NOM : AUFORT Prénom : William Jury :

~~Algèbre~~ ← Entourez l'épreuve → ~~Analyse~~ Informatique

Sujet choisi : 927 : Exemples de preuves d'algorithmes : correction, terminaison.

Autre sujet :

I] Terminaison

__Déf 1__ On dit qu'un programme __termine__ sur x son argument si le calcul de P(x) nécessite un nombre fini d'étapes.

a) Cas des algorithmes itératifs : variants de boucle

__Remarque 2__ Seules les boucles WHILE posent un problème pour la terminaison.

__Déf 3 :__ Un ~~ensemble~~ ensemble $(E, \leq)$ est dit __bien fondé__ s'il n'existe pas de suite infinie d'élément de E strictement décroissante.

__Déf 4__ Un __variant__ de boucle est une fonction de l'ensemble des états du programme dans $(E, \leq)$ bien fondé, strictement décroissante à chaque passage de boucle.

__Prop 5 :__ Si une boucle admet un variant, alors elle termine.

__Ex 6 :__ Pour une boucle for $i = a$ to b, $i \mapsto b - i$ est un variant.

__Ex 7 :__ EUCLIDE(x,y) =

```
a ← x
b ← y
tant que b ≠ 0
  tmp ← a ; a ← b ; b ← tmp mod b
retourner a
```

Alors $(a,b) \mapsto b$ est un variant de boucle, donc EUCLIDE termine.

__Ex 8 :__ L'algorithme de Moore utilisé pour calculer un automate minimal termine, avec comme variant $|Q|$ − nombre d'éléments de la partition en cours.

b) Cas des algorithmes récursifs

__Prop 9 :__ Soit f une fonction récursive, A l'ensemble de ses arguments, $\varphi : A \to (E, \leq)$ bien fondée. Soit $B \subset A$ l'ensemble des cas de bases (ie $\varphi(x)$ minimal dans $\varphi(A)$). Si $\forall b \in B$, $f(b)$ termine et si dans $f(x)$ n'apparaissent que des appels à $f(y)$ avec $\varphi(y) < \varphi(x)$ en nombre fini, alors $f(x)$ termine $\forall x \in A$.

__Remarque 10 :__ φ est en quelque sorte un variant.

__Ex 11 :__ La fonction d'Ackermann termine.

$$Ack(n,p) = \begin{cases} p+1 & \text{si } n = 0 \\ ack(n-1, 1) & \text{si } p = 0 \\ ack(n-1, ack(n, p-1)) & \text{sinon} \end{cases} \quad \text{via } (\mathbb{N}^2, \leq_{lex})$$

__Ex 12 :__ Le tri rapide termine. (φ = longueur de liste)

__Contre-exemple 13 :__ $f(m,n) = \begin{cases} 1 & \text{si } m = 0 \\ f(m-1, f(m,n)) & \text{sinon} \end{cases}$

c) Des cas de non-terminaison

__Remarque 14__ On a exposé ici des cas où on peut prouver la terminaison. Le problème général est lui indécidable (comme le problème de l'arrêt des Machines de Turing)

__Ex 15__ SYRACUSE(n) =

```
si n = 1 retourner 1
si n est impair, SYRACUSE(3n+1)
sinon SYRACUSE(n/2)
```


__Références__ __Surtout :__ Winskel, __The Formal Semantics of Programming Languages__ (pour le III)
Albert, __Cours et exercices d'informatique__ (pour I et II)
Cormen (pour des exemples)
David Nour Raffali : __Introduction à la logique__ (dév 1)
Pour l'intro / la culture : Apt/Olderog, Backhouse, Gries.

On ne sait pas si SYRACUSE termine sur toute entrée $n \geq 1$. En particulier, le théorème 9 ne s'applique pas $(3n+1 > n)$.

Ex 16 : "while true do skip" ne termine pas.

II] Correction

Déf 17 On dit qu'un programme est correct (partiellement) si sur l'entrée x, lorsqu'il termine il renvoit une valeur conforme à sa spécification. S'en plus il termine sur toute entrée, on parle de correction totale.

Remarque 18 Sauf mention contraire, dans toute la leçon, on s'intéresse à la correction partielle

a) Cas des algorithmes itératifs : invariants

Remarque 19 Comme en Ia, seul le comportement des boucles WHILE est difficile à analyser. Ici, on s'intéresse aux propriétés préservées par les boucles, qui nous aident à comprendre l'algo.

Déf 20 Un invariant de boucle est une propriété préservée après passage de la boucle.

Ex 21 : Dans un tri par insertion :

```
TRI_INSERTION(A,n):
  pour j=2 à n faire                ← L'invariant de la boucle pour est « Au début de chaque itération, les j-1 premiers éléments sont triés »
    tmp ← A[j]
    i ← j-1
    tant que i>0 et A[i]>tmp faire
      A[i+1] ← A[i];
      i ← i-1;
    A[i+1] ← tmp.
```

Les éléments initiaux $A[1,\dots,j-1]$ mais triés.

A la fin de la boucle, on obtient donc A trié.

