

Preuve de sûreté de programmes impératifs

Quentin Peyras

EPITA

3 décembre 2024

1 Introduction

2 Preuve de programmes impératifs

- Sémantique de programme impératifs
- Annotations et programmation par contrat
- Weakest Précondition
- Why3

Sommaire

1 Introduction

2 Preuve de programmes impératifs

- Sémantique de programme impératifs
- Annotations et programmation par contrat
- Weakest Précondition
- Why3

Vérification déductive de programmes impératifs

- Application de règles de déduction pour certifier les propriétés d'un programme
- TP noté (50% de la note finale)

Interprétation abstraite

- Approximation de la mémoire par des structures abstraites
- TP noté (50% de la note finale)

Sommaire

1 Introduction

- ## 2 Preuve de programmes impératifs
- Sémantique de programme impératifs
 - Annotations et programmation par contrat
 - Weakest Précondition
 - Why3

Sommaire

1 Introduction

2 Preuve de programmes impératifs

- Sémantique de programme impératifs
- Annotations et programmation par contrat
- Weakest Précondition
- Why3

Impératif vs fonctionnel pur

Fonctionnel pur :

- Pas d'effet de bord
- Toute expression a une unique valeur (celle dans son contexte de définition)
- L'ordre d'évaluation n'influence pas le résultat final

Impératif :

- Modification de l'environnement
- La valeur d'une expression change en fonction de l'environnement
- La valeur d'une expression dépend de l'ordre d'évaluation

Exemple de code OCaml et C

Code OCaml :

```
let x = 4 in  
let f y = x + y in  
let x = 1 in  
f x;;
```

Code C :

```
int x = 4;  
int f(int y) {  
    return x + y;  
}  
int main() {  
    x = 1;  
    printf("%d\n", f(x));  
    return 0;  
}
```

Quel est le résultat affiché par chaque programme ?

Conséquence sur la preuve

Fonctionnel pur :

- Chaque expression a une unique valeur
- Une propriété d'une expression est toujours vraie
- Possible de raisonner en remontant au contexte de définition

Impératif :

- La valeur d'une expression dépend du l'environnement
- Une propriété d'une expression est conditionnée à des conditions sur l'environnement d'évaluation
- Impossible de raisonner indépendamment de l'état de l'environnement

Sémantique opérationnelle des petits pas

- modèle de mémoire = ensemble d'états que peut prendre la mémoire
- les programme P
- la sémantique est alors une fonction : $P \times M \rightarrow M$

Modèle de mémoire simple : affectation des variables

- modèle de mémoire = table d'association variables/valeurs
- $x := 1, [] \rightarrow [x = 1]$
- simpliste, pas de pointeurs, pas de fonctions, etc

Évaluation d'expression

Définition (Expression)

Élément syntaxique d'un programme qui renvoi une valeur.

- En programmation fonctionnelle tout est expression.
- En programmation impérative un programme se divise entre expressions et instructions, l'évaluation d'une expression e est une instruction $\text{eval}(e)$.

Sémantique des expressions (petits pas)

Variable

x est un nom de variable.

$$\text{eval}(x), m \rightarrow e[x], m$$

Constante

v est une valeur constante d'un type de base (booléen, entier, flottant, ...) ou le code d'une fonction.

$$\text{eval}(v), m \rightarrow v, m$$

Sémantique des expressions (petits pas)

Opérations

$\#$ est une opération $+, -, *, /, ==, \dots$, a et b sont des expressions et :

- $\text{eval}(a), m \rightarrow v_a, m'$
- $\text{eval}(b), m' \rightarrow v_b, m''$

$$\text{eval}(a\#b), m \rightarrow v_a\#v_b, m''$$

Sémantique des petits pas

Affectation

Si $\text{eval}(e), m \rightarrow v, m'$, alors $x := e, m \rightarrow (), m' \cup [x = v]$

Séquence

Si $P, m \rightarrow (), m'$, alors $P; Q, m \rightarrow Q, m'$

Sémantique des petits pas

If-Then-Else

- Si $eval(B), m \rightarrow \text{False}, m'$ alors $\text{if } B \text{ then } P \text{ else } Q, m \rightarrow Q, m'$
- Si $eval(B), m \rightarrow \text{True}, m'$ alors $\text{if } B \text{ then } P \text{ else } Q, m \rightarrow P, m'$

While

- Si $eval(B), m \rightarrow \text{False}, m'$ alors $\text{while } B \text{ do } P, m \rightarrow (), m'$
- Si $eval(B), m \rightarrow \text{True}, m'$ alors $\text{while } B \text{ do } P, m \rightarrow P; \text{ while } B \text{ do } P, m'$

Exercice

Appliquez la sémantique des petits pas à `division(5,2)` (avec environnement initial vide) :

```
void division(int a, int b) {  
    q = 0;  
    r = a;  
    while(r >= b){  
        q = q + 1;  
        r = r - b;  
    }  
}
```

Comment prouver que ce programme est correct ?

