Fiabilidade dos Sistemas Informáticos

Verificação de Programas Imperativos
Lógica de Hoare e outros métodos baseado em Pré/Pós-Condições e Invariantes

Simão Melo de Sousa

Computer Science Department
University of Beira Interior, Portugal

2005-2006

Outline

Plano

Lógica de Floyd-Hoare: Origens

- Robert W. Floyd. "Assigning meaning to programs". In Proc. Symp. in Applied Mathematics, volume 19, 1967.
- C. A. R. (Tony) Hoare. "An axiomatic basis for computer programming". Communications of the ACM, 12 (10): 576-585. October 1969.
- Edsger W. Dijkstra, "Guarded command, nondeterminacy and formal derivation of programs". Communication of the ACM, 18 (8):453-457, August 1975.
- Referência principal para estes apontamentos: J-F. Monin. Understanding Formal Methods. Springer Verlag, 2002.

Objectivo

- Permitir a demonstração de correcção de programas imperativos.
- um estado lógico = conjunto de valores de interesse para um programa
- ideia: ver cada componente do programa analizado como um modificador de estados lógicos.
- $\Longrightarrow$ Cada programa tem um comportamento lógico particular que se pode comprovar

Relação entre estado lógico e programas

Programar....

1. O quê? Descrição da tarefa por realizar, do problema por resolver.
2. Como? O algoritmo, Descrição do método de resolução
3. A realização: o programa em si, concretização do método

Necessidade de comprovar que o algoritmo (podem existir vários) e, mais ainda, a própria implementação resolvem o problema proposto.
Assim sendo:

- Estado lógico: Especificação lógica do comportamento e das relações entre os dados manipulados. Semântica, em termos lógicos, do programa.
- Lógica de Hoare (e afins): método/enquadramento para comprovar a adequação do comportamento de programas com as suas especificações lógicas (que devem modelar problemas e algoritmos).

Plano

Linguagem de Estudo

A linguagem `While`

Para ilustrar de uma forma simples mas completa o conceito de lógica de Hoare vamos introduzir uma linguagem de estudo: a linguagem `While`, sobre a qual estudaremos os mecanismos de anotação, asserção e demonstração.

Sintaxe BNF

operadores	op	::=	$+ \mid - \mid * \mid / \mid mod$
expressões	e	::=	$n \mid x \mid x[e] \mid e\ op\ e$
condições	c	::=	$e \mid e > e \mid e = e \mid e\,!= e \mid e >= e$
			$\mid\ !(c) \mid c\&c \mid c\mid c$
instruções	i	::=	$skip$
			$\mid\ x := e$
			$\mid\ (c)?S : S$
			$\mid\ while(c)\{S\}$
sequências	S	::=	$\emptyset \mid i; S \mid \{S\}$

Plano

Triplos de Hoare

Um triplo de hoare é

$$\{P\}\ S\ \{Q\}$$

Onde S é um programa `while` e P e Q são predicados sobre as variáveis de S.

- P: Pre-Condição Q: pós-Condição
- Semântica Informal: Se a execução de S é iniciada num estado lógico em que P é verificado então esta execução conduz a um estado lógico verificando Q

Conceitos de Base

Exemplos

- $\{x = 29\}$ `x:= x + 1` $\{x > 20\}$
- $\{x + 1 = 30\}$ `x:= x + 1` $\{x = 30\}$
- $\{x \neq 0\}$ `?(x<0)x:=-x:x:=x` $\{x > 0\}$
- $\{m \leq n\}$ `x := (m+n)/2` $\{m \leq x \leq n\}$

Variáveis `while` e variáveis lógicas

No triplo $\{x + 1 = 30\}$`x:= x + 1`$\{x = 30\}$

- a variável x de $x + 1 = 30$ é uma variável lógica (cujo valor é imutável)
- a variável x de $x := x + 1$ é uma variável informática (representa um espaço mémoria cujo conteúdo pode mudar)
- logo, o x da pre-condição representa um valor que não é o valor do x da pós-condição: o valor presente no espaço memória x da instrução antes da sua execução.

Conceitos de Base

Exemplos

- $\{x = 29\}$ `x:= x + 1` $\{x > 20\}$
- $\{x + 1 = 30\}$ `x:= x + 1` $\{x = 30\}$
- $\{x \neq 0\}$ `?(x<0)x:=-x:x:=x` $\{x > 0\}$
- $\{m \leq n\}$ `x := (m+n)/2` $\{m \leq x \leq n\}$

Triplos precisos

Para um determinado programa S, existem muitos pares (P, Q) tais que $\{P\}\ S\ \{Q\}$. No entanto nem todos esses pares têm interesse. Os triplos $\{P\}\ S\ \{Q\}$ considerados pertinentes são tais que P seja o mais fraco possível e Q o mais forte possível. (A é mais fraco do que B ou B mais forte do que A, se $B \implies A$)

max

$\{ x \in \mathbb{N} \wedge y \in \mathbb{N} \wedge z \in \mathbb{N} \}$ `?(x>y)z:=x:z:=y` $\{ (x > y \implies z = x) \wedge (x \leq y \implies z = y) \}$

ou ainda

$\{ x \in \mathbb{N} \wedge y \in \mathbb{N} \wedge z \in \mathbb{N} \}$ `?(x>y)z:=x:z:=y` $\{ (x \leq z \wedge y \leq z \wedge (\forall t \in \mathbb{N},\ (x \leq t \wedge y \leq t) \implies z \leq t))) \}$

Par Seguinte

$\{ x \in \mathbb{N} \wedge x = x_0 \}$ `?(x mod 2 = 0)x:=x+2:x:=x+1` $\{ x > x_0 \wedge \forall z \in \mathbb{N}, (z > x_0 \wedge ((par\ z)) \implies x \leq z) \}$

Multiplicação

$\{a, b \in \mathbb{N} \wedge b > 0\}$

```
res:=0; i:= 0; while (i!=b) {i:=i+1; res := res + a}
```

$\{res = a \times i \wedge i = b\}$

Maior Elemento

$\{N \in \mathbb{N} \wedge N > 0\}$

```
res:=a[0]; i:=1;
while (i<=N)
   {?(a[i]>res)res:=a[i]:skip; i:=i+1;}
```

$\{N \in \mathbb{N} \wedge N > 0 \wedge \forall x \in \{0 \cdots N\}, res \geq a[x]\}$

Demonstrar $\{P\}\ I_1; I_2; \cdots ; I_n\ \{Q\}$

Uma solução:

Encontrar as condições intermédias $P_1 \cdot P_{n-1}$ tais que de P se possa chegar a Q *passo a passo*.

$$
\begin{array}{l}
\{P\} \\
I_1 \\
\{P_1\} \\
\vdots \\
I_{n-1} \\
\{P_{n-1}\} \\
I_n \\
\{Q\}
\end{array}
$$

Para tal...

É preciso determinar para cada elemento da linguagem como a pré e a pós condição se interligam.

Regras de Inferências

$$\text{w}\ \frac{\vdash R \implies P \quad \{P\}\ S\ \{Q\}}{\{R\}\ S\ \{Q\}} \qquad \text{s}\ \frac{\vdash Q \implies R \quad \{P\}\ S\ \{Q\}}{\{P\}\ S\ \{R\}}$$

$$\text{skip}\ \frac{}{\{P\}\ skip\ \{P\}}$$

$$:=\ \frac{}{\{P(e)\}\ x := e\ \{P(x)\}} \qquad ;\ \frac{\{P\}\ i\ \{R\} \quad \{R\}\ S\ \{Q\}}{\{P\}\ i; S\ \{Q\}}$$

$$?:\ \frac{\{P \wedge e\}\ S_t\ \{Q\} \quad \{P \wedge \neg e\}\ S_e\ \{Q\}}{\{P\}\ ?(e)S_t; S_e\ \{Q\}} \qquad \text{while}\ \frac{\{I \wedge e\}S\{I\}}{\{I\}\ while(e)S\ \{I \wedge \neg e\}}$$

$$\text{while}_{wf}\ \frac{\vdash I \wedge e \implies v \in \mathbb{N} \quad \{I \wedge e \wedge v = v_0\}S\{I \wedge v_0 > v\}}{\{I\}\ while(e)S\ \{I \wedge \neg e\}}$$

Correcção Parcial vs. Correcção Total

Ciclos e Problema da Terminação

Existem duas regras (*while* e $while_{wf}$) para estruturas cíclicas. Porquê?

