Preuves de programmes

Martin Strecker

- Motivation: Pourquoi?
- Motivation: Comment?
- Conventions et notions de base
- Règles d'inférence
- Correction partielle vs. correction totale

Preuves de programmes: Pourquoi? (1)

Exemple: Pour un nombre n, trouver le plus grand diviseur d de n (sauf $n \ldots$)

Première solution:

```
int n, d;

printf("Entrer n: ");
scanf("%d", &n);
d = n - 1;

while (n % d != 0) {
  d = d - 1;
}

printf("d est: %d\n", d);
```

Preuves de programmes: Pourquoi? (2)

La solution est-elle correcte? Quelques tests:

```
> div
Entrer n: 12345
d est: 4115
```

4115 est diviseur de 12345, mais le plus grand??

```
> div
Entrer n: 6
d est: 3
> div
Entrer n: 5
d est: 1
```

C'est correct. Mais et-ce qu'on a traité tous les cas importants??

Comparaison: Test vs. Preuves de programmes (1)

Test:

Avantages:

- Intuitives
- Souvent faciles à mettre en oeuvre

Désavantages: Difficile à savoir si

- on a traité tous les cas critiques
- le résultat fourni par le programme est correct:
  - Test d'un programme qui traite des matrices de taille $10^3 \times 10^3$??
  - Test d'un compilateur ??

Comparaison: Test vs. Preuves de programmes (2)

Preuves:

Avantages:

- Vérification exhaustive
- Demande une programmation structurée et disciplinée

Désavantages:

- difficile ou impossible à automatiser (indécidabilité des logiques "intéressantes")
- donc: requiert souvent une intervention manuelle
- donc: un travail exigeant ...

- Motivation: Pourquoi?
- Motivation: Comment?
- Conventions et notions de base
- Règles d'inférence
- Correction partielle vs. correction totale

Comment raisonner sur des programmes ? (1)

Exemple 1: Affectation simple

(1) $\{x = 4\}$
(2) `x = x - 1;`
(3) $\{x = 3\}$

Raisonnement possible:

1. $\{x = 4\}$, donc $\{x - 1 = 3\}$
2. Affectation: `x - 1` “devient” `x`
3. Après affectation: $\{x = 3\}$

Comment raisonner sur des programmes ? (2)

Exemple 2: Affectation moins simple

(1) $\{y = 5 * x + 15\}$
(2) `x = 2 * x + 6;`
(3) $\{2 * y = 5 * x\}$

Raisonnement possible:

1. $\{y = 5 * x + 15\}$, donc $\{2 * y = 10 * x + 30 = 5 * (2 * x + 6)\}$

2. Affectation: `2 * x + 6` "devient" `x`

3. Après affectation: $\{2 * y = 5 * x\}$

Problème: Calculation peu ciblée.

Comment raisonner sur des programmes ? (3)

Comment faire mieux? En lieu de raisonner en avant
. . . raisonner en arrière !

(1) $\{x = 4\}$
(2) `x = x - 1;`
(3) $\{x = 3\}$

1. A démontrer: $\{x = 3\}$ après affectation

2. Substituer: $\{(x = 3)[x \leftarrow x - 1]\}$

3. Calculer: $\{(x = 3)[x \leftarrow x - 1]\} \equiv \{x - 1 = 3\} \equiv \{x = 4\}$

Exercice: Faire l'Exemple 2 !!

Comment raisonner sur des programmes ? (4)

Est-ce qu'il y a un problème avec le raisonnement suivant?

(1) $\{x = 4\}$
(2) `x = x + 3;`
(3) $\{x > 5\}$

Non !

1. A démontrer: $\{x > 5\}$ après affectation
2. Substituer: $\{(x > 5)[x \leftarrow x + 3]\}$
3. Calculer: $\{(x > 5)[x \leftarrow x + 3]\} \equiv \{x + 3 > 5\} \equiv \{x > 2\}$
4. *Implication:* $(x = 4) \longrightarrow (x > 2)$