Ex 22 : L'algorithme élémentaire d'unification est correct (et termine) :

```
UNIFIE(E'): E ← E', σ ← Id.
  Tant que (E ≠ ∅)
    si E = E' ⊎ {f(u1,...,up) = g(v1,...,vq)}
      si f=g
        E ← E' ∪ {ui = vi}_{i=1..p=q}
      sinon, echec par clash
    si E = E' ⊎ {x=x} : E ← E'.
    si E = E' ⊎ {x=u} ou E' ⊎ {u=x}
      si x apparait dans u, echec par "occurcheck"
      sinon σ ← σ[u/x]
            E ← E'[u/x]                    (DEV 1)
  retourner σ
```

Ex 23 : L'algorithme de Moore (cf ex 8) est correct, l'invariant de boucle est « la relation d'équivalence associée à P est plus fine que R associée à $\{E, \bar{E}\}$ et moins fine que l'équivalence de Nérode. »

b) Cas des algorithmes récursifs

Remarque 24 : Ces algorithmes traitent en général de structures inductives, qui nous permettent de faire des preuves par induction :

Proposition 25 : Si $(E, \leq)$ bien fondé, et P prédicat sur E, Si P est vraie pour les éléments minimaux de E et si $\forall x \in E, (\forall y < x, p(y)) \Rightarrow p(x)$, alors $\forall x \in E\ p(x)$

Théorème 26 : Avec les mêmes notations qu'à la prop 9, si P est un prédicat sur les valeurs calculées par f tel que $\forall b \in B, P(b)$ vrai et si dans l'appel de $f(x)$ n'apparaissent que des appels à $f(y)$ avec $\varphi(y) < \varphi(x)$ et $(\forall y\ P(y)) \Rightarrow P(x)$, alors $P(x)$ est vraie $\forall x$.

Remarque 27 La ressemblance avec la proposition 9 fait qu'on prouve souvent la terminaison et la correction ensemble, c'est souvent le prédicat utilisé.

Ex 28 Le tri rapide est correct en utilisant le prédicat $P(l)$: « tri-rapide l termine et vaut la liste triée des éléments de l »

c) Programmer c'est prouver

Remarque 29 L'approche faite jusqu'à présent pourrait laisser croire que la preuve suit toujours l'algo. Or les algorithmes utilisent souvent des propriétés sur les objets choisis (qui se retrouvent sous forme d'invariants ou de récursion) nécessitants des preuves. La preuve et la conception des algorithmes se font donc en même temps en pratique.

Ex 30 : Dans le paradigme de programmation dynamique, l'algorithme est souvent issu d'une relation de récurrence qu'il faut prouver.

Ex 31 : L'algorithme de Floyd-Warshall pour le calcul de tous les plus courts chemins : il suffit de vérifier que $d_{ij}^{(k)} = \begin{cases} w_{ij} \text{ si } k=0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) \text{ sinon} \end{cases}$ où $d_{ij}^{(k)}$ est la longueur du plus court chemin de i à j passant par les sommets intermédiaires dans $\{1,\dots,k\}$.

III] Formalisation : Logique de Hoare

Remarque 32 On souhaite formaliser la correction partielle dans une optique d'automatisation des preuves. Il faut pour cela se fixer un ensemble de programmes et formaliser la correction partielle avec

a) Le langage IMP

a) Syntaxe

Déf 33 On se donne Loc un ensemble de registres $\text{Loc} = \{x, y, z, \dots\}$.

Déf 34 (Syntaxe): On définit la syntaxe des éléments de Aexp, Bexp, Com, respectivement expressions arithmétiques, booléens, et commandes de IMP:

Aexp: $a ::= n \in \mathbb{N} \mid X \in Loc \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$

Bexp: $b ::= true \mid false \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1$

Com: $c ::= skip \mid X := a \mid c_0 ; c_1 \mid if\ b\ then\ c_1\ else\ c_2 \mid while\ b\ do\ c.$

② Sémantique

Déf 35: Un état du programme est une fonction σ de Loc dans $\mathbb{N}$.

Déf 36: On définit la relation d'évaluation des expressions arithmétiques $\langle a, \sigma \rangle \to n$ par:

$$\langle n, \sigma \rangle \to n \qquad \langle X, \sigma \rangle \to \sigma(X)$$

$$\frac{\langle a_0, \sigma \rangle \to n_0 \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1}$$

de même pour $-, \times$.

De même, on définit $\langle b, \sigma \rangle \to t$ pour l'évaluation des expressions booléennes.

