Lógica de Floyd-Hoare

- Método axiomático para provar que determinados programas são corretos.
- Introduzido por Charles Antony Richard Hoare e Robert Floyd na década de 1960.
- Também utilizado para especificar a semântica (axiomática) de linguagens de programação.
- Por que devemos especificar e verificar código?
    - Para redução de falhas que são introduzidas durante a especificação do sistema e, assim, reduzir o seu custo final;
    - Código bem especificado e verificado é mais fácil de reutilizar, uma vez que temos um claro entendimento do que o sistema deve fazer;
    - Para sistemas críticos (aviões, usinas nucleares, ...) isto se torna muito mais importante, pois se espera que os sistemas possam sofrer auditorias sobre o quanto eles garantem que irão funcionar conforme desejado;
    - A especificação de um sistema auxilia na documentação do mesmo e, feita de uma maneira formal, auxilia no processo de implementação final do sistema;
    - Atualmente empresas enfrentam problemas com sistemas legados, pois os responsáveis pelo seu desenvolvimento não os documentam de uma maneira precisa e, na maioria das vezes, não se encontram mais na empresa;
    - Sistemas de software têm um tempo de vida muitas vezes maior do que as pessoas – o *bug* do milênio (ano 2000) é apenas um exemplo.

- Abordagem que usaremos:
    - ➔ aplicar o método de Floyd-Hoare sobre uma linguagem de programação imperativa simplificada.

[Dados de Entrada] → [Programa] → [Dados de Saída]

Triplas de Hoare

`(|P|) C (|Q|)` , onde:

- P e Q são asserções (fórmulas na lógica de predicados de primeira ordem com igualdade);
- P : precondição (asserções definidas sobre o estado inicial do programa, que denotam sob quais restrições o programa está preparado para funcionar);
- Q : pós-condição (asserções sobre o estado final do programa, que denotam condições sobre os dados resultantes do programa);
- C : comandos de uma linguagem de programação.

Exemplo:

```
(|x > 4|)   x:= x + 1   (|x > 5|)
```

Exemplo 1:

Problema: dados dois valores de entrada sobre o domínio dos números inteiros maiores ou iguais a 10, o programa deverá calcular a soma desses dois valores.

Especificação:
- sejam x o primeiro valor de entrada e y o segundo valor de entrada.
- seja z o valor calculado pelo programa.

Solução (A) - OK
precondição: (| (x ≥ 10) ∧ (y ≥ 10) |)
pós-condição: (| (z = x + y) |)

Solução (B) – Exemplo de super-especificação
precondição: (| (x ≥ 10) ∧ (y ≥ 10) ∧ ((x+y) ≥ 0) desnecessário |)
pós-condição: (| (z = x + y) ∧ (z ≥ 20) desnecessário |)

Solução (C) – Exemplo de especificação desnecessariamente complexa
precondição: (| (x ≥ 10) ∧ ((x+y) ≥ (10+x)) não é claro |)
pós-condição: (| (z = x + y) |)

Solução (D) – Exemplo de especificação incompleta
precondição: (| (x ≥ 10) ???? falta algo! |)
pós-condição: (| (z = x + y) |)

Exemplo 2:

Problema: dado um número inteiro, calcular o fatorial do número de forma iterativa (não recursiva).

Especificação:
- x: o número (entrada)

- fat: o resultado do programa

Solução:

precondição: (| (x ≥ 0) ∧ (temp = x) |)
pós-condição: (| (fat = temp!) |)

Exemplo de possível programa em Java:

```
int fatorial (int x){
     int fat = 1;
     while (x != 1){
          fat = fat * x;
          x = x - 1;
     }
     return fat;
}
```

Note que x será alterado no programa, portanto reservamos uma cópia lógica em temp.

