:
+ ne cause pas d'erreur
+ termine
+ quand elle termine correctement, c'est l'identité

I Validation (preuve de correction)

TRm (Rice): Toute propriété "non triviale" liant les entrées et les sorties d'un programme est indécidable.

1) Typage

Idée: On associe aux variable et fonctions d'un programme, un type afin de s'assurer que les opérations effectuées le sont sur des variables ayant un type adapté.

Remarque 1: La plupart des langages de programation sont dotés d'un typeur qui verifie que le type des variables et fonction sont coherent.

Remarque 2: La plupart des systèmes de type panède des algorithmes d'inférence de type. Alors, il n'est pas necessaire d'expliciter les types des variables et de fonctions pour que le programme sait typer

Remarque 3: le fait qu'une fonction ait un type de sortie ne prédit en aucun cas que cette fonction termine ou ne lève pas d'erreur.

Exemple: avant de verifier si f est l'identité on peut appeler le typeur pour verifier que f a le type: int → int.

2) Sémantique

idée: on va modéliser et formaliser le comportement des element d'un langage de programation afin de prouver que le comportement d'un programme est correct.

a) Sémantique à grands pas

Idée : on va associer à un programme p et un état σ l'état σ' obtenu après l'exécution de p à partir de σ.

Definition : Une sémantique opérationnelle à grand pas d'un langage est un ensemble de triplet $\langle \sigma, p, \sigma' \rangle$ où σ et σ' sont des états et p est un programme du langage.

On va définir ces triplets par un ensemble de règles définies inductivement sur le langage.

Remarque : Les triplets de la sémantique à grand pas n'existe que pour les programmes qui terminent.

b) Sémantique à petits pas

Idée : on associe à un programme p et un état σ l'état σ obtenu et le programme p' restant à faire après une étape d'exécution de p à partir de σ.

Definition : Une sémantique opérationnelle à petits pas d'un langage est un ensemble de quadruplets $\langle \sigma, p, \sigma', p' \rangle$ où σ et σ' sont des états et p et p' sont des programmes du langage.

On définit ces quadruplets par un ensemble de règles définies inductivement sur le langage.

Remarque : On peut définir la sémantique à petits pas d'un programme ne terminant pas.

c) Exemple de sémantiques d'un langage Dev 1

3) Outils mathématique pour la preuve de correction.

a) ordre bien fondé

Definition : L'ordre $\leqslant$ est bien fondé sur l'ensemble E si il n'existe pas de suite infinie strictement décroissante pour $\leqslant$ dans E. Alors $(E, \leqslant)$ est un ensemble bien fondé.

Usage : Les inductions sur des ensemble bien fondés sont très utile pour la validation en programmation fonctionelle

b) variants et invariants

Definition (invariant) : Une quantité i (dépendant des variables) est un invariant du programme p si la valeur de i est constante durant l'exécution de p.

Definition (variant de terminaison) : Une quantité i est un variant de terminaison du programme si i est constament dans $\mathbb{N}$ et si la valeur de i décroit strictement à chaque étape de l'exécution de p.

c) Structures de Kripke Dev 2

II vérification (tests de corrections)

1) White / black box.

idée: On dit que l'on manipule le programme à tester comme une "boîte blanche" si l'on a accès au code et qu'on le manipule comme une "boîte noire" si on ne peut que l'exécuter.

Remarque: Pour la validation on était nécessairement en mode boîte blanche.

2) Tests aléatoires

idée: pour essayer de couvrir un maximum de cas, on peut générer des jeux de tests avec des valeurs aléatoires.

3) Tests adaptatifs

Idées: On modifie répétitivement un jeu de test dépendant le code traversé (= white box) / le résultat et le temps d'exécution (= black box) dans le but de mieux circonscrire les cas critiques ou faux.

4) Évaluations de la qualité des tests.

4.1) Couverture

a) statement coverage

Idée: Compter la proportion de ligne de code exécutées par au moins un test.

b) branch coverage

idée: On compte la proportion de branchement conditionels du code machine dont les deux branches sont exécutés aux moins une fois.

Plus fin que le statement coverage.

c) couverture des limites

idée: On vérifie que des chemins d'exécutions considérés comme les cas limite sont bien exécutés au moins une fois.

d) Modified condition/decision coverage (MC/DC)

Def: Le jeu de test est MC/DC si

- Tout les points d'entrée et de sortie du code sont appelés au moins une fois.
- Chaque formule booléenne est vrai et fausse une fois.
- Chaque condition d'une formule est vrai et fausse au moins une fois.
- Chaque condition d'une formule affecte indépendement le résultat.