Insper

# Lógica da Computação - 2020/1

Aula 26/T14 - 03/Jun/2020

Raul Ikeda - rauligs@insper.edu.br

**Objetivos**

1. Verificação de Programas

**Especificação de Programas**

Frequentemente nos deparamos com pedaços de código que foram produzidos por terceiros. Seja uma biblioteca ou algo que algum colega criou e que você precisa utilizar no seu programa. Como você consegue dizer se o programa reutilizado faz realmente o que precisa ser feito?
Normalmente utiliza-se um sistema de **validação** realizando testes com diferentes entradas e verificando a saída do programa. Contudo esse método não verifica realmente se o programa está correto para casos mais complexos e não determinísticos. Para realizar uma **verificação** de fato, é necessário:

- Uma especificação precisa e não ambígua.
- Uma implementação em uma linguagem muito bem definida e também sem ambiguidades.
- Uma forma de traduzir a especificação na linguagem de programação utilizada.
- Um sistema de provas para demonstrar que o programa realmente satisfaz as especificações.

**Linguagem de Programação como Lógica de Primeira Ordem**

```
>> Ver Corrêa da Silva et al. Cap. 6.3.
```

Considere agora que um programa de computador é composto por sequências de comandos que levam de um **estado inicial** a um **estado final**. A especificação do programa deve portanto explicar quais são essas transformações, especialmente no âmbito dos estados envolvidos. Analise as seguintes definições abaixo:

**Precondições**: restrições acerca do estado inicial do programa, tanto sob ponto de vista dos dados de entrada, quanto valores iniciais das variáveis.
**Póscondições**: discorrem sobre o estado final do programa. Basicamente o que o programa precisa realmente entregar na saída.
**Invariantes**: são asserções que devem ser sempre válidas ao longo da duração do programa. Como consequência, essas asserções devem ser válidas tanto no estado inicial, quanto no estado final.

**Exemplo**: “Dado um número inteiro n, o programa FAT deverá calcular o fatorial de n.”

```
>> Ver Corrêa da Silva et al. Cap. 6.3.2.
```

**Lógica de Hoare**

Considere a Tripla de Hoare:

$$\langle\varphi\rangle Prog\langle\psi\rangle$$

Onde:

- $\varphi$ são as pré-condições definidas pela especificação do problema.
- $Prog$ é o programa implementado.
- $\psi$ são as pós-condições que representam o estado final após as pré-condições passarem pelo programa.

A ideia é demonstrar que partindo das pré-condições, após passar pelo programa, as pós condições são satisfeitas, sem precisar definir alguma entrada ou realizar testes unitários. Para tanto, devemos traduzir as instruções de uma linguagem de programação em um sistema que permite realizar a prova. Também é preciso assumir que $Prog$ termina.

O sistema de provas deve realizar a prova por partes de tal forma que, dado um programa $P$:

$C_0$; #Comando 0
$C_1$;
...
$C_n$;

No qual desejamos provar que $\vdash \langle\varphi\rangle P\langle\psi\rangle$, pode ser interpretado como:

$$\langle\varphi\rangle C_0; \langle\varphi_1\rangle C_1; \langle\varphi_2\rangle ... \langle\varphi_n\rangle C_n; \langle\psi\rangle$$

E agora precisaríamos provar individualmente:

$$\vdash \langle\varphi\rangle C_0 \langle\varphi_1\rangle$$
$$\vdash \langle\varphi_1\rangle C_1 \langle\varphi_2\rangle$$
$$...$$
$$\vdash \langle\varphi_n\rangle C_n \langle\psi\rangle$$

**Exemplo 1 - Atribuição: Verifique o programa abaixo que, dado um número natural, ele retorna o próximo número no domínio dos inteiros.**

```
a = x + 1;
```

**Exemplo 2 - Condicional: Verifique o programa abaixo que, dado um número inteiro, ele retorna o próximo número no domínio dos naturais.**

```
a = x + 1;
if (a < 0)
  num = 0;
else
  num = a;
```

**Exemplo 3 - Loop: Verifique se o programa abaixo calcula o fatorial de um número inteiro n.**

```
fat = 1;
i = 0;
while (i != n)
{
  i = i + 1;
  fat = fat * i;
}
```

**Lista de Exercícios**

- Corrêa da Silva et al. Exercícios: 7.2, 7.3, 7.4, 7.5 e 7.6

**Próxima aula:**