Exercícios:

1) Formular uma especificação para um programa que, dados dois números naturais, deverá calcular o máximo divisor comum para eles (considere uma função matemática MDC).

   x e y: os dois valores de entrada
   mdc: o resultado do programa
   precondição: (| (x > 0) ∧ (y > 0) |)
   pós-condição: (| (mdc = MDC(x,y)) |)

2) Formular uma especificação para um programa que, dados dois números inteiros de entrada, a soma desses dois números no estado inicial deverá ser idêntica à soma deles no estado final do programa. Note que o programa poderá eventualmente modificar os valores das variáveis de entrada.

   x e y: os números de entrada
   S: variável para garantir o que foi especificado
   precondição: (| (S = x + y) |)
   pós-condição: (| (S = x + y) |)

Linguagem de Programação Simplificada

- Paradigma: Linguagem do tipo imperativa
- Tipos de dados: int e bool
- Comandos: atribuição (:=), sequência (;), condição (if), laço (while)

SINTAXE (BNF):

Expressões aritméticas:

```
E ::= n | x | E+E | E-E | E*E | -E
```

Onde n é um número inteiro, x é qualquer variável

Expressões lógicas:

```
B ::= true | false | !B | B&&B | B||B |
      E<E | E==E | E!=E
```

Comandos:

```
C ::= x := E |
      C ; C |
      if B {C} |
      if B {C} else {C} |
      while B {C}
```

Exercício:

1) Dê alguns exemplos de programas utilizando a sintaxe definida.

Cálculo de Floyd-Hoare

Para cada comando possível em um programa teremos uma regra que define sua "semântica axiomática".
Além disso, teremos regras extras de consequências, chamadas de PreForte e PosFraco.
O cálculo de demonstração a ser utilizado em uma demonstração $\vdash_{parcial} (|\phi_0|)C(|\phi_n|)$ será composto por uma sequência entremeando fórmulas e códigos de programa. A cada fórmula escreve-se uma justificativa.

$(|\phi_0|)$
C_1
$(|\phi_1|)$ justificativa
C_2
...
$(|\phi_{n-1}|)$ justificativa
C_n
$(|\phi_n|)$ justificativa

Como encontrar as fórmulas intermediárias? É mais conveniente começar a demonstração de trás para frente! Procure refazer os exemplos apresentados usando essa técnica.

O Cálculo de Floyd-Hoare, na versão aqui apresentada, irá obter a chamada "pré-condição mais fraca". Essa condição é a "mais fraca" no sentido de que qualquer outra pré-condição possível implicará nela.

Exemplo:

$(| \; y \leq 4 \; |) \quad x := y+1 \quad (| \; x \leq 5 \; |)$

$y \leq 4 \wedge z = 0 \Rightarrow y \leq 4$

ATRIBUIÇÃO

$(| \quad P \quad |) \quad x := 4 \quad (| \; y > x \; |)$ (Qual fórmula deve ser P ?)

$(| \; y > 4 \; |) \quad x := 4 \quad (| \; y > x \; |)$

Constrói-se a precondição a partir da pós-condição:

$$\frac{}{(| \; Q[x/t] \; |) \; x:=t \; (| \; Q \; |)} \; \text{Atribuição}$$

Exemplo:

$(| \quad ????? \quad |) \quad x := x + 1 \quad (| \; (x < 5) \; |)$

3. $(| \quad x < 4 \quad |) \quad x := x + 1 \quad (| \; (x < 5) \; |)$
2. $(| \; x+1 < 5 \quad |) \quad x := x + 1 \quad (| \; (x < 5) \; |)$
1. $(| \; (x < 5)_{[x/(x+1)]} \; |) \quad x := x + 1 \quad (| \; (x < 5) \; |)$ Atrib

CONSEQUÊNCIA (PreForte e PosFraco)

Algumas vezes, as pré e pós condições que obtemos pelas regras do Cálculo de Hoare podem não ser exatamente a que desejamos. Elas podem ser logicamente equivalentes, mas ter uma forma sintática diferente ou podem ser logicamente mais fracas (no caso de pré-condições) ou logicamente mais fortes (no caso de pós-condições).

Exemplo:
Verifique o que acontece quando obtemos uma precondição por *Atrib* que não corresponde exatamente ao que precisávamos para uma dada prova...

