Inatel

M020 – Matemática Discreta

1. LÓGICA FORMAL

1.6 Demonstração de Correção

Marcelo Vinícius Cysneiros Aragão

marcelovca90@inatel.br

Introdução

- À medida que nossa sociedade torna-se mais dependente de computadores, é cada vez mais importante que os programas para computador sejam confiáveis e sem erros.
- A **verificação do programa** tenta garantir que o programa de computador está correto.
- “Correção” nesse contexto tem uma definição um pouco diferente da utilizada no nosso dia-a-dia: um programa está **correto** se ele se comporta de acordo com suas especificações.

Introdução

- No entanto, isso não significa, necessariamente, que o programa resolve o problema que pretendia resolver; as especificações do programa podem não ser compatíveis ou não prever todos os aspectos das necessidades do cliente.
- A **validação do programa**, que não discutiremos, tenta garantir que o programa, de fato, atende às necessidades originais do cliente.
- Em um projeto grande de desenvolvimento de um programa, “programa V & V” ou “garantia de qualidade do programa” é considerado tão importante que, muitas vezes, um grupo de pessoas diferentes dos programadores fica encarregado dessas tarefas.

Introdução

- A verificação de um programa pode ser abordada das seguintes formas:
  - **Testes de programa**
  - **Demonstração de correção**

Testes de Programa e Demonstração de Correção

- Os **testes de programa** tentam mostrar que valores particulares de dados de entrada geram respostas aceitáveis.
- Os testes de programa formam uma parte importante de qualquer esforço de desenvolvimento, mas é um fato bem conhecido que “os testes podem provar a existência de erros, nunca sua ausência”.

Testes de Programa e Demonstração de Correção

- Se um teste, executado sob um certo conjunto de condições e com um determinado de conjunto de dados de entrada, revelar um *bug* no código, esse erro pode ser corrigido.
- Mas, exceto para programas bem simples, testes múltiplos que não detectam erros não garantem que o programa não os tenha, que não exista algum *bug* escondido no meio do código esperando a hora certa para atacar.

Testes de Programa e Demonstração de Correção

- Complementar aos testes, cientistas de computação desenvolveram uma abordagem mais matemática para “provar” que um programa está correto.
- A **demonstração de correção** usa técnicas de um sistema de lógica formal para provar que, se as variáveis de entrada satisfazem certos predicados ou propriedades especificadas, então as variáveis de saída, produzidas pela execução do programa, satisfazem outras propriedades especificadas.

Testes de Programa e Demonstração de Correção

- Para distinguir entre demonstração de correção e testes de programa, considere um programa para calcular o comprimento c da hipotenusa de um triângulo retângulo dados os valores positivos a e b para o comprimento dos lados.
  - A demonstração da correção do programa estabeleceria que, sempre que a e b satisfizerem as propriedades $a > 0$ e $b > 0$, então, após a execução do programa, o predicado $a^2 + b^2 = c^2$ seria satisfeito.
  - Testar um tal programa corresponderia a escolher diversos valores particulares para a e b, calcular o resultado c e verificar a igualdade de $a^2 + b^2$ com c^2 em cada caso. No entanto, podem ser testados apenas valores representativos de a e b e não todos os valores possíveis.

Testes de Programa e Demonstração de Correção

- Novamente, testes e demonstração de correção são **aspectos complementares** da verificação de um programa.
- Idealmente, todos os programas passam por testes; entretanto, eles podem, ou não, passar, também, por uma demonstração de correção.
- A demonstração de correção é usada, em geral, apenas em seções pequenas e críticas do código e não no programa inteiro.

Asserções

- Para descrever a demonstração de correção mais formalmente, vamos denotar por X uma coleção arbitrária de valores de entrada para algum programa, ou segmento de programa, P.
- As ações de P transformam X em um grupo correspondente de valores de saída Y; a notação $Y = P(X)$ sugere que os valores de Y dependem de X através das ações do programa P.

Asserções

- Um predicado $Q(X)$ descreve as condições que os valores de entrada devem satisfazer.
- Por exemplo, se um programa deve encontrar a raiz quadrada de um número positivo, então X consiste em um valor de entrada, x, e $Q(X)$ pode ser "$x > 0$".
- Um predicado R descreve as condições que os valores de saída devem satisfazer. Essas condições muitas vezes envolvem, também, os valores de entrada, de modo que R tem a forma $R(X, Y)$ ou $R[X, P(X)]$.

Asserções

- No nosso caso da raiz quadrada, se y é o único valor de saída, então y deve ser a raiz quadrada de x, de modo que $R(x, y)$ poderia ser "$y^2 = x$".
- O programa P está correto se o condicional

$$(\forall X)(Q(X) \rightarrow R[X, P(X)])$$

for válido. Em outras palavras, sempre que Q for verdadeiro com os valores de entrada, R é verdadeiro com os valores de entrada e saída.

Asserções

- Para o caso da raiz quadrada, o condicional anterior é

$$(\forall x)(x > 0 \rightarrow [P(x)]^2 = x)$$

- O condicional dado está em uma notação de fbf predicada padrão, mas a notação tradicional usada na correção do programa é $\{Q\}P\{R\}$.

Asserções

- $\{Q\}P\{R\}$ é chamada de uma **tripla de Hoare**, em homenagem ao cientista de computação inglês Anthony Hoare.
- