Vérification de Propriétés de Programmes Impératifs

une introduction

Olivier Ponsini

Université de Nice-Sophia Antipolis — Master 1

14 décembre 2005

Contexte et Motivation

- Omniprésence de l'informatique
- Besoin de fiabilité logicielle
  - tests
    - montrer la présence d'erreurs, pas leur absence !
  - méthodes formelles
    - langages, techniques et outils mathématiques pour la spécification et la vérification des systèmes

Contexte et Motivation

Réalité industrielle

Difficile diffusion des méthodes formelles

- Coût financier de mise en œuvre
  - temps
  - expertise
- Vide juridique quant à la responsabilité des développeurs
- En train d'évoluer
  - pertes financières dues à des erreurs
  - normes (Critères Communs)

Le Problème de la Correction des Programmes

Définition

Le programme satisfait-il ses spécifications ?

- Une (petite) partie du problème de la fiabilité des systèmes
- La partie sans doute la mieux comprise pour l'instant

Spécifier

Pourquoi spécifier formellement ?

- Décrire une infinité de comportements
  - révéler ambiguïtés, incohérences, oublis
  - communiquer efficacement
  - documenter (guide le développement, facilite la maintenance et la réutilisation)
- Automatiser les traitements ultérieurs
  - prototype (si spécification exécutable)
  - vérification

Spécifier

Limites

- Correction de la spécification
  - essence du problème informelle
  - transition informel/formel invérifiable de façon absolue
- Complétude de la spécification
  - abstraction mathématique du monde réel
  - compilateur ? matériel ?
  - assomptions irréductibles sur la modélisation du monde physique

Spécifier

Limites

Ne pas avoir une confiance démesurée dans les méthodes formelles

- Correct, mais toujours sous certaines hypothèses
- Choix du niveau acceptable de modélisation et de formalisation

Spécifier

Malgré tout !

Le Processus de spécification formel est virtueux

- Améliore la compréhension du problème réel
- Seule méthode à admettre un traitement rigoureux
- Possibilité d'automatisation
  - spécification manuelle par nature
  - vérification
    - indécidable en général
    - automatique pour des cas particuliers (démonstrateur de théorèmes)
    - au moins vérifier mécaniquement la preuve de la vérification (assistant de preuves)

Approches Existantes

- Synthèse de programme
  - programmes corrects par construction
    - spécification exécutable
    - logiques constructives (preuve = programme)
  - génération de code (raffinements successifs)
- Vérification de modèle
  - modélisation finie du système (automate)
  - parcours exhaustif automatique des états du modèle
  - explosion de l'espace des états

Approches Existantes

- Vérification par démonstration
  - raisonner dans des systèmes de calcul permettant de traiter un nombre infini d'états
  - semi-automatique
  - adaptée à la vérification de propriétés
  - adaptée aux programmes séquentiels manipulant des données complexes
  - adaptée à la vérification de portions de code critiques

Exemple

```
int somme_n(int n) {
  int s = 0;

  while(n > 0) {
    s = s + n;
    n = n - 1;
  }
  return s;
}
```

Propriété : $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Comment raisonner sur les programmes ?

- Donner un sens au langage de programmation
- Définir la sémantique d'un langage formel = lui donner une signification mathématique
- Rôles
  - comprendre les programmes
  - tester et vérifier les programmes
  - écrire et comprendre des spécifications
  - écrire des compilateurs
  - classifier les langages, en concevoir de nouveaux

Sémantiques

- Approche opérationnelle
- Approche dénotationnelle
- Approche axiomatique
- Approche algébrique

Exemples de méthodes

- Logique de Floyd-Hoare
- SOSSub$_{\mathcal{C}}$

Logique de Floyd-Hoare [Floyd 67, Hoare 69]

- Approche axiomatique
- Annotation des programmes
- Règles logiques pour raisonner

Triplets de Hoare

$$\{P\}\mathtt{I}\{Q\}$$

- $\mathtt{I}$ Une instruction
- P L'antécédent (pré-condition)
- Q Le conséquent (post-condition)

Définition

Si la propriété P est vraie pour les valeurs des variables du programme avant l'exécution de $\mathtt{I}$, et si l'exécution termine, alors la propriété Q est vraie après l'exécution de I.

Correction Partielle et Totale

- Partielle : $\models_{par} \{P\}\mathtt{I}\{Q\}$, ssi pour tous états σ, σ'
  - si $\sigma \models P$ et $(\mathtt{I}, \sigma) \rightarrow \sigma'$ alors $\sigma' \models Q$
- Totale : $\models_{tot} \{P\}\mathtt{I}\{Q\}$, ssi pour tout état σ
  - si $\sigma \models P$ alors $\mathtt{I}$ termine et
  - pour tout état σ', si $(\mathtt{I}, \sigma) \rightarrow \sigma'$ alors $\sigma' \models Q$

Calcul de Hoare

- Axiome

$$\{P[E/x]\}\texttt{x=E}\{P\}$$

- Règles d'inférence
  - séquence $\dfrac{\{P\}\texttt{I}_1\{P'\} \quad \{P'\}\texttt{I}_2\{Q\}}{\{P\}\texttt{I}_1\texttt{ ; }\texttt{I}_2\{Q\}}$
  - conditionnelle $\dfrac{\{\texttt{C} \wedge P\}\texttt{I}_1\{Q\} \quad \{\neg\texttt{C} \wedge P\}\texttt{I}_2\{Q\}}{\{P\}\texttt{if(C) I}_1\texttt{ else I}_2\{Q\}}$

Calcul de Hoare

- boucle $\dfrac{\{\texttt{C} \wedge P\}\texttt{I}\{P\}}{\{P\}\texttt{while(C) I}\{\neg\texttt{C} \wedge P\}}$
- conséquence $\dfrac{P \Rightarrow P' \quad \{P'\}\texttt{I}\{Q'\} \quad Q' \Rightarrow Q}{\{P\}\texttt{I}\{Q\}}$

Correction du calcul de Hoare

Théorème

Si $\vdash \{P\}\texttt{I}\{Q\}$ alors $\models_{par} \{P\}\texttt{I}\{Q\}$

Exemple

$\{N \geq 0\}$
`int somme_n(int n) {`
$\{n = N \wedge N \geq 0\}$
`  int s = 0 ;`
$\{n = N \wedge N \geq 0 \wedge s = 0\}$
$\{I = (n \geq 0 \wedge somme(N) = somme(n) + s)\}$
`  while(n > 0) {`
$\{n > 0 \wedge I\}$
$\{n \geq 1 \wedge \frac{N(N+1)}{2} = \frac{(n-1)n}{2} + s + n\}$
`    s = s + n ;`
$\{n - 1 \geq 0 \wedge \frac{N(N+1)}{2} = \frac{(n-1)n}{2} + s\}$
`    n = n - 1 ;`
$\{I\}$
$\{I = (n \geq 0 \wedge \frac{N(N+1)}{2} = \frac{n(n+1)}{2} + s)\}$
`  }`

$$\frac{\{C \wedge I\}\texttt{E}\{I\}}{\{I\}\texttt{while(C) E}\{\neg C \wedge I\}}$$

Logique Équationnelle

Langage

- Sous-ensemble de la logique du premier ordre
- Substitution de termes égaux
- Langage
  - un ensemble $\mathcal{F}$ de symboles de fonction $(f, g \ldots)$ munis d'une arité
  - un ensemble $\mathcal{V}$ de variables $(x, y \ldots)$
  - l'ensemble $\mathcal{T}$ des termes sur $\mathcal{F}$ et $\mathcal{V}$, défini récursivement :
    - $x \in \mathcal{V}$ est un terme
    - soient $t_1, \ldots, t_n \in \mathcal{T}$, $f \in \mathcal{F}$ d'arité n, $f(t_1, \ldots, t_n) \in \mathcal{T}$