Donc, $\{x = 4\}$ est une bonne précondition ... *mais pas la plus faible !*

Comment raisonner sur des programmes ? – Schéma général

But: Démontrer la correction du programme `prog` par rapport a sa spécification:

$$\{P\}\ \texttt{prog}\ \{Q\}$$

Calculer la plus faible précondition *pfp*
(en anglais: weakest precondition, *wp*):

$$\mathit{pfp}(\texttt{prog}, Q)$$

Démontrer implication:

$$P \longrightarrow \mathit{pfp}(\texttt{prog}, Q)$$

Comment ça fonctionne pour des programmes plus complexes?
$\rightsquigarrow$ voir la suite

- Motivation: Pourquoi?
- Motivation: Comment?
- Conventions et notions de base
- Règles d'inférence
- Correction partielle vs. correction totale

Rappel: Logique des prédicats (1)

Syntaxe des formules:

- Constantes propositionnelles: $\bot$ (faux), $\top$ (vrai)
- Prédicats: $P(t_1, \ldots, t_n)$ (où $t_1, \ldots, t_n$ sont des termes)
- Négation: $\neg F$
- Conjonction ("et"): $F \wedge G$, disjonction ("ou"): $F \vee G$, implication: $F \longrightarrow G$
- Quantification universelle ("quelque soit"): $\forall x.P$
- Quantification existentielle ("il existe"): $\exists x.P$

Rappel: Logique des prédicats (2)

Dans la formule $\forall x.P$ (ou bien $\exists x.P$), la variable x est *liée* dans P.

Toute variable qui n'est pas liée est *libre*.

Exemple: Dans $\forall x.P(x,y) \wedge \exists z.Q(x,z)$, les variables x et z sont liées, y est libre

Toute variable liée peut être *renommée*.

Exemples: Soit $F \equiv \forall x.P(x,y) \wedge \exists z.Q(x,z)$

- Renommer x par v dans F donne ... ???
- Renommer x par y dans F n'est pas possible. Pourquoi???
- Renommer x par z dans F n'est pas possible. Pourquoi???

Rappel: Substitution

Expressions: Une substitution $e[x \leftarrow t]$ remplace toute occurrence de la variable x dans l'expression e par le terme t.

Exemples:

- $(x + 5 = x)[x \leftarrow x - 2]$ est $(x - 2) + 5 = x - 2$
- *Attention aux parenthèses!!* $(7 - v)[v \leftarrow y - 2]$ est $7 - (y - 2)$

Formules: $F[x \leftarrow t]$ . . . pareil. [*Notation feuilles TD:* F_t^x]
Attention: renommer des variables liées, si nécessaire !

Exemple: $(\forall x.P(x, y) \wedge \exists z.Q(x, z))[y \leftarrow x + 4]$

- Solution incorrecte: $(\forall x.P(x, x + 4) \wedge \exists z.Q(x, z))$
- Solution correcte ???

Langage de programmation (1)

Dans un premier temps, un considère un langage C réduit.

Commandes c (définition inductive):

- Affectation: `x = e`
- Séquence: vide ou `c1; c2`
- Sélection: `if (b) {c1} else {c2}`
- Boucle: `while (b) {c}`

Fonctions, procédures, entrées / sorties: plus tard

Langage de programmation (2)

Traitement de

- sélection sans `else`:
  Traduire en `if - then - else`:
  `if (b) {c1}` devient
  `if (b) {c1} else {}`

- boucle `for`:
  Traduire en initialisation; boucle `while`

- boucle `do .. while`:
  Traduire en séquence; boucle `while`

Traitement d'erreurs (1)

Nous considérons les cas d'erreurs suivants:

- Arithmétique: Division par zéro: `x / 0`; modulo zéro: `x % 0`
- Dépassement de bornes lors d'un accès à un tableau `a[i]`: $i < 0$ ou $i > n - 1$

Nous ne prenons pas en compte:

- Dépassement de bornes lors d'opérations arithmétiques (ex.: `a + b > INT_MAX`) supposition idéalisante: arithmétique sans limites ...