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

- Automatisable

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

- Automatisable
- Explosion combinatoire

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

- Automatisable
- Explosion combinatoire
- Ne donne pas d'intuition sur pourquoi le programme marche

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

- Automatisable
- Explosion combinatoire
- Ne donne pas d'intuition sur pourquoi le programme marche

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

- Automatisable
- Explosion combinatoire
- Ne donne pas d'intuition sur pourquoi le programme marche

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- Difficile à automatiser

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

- Automatisable
- Explosion combinatoire
- Ne donne pas d'intuition sur pourquoi le programme marche

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- Difficile à automatiser
- Suit un raisonnement logique et compréhensible

Triplet de Hoare

Motivation

Comment peut-on raisonner à propos de l'état d'un programme ?

Option 1

On regarde tous les états possibles du programme, c'est du model-checking :

- Automatisable
- Explosion combinatoire
- Ne donne pas d'intuition sur pourquoi le programme marche

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- Difficile à automatiser
- Suit un raisonnement logique et compréhensible
- Moins gourmand en temps de calcul

Triplet de Hoare

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- Précondition : propriété vérifiée avant l'instruction (ou la série d'instructions)
- Postcondition : propriété vérifiée après l'instruction (ou la série d'instructions)

Triplet de Hoare

Un triplet de Hoare est donc un triplet composé :

Triplet de Hoare

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- Précondition : propriété vérifiée avant l'instruction (ou la série d'instructions)
- Postcondition : propriété vérifiée après l'instruction (ou la série d'instructions)

Triplet de Hoare

Un triplet de Hoare est donc un triplet composé :

- **Précondition** : Condition qui doit être vraie avant l'exécution du programme.

Triplet de Hoare

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- Précondition : propriété vérifiée avant l'instruction (ou la série d'instructions)
- Postcondition : propriété vérifiée après l'instruction (ou la série d'instructions)

Triplet de Hoare

Un triplet de Hoare est donc un triplet composé :

- **Précondition** : Condition qui doit être vraie avant l'exécution du programme.
- **Commande** : Le programme à exécuter.

Triplet de Hoare

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- Précondition : propriété vérifiée avant l'instruction (ou la série d'instructions)
- Postcondition : propriété vérifiée après l'instruction (ou la série d'instructions)

Triplet de Hoare

Un triplet de Hoare est donc un triplet composé :

- **Précondition** : Condition qui doit être vraie avant l'exécution du programme.
- **Commande** : Le programme à exécuter.
- **Postcondition** : Condition qui doit être vraie après l'exécution du programme.

Triplet de Hoare

Option 2

On décrit quelle propriété le programme satisfait entre chaque instruction :

- **Précondition** : propriété vérifiée avant l'instruction (ou la série d'instructions)
- **Postcondition** : propriété vérifiée après l'instruction (ou la série d'instructions)

Triplet de Hoare

Un triplet de Hoare est donc un triplet composé :

- **Précondition** : Condition qui doit être vraie avant l'exécution du programme.
- **Commande** : Le programme à exécuter.
- **Postcondition** : Condition qui doit être vraie après l'exécution du programme.

Les triplet de Hoare suivent des règles de déduction logiques en fonction des instructions exécutées.

Affectation

$$\overline{\{ I[x \mapsto E] \} \ x := E \ \{ I \}}$$

Implication logique

$$\frac{I \vdash J \quad \{J\} P \{K\} \quad K \vdash L}{\{I\} P \{J\}} \text{ (affaiblissement)}$$

Composition

$$\frac{\{I\} P \{J\} \quad \{J\} Q \{K\}}{\{I\} P; Q \{K\}} (;)$$

Exercice

Proposez une règle pour la conditionnelle If-then-else et une règle pour la boucle While.

