INF05516 - Semântica formal N
Ciência da Computação - UFRGS
2006-2

Marcus Ritt
mrpritt@inf.ufrgs.br

02/10/2006

Agenda

Última aula:

- Prova 1.

Hoje:

- Introdução à semântica axiomática.

Um exemplo

- No exercício 2.4 provamos que

```
y := 1
while ¬(x=0) do (
  y := y × x;
  x := x − 1
)
```

  e uma implementação correta para $x!$, se $x \geq 0$.
- Se $x < 0$ o programa não termina: $x \geq 0$ é uma condição necessário.
- $x \geq 0$ se chama *pré-condição*.
- O objetivo do programa é calcular o fatorial. Na prova obtemos a *pós-condição* $y = n!$ se n foi o valor inicial de x.
- A pos-condição $y = x!$ é falso: x tem o valor 0 depois da execução!

Introdução

- Em geral, um programa é *correto* se, dada uma pré-condição Φ e uma pós-condição Ψ, para cada estado inicial, que satisfaz a pré-condição, o programa
  - para e
  - o estado final satisfaz a pós-condição.
- A primeira característica se chama *terminação* a segunda *corretude parcial*.
- Na prática a corretude parcial e a questão da terminação são mais fácil de tratar separado. Obtemos e "equação"

  $$\text{Corretude parcial} + \text{Terminação} = \text{Corretude total}$$

- Para um programa $\mathbf{c}$ com pré-condição Φ a pós-condição Ψ escrevemos
  - $\{\Phi\}\mathbf{c}\{\Psi\}$ se $\mathbf{c}$ é parcialmente correto.
  - $\{\Phi\}\mathbf{c}\{\Downarrow \Psi\}$ se $\mathbf{c}$ é totalmente correto.
- Essa notação se chama tripla de Hoare.

Exemplo

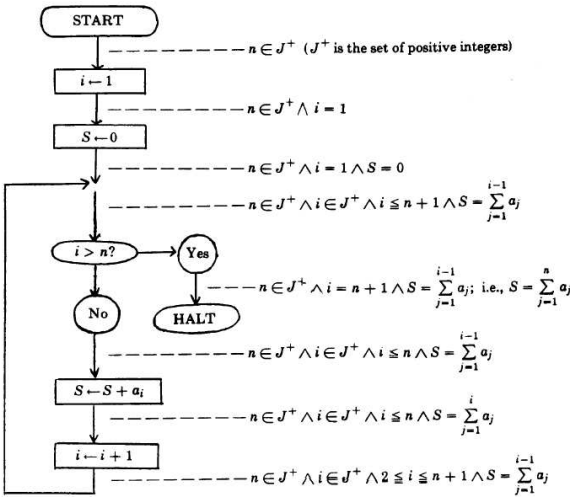
- Seja $\sigma(x) > 0$

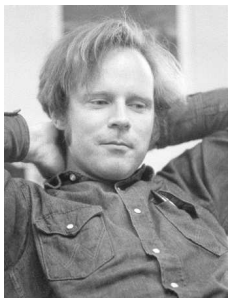
```
C ≡
while ¬(x=1) do (
  y:=2; z:=1;
  while y < x do ( y:=y+2; z:=z+1 );
  if (y = x) then
    x := z
  else
    x := 3*x+1
)
```

- Intuitivamente, se o programa termina, temos $\sigma(x) = 1$, $\sigma(y) = 2$ e $\sigma(z) = 1$.
- Logo $\{x > 0\}C\{x = 1 \wedge y = 2 \wedge z = 1\}$.
- E $\{x > 0\}C\{\Downarrow x = 1 \wedge y = 2 \wedge z = 1\}$?

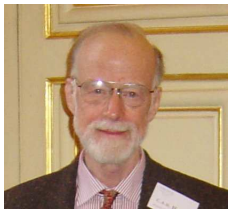
Semântica axiomática

- A *semântica axiomática* consiste de:
  - uma linguagem para especificar pré-condições e pós-condições.
  - regras para provar a corretude.
- Floyd inventou a semântica axiomática para fluxogramas [1].
- Hoare inventou a notação atual [2] e a abordagem para linguagems com procedimentos.





Robert W Floyd (*1936 +2001)



Charles Antony Richard Hoare (*1934)
(Foto de David Monniaux)

Semântica axiomática

- A semântica axiomática é mais abstrato que outras semânticas.
  - Ela só considera as características necessárias para provar a corretude de uma especificação.
  - Assim é mais fácil de raciocinar sobre programas.
- O objetivo é
  - de analisar uma descrição informal de um tarefa;
  - produzir uma especificação formal baseado em uma lógica adequada
  - e provar que uma implementação satisfaz a sua especificação formal.

