Introduction à l'algorithmique : Correction

Serge Haddad

LSV, ENS Paris-Saclay & CNRS & Inria

L3

Recherche de maximum

Max(T, n)

$max \leftarrow T[1]$;

$\{max = T[1]\}$

For i **from** 2 **to** n **do**

$\{max = max(T[k] \mid 1 \leq k < i)\}$

If $T[i] > max$ **then**

$\{T[i] > max(T[k] \mid 1 \leq k < i)\}$

$max \leftarrow T[i]$;

$\{max = T[i] \wedge max > max(T[k] \mid 1 \leq k < i)\}$

Else

$\{max = max(T[k] \mid 1 \leq k < i) \wedge T[i] \leq max\}$

skip ;

$\{max = max(T[k] \mid 1 \leq k < i) \wedge T[i] \leq max\}$

$\{max = max(T[k] \mid 1 \leq k \leq i)\}$

$\{max = max(T[k] \mid 1 \leq k \leq n)\}$

return(max)

Recherche de maximum et minimum

Application de l'algorithme précédent : $2(n-1)$ comparaisons mais ...

MinMax(T, n) *(n impair)*

$max \leftarrow T[1]$; $min \leftarrow T[1]$;

For i **from** 1 **to** $\frac{n-1}{2}$ **do**

If $T[2i] > T[2i+1]$ **then**

If $T[2i] > max$ **then** $max \leftarrow T[2i]$;

If $T[2i+1] < min$ **then** $min \leftarrow T[2i+1]$;

Else

If $T[2i+1] > max$ **then** $max \leftarrow T[2i+1]$;

If $T[2i] < min$ **then** $min \leftarrow T[2i]$;

return(min, max)

Soit $\frac{3(n-1)}{2}$ comparaisons !

Algorithmique

Objectifs

- Résoudre des problèmes
- correctement
- et efficacement

Moyens

- Spécification formelle *(voir aussi le cours de sémantique)*
- Systèmes formels de preuve *(voir aussi le cours de logique)*
- Analyse asymptotique de complexité *(voir aussi le cours de complexité)*
- Structures de données appropriées
- Paradigmes algorithmiques

Plan

Correction d'un algorithme

Un algorithme est *correct* si :

- il se termine ;
- il implémente sa spécification ;

Observation. Un programme se termine toujours si :

- il ne comporte que des boucles For ;
- il n'inclut pas des fonctions (mutuellement) récursives ;
- il n'effectue pas d'opérations illégales (division par zéro, déréférencement d'un pointeur nul, etc.)

Pourquoi ne pas imposer des contraintes syntaxiques sur les programmes ?

La fonction d'Ackermann

Ackermann2(m, n)

  If $m = 0$ **then return** $n + 1$;

  If $n = 0$ **then return** Ackermann2$(m - 1, 1)$;

  return Ackermann2$(m - 1,$ Ackermann2$(m, n - 1))$;

Ackermann(n)

  return Ackermann2(n, n) ;

La fonction d'Ackermann ne peut être implémentée par un algorithme à base de For, de If et d'opérations arithmétiques $(+, -, \times)$.

Idée de preuve.

Pour tout algorithme Calcul de ce type,

il existe n tel que Ackermann$(n) >$ Calcul(n).

Comment établir la correction ?

La correction peut s'établir en deux étapes.

- Correction partielle :
  « Si l'algorithme se termine alors il satisfait sa spécification. »
- Terminaison :
  « L'algorithme se termine. »

Un double intérêt.

- La correction partielle admet des preuves plus naturelles
  (que la correction totale).
- La correction partielle peut simplifier la preuve de terminaison.

Comment établir la terminaison ?

Pas de méthode automatique !

Supposons qu'il existe un programme `Termine(Code,Donnée)` qui teste si un programme se termine sur une donnée.

Soit le programme

```
Fou(Code)
        Tant que Termine(Code,Code) faire fait;
```

`Fou(Fou)` termine-t-il ?