- A regra *while* não se pronuncia sobre quando a condição *e* deixará de ser verdade $\Longrightarrow$ **Correcção Parcial**: se terminar então o estado final satisfaz a pós-condição
- A regra $while_{wf}$ explicita um valor inteiro que decresce a cada interação: Iteração bem fundada $\Longrightarrow$ **Correcção Total**: a iteração termina e a sua execução leva a um estado final que satisfaz a pós-condição

Demonstração: Nova Perspectiva

Mesmo Objectivo:

$$
\begin{array}{l}
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow
\end{array}
\begin{array}{l}
\{P\} \\
I_1 \\
\{P_1\} \\
\vdots \\
I_{n-1} \\
\{P_{n-1}\} \\
I_n \\
\{Q\}
\end{array}
\downarrow
$$

Mas: Fazer bom uso das duas primeiras regras

Para uma instrução I_j, uma pré-condição P e um pós-condição Q, definir dois predicados A e B tais que se verifiquem:

- $\{A\}\ I_j\ \{B\}$,
- $B \implies Q$ (i.e. regra s),
- $P \implies A$ (i.e. regra w).

Sugere:

1. começar da pós-condição e subir até a pré-condição. De P calcular os sucessivos P_i até ao fim (i.e. até P_n) e demonstrar então que $P_n \implies Q$
2. começar da pré-condição e descer até a pós-condição. De Q calcular os sucessivos Q_i até ao início (i.e. até Q_n) e demonstrar então que $P \implies Q_n$

Demonstração: Nova Perspectiva

Ou seja:

Weakest Pre − condition Calculus ↑

$\{P\}$
Confrontar P *e* P_0
$\{P_0\}$
I_1
$\{Q_1\}$
Confrontar Q_1 *e* P_1
$\{P_1\}$
I_2
$\{Q_2\}$
Confrontar Q_2 *e* P_2
$\{P_2\}$
.
.
.
I_{n-1}
$\{Q_{n-1}\}$
Confrontar Q_{n-1} *e* P_{n-1}
$\{P_{n-1}\}$
I_n
$\{Q_n\}$
Confrontar Q_n *e* Q
$\{Q\}$

↓ *Strongest Post − Condition Calculus*

Complementos: caso do cíclo

Como provar que $\{P\}$ `while(B)S` $\{Q\}$

Descobrir um predicado I chamado *Invariante* e uma medida estritamente descrescente v (o *variante* ver acetato correcção total vs parcial), tais que

$P \implies I$,
$I \wedge \neg B \implies Q$,
$I \wedge B \implies v \in \mathbb{N}$,
$\{I \wedge C \wedge (v = v_0)\} S \{I \wedge (v < v_0)\}$,

Ou seja:

P
$\Downarrow$
I
`while(B)`
$\{I \wedge C \wedge (v = v_0)\}$
`S`
$\{I \wedge (v < v_0)\}$
;
$\{I \wedge \neg C\}$
$\Downarrow$
Q

Desafio $\implies$ encontrar o bom **invariante**

Plano

Automatização do Processo Ascendente

wp : weakest pre-condition calculus (Dijkstra)

$$
\begin{array}{lcl}
wp(x := e,\ Q) & = & Q[x := e] \\
wp(skip,\ Q) & = & Q \\
wp(S; T,\ Q) & = & wp(S, wp(T, Q)) \\
wp(?(B)S : T,\ Q) & = & (B \implies wp(S, Q)) \wedge (\neg B \implies wp(T, Q))) \\
wp(while((B)\{\{I, v\}S\},\ Q) & = & I \wedge (I \wedge B \implies wp(S, I)) \\
 & & \quad (I \wedge \neg B \implies Q) \\
 & & \quad (I \implies v \in \mathbb{N}) \\
 & & \quad (I \wedge B \wedge v = v_0 \implies wp(s, v < v_0))
\end{array}
$$