Linguagem

- Qual seria uma linguagem adequada para as asserções?
- Usando IMP, precisamos condições
  - sobre expressões booleanas: usa lógica propositional
  - sobre expressões aritméticas: usa variáveis para números inteiros, predicados como $=$ e $<$, funções como $+$, $-$ e $\times$ e quantores.
- Essentialmente, chegamos na lógica de predicados (primeira ordem) sobre os números naturais, usando um conjunto de variáveis V com $\mathbf{v} \in V$:

$$\mathbf{a} ::= \mathbf{n}|\mathbf{x}|\mathbf{v}|\mathbf{a}+\mathbf{a}'|\mathbf{a}-\mathbf{a}'|\mathbf{a}\times\mathbf{a}'$$
$$\mathbf{A} ::= \mathbf{t}|\mathbf{a}=\mathbf{a}'|\mathbf{a}<\mathbf{a}'|\neg\mathbf{b}|\mathbf{b}\vee\mathbf{b}'|\mathbf{b}\wedge\mathbf{b}'|\mathbf{b}\rightarrow\mathbf{b}'|\forall\mathbf{v}\mathbf{A}|\exists\mathbf{v}\mathbf{A}$$

Interpretações

- O que significa a expressão $i < j$, com $i \in V, j \in V$?
- O valor depende dos valores de i e j: Para atribuir um valor de verdade, precisamos uma *interpretação* $I \in [V \to \mathbb{Z}]$.
- O que significa a expressão $x = 2$, com $x \in \mathrm{Ident}$?
- O valor depende do estado σ!
- Usamos o seguinte notação: Uma asserção A é correto num estado σ usando a interpretação I

$$\sigma \models^I A$$

- Uma asserção de corretude parcial é correto

$$\sigma \models^I \{\Phi\}\mathbf{c}\{\Psi\}$$

se e somente se

$$\sigma \models^I \Phi \wedge \mathbf{c}, \sigma \Downarrow \sigma' \rightarrow \sigma' \models^I \Psi.$$

Interpretações...

- Em geral, nosso interesse é uma tripla (especificação!) que está válido em qualquer estado e interpretação.
- Se $\forall \sigma \in \Sigma : \forall I \in [V \to \mathbb{Z}] : \sigma \models^I \{\Phi\}\mathbf{c}\{\Psi\}$ é correto, escrevemos

$$\models \{\Phi\}\mathbf{c}\{\Psi\}.$$

(lê: $\{\Phi\}\mathbf{P}\{\Psi\}$ é (semânticamente) correto.)

Regras 13 / 25

Motivação

- Com nossa definição obtemos $\models \{x > 0 \wedge x = n\}\texttt{C}\{y = n!\}$ para o exemplo da introdução.
- A prova foi feito usando a semântica operacional.
- Na semântica axiomática queremos abstrair da semântica concreta para simplificar o raciocínio sobre programas.
- Essa abstração usa *regras de prova*.

Substituição

- Asserções como $\forall x P(x)$ e $\forall y P(y)$ tem o mesmo significado.
- Se $\forall x : P(x)$, então $P(3)$ se substituimos 3 para x em P.
- Em geral, escrevemos $A[\mathbf{a}/\mathbf{v}]$ para a substituição da expressão aritmética $\mathbf{a}$ para a variável $\mathbf{v}$.
- Substitui com cautela
  - Não pega variáveis livres

$$(\forall i(i = j))[i/j] \qquad \forall i(i = i)$$
$$(\forall i(i = j))[i/j] \qquad \forall k(k = i)$$

  - Não substitua variáveis ligadas

$$\forall i(i > 0)[j/i] \qquad \forall i(j > 0)$$

Regras para corretude parcial

$$\frac{}{\{\Phi\}\texttt{skip}\{\Phi\}}\ \text{skip}$$

$$\frac{}{\{\Phi[a/x]\}\mathbf{x}\texttt{:=}\mathbf{a}\{\Phi\}}\ \text{assign}$$

$$\frac{\{\Phi\}\mathbf{c_1}\{\chi\} \quad \{\chi\}\mathbf{c_2}\{\Psi\}}{\{\Phi\}\mathbf{c_1}\texttt{;}\mathbf{c_2}\{\Psi\}}\ \text{seq}$$

$$\frac{\{\Phi \wedge b\}\mathbf{c_1}\{\Psi\} \quad \{\Phi \wedge \neg b\}\mathbf{c_2}\{\Psi\}}{\{\Phi\}\texttt{if }\mathbf{b}\texttt{ then }\mathbf{c_1}\texttt{ else }\mathbf{c_2}\{\Psi\}}\ \text{if}$$

$$\frac{\{\Phi \wedge b\}\mathbf{c}\{\Phi\}}{\{\Phi\}\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}\{\Phi \wedge \neg b\}}\ \text{while}$$

$$\frac{\vdash \Phi \to \Phi' \quad \{\Phi'\}\mathbf{c}\{\Psi'\} \quad \vdash \Psi' \to \Psi}{\{\Phi\}\mathbf{c}\{\Psi\}}\ \text{impl}$$