If-then-else

$$\frac{\{I \wedge B\} P \{J\} \quad \{I \wedge \neg B\} Q \{J\}}{\{I\} \text{ if } B \text{ then } P \text{ else } Q \{J\}} \text{ (ite)}$$

Exercice

Donnez une dérivation du triplet de Hoare suivant :

$$\{x > 2\} \text{ if } x > 1 \text{ then } y := 1 \text{ else } y := -1 \{y > 0\}$$

While

$$\frac{\{I \wedge B\} P \{I\}}{\{I\} \text{ while } B \text{ do } P \{I \wedge \neg B\}} \text{ (while)}$$

Exercice 1

Donnez une dérivation du triplet de Hoare suivant :

$$\{n \geq 12\} \text{ while } n > 0 \text{ do } n := n - 1 \text{ done } \{n = 0\}$$

Exercice 2

Donnez une dérivation du triplet de Hoare suivant :

$$\{n \geq 12\} \text{ while } n > 0 \text{ do } n := n + 1 \text{ done } \{n = 0\}$$

While avec terminaison

$$\frac{I \Rightarrow V \geq 0 \quad \{I \wedge B \wedge V = n\} P \{I \wedge V < n\}}{\{I\} \text{ while } B \text{ do } P \{I \wedge \neg B\}} \text{ (total while)}$$

Exercice

Donnez une dérivation du triplet de Hoare suivant en assurant la terminaison :

$$\{n > k\} \text{ while } n > k \text{ do } n, k := n + 1, k + 2 \text{ done } \{n = k\}$$

Sommaire

1 Introduction

2 Preuve de programmes impératifs

- Sémantique de programme impératifs
- **Annotations et programmation par contrat**
- Weakest Précondition
- Why3

Annotation de programme

Motivation

Utiliser des arbres de dérivation de triplet de Hoare n'est pas adapté à la certification de code :

- Inutilisable pour du code de plus de quelques lignes
- Beaucoup de combinaisons de séquences qui n'apportent pas grand chose

Solution

On travaille directement dans le code :

- On ajoute des commentaires décrivant les formules vérifiées à chaque étape du programme
- Rien ne change à la compilation
- On peut certifier le programme en vérifiant la validité des triplets de Hoare ligne par ligne automatiquement

Cela donne un programme annoté à la fois compilable et certifiable.

Programmation par contrat

Problème

Comment traiter la définition et l'appel de fonction ?

Programmation par contrat

Contrat d'une fonction

Programmation par contrat

Contrat d'une fonction

Le contrat d'une fonction permet de définir son comportement lors de son utilisation. Un contrat doit contenir :

Programmation par contrat

Contrat d'une fonction

Le contrat d'une fonction permet de définir son comportement lors de son utilisation. Un contrat doit contenir :

- la précondition de la fonction, c'est à dire les propriétés que doivent satisfaire les entrées de la fonction

Programmation par contrat

Contrat d'une fonction

Le contrat d'une fonction permet de définir son comportement lors de son utilisation. Un contrat doit contenir :

- la précondition de la fonction, c'est à dire les propriétés que doivent satisfaire les entrées de la fonction
- la postcondition de la fonction, c'est à dire les propriétés satisfaites par les valeurs de retour de la fonction

Programmation par contrat

Contrat d'une fonction

Le contrat d'une fonction permet de définir son comportement lors de son utilisation. Un contrat doit contenir :

- la précondition de la fonction, c'est à dire les propriétés que doivent satisfaire les entrées de la fonction
- la postcondition de la fonction, c'est à dire les propriétés satisfaites par les valeurs de retour de la fonction

Le mot contrat vient du fait que si l'utilisateur s'engage à utiliser la fonction dans un cadre respectant la précondition alors le programmeur s'engage à ce que sa fonction satisfasse la post-condition.

Programmation par contrat

Contrat d'une fonction

Le contrat d'une fonction permet de définir son comportement lors de son utilisation. Un contrat doit contenir :

- la précondition de la fonction, c'est à dire les propriétés que doivent satisfaire les entrées de la fonction
- la postcondition de la fonction, c'est à dire les propriétés satisfaites par les valeurs de retour de la fonction

Le mot contrat vient du fait que si l'utilisateur s'engage à utiliser la fonction dans un cadre respectant la précondition alors le programmeur s'engage à ce que sa fonction satisfasse la post-condition.

Remarque

Pour obtenir du code exécutable certifié il faut également que le compilateur utilisé soit certifié.

Exercice

Annotez le programme suivant en ajoutant les conditions à chaque étape du programme :

```
int division(int a, int b) {  
    q = 0;  
    r = a;  
    while(r >= b){  
        q = q + 1;  
        r = r - b;  
    }  
    return q;  
}
```


Sommaire

1 Introduction

2 Preuve de programmes impératifs

- Sémantique de programme impératifs
- Annotations et programmation par contrat
- **Weakest Précondition**
- Why3

Weakest Precondition

Motivation

- Annoter tout un programme est fastidieux
- Beaucoup d'annotations se déduisent sans difficulté des instructions et de la post-condition
- Comment automatiser cela ?