Provar que o seguinte programa está correto:

$(| \quad x < 3 \quad |) \quad x:= x + 1 \quad (| \; (x < 5) \; |)$

Resolução:

$(| \; x+1 < 5 \quad |) \quad x:= x + 1 \quad (| \; (x < 5) \; |)$

$(| \; (x < 5)_{[x/(x+1)]} \; |) \quad x:= x + 1 \quad (| \; (x < 5) \; |)$ Atrib

Repare que queríamos provar x<3, porém obtivemos x+1<5 depois de aplicar *Atrib*. Então temos que usar outra regra:

$$\frac{P \Rightarrow Q,\ (|Q|)\ C\ (|R|),\ R \Rightarrow S}{(|P|)\ C\ (|S|)}\ \text{Consequência}$$

Observe que esta regra pode ser separada em duas (uma para o fortalecimento da pré-condição e outra para o enfraquecimento da pós-condição):

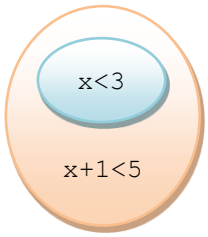
$$\frac{P \Rightarrow Q,\ (|Q|)\ C\ (|R|)}{(|P|)\ C\ (|R|)}\ \text{PreForte}$$

$$\frac{(|P|)\ C\ (|Q|),\ Q \Rightarrow R}{(|P|)\ C\ (|R|)}\ \text{PosFraco}$$

Veja o exemplo de aplicação da regra de fortalecimento da pré-condição. Agora temos que provar que:

$(x<3) \Rightarrow (x+1<5)$

... ou seja, que o primeiro é um caso especial do segundo (o segundo é uma asserção mais fraca, portanto).



Observe que a asserção obtida pela aplicação da regra do Cálculo de Hoare para a Atribuição é uma asserção mais fraca. A aplicação da regra da Consequência para Fortalecimento da pré-condição é que permite obtermos a prova desejada.

<u>Exemplo completo com o uso de Atribuição + PreForte</u>:

Prove: $(|x < 3|)\ \ x := x + 1\ \ (|x < 5|)$

4.	$(\|x < 3\|)$	
3.	$(\|x + 1 < 5\|)$	**PreForte**
2.	$x := x + 1$	

1.	(|x < 5|)	**Atribuição**

Observe que a implicação (x<3)⇒(x+1<5) não será provada, pois estamos interessados na estrutura do Cálculo de Hoare e não em provas de teorias aritméticas. Assumimos que um provador de teoremas automatizado será capaz de construir tais provas (por exemplo, em um sistema automatizado como Dafny).

Exercício:

1) Prove: (|x < 5|) x := x + 1 (|x < 7|)

COMPOSIÇÃO

Agora vamos ver como combinar dois comandos em sequência:

$$\frac{(|P|)\,C1\,(|Q|),\ (|Q|)\,C2\,(|R|)}{(|P|)\ C1;C2\ (|R|)}\ \text{Composição}$$

A regra Composição permite a sequência de comandos. Determinamos a asserção intermediária (|Q|) a partir da pós-condição.

Exemplo:
Determinar a precondição:

```
(|     ?     |)   x := x + 1 ; y := y + x   (| y > 6 |)

3. (| y + x + 1 > 6 |)  x := x + 1 ; y := y + x  (| y > 6 |)   Comp 1,2
2. (| y + x + 1 > 6 |)   x := x + 1  (| y + x > 6 |)  Atrib (em C1)
1. (| y + x > 6 |)   y := y + x  (| y > 6 |)          Atrib (em C2)
```

Exemplo completo com o uso de Atribuição + PreForte + Composicao:

Prove: (|x > 2 ∧ y > 3|) x := x + 1 ; y := y + x (|y > 6|)

	(|x>2 ∧ y>3|)	
	(|y+x+1 > 6|)	**PreForte**
	x := x + 1	
	(|y+x > 6|)	**Atribuição**
	y := y + x	
	(|y > 6|)	**Atribuição**

Procure descrever informalmente por que a implicação indicada é válida. Ou seja, a condição de prova que será necessária é provar x>2 ∧ y>3 ⇒ y+x+1>6.

