Leçon Axel Kugelman :

Preuves et corection de programme

Plan

Motivation : algo de + en + compliqués, bugs désastreux (Ariane 5), difficile à maintenir.

Preuve et corection de programme : donner des garanties.

Ouverture : Techniques vraiment utiles (sel4) ou encore compilation formellement vérifié.

Q : 1 seul dupt, 2e ?
- ↳ ex de preuve de triplet de Hoare
- → preuve du thm 5
- → complétude logique de Hoare.
- → preuve d'un prog. : exemple en général.

Dupt : (Le jury peut demander des précisions / poser des questions sur les dupts au moment du choix du dupt).

Implémentation d'un interpréteur abstrait.

Langage : vd : entier, ~~string~~ variable, + ou −.

instr : Assign, input, assert, ifElse, while.

prog : list instr.

Domaine abstrait : intervalles
↳ chaque val. va avoir l'intervalle des val. qu'elle peut prendre.

(Interprétation des instructions)

Mémoire : ~~environnement~~ : dictionnaire : variable → intervalle

~~valeur~~ : ~~int~~ valeur

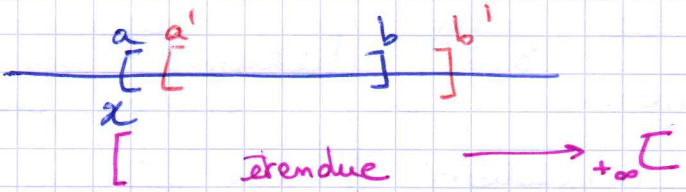
Assign, Input → indiquer ds l'env. la nlle valeur (pr assign on calcule la valeur).

assert ($x \leq n$ ou $x > n$) on vérifie $[n+1; +\infty[\cap (\text{intervalle } x) \neq \emptyset$?
↳ purement indicatif, message mais pas bloquant.

if ($x \leq n$) P_1 else P_2 : on ~~modifie~~ fait un nouvel env. selon si on passe dans P_1 ou P_2 pour avoir une ~~valeur~~ intervalle pour x à jour.

- A la fin du if → union des env obtenus [min(e_1, e_2); max(e_1, e_2)] pr chaque var.
  + $]-\infty ; +\infty[$ si pas défini ds un des 2

- While ($x \leq n$) ; Pb pour terminaison.
  ↳ on cherche un pt fixe
  Fusion d'env. avec élargissement d'intervalle:
  $x \in [a, b]$ au début de la boucle
  ↳ $x \in [a', b']$ $a' \geq a$ $b' \geq b$



- Test / démo du prog.

Questions : Dvpt

- Montrer des propr. sur un prog. Fonctionne si l'impl. est correcte. Quelles garanties ?
  → Pr chaque instr. surapproxim. les valeurs possibles.
- Quelle est la sémantique du programme ?
  → Pr savoir si correct, doit savoir ce que c'est sensé faire donc avoir sémantique.
  Etat mémoire après exec. d'une instruction
  par ex: * input : variable x va être affectée à une valeur ds un intervalle.
  * Plus.
- Formuler plus préciser sémantique du prog ? + lien entre sémantique et sémantique abstraite présentée ?
  → sémantique = état mémoire à la fin exec des instructions.
  sémantique abstraite : Pr tte entrée, chaque var doit avoir une valeur ds d'intervalle de la sémantique abstraite.

Leçon - Sémantique While : pt fixe. Qu'est-ce qui se passe avec l'abstraction ?

→ Si on s'intéresse à ttes les exécutions possibles, while donne lfp.

Elargissement : force apparition d'un point fixe (par grossissement des env.) pas forcément le plus petit.

- Code : Analyseur syntaxique ?

→ Non, joli affichage pr l'exemple.

Q. plus large :

- Lien entre dupt et typage ?

→ Changer domaine abstrait : domaine de type et en plus vérifier type correct.

Version simple : typage explicite

- Correction interprétation abstraite sur les types ?

→ Les var. définies sont du type correspondant ds l'interpréteur abstrait.

- Plan : ≠ entre techniques fonctionnelles et impératif ? est-ce qu'on peut changer de pt de vue ? Utiliser une technique ds l'autre cas ?