Plano

Troca

$\{x = x_0, y = y_0\}$
t:=x;
$\{t = x_0, y = y_0\}$
$\Updownarrow$
$\{t = x_0, y = y_0\}$
x:=y;
$\{t = x_0, x = y_0\}$
$\Updownarrow$
$\{t = x_0, x = y_0\}$
y:=t;
$\{x = y_0, y = x_0\}$

$$;\frac{:=\dfrac{}{\{t = x_0, y = y_0\}\ x := y;\ \{t = x_0, x = y_0\}} \qquad :=\dfrac{}{\{t = x_0, x = y_0\}\ y := t;\ \{y = x_0, x = y_0\}}}{\{t = x_0, y = y_0\}\ x := y; y := t; \{y = x_0, x = y_0\}}$$

$$;\frac{:=\dfrac{}{\{x = x_0, y = y_0\}\ t := x;\ \{t = x_0, y = y_0\}} \qquad \{t = x_0, y = y_0\}\ x := y; y := t; \{y = x_0, x = y_0\}}{\{x = x_0, y = y_0\}\ t := x; x := y; y := t;\ \{x = y_0, y = x_0\}}$$

Par Seguinte

$\{(x = x_0) \wedge (x \in \mathbb{N})\}$
$\Updownarrow$
$\{(x = x_0) \wedge (x \in \mathbb{N})\}$
?(x mod 2 = 0)
$\{(x = x_0) \wedge (x \in \mathbb{N}) \wedge (par\ x)\} \implies \{(par\ x + 2) \wedge (x + 2 > x_0)\}$
x:=x+2;
$\{(par\ x) \wedge (x > x_0)\}$
:
$\{(x = x_0) \wedge (x \in \mathbb{N}) \wedge \neg(par\ x)\} \implies \{(par\ x + 1) \wedge (x + 1 > x_0)\}$
x:=x+1;
$\{(par\ x) \wedge (x > x_0)\}$
$\{(par\ x) \wedge (x > x_0)\}$
$\Updownarrow$
$\{(par\ x) \wedge (x > x_0)\}$

Max

$\{(x, y, z \in \mathbb{N}) \wedge (x = x_0) \wedge (y = y_0)\}$
$\Updownarrow$
$\{(x = x_0) \wedge (x \in \mathbb{N})\}$
?(x < y = 0)
$\{(x, y, z \in \mathbb{N}) \wedge (x = x_0) \wedge (y = y_0) \wedge (x > y)\} \implies \{(x_0 > y_0 \implies x = x_0) \wedge (x_0 \leq y_0 \implies z = y_0)\}$
z:=x;
$\{(x_0 > y_0 \implies z = x_0) \wedge (x_0 \leq y_0 \implies z = y_0)\}$
:
$\{(x, y, z \in \mathbb{N}) \wedge (x = x_0) \wedge (y = y_0) \wedge (x \leq y)\} \implies \{(x_0 > y_0 \implies z = x_0) \wedge (x_0 \leq y_0 \implies y = y_0)\}$
z:=y;
$\{(x_0 > y_0 \implies z = x_0) \wedge (x_0 \leq y_0 \implies z = y_0)\}$
$\{(x_0 > y_0 \implies z = x_0) \wedge (x_0 \leq y_0 \implies z = y_0)\}$
$\Updownarrow$
$\{(x_0 > y_0 \implies z = x_0) \wedge (x_0 \leq y_0 \implies z = y_0)\}$