Relation bien fondée

Une relation binaire $\prec$ sur un ensemble E est *bien fondée*
s'il n'existe pas de suite infinie $\{e_n\}_{n\in\mathbb{N}}$ telle que : $\forall n \in \mathbb{N}\ e_{n+1} \prec e_n$

$(\mathbb{N}, <)$ est un ensemble bien fondé.

Quelques propriétés.

- Si $\prec$ est bien fondée alors $\prec$ est irréflexive ;
- Si $\prec$ est bien fondée alors $\prec^+$ est bien fondée ;
- Si $\prec$ est bien fondée alors $\prec^*$ est une relation d'ordre partiel.

Construire des relations bien fondées : l'ordre lexicographique.

- Soit $(E_1, \prec_1)$ et $(E_2, \prec_2)$ des ensembles bien fondés ;
- Alors $(E_1 \times E_2, \prec)$ est un ensemble bien fondé où :

$$(e'_1, e'_2) \prec (e_1, e_2) \Leftrightarrow e'_1 \prec e_1 \vee (e'_1 = e_1 \wedge e'_2 \prec e_2)$$

$(\mathbb{N}^k, <)$ est un ensemble bien fondé.

Terminaison des boucles

Spécification d'un ensemble bien fondé $(E, \prec)$ et d'une fonction $\varphi(\mathbf{x})$ des variables telle que :

- $\varphi(\mathbf{x})$ appartient à E à chaque début de boucle ;
- $\varphi(\mathbf{x})$ décroît à chaque tour de boucle.

While $i > 0$ **and** $j > i$ **do**

$i \leftarrow i * j$;

$j \leftarrow 2 * j$;

$(E, \prec) = (\mathbb{N}^2, <)$ et $\varphi(i, j) = (\max(3 - i, 0), j - i)$.

- i croît à chaque tour de boucle ;
- Lorsque $i \geq 3$, $j - i$ décroît à chaque tour de boucle.

Terminaison des appels récursifs

Spécification d'un ensemble bien fondé $(E, \prec)$ et d'une fonction $\varphi(\mathbf{x})$ des paramètres de la fonction récursive f telle que :

- $\varphi(\mathbf{x})$ appartient à E à chaque appel de fonction ;
- $\varphi(\mathbf{x})$ décroît entre deux appels récursifs.

Ackermann2(m, n)

If $m = 0$ **then return** $n + 1$;

If $n = 0$ **then return** Ackermann2$(m - 1, 1)$;

return Ackermann2$(m - 1,$ Ackermann2$(m, n - 1))$;

$(E, \prec) = (\mathbb{N}^2, <)$ et φ est l'identité.

- $(m - 1, 1) \prec (m, 0)$;
- $(m, n - 1) \prec (m, n)$;
- $(m - 1,$ Ackermann2$(m, n - 1)) \prec (m, n)$.

Plan

Correction partielle : les triplets de Hoare

Un triplet de Hoare $\{\varphi\}\mathrm{pg}\{\psi\}$ est défini par :

- pg un programme ;
- φ une formule logique incluant des variables de pg et $\mathbf{x}$, des variables logiques, appelée *précondition* ;
- ψ une formule logique incluant des variables de pg et des variables logiques, appelée *postcondition*.

Le système des triplets de Hoare sert à la fois à la spécification et à la vérification.

Spécification de la factorielle. $\{in = x \wedge x \geq 0\}\mathrm{Fact}(in, out)\{out = x!\}$

Que pensez-vous de $\{in \geq 0\}\mathrm{Fact}(in, out)\{out = in!\}$?

Un triplet de Hoare est *valide* si pour toute interprétation $\mathbf{v}$ des variables logiques et tout état initial du programme $\mathbf{s}$:

- Si $\mathbf{s}$ satisfait $\varphi[\mathbf{x}\backslash\mathbf{v}]$;
- Si pg se termine avec l'entrée $\mathbf{s}$ et conduit à l'état $\mathbf{s} \bullet \mathrm{pg}$;
- Alors $\mathbf{s} \bullet \mathrm{pg}$ satisfait $\psi[\mathbf{x}\backslash\mathbf{v}]$.