Weakest precondition

La plus faible précondition, notée $wp(P, Q)$, d'un programme P par rapport à une postcondition Q est la formule logique la plus faible permettant d'assurer que Q soit satisfaite après l'exécution de P .

Weakest Precondition

Définition (Weakest precondition)

Elle est définie inductivement comme suit :

Weakest Precondition

Définition (Weakest precondition)

Elle est définie inductivement comme suit :

- $wp(x := a, Q) = Q[x/a]$, où $Q[x/a]$ est Q où toutes les occurrences de x sont remplacées par a .

Weakest Precondition

Définition (Weakest precondition)

Elle est définie inductivement comme suit :

- $wp(x := a, Q) = Q[x/a]$, où $Q[x/a]$ est Q où toutes les occurrences de x sont remplacées par a .
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$

Weakest Precondition

Définition (Weakest precondition)

Elle est définie inductivement comme suit :

- $\text{wp}(x := a, Q) = Q[x/a]$, où $Q[x/a]$ est Q où toutes les occurrences de x sont remplacées par a .
- $\text{wp}(S_1; S_2, Q) = \text{wp}(S_1, \text{wp}(S_2, Q))$
- $\text{wp}(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) = (b \wedge \text{wp}(S_1, Q)) \vee (\neg b \wedge \text{wp}(S_2, Q))$

Weakest Precondition

Définition (Weakest precondition)

Elle est définie inductivement comme suit :

- $wp(x := a, Q) = Q[x/a]$, où $Q[x/a]$ est Q où toutes les occurrences de x sont remplacées par a .
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
- $wp(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) = (b \wedge wp(S_1, Q)) \vee (\neg b \wedge wp(S_2, Q))$
- $wp(\text{while } b \text{ do } S \text{ done}, Q) = (inv \wedge b \wedge wp(S, inv)) \vee (\neg b \wedge inv)$, où inv est le plus faible invariant de la boucle.

Weakest Precondition

Définition (Weakest precondition)

Elle est définie inductivement comme suit :

- $wp(x := a, Q) = Q[x/a]$, où $Q[x/a]$ est Q où toutes les occurrences de x sont remplacées par a .
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
- $wp(\text{if } b \text{ then } S_1 \text{ else } S_2, Q) = (b \wedge wp(S_1, Q)) \vee (\neg b \wedge wp(S_2, Q))$
- $wp(\text{while } b \text{ do } S \text{ done}, Q) = (inv \wedge b \wedge wp(S, inv)) \vee (\neg b \wedge inv)$, où inv est le plus faible invariant de la boucle.

Attention il est difficile d'automatiser le calcul de l'invariant de boucle ! Il en est de même du variant de boucle.

Retour des annotations

Invariant de boucle

Pour calculer la weakest precondition, il est nécessaire d'annoter le programme avec les invariants de boucle.

Annotations

Un programme annoté consiste finalement à un programme dans lequel :

- les prototypes de fonctions sont annotés avec leur contrat contenant :
 - ses préconditions
 - ses postconditions
- les boucles sont annotées avec leur invariant
- pour prouver la terminaisons elles sont également annotées avec leur variant

C'est ce principe que suit l'outil Why3.

Exercice

Annotez le programme suivant en ajoutant un contrat et un invariant de boucle (il faut utiliser l'appartenance mathématique pour les formaliser) :

```
List search(List searched, int key) {  
    List current = searched;  
    int n = 0;  
    while (current != NULL) {  
        if (current->value == key) {  
            return n;  
        }  
        current = current->next;  
        n++;  
    }  
    return NULL;  
}
```

Sommaire

1 Introduction

2 Preuve de programmes impératifs

- Sémantique de programme impératifs
- Annotations et programmation par contrat
- Weakest Précondition
- **Why3**

Un exemple d'outil : Frama-C

Qu'est-ce que Frama-C ?

Frama-C est un environnement de développement dédié à l'analyse et à la certification de programmes C (langage dominant dans l'embarqué).

- Frama-C offre différentes méthodes de vérifications (weakest precondition, interprétation abstraite, ...)
- Frama-C fait appel à différents outils pour ces vérifications

En particulier, Frama-C utilise l'assistant de preuve Why3

Vérification de programme avec Frama-C et Why3