→ Les preuves s'adaptent mieux à la structure du programme comme ça. On peut utiliser variant pr récurs°
ex : factorielle ~~invariant~~

- Et du coup, inversement ? Terminaison bien fondé pr impératif ?

→ Transformer invariant / variant en ppté d'ordre bien fondé. Transformer boucle while en boucle récursive.

- Un variant est une qté (notes) ? Pas plutôt une ppté ?

→ Version moins générale que ppté. On utilise la ppté variant = qté ou v. $\leq$ ... $v >$ ...

- Triplet de Hoare : Correction vs correction partielle ?

→ Règle while ~~$\exists E = true \wedge Inv = z \} C \{ Inv$~~

$\{E = true \wedge I \wedge V = z\}\ C\ \{I \wedge V < z\}$ $\quad I \Rightarrow V \geq 0$

↳ correction totale (avec terminaison).

$\{E = true \wedge I\}\ C\ \{I\}$

↳ correction partielle (terminaison non garantie).

- Interêt correction partielle ? A l'air moins fort...

→ on peut avoir envie qu'un prog. ne termine pas. (? pas sûre)
+ moins de var. à introduire donc preuves plus simples.

- Variant pr un prog. qui termine ?

→ Nb de boucles restants mais nécessite qu'on le connaissance.

Pas de réponse générale : pb de l'arrêt.

- Prog. récursifs : Thm 2 ?? "les autres étapes de calcul terminent" ?

→ Si boucle infinie avec appel boucle, ppté ordre bien fondé n'aide pas.

ex:
```
f n =
  ⋮
  f(n-1)      } termine et pas d'appels à f.
  ⋮
```

- Qu'est-ce qu'on teste sur n' : nouvel arg. de f ?

→ Vérifier si dans l'ensemble bien fondé.

ex: pgcd a b $\qquad (E, \leqslant) \to (\mathbb{N}^2, \leqslant)$ ordre lexicographique à droite.
⋮
pgcd b a%b

- Modèle d'un prog ?

(trou).

Leçon

Remarques :

→ Pb de terminologies : pas d'accord avec les defs.

Q : - Dvpt : if then else : si une branche impossible ?

→ Si env vide renvoie None et ne calcule rien.

- Possible aucune des deux branches ?

→ Non : au moins un des 2 côtés, non vide. de la coupe

- Même test pr boucle while ?

→ Oui, pr vérifier si on rentre ds la boucle ou pas du tout.

- ~~Pb~~ On a le droit à des annexes au plan : par ex : ici les règles de sémantique par ex. Limite de pages ?
- Organisation du plan : I - vague, puis de + en + précis et III + intéressant : à rééquilibrer. Trop de place pr le début (et trop de temps) !
- Bon pt : • exemples.
  • Motivation
- ⚠ Traitement différencié des prog. itératifs / inductifs (d'où question) ⚠ vérifier variant ⩾ 0 !
- Correction partielle : Trouver un variant peut être compliqué, alors la version sans le variant peut être (entier !) plus pratique.
- Manque de place : moins détaillé les ex (par ex celui sur factorielle).
- ⚠ Un peu dangereux car à la limite du programme il peut y avoir des questions poussées assez agressives.
- 1ère partie basée sur quoi ? → Cours méthode de prog. R. Demangeon principalement, éléments math. aussi.
  (Defs. un peu bancales.
- Mettre un ex. d'induct un peu plus intéressant que fact. (où on fait juste −1).
- Ajouter les refs d'où sont prises les infos.

Peut-être trop long pr un vrai dvpt (long à coder)
→ domaines abstraits plus faciles (positif / négatif ou pair/impair) par ex.

- Le jury peut poser des questions sur tout ce qui est en lien ac ce dont on parle + ce qui est au programme.
→ teste les limites !
- On peut (~~ou plutôt on ne peut pas le contraire et on traite le choix ?~~) ne pas couvrir tout le programme sur le thème. On peut mentionner l'existence des notions dans la pés. du plan (par ex ici les graphes de flots de contrôle).