Logique Équationnelle

Langage

- Formules de la logique équationnelle conditionnelle :

$$e_1 \wedge e_2 \wedge \cdots \wedge e_n \Rightarrow e_0$$

avec $\forall 0 \leq i \leq n,\ e_i \equiv t_i = u_i$

Exemples

$$(x.y).z = x.(y.z) \qquad z \geq 0 = \mathit{true} \Rightarrow \mathit{abs}(z) = z$$
$$x.e = x \qquad z < 0 = \mathit{true} \Rightarrow \mathit{abs}(z) = -z$$
$$x.x^{-1} = e$$

Logique Équationnelle

Axiomes

$$t = u \wedge t' = u' \Rightarrow t = u$$
$$t = t$$
$$t = v \wedge u = v \Rightarrow t = u$$
$$t_1 = u_1 \wedge \cdots \wedge t_n = u_n \Rightarrow f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)$$
pour tout f d'arité n

Logique Équationnelle

Règles d'inférence

$$\frac{F \Rightarrow t' = u' \wedge E \wedge t' = u' \Rightarrow t = u}{E \wedge F \Rightarrow t = u}$$

$$\frac{E \Rightarrow t = u}{\sigma(E) \Rightarrow \sigma(t) = \sigma(u)}$$
pour toute substitution σ

Avec $E = \{t_1 = u_1 \wedge \cdots \wedge t_n = u_n\}(n \geq 0)$,
$\sigma(E) = \{\sigma(t_1) = \sigma(u_1) \wedge \cdots \wedge \sigma(t_n) = \sigma(u_n)\}$,
$F = \{t'_1 = u'_1 \wedge \cdots \wedge t'_m = u'_m\}(m \geq 0)$.

Exemples

$$x + 0 = x$$
$$x + s(y) = s(x + y)$$

Montrer que $1 + 2 = 3$

$$a = b$$
$$f(x) = g(x)$$

Montrer que $g(b) = f(a)$

Systèmes de Réécriture

- Ensemble d'équations orientées en règles
- Processus de réécriture : application des règles
- Propriétés
  - terminaison
  - confluence

Exemples

$$(x.y).z \longrightarrow x.(y.z) \qquad z \geq 0 = true \Rightarrow abs(z) \longrightarrow z$$
$$x.e \longrightarrow x \qquad z < 0 = true \Rightarrow abs(z) \longrightarrow -z$$
$$x.x^{-1} \longrightarrow e$$

Exemples (bis)

$$x + 0 \rightarrow x$$
$$x + s(y) \rightarrow s(x + y)$$

Montrer que $1 + 2 = 3$

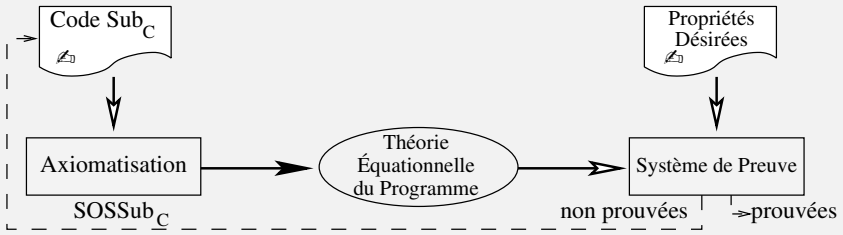
$$a \rightarrow b$$
$$f(x) \rightarrow g(x)$$

Montrer que $g(b) = f(a)$

SOSSub$_{\mathcal{C}}$

Motivation et aperçu

- Exprimer programmes et propriétés dans la logique équationnelle
  - approche algébrique
  - la sémantique des programmes
  - sans modèle de la mémoire



Pourquoi la Logique Équationnelle ?

- Simplicité du formalisme
  - syntaxe concise
  - sémantique claire
- Puissance
- Capacité à être automatisée
  - algorithmes efficaces
  - nombreux outils existants
- Largement répandue

Processus de Preuve

Exemple

Équations du programme

$$\begin{aligned} somme\_n(n) &= loop(n, 0) \\ \neg(n > 0) &\Rightarrow loop(n, s) = s \\ n > 0 &\Rightarrow loop(n, s) = loop(n - 1, s + n) \end{aligned}$$