Déf 37: On définit la relation d'exécution d'un programme $\langle c, \sigma \rangle \to \sigma'$ par:

$$\langle \sigma, skip \rangle \to \sigma \qquad \frac{\langle a, \sigma \rangle \to n}{\langle X := a, \sigma \rangle \to \sigma[n/X]} \qquad \frac{\langle c_0, \sigma \rangle \to \sigma' \quad \langle c_1, \sigma' \rangle \to \sigma''}{\langle c_0 ; c_1, \sigma \rangle \to \sigma''}$$

$$\frac{\langle b, \sigma \rangle \to true \quad \langle c_0, \sigma \rangle \to \sigma'}{\langle if\ b\ then\ c_0\ else\ c_1, \sigma \rangle \to \sigma'} \quad \left(\text{de même pour } \langle b, \sigma \rangle \to false\right)$$

$$\frac{\langle b, \sigma \rangle \to false}{\langle while\ b\ do\ c, \sigma \rangle \to \sigma} \qquad \frac{\langle b, \sigma \rangle \to true, \ \langle c, \sigma \rangle \to \sigma' \quad \langle while\ b\ do\ c, \sigma' \rangle \to \sigma''}{\langle while\ b\ do\ c, \sigma \rangle \to \sigma''}$$

Rem 38: $\sigma[m/x]$ est l'état $Loc \to \mathbb{N}$ tel que

$$\sigma[m/x](y) = \begin{cases} \sigma(y) & \text{si } y \neq x \\ m & \text{si } y = x. \end{cases}$$

Remarque 39: $(\sigma, c) \to \sigma'$ signifie que l'exécution de c à partir de l'état σ donne l'état σ'.

b) Triplet de Hoare

Remarque 40: Pour pouvoir prouver des choses sur les programmes, on va étendre Bexp (et donc Aexp) avec les quantificateurs $\forall, \exists$ sur des variables entières. On obtient alors l'ensemble des assertions (booléennes).

Déf 41: Un triplet de Hoare est un triplet de la forme $\{A\}\ c\ \{B\}$, où A, B sont des assertions, et c un programme IMP. (A: pré-condition, B: post-condition)

① Sémantique

Déf 42: Une interprétation est une fonction I de l'ensemble des variables entières dans $\mathbb{N}$.

Déf 43: On définit naturellement l'évaluation d'une expression arithmétique dans (I, σ), et le jugement $\sigma \models^I A$ où A est une assertion, décrivant si σ satisfait A dans l'interprétation I. Ces définitions se font par induction structurelle.

Déf 44: On définit le jugement $\sigma \models^I \{A\} c \{B\}$ par

$$\sigma \models^I \{A\} c \{B\} \iff \forall \sigma' \left(\sigma \models^I A \ \text{et} \ \langle \sigma, c \rangle \to \sigma' \implies \sigma' \models^I B \right)$$

Déf 45: On dit qu'un triplet de Hoare $\{A\} c \{B\}$ est valide si $\forall I, \forall \sigma, \ \sigma \models^I \{A\} c \{B\}$, on note alors $\models \{A\} c \{B\}$.

② Règles de preuves

Déf 46: Règles de Hoare:

1. $$\frac{}{\{A\}\ skip\ \{A\}}$$

2. $$\frac{}{\{B[a/x]\}\ x := a\ \{B\}}$$

3. $$\frac{\{A\}\ c_1\ \{C\} \quad \{C\}\ c_2\ \{B\}}{\{A\}\ c_1 ; c_2\ \{B\}}$$

4. $$\frac{\{A \wedge b\}\ c_1\ \{B\} \quad \{A \wedge \neg b\}\ c_2\ \{B\}}{\{A\}\ if\ b\ then\ c_1\ else\ c_2\ \{B\}}$$

5. $$\frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ while\ b\ do\ c\ \{A \wedge \neg b\}}$$

6. $$\frac{\models (P_1 \Rightarrow P_2) \quad \{P_2\}\ c\ \{Q_2\} \quad \models Q_2 \Rightarrow Q_1}{\{P_1\}\ c\ \{Q_1\}}$$

Déf 47: On dit que $\{A\}\ c\ \{B\}$ est prouvable s'il existe une dérivation dans le système de règles précédent. On note alors $\vdash \{A\}\ c\ \{B\}$. (induction sur $\vdash$)

Théorème 48 (Cohérence): Si $\vdash \{A\}\ c\ \{B\}$, alors $\models \{A\}\ c\ \{B\}$. (DEV2)

Remarque 49: Autrement dit, une dérivation avec ces règles fournit une preuve de la commande c! (DEV3)

Exemple 50: Preuve de la factorielle en logique de Hoare: $\{X = m, m \geq 0, Y = 1\}\ fact\ \{Y = m!\}$ où

$$fact \equiv while(X > 0)\ (Y := X \times Y ; X := X - 1)$$

c) Pour aller plus loin: Correction totale automatisation

Remarque 51: Dans la règle du while, A joue le rôle d'invariant. Pour pouvoir également prouver la correction (terminaison), il suffit de modifier la règle pour intégrer un variant:

$$\frac{\{A \wedge b\}\ c\ \{A\} \quad \{A \wedge b \wedge V = z\}\ c\ \{V < z\} \quad \models b \Rightarrow V \geq 0}{\{A\}\ while\ b\ do\ c\ \{A \wedge \neg b\}}$$

Remarque 52: La dernière règle (règle de conséquence) empêche d'obtenir une automatisation complète des preuves. Comment trouver P_2, Q_1, et surtout comment prouver $\models P_1 \Rightarrow P_2$ et $\models Q_1 \Rightarrow Q_2$? Une idée peut être d'annoter le programme en donnant, par exemple, les invariants (ex: logiciel WHY).

Remarque 53: La recherche automatique d'invariant est un problème de recherche actuel.