Traitement d'erreurs (2)

Evaluable: Le prédicat *évaluable*(e) fournit une condition

- sous laquelle l'expression e peut être évaluée
- sans produire d'erreur

Définition (incomplète):
évaluable$(v) = \top$ pour v variable
évaluable$(e_1 + e_2) =$ *évaluable*$(e_1) \wedge$ *évaluable*(e_2)
évaluable$(e_1/e_2) =$ *évaluable*$(e_1) \wedge$ *évaluable*$(e_2) \wedge e_2 \neq 0$
évaluable$(a[i]) =$ *évaluable*$(a) \wedge$ *évaluable*$(i) \wedge 0 \leq i < taille(a)$

Exemple: *évaluable*$(a[k+5]) = 0 \leq k+5 < taille(a)$

- Motivation: Pourquoi?
- Motivation: Comment?
- Conventions et notions de base
- Règles d'inférence
- Correction partielle vs. correction totale

Règles d'inférence: Plan

L'ensemble des commandes a été défini de manière *inductive*.

Nous allons définir une fonction *pfp*(c, Q) par *récursion* sur les commandes c

Donc: une règle par constructeur

pfp: Affectation

Règle:

$$\mathit{pfp}(\texttt{x = e}, Q) = Q\,[\,x \leftarrow\ e] \wedge \mathit{évaluable}(e)$$

Exemples:

- $\mathit{pfp}(\texttt{x = x - 1}, x > 0) \equiv$
  $x > 0\,[\,\texttt{x} \leftarrow\ \texttt{x - 1}] \wedge \mathit{évaluable}(x-1) \equiv$
  $x - 1 > 0 \wedge \top$

- Démontrer: $\{x > 0\}\ \texttt{x = x - 1}\ \{x > 0\}$

  1. Calculer $\mathit{pfp}(\texttt{x = x - 1}, x > 0) \equiv (x - 1 > 0)$
  2. Implication $(x > 0) \longrightarrow (x - 1 > 0)\ \equiv \bot$

pfp: Séquence (1)

Règles:

- Séquence non vide:
  $$\mathit{pfp}(\texttt{c1; c2}, Q) = \mathit{pfp}(\texttt{c1}, \mathit{pfp}(\texttt{c2}, Q))$$

- Séquence vide:
  $$\mathit{pfp}(\{\}, Q) = Q$$

Exemple:

$\mathit{pfp}(\texttt{x = x + 5; y = x - y}, (x = 7 \wedge y = 10)) \equiv$
$\mathit{pfp}(\texttt{x = x + 5}, (x = 7 \wedge x - y = 10)) \equiv$
$(x + 5 = 7 \wedge (x + 5) - y = 10) \equiv$
$x = 2 \wedge y = -3$

pfp: Séquence (2)

Exemple: Échange de deux variables

Démontrer:

$\{x = A \wedge y = B\}$

```
x = x + y;
y = x - y;
x = x - y
```

$\{x = B \wedge y = A\}$

pfp: Séquence (3)

Solution:

$$pfp(\texttt{x = x + y; y = x - y; x = x - y}, (x = B \wedge y = A)) \equiv$$
$$pfp(\texttt{x = x + y; y = x - y}, (x - y = B \wedge y = A)) \equiv$$
$$pfp(\texttt{x = x + y}, (x - (x - y) = B \wedge x - y = A)) \equiv$$
$$((x + y) - ((x + y) - y) = B \wedge (x + y) - y = A) \equiv$$
$$y = B \wedge x = A$$

pfp: Sélection (1)

Règle:

$$\begin{aligned}&\mathit{pfp}(\texttt{if (b) c1 else c2}, Q) = \\ &b \longrightarrow \mathit{pfp}(\texttt{c1}, Q) \wedge \\ &\neg b \longrightarrow \mathit{pfp}(\texttt{c2}, Q)\end{aligned}$$

Exemple:

$$\begin{aligned}&\mathit{pfp}(\texttt{if (x <= y) m = x else m = y}, (x > m)) \equiv \\ &(x \leq y \longrightarrow \mathit{pfp}(\texttt{m = x}, (x > m))) \wedge \\ &(x > y \longrightarrow \mathit{pfp}(\texttt{m = y}, (x > m))) \quad \equiv \\ &(x \leq y \longrightarrow x > x) \wedge (x > y \longrightarrow x > y) \quad \equiv \\ &(x > y \vee x > x) \wedge \top \quad \equiv x > y\end{aligned}$$

pfp: Sélection (2)

Démontrer

$\{x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}\}$

```
if (y % 2 == 0) {
  x = x * x;
  y = y / 2;
  }
else {
  z = z * x;
  y = y - 1;
  }
```

$\{x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}\}$

pfp: Sélection (3)