Multiplicação

$\{(a, b \in \mathbb{N}) \wedge (b > 0)\}$
$\Downarrow$
$\{0 = a \times 0\}$
res:=0;
$\{res = a \times 0\}$
$\Updownarrow$
$\{res = a \times 0\}$
i:= 0; $\{res = a \times i\}$
$\Updownarrow$
$\{res = a \times i\}$
while (i!=b) {
$\{(res = a \times i) \wedge (i \neq b)\}$
$\Downarrow$
$\{(res + a = a \times (i + 1))\}$
i:=i+1;
$\{(res + a = a \times i)\}$
$\Updownarrow$
$\{(res + a = a \times i)\}$
res := res + a;
$\{(res = a \times i)\}$
}
$\{(res = a \times i) \wedge (\neg i \neq b)\}$
$\Downarrow$
$\{(res = a \times b)\}$

Pesquisa em Tabela (J.F. Monin)

$\{(p, q \in \mathbb{N}) \wedge (p \leq q) \wedge (P \subseteq \{p \cdots q-1\})\}$
x:=p;y:=q;
while (x!=y)
$\{p \leq x < q\}$
?(P x)y:=x; : x:=x+1;
$\{x \in \mathbb{N} \wedge (p \leq x \leq q) \wedge (x < q \implies (P\ x))$
$\wedge((x = q) \implies (\forall i \in \mathbb{N}, p \leq i < q \implies \neg(P\ i)))\}$

variante: $y - x$
Invariante: $I_1 \wedge I_2 \wedge I_3$
onde
$I_1 = x, y \in \mathbb{N} \wedge p \leq x \leq y \leq q$ Domínio de x e de y
$I_2 = \forall i \in \mathbb{N}, p \leq i < x \implies (P\ i)$
$I_3 = y < q \implies (P\ x)$

Plano

O conceito de Base

- Permitir que o programador comente o seu código com anotações/asserções de comporamento.
- Caso simples: uma anotação no início e uma no fim.
- Caso elaborado: todas as anotações formando a prova em lógica de Hoare
- é preciso saber encontrar o bom nível de detalhe. Em geral, o caso simples + invariantes (i.e. anotações nos pontos difíceis do programa).
- estas anotações podem ser vistas como um resumo da demonstração de correcção do programa

Anotação e Correcção

Dis-se que um programa é anotado correcto quando as asserções que contém são demonstráveis (na lógica de Hoare). Duma certa forma quando o resumo descrto pelas asserções corresponde realmente a uma demonstração. Ou seja, quando:

1. De cada vez que um triplo $\{P\}\ S\ \{Q\}$ aparece no programa anotado, existe uma demonstração em lógica de Hoare;
2. Cada vez que duas asserções $\{P\}$ e $\{Q\}$ são seguem (são contíguas) no programa anotado, então temos $P \implies Q$

Asim podemos mostrar que qualquer programa anotado correcto corresponde a uma demonstração em Lógica de Hoare (e reciprocamente)

anotações + wp

- escrever um programa contendo anotações em pontos estratégicos (pré e pós condições gerais mais invariantes nos ciclos e anotações em pontos difíceis do programa);
- transmitir o código anotado a uma ferramenta verifica a correcção das anotações. Por exemplo a uma ferramenta que gera todas fórmulas $P \implies wp(S, Q)$ para cada triplo $\{P\}\ S\ \{Q\}$ (Chamadas **Condições de Verificação**);
- Demonstrar estas condições de verificação.

Plano

Anotações, porquê?

- Conceito na base de muitos métodos formais
- Caso de sucesso: linguagem EIFFEL
- Linguagens de anotações: JML para o Java (http://www.cs.iastate.edu/~leavens/JML) , *Spec*$\sharp$
- Como verificar por computador as anotações?
  1. em tempo de execução (asserções como um mecanismo particular de excepções). Veja por exemplo o mecanismo `assert` presente em várias linguagens (C, OCaml incluido)
  2. automaticamente: Model Checking, Analizadores estáticos (ESC/JAVA, etc...).
  3. demonstração assistida: Jack, Caduceus/Krakatoa/Why, LoopTool, KeyTool