Exemplo 1

$$\{\text{true}\}\texttt{if x>0 then skip else x:=0-x}\{\neg(x<0)\}?$$

$$\dfrac{\dfrac{}{\{x\geq 0\}\texttt{skip}\{x\geq 0\}}\,\text{skip}\quad\dfrac{x<0\rightarrow x\leq 0\quad\dfrac{}{\{x\leq 0\}\texttt{x:=0-x}\{x\geq 0\}}\,\text{assign}}{\{x<0\}\texttt{x:=0-x}\{x\geq 0\}}\,\text{impl}}{\{\text{true}\}\texttt{if } (x\geq 0)\texttt{ then skip else x:=0-x}\{x\geq 0\}}\,\text{if}$$

Exemplo

Queremos provar o exemplo do exercício 2.4 usando as regras de prova:

$$\{x=n\wedge x>0\}\texttt{y:=1;while } \neg\texttt{(x=0) do (y:=y*x;x:=x-1)}\{y=n!\}?$$

$$\dfrac{\{x=n\wedge 1=1\}\texttt{y:=1}\{x=n\wedge y=1\}\qquad\dfrac{x=n\wedge y=1\rightarrow\dfrac{\dfrac{\dfrac{}{\{yx!=n!\wedge x\neq 0\}\texttt{y:=y*x}\{\,\}\{y(x-1)!=n!\}\texttt{x:=x-1}\{yx!=n\}}\,\text{seq}}{\{yx!=n!\wedge x\neq 0\}\texttt{y:=y*x;x:=x-1}\{yx!=n\}}\,\text{while}}{\{yx!=n!\}\texttt{while } \neg\texttt{(x=0) do } \ldots\{yx!=n!\wedge x=0\}}\rightarrow y=n!}\,\text{impl}}{\{x=n\wedge y=1\}\texttt{while } \neg\texttt{(x=0) do (y:=y*x;x:=x-1)}\{y=n!\}}\,\text{seq}}{\{x=n\}\texttt{y:=1;while } \neg\texttt{(x=0) do (y:=y*x;x:=x-1)}\{y=n!\}}$$

Aqui fica claro, que essa notação não é ótimo!

Trabalhar com sequências

Ao invès árvores de prova vamos user o seguinte notação (que se chama *tableau*): Para um programa $\mathbf{c_1;c_2;\cdots;c_n}$ a regra "seq" justifica de escrever

$$\begin{array}{c}\{\Phi\}\\ \mathbf{c_1}\\ \{\chi_1\}\\ \mathbf{c_2}\\ \cdots\\ \mathbf{c_{n-1}}\\ \{\chi_{n-1}\}\\ \mathbf{c_n}\\ \{\Psi\}\end{array}$$

se conseguimos de justificar as *condições intermediárias* χ_i, $1 \leq i < n$.

Trabalhar com consequências

A regra "impl" se chama também *regra da consequência*. Ela pode ser escrita

$$\begin{array}{c}\{\Phi\}\\ \{\Phi'\}\\ \mathbf{c}\\ \{\Psi'\}\\ \{\Psi\}\end{array}$$

Trabalhar com a atribuição

- Consequentemente escrevemos

$$\{\Phi[a/x]\}$$
$$\texttt{l:=a}$$
$$\{\Phi\}$$

  para a atribuição.

Trabalhar com a atribuição...

- Porque a regra de atribuição tem a forma

$$\frac{}{\{\Phi[a/x]\}\mathbf{x:=a}\{\Phi\}}\,\text{assign}$$ e não $$\frac{}{\{\Phi\}\mathbf{x:=a}\{\Phi[a/x]\}}\,\text{assign?}$$

- A aplicação das duas regras resulta em

$$\{5=5\}\texttt{x:=5}\{x=5\}$$
$$\{x=6\}\texttt{x:=5}\{5=6\}\,!$$

- Trabalhar com a atribuição é mais fácil de baixo para cima.
- Logo vamos tratar uma sequência de baixo para cima, começando com a pos-condição.

Exemplo

Entrada x, y
Saida x, y

```
h := x;
x := y;
y := h
```

Especificação?

$$\{x = x_0 \wedge y = y_0\}\texttt{h:=x;x:=y;y:=x}\{x = y_0 \wedge y = x_0\}$$

.

Exemplo...

Prova?

$$\{y = y_0 \wedge x = x_0\}$$
$$\texttt{h:=x;}$$
$$\{y = y_0 \wedge h = x_0\}$$
$$\texttt{x:=y;}$$
$$\{x = y_0 \wedge h = x_0\}$$
$$\texttt{y:=h}$$
$$\{x = y_0 \wedge y = x_0\}$$

Referências

[1] Richard W Floyd. Assigning meaning to programs. In *Proceedings AMS Symposia in Applied Mathematics, Mathematical aspects in Computer Science*, volume 19, pages 19–31, 1967.

[2] Charles Antony Richard Hoare. An axiomatic basis of computer programming. *Communications of the ACM*, 12(10):576–580, 1969.