Solution: Soit $F \equiv x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}$

pfp(`if(y % 2 == 0) {..} else {..}`, F) $\equiv$

$y\%2 = 0 \longrightarrow$ *pfp*(`x = x * x; y = y / 2`, F) $\wedge$
$y\%2 \neq 0 \longrightarrow$ *pfp*(`z = z *x; y = y - 1`, F) $\equiv$

$y\%2 = 0 \longrightarrow (x * x)^{(y/2)} * z = A^B \wedge (x * x) \in \mathbb{Z} \wedge (y/2) \in \mathbb{Z} \wedge$
$y\%2 \neq 0 \longrightarrow x^{y-1} * z * x = A^B \wedge x \in \mathbb{Z} \wedge (y-1) \in \mathbb{Z} \equiv$

$y\%2 = 0 \longrightarrow x^y * z = A^B \wedge (x * x) \in \mathbb{Z} \wedge (y/2) \in \mathbb{Z} \wedge$
$y\%2 \neq 0 \longrightarrow x^y * z = A^B \wedge x \in \mathbb{Z} \wedge (y-1) \in \mathbb{Z}$

ce qui est impliqué par: $x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}$

pfp: Sélection (4)

Exemple: Règle pour `if (b) c1`

Développer cette règle, en utilisant

- La traduction en `if - then - else`
- La règle pour la sélection
- La règle pour la séquence vide

Solution: Voir feuille de TD

pfp: Boucle (1)

Exemple: Soit `prog` le programme:

```
while (x < 3) {
  x = x + 1;
  y = y + 2;
  }
```

$\{x = 3 \wedge y = 6\}$

Quelle est *pfp*(`prog`, $x = 3 \wedge y = 6$) ??

- Lors du dernier passage de la boucle: $x = 2 \wedge y = 4$
- ... avant-dernier ...: $x = 1 \wedge y = 2$
- ... premier ...: $x = 0 \wedge y = 0$

Donc: *pfp*(`prog`, $x = 3 \wedge y = 6$) $= (x = 0 \wedge y = 0)$

pfp: Boucle (2)

Problèmes:

- En général: nombre d'itérations inconnu *a priori*
- La *pfp* est différente à chaque itération

Observation: Une propriété reste inchangée: $y = 2 * x \wedge x \leq 3$
$\rightsquigarrow$ *invariant* de la boucle

pfp: Boucle (2)

Notation pour des boucles avec invariant `I`

```
/* INV: { I } */
while (b)
  c
```

Exemple:

```
/* INV: { y = 2 * x } */
while (x < 3)
 { x = x + 1;
   y = y + 2;
 }
```

pfp: Boucle (3)

Règles pour une boucle de la forme

```
/* INV: { I } */
while (b)
  c
```

Calcul de la pfp:

$$pfp(\texttt{/* INV: I */ while (b) c}, \quad Q \wedge \neg b) = I$$

Correction de l'invariant:

$$(I \wedge b) \longrightarrow pfp(\texttt{c}, I)$$

Correction d'une boucle (1)

Comment vérifier?

$\{P\}$

```
/* INV: { I } */
while (b)
  c
```

$\{Q\}$

1. Initialisation correcte: $P \longrightarrow I$
2. Préservation de l'invariant: $(I \wedge b) \longrightarrow \mathit{pfp}(\mathsf{c}, I)$
3. Sortie de la boucle: $(I \wedge \neg b) \longrightarrow Q$

Correction d'une boucle (2)

Exemple:

$\{\top\}$
```
x = 0; y = 0;
```
$\{x = 0 \wedge y = 0\}$
```
/* INV: { y = 2 * x ∧ x ≤ 3}  */
while (x < 3)
  { x = x + 1;    y = y + 2;  }
```
$\{x = 3 \wedge y = 6\}$

1. Initialisation correcte: $(x = 0 \wedge y = 0) \longrightarrow (y = 2 * x)$
2. Préservation de l'invariant:
   $(y = 2 * x) \wedge (x \leq 3) \wedge (x < 3) \longrightarrow (y + 2) = 2 * (x + 1) \wedge (x + 1) \leq 3$
3. Sortie de la boucle: $y = 2 * x \wedge x \leq 3 \wedge \neg(x < 3) \longrightarrow x = 3 \wedge y = 6$

Correction d'une boucle (3)

Exemple: Calcul efficace de puissance

$\{A \in \mathbb{Z} \wedge B \in \mathbb{Z}\}$

```
x = A; y = B; z = 1;
/* INV: {x^y * z = A^B ∧ x ∈ ℤ ∧ y ∈ ℤ}   */

while (y != 0)
   if (y % 2 == 0) {
      x = x * x; y = y / 2;
   }
   else {
      z = z * x; y = y - 1;
   }
```

$\{z = A^B\}$

Faire la vérification !

Correction d'une boucle (4)

Solution: Soit $A \in \mathbb{Z} \wedge B \in \mathbb{Z}$

1. Initialisation correcte:
   $x = A \wedge y = B \wedge z = 1 \longrightarrow x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}$

2. Préservation de l'invariant:
   $x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z} \wedge (y \neq 0) \longrightarrow x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}$
   (Voir exemple "sélection")

3. Sortie de la boucle:
   $x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z} \wedge (y = 0) \longrightarrow z = A^B$

- Motivation: Pourquoi?
- Motivation: Comment?
- Conventions et notions de base
- Règles d'inférence
- Correction partielle vs. correction totale

Motivation: Correction totale

Le programme “calcul de puissance” a été prouvé “correct”
– et pourtant il y a un problème.

Lequel?

Problème: Non-terminaison si `B` est négatif!

Correction partielle / totale: Définitions

Deux notions de correction:

- `prog` est *partiellement correct* par rapport à $\{P\}$ `prog` $\{Q\}$ si:
  - pourvu que P est satisfait avant exécution, et
  - pourvu que `prog` termine (... mais ce n'est pas assuré !)
  - alors Q est satisfait après

- `prog` est *totalement correct* par rapport à $\{P\}$ `prog` $\{Q\}$ si:
  - pourvu que P est satisfait avant exécution
  - alors `prog` termine
  - et Q est satisfait après exécution

Correction partielle / totale: Règles

Correction partielle: Les règles 1 . . . 3 vues auparavant

Correction totale: En ajoutant deux règles:

Trouver une *variante* f telle que:

4. f strictement positif initialement: $I \wedge b \longrightarrow f > 0$

5. f strictement décroissante: $(T = f) \wedge I \wedge b \longrightarrow \mathit{pfp}(\mathsf{c}, T > f)$

Correction partielle / totale: Exemple (1)

Comment "réparer" le calcul de la puissance?

- Renforcer la précondition:
  en lieu de $\{A \in \mathbb{Z} \wedge B \in \mathbb{Z}\}$ prendre $\{A \in \mathbb{Z} \wedge B \in \mathbb{N}\}$

- Renforcer l'invariant:
  en lieu de `/* INV:` $\{x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{Z}\}$ `*/`
  prendre `/* INV:` $\{x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{N}\}$ `*/`
  et mettre à jour les preuves des étapes 1 .. 3 !!

- Choisir variant: f est y

- Démontrer: f strictement positif initialement:
  $x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{N} \wedge y \neq 0 \longrightarrow y > 0$

Correction partielle / totale: Exemple (2)

- Démontrer: f strictement décroissante:

  $(T = y) \wedge x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{N} \wedge y \neq 0 \longrightarrow$
  $(y\%2 = 0 \longrightarrow T > y/2) \wedge$
  $(y\%2 \neq 0 \longrightarrow T > y - 1)$

  ... après quelques simplifications:

  $x^y * z = A^B \wedge x \in \mathbb{Z} \wedge y \in \mathbb{N} \wedge y \neq 0 \longrightarrow$
  $(y\%2 = 0 \longrightarrow y > y/2) \wedge$
  $(y\%2 \neq 0 \longrightarrow y > y - 1)$

Résumé

- Vérification de programmes:
  - Tests: incomplets, mais (apparemment?) faciles
  - Preuves: vérification exhaustive

  $\rightsquigarrow$ Techniques complémentaires

- Schéma de preuve: Propagation d'assertions "d'arrière en avant"

- ... en calculant la *pfp*

- Deux notions de correction: partielle (sans terminaison) / totale (avec)

Regarder l'exemple initial de plus près. Est-il vraiment "correct"? Sous quelles conditions?