Exercício:

1) Prove: (|a > 0 ∧ b > 0|) x := a ; y := b (|x+y > 0|)

2) Prove: `(|a > b|)  x := -a ; y := -b  (|x < y|)`

3) Os dois programas a seguir realizam a troca de valores entre suas variáveis. Mostre que ambos estão corretos.

a) Prove: `(|x < y|)  temp := x ; x := y ; y := temp  (|y < x|)`

b) Prove: `(|x < y|)  y := y + x ; x := y - x ; y := y - x  (|y < x|)`

CONDICIONAL

Com relação aos condicionais, vamos primeiro lidar com a versão mais simples. Nesta regra, a primeira premissa trata do caso em que a expressão booleana *B* é avaliada como *true*. A segunda premissa, portanto, leva em conta o caso em que *B* é *false*.

$$\frac{(|P \wedge B|)\,C\,(|Q|),\quad P \wedge \neg B \Rightarrow Q}{(|P|)\ \text{if } B\ \{C\}\ (|Q|)}\ \text{Condicional1}$$

Exemplo:

`(|T|)  if x<0 {x := -x}  (| x ≥ 0 |)`

As premissas para a regra, então, seriam (devemos provar as duas):

a) `(|T ∧ x<0|)  x := -x  (| x ≥ 0 |)`

b) `(| (T ∧ ¬(x<0)) ⇒ (x ≥ 0) |)`

Observe que queremos calcular a condição mais fraca ϕ tal que:

`(|`ϕ`|) if B {C} (|Q|)`

Esta ϕ pode ser calculada da seguinte maneira:

a) Empurre `Q` para cima através de `C`; vamos chamar o resultado de ϕ_1.

b) Defina ϕ como $(B \Rightarrow \phi_1) \wedge (\neg B \Rightarrow Q)$.

Exemplo completo com o uso de Atribuição + PreForte + Condicional:

Prove: `(|T|)  if x<0 {x := -x}  (| x ≥ 0 |)`

	`(|T|)`	
	`(| x<0 ⇒ -x≥0 ∧ ¬(x<0) ⇒ x≥0 |)`	**PreForte**
	`if (x<0)`	
	`}`	
	`(| -x ≥ 0 |)`	**Condicional1**
	`x := -x`	
	`(| x ≥ 0 |)`	**Atribuição**
	`}`	

	(| x ≥ 0 |)	**Condicional1**

Procure descrever informalmente porquê a implicação indicada é válida.

Exercícios:

1) Prove: (|x < 10|) if x >= 5 {x := 4} (|x < 5|)
2) Prove: (| T |) if x < y {x := y} (|x ≥ y|)
3) Prove: (| T |) if x > y {x := y+1; y := x+1} (|x ≤ y|)

Agora vamos ver a regra para o condicional completo (com *else*).

$$\frac{(|P \wedge B|)\,C1\,(|Q|),\ (|P \wedge \neg B|)\,C2\,(|Q|)}{(|P|)\ \text{if } B \text{ } \{C1\} \text{ else } \{C2\}\ (|Q|)}\ \text{Condicional2}$$

Observe que queremos calcular a condição mais fraca ϕ tal que:

(|ϕ|) if B {C1} else {C2} (|Q|)

Esta ϕ pode ser calculada da seguinte maneira:

a) Empurre Q para cima através de C1; vamos chamar o resultado de ϕ_1.
b) Empurre Q para cima através de C2; vamos chamar o resultado de ϕ_2.
b) Defina ϕ como $(B \Rightarrow \phi_1) \wedge (\neg B \Rightarrow \phi_2)$.

Exemplo completo com o uso de Atribuição + PreForte + Condicional:

Prove:

```
(|T|)
a:=x+1;
if (a-1 == 0) {
  y:=1;
} else {
  Y:=a;
}
(| y = x+1 |)
```

	(|T|)	
	(| x+1-1=0 ⇒ 1=x+1 ∧ ¬(x+1-1=0) ⇒ x+1=x+1 |)	**PreForte**
	a := x+1;	
	(| a-1=0 ⇒ 1=x+1 ∧ ¬(a-1=0) ⇒ a=x+1 |)	**Atribuição**
	if (a-1 == 0) {	
	(| 1 = x + 1 |)	**Condicional2**
	y := 1;	
	(| y = x + 1 |)	**Atribuição**
	}	
	(| a = x + 1 |)	**Condicional2**
	y := a;	

	(| y = x + 1 |)	**Atribuição**
	}	
	(| y = x + 1 |)	**Condicional2**

Procure descrever informalmente porquê a implicação indicada é válida.

Exercícios:

1) Prove:

```
(|T|)
if (x<y){
  max := y
}else{
  max := x
}
(| max ≥ x ∧ max ≥ y |)
```

2) Prove:

```
(|T|)
if (x<y){
  y := y-1
}else{
  x := -x;
  y := -y
}
(| x ≤ y |)
```