Méthodes de Conception de Programmes

Informations Pratiques

0.1 Objectifs du cours

1. Spécifier
2. Concevoir
3. Démontrer

0.2 Evaluations

- Devoirs (1pt): Un exercice après chauque TP, à rendre sur Moodle.
- Projet(4pts): Spécifier, concevoir et prouver un programme en 3 phases à partir de la S6 avec évaluations croisées en groupes de 3 (libres).
- Examen (15pts): écrit et exercices(en septembre = 100% de note finale).

0.3 Dafny

Dafny est un langage de programmation avec spécifications et vérifiacation automatique.

I. Introduction

1.1 Programmes et bugs

1.1.1) Comment éviter les bugs? Le débogage?

Si l'on cherche à écrire un programme correct, le débogage n'est **pas** une solution **efficace**. En effet, il force le programmeur à fonctionner par **essais erreur**. On peut essayer d'écrire des tests mais l'absence d'échecs dans la résolution de ceux-ci ne démontres pas que le programme est correct. **les tests** ne sont **pas** non plus **efficaces** pour démontrer que le programme est faux. (Chaque test augmente la confiance mais ne garantie rien)

"L'absence de preuves n'est pas une preuve de l'absence" "Program testing can used to show the prescence of bugs, but never to show their abscence"-Dijkstra

On Définira donc une **Science de la programmation** par l'usage de la méthode scientifique pour construire des programmes corrects. On débute ar **spécifier** *les spécifications des programmes*, on **vérifie** *ensuite les programmes par rapport à leurs spécifications* et on **construira** *enfin le programme sur la base de ces spécifications*

1.2 Problème, solution et preuve

1.2.1) Conception de programme

Théorie du problème: Quelles structures, opérations, attributs interviennent? Quelles sont les propriétés utiles? **Solution**: *"Comment résoudre le problème?"* **Preuve**: *"La solution est-elle correcte?"* **Représentation**: *"Comment réaliser la solution sous forme de programme?"*

Le problème réside dans la définition de correct. Il existe plusieurs définition qui peuvent convenir au caractère correct d'un programme. On définira donc un **programme correct** comme suit : *Un programme correct par rapport aux spécifications du problème*

1.2.2) Les contracts

Il existe 4 formes de contracts qui jugent la validité d'un programme.

La version la plus simple est appellée **Correction**, il dit que *'SI pré ALORS post'*. Cette vision du contract est un peu trop simpliste pour traiter tous les cas et fait abstraction de l'arrêt du programme.

On obtiens donc les deux nouveaux contracts **Terminaison** : *'SI pré ALORS fin'* et **Correction Partielle** : *'SI pré ET fin ALORS post'*. Pour prouver l'exactitude d'un programme, il suffit de démontrer que les deux contracts sont remplis.

On peut aussi utiliser le contract dit de **Correction Totale** (Correction Partielle et Terminaison) : *'SI pré ALORS fin ET post'*.

1.2.3) La preuve de l'invariant

Un **invariant** est *une formule mathématique qui reste valide durant toute l'itération.*
Les 3 preuves pour obtenir un invariant de boucle correct: * Si les préconditions sont vraies initialement, alors l'invariant est vrai à l'entrée dans la boucle. * Si l'invariant est vrai avant une itération de la boucle, alors l'invariant est vrai après l'itération. * Si l'invariant est vrai à la sortie de boucle, alors les post-conditions sont vraies finalement.

1.2.4) Preuve et réccurence

La preuve sur l'itération est une récurrence : un **cas de base** (L'invariant est vrai après 0 itérations), un **cas inductif** (si l'invariant est vrai après N itérations, alors l'invariant est vrai après N+1 itérations), **Conclusion**(l'invariant est toujours vrai)

1.2.5) Un problème de précision

Les *spécifications*(Quoi?), le *programme*(Comment?) et la *preuve*(Pourquoi?) doivent être exprimés de manière **précise** et **non-ambigüe**

1.3 Spécification de programmes

1.3.1) Les assertions

Une assertion : *expression mathématique qui doit être évaluée à vraie* Une assertion : [p] (= en ce point p doit être vrai).

Les assertions sont souvent utilisées dans de la *programmation par contraintes* pour vérifier la bonne éxécution et la véracité d'un programme.(ex:[0<N<100]).

1.3.2) Triplet de Hoare

[P] : pré
S : prog
[S] : post
Le triplet est valide ssi: S réalise Q si P (correction totale)

1.3.3) Les variables auxiliaires

Les variables auxiliaires sont des variables utilisées dans les pré et/ou post qui ne sont nides variables du programme et dont la valeur ne change pas au cours de l'exécution.

II. Programmes Séquentiels

2.1 Concepts clés

Tableau : [P]S[R]S'[Q]=[P]S[R],[R]S'[Q]
Règle de conséquence : Si P->Q alors [P][Q]

|a| = taille du tableau a

2.2 La preuve

Chaque type de preuve possède son propre pattern de preuve.

S3

l'induction expérimentale n'est pas forcément valide.

TP S3

L'induction est une méthode de preuve reccursive. on prouve que K+1 est vrai si K est vrai.

L'induction simple : prouve K et k+1 **L'induction Complète** : utilise nimporte quel nombre. **L'induction structurale** : (ex preuve d'un arbre) L'application de l'induction simple aux strctures.

Cours S4

TP S4