La validité d'un triplet de Hoare est noté $\models \{\varphi\}\mathrm{pg}\{\psi\}$.

Etablir un triplet : affectation

Un *système de preuve* est un ensemble de *règles* de la forme $\dfrac{\texttt{hypothèses}}{\texttt{conclusions}}$.

Une *preuve* est une suite finie d'instances de règles telle que :

- toute hypothèse d'une instance de règle est la conclusion d'une instance d'une règle précédente ;
- d'où la nécessité de règles sans hypothèses appelées *axiomes*.
- $\vdash$ `assertion` signifie qu'on a établi une preuve de `assertion`.

Affectation. La précondition d'une affectation est la postcondition où l'expression affectée à la variable se substitue à elle.

$$\boxed{\frac{}{\{\varphi[X\backslash E]\}X \leftarrow E\{\varphi\}}}$$

Exemple. $\vdash \{Y^2 + X \leq 17\}X \leftarrow Y^2 + X\{X \leq 17\}$

Etablir un triplet : concaténation

La règle de la concaténation s'appuie sur la transitivité de l'implication.

$$\frac{\{\varphi\}\ \mathtt{pg}\ \{\theta\} \qquad \{\theta\}\ \mathtt{pg}'\{\psi\}}{\{\varphi\}\ \mathtt{pg}\ ;\ \mathtt{pg}'\{\psi\}}$$

Exemple.

$$\begin{array}{l} \{x+y=a+b \wedge y=b\} \Leftrightarrow \{x=a \wedge y=b\} \\ x \leftarrow x+y\,; \\ \{x=a+b \wedge x-y=a\} \Leftrightarrow \{x=a+b \wedge y=b\} \\ y \leftarrow x-y\,; \\ \{x-y=b \wedge y=a\} \Leftrightarrow \{x=a+b \wedge y=a\} \\ x \leftarrow x-y\,; \\ \{x=b \wedge y=a\} \end{array}$$

Etablir un triplet : conditionnelle

La règle de l'instruction conditionnelle envisage les deux résultats possibles du test.

$$\frac{\{\varphi \wedge c\}\ \mathrm{pg}\ \{\psi\} \qquad \{\varphi \wedge \neg c\}\ \mathrm{pg}'\{\psi\}}{\{\varphi\}\ \textbf{if}\ c\ \textbf{then}\ \mathrm{pg}\ \textbf{else}\ \mathrm{pg}'\{\psi\}}$$

Exemple.

If $x\%2 = 1$ **then**

$\{x - 1\%2 = 0\} \Leftrightarrow \{x\%2 = 1\}$

$x \leftarrow x - 1\,;$

$\{x\%2 = 0\}$

Else

$\{x - 2\%2 = 0\} \Leftrightarrow \{\neg x\%2 = 1\}$

$x \leftarrow x - 2\,;$

$\{x\%2 = 0\}$

Etablir un triplet : itération

La règle de l'itération nécessite l'introduction d'un invariant.

$$\frac{\{\varphi \wedge c\}\ \text{pg}\ \{\varphi\}}{\{\varphi\}\ \textbf{while}\ c\ \textbf{do}\ \text{pg}\ \{\varphi \wedge \neg c\}}$$

$\{n \geq 0\}$

$a \leftarrow 0\,;\ s \leftarrow 1\,;\ t \leftarrow 1\,;$

$\{s = 1 \wedge a = 0 \wedge t = 1 \wedge 0 \leq n\} \Rightarrow \{s = (a+1)^2 \wedge a^2 \leq n \wedge t = 2a+1\}$