Propriété du programme à vérifier

$$n \geq 0 \Rightarrow somme\_n(n) = n * (n + 1)/2$$

Langage Sub$_{\mathcal{C}}$

- Langage impératif typé : entiers et listes
- Inspiré du langage C
- Constructions disponibles :
  - affectation
  - séquence
  - conditionnelle
  - itération
  - sous-programme

Langage Sub$_{\mathcal{C}}$

- Opérateurs disponibles :
  - opérateurs usuels
    - arithmétiques
    - logiques
    - relationnels
  - opérateurs sur les listes
    - `NULL`
    - `element(L)`
    - `next(L)`
    - `add(elt, L)`

Problème

Axiomatisation

Entrée : Un programme Sub$_{\mathcal{C}}$ P
Sortie : Un ensemble d'équations qui expriment la sémantique du programme P

- Utiliser le code source non annoté
- Traduire des programmes incomplets
- Abstraire les programmes par des fonctions de transfert des entrées vers les sorties
- Ne pas expliciter la mémoire dans les équations

Principes

Définir la sémantique du langage dans le but d'obtenir celle du programme

- Environnement d'un programme = ensemble d'équations
- Décrire la sémantique des constructions par des modifications dans l'environnement d'un programme
- Évaluer symboliquement les programmes avec cette sémantique

Pierre angulaire de la traduction

- Système de réécriture

Détails de l'Axiomatisation

- La définition de la sémantique est donnée par les règles d'un système de réécriture
  - définition formelle
  - définition exécutable

Règles pour l'affectation

$$Comp(Assign(var, exp), EmptyEnv) \longrightarrow Env((var, exp), EmptyEnv)$$

$$Comp(Assign(var, exp), Env(pair, env)) \longrightarrow UpdateEnv((var, exp), Env(pair, env))$$

Détails de l'Axiomatisation

- Les trois étapes de l'axiomatisation d'un programme
  1. le code source est transformé, syntaxiquement, en un terme
  2. le terme est réécrit en l'environnement final
  3. l'environnement est formulé en équations

Intuition de la Sémantique

- Affectation
  x=2 ; représentée par $(x, 2)$ signifie $x = 2$
- Conditionnelle
  - partage les sous-programmes en chemins d'exécution
  - introduit une équation conditionnelle
- Itération
  - fonction récursive terminale

Intuition de la Sémantique

Conditionnelle

```
int f(int n, int a) {
  n = n + a ;
  if(a > 0)
    n = -n ;
  else
    n = n + a ;
  return n ;
}
```

$a > 0 = true \Rightarrow f(n, a) = -(n+a)$

$\neg(a > 0) = true \Rightarrow f(n, a) = (n + a + a)$

$Choice(Branch(a > 0, (n, -(n + a))),$
$Branch(\neg(a > 0), (n, n + a + a)))$

Intuition de la Sémantique

Itération

```
int somme_n(int n) {
  int s = 0 ;

  s' = s ;
  n' = n ;
  s = loop_s(n', s') ;
  n = loop_n(n', s') ;
  return s ;
}
```

```
int loop_n(int n, int s) {
  if(n > 0) {
    s = s + n ;
    n = n - 1 ;
    return loop_n(n, s) ;
  }
  else return n ;
}
```


Intuition de la Sémantique

Itération

```
while(n > 0) {
  s = s + n ;
  n = n - 1 ;
}
```

- Environnement

$$WhileClosure(n > 0, \{(s, s + n) \cdot (n, n - 1)\})$$

- Équations

$$n > 0 = true \Rightarrow loop_s(n, s) = loop_s(n - 1, s + n)$$
$$\neg(n > 0) = true \Rightarrow loop_s(n, s) = s$$

Exemple

Inversion des éléments d'une liste

```
list inverseSubC(list L) {
  list r;
  while(L != NULL) {
    r = add(element(L), r);
    L = next(L);
  }
  return r;
}
```

$$
\begin{aligned}
inverseSubC(L) &= loop(L, NULL) \\
L = NULL &\Rightarrow loop(L, r) = r \\
\neg(L = NULL) = true &\Rightarrow loop(L, r) = \\
&\quad loop(next(L), add(element(L), r))
\end{aligned}
$$