While $s \leq n$ **do**

$\{s = (a+1)^2 \wedge a^2 \leq n \wedge t = 2a+1 \wedge s \leq n\} \Rightarrow$

$\{s = (a+1)^2 \wedge (a+1)^2 \leq n \wedge t = 2a+1\}$

$a \leftarrow a+1\,;$ $\{s = a^2 \wedge a^2 \leq n \wedge t = 2a-1\}$

$s \leftarrow s+t+2\,;$ $\{s = (a+1)^2 \wedge a^2 \leq n \wedge t = 2a-1\}$

$t \leftarrow t+2\,;$ $\{s = (a+1)^2 \wedge a^2 \leq n \wedge t = 2a+1\}$

$\{s = (a+1)^2 \wedge a^2 \leq n \wedge t = 2a+1 \wedge s > n\} \Rightarrow \{a = \lfloor \sqrt{n} \rfloor\}$

Etablir un triplet : règles annexes

- La règle du **skip** pour les conditionnelles incomplètes.

$$\frac{}{\{\varphi\}\mathbf{skip}\{\varphi\}}$$

- La règle d'affaiblissement déjà utilisée implicitement.

$$\frac{\models (\varphi \Rightarrow \varphi') \qquad \{\varphi'\}\ \mathrm{pg}\ \{\psi'\} \qquad \models (\psi' \Rightarrow \psi)}{\{\varphi\}\ \mathrm{pg}\ \{\psi\}}$$

Correction et complétude

- Un système de preuve est *correct* si :

$$\vdash \varphi \text{ implique } \models \varphi$$

Généralement les systèmes de preuve sont corrects par construction.

- Un système de preuve est *complet* si :

$$\models \varphi \text{ implique } \vdash \varphi$$

Prouver la complétude d'un système est difficile.

Il existe des théories qui n'admettent pas de système de preuve « raisonnable ». *(cf Gödel)*

Le système de preuve de Hoare est correct et complet pour les programmes arithmétiques.

Etablir un triplet : recursivité

Pour établir un triplet $\{\varphi\}\mathtt{f}\{\psi\}$ pour une fonction `f` avec appels récursifs,

- on applique les règles du système du preuve avec une exception ;
- on suppose le triplet prouvé pour les appels récursifs.

$\{in = x \wedge x \geq 0\}$

Fact(in, out)

If $in = 0$ **then**

$\{in = 0 \wedge x = 0\}$

$out \leftarrow 1$;

$\{in = 0 \wedge x = 0 \wedge out = 1\}$

Else

$\{in - 1 = x - 1 \wedge x - 1 \geq 0\}$

Fact$(in - 1, out)$;

$\{in - 1 = x - 1 \wedge x - 1 \geq 0 \wedge out = (x - 1)!\}$

$out \leftarrow in * out$;

$\{in = x \wedge x - 1 \geq 0 \wedge out = x!\}$

$\{in = x \wedge x \geq 0 \wedge out = x!\}$

Synthèse d'une plus faible précondition

Une approche alternative pour établir $\{\varphi\}\mathrm{pg}\{\psi\}$:

- Considérer uniquement le programme pg et la postcondition ψ.
- Générer la *plus faible précondition* $wp(\psi, \mathrm{pg})$
  telle que le triplet $\{wp(\psi, \mathrm{pg})\}\mathrm{pg}\{\psi\}$ soit valide.
- Vérifier que $\models \varphi \Rightarrow wp(\psi, \mathrm{pg})$.

La plus faible précondition est calculable par induction.

- $wp(\psi, \texttt{skip}) \Leftrightarrow \psi$
- $wp(\psi, X \leftarrow E) \Leftrightarrow \psi[X \backslash E]$
- $wp(\psi, \mathrm{pg}; \mathrm{pg}') \Leftrightarrow wp(wp(\psi, \mathrm{pg}'), \mathrm{pg})$
- $wp(\psi, \textbf{if } c \textbf{ then } \mathrm{pg} \textbf{ else } \mathrm{pg}') \Leftrightarrow (wp(\psi, \mathrm{pg}) \wedge c) \vee (wp(\psi, \mathrm{pg}') \wedge \neg c)$

Le calcul de la plus faible précondition d'une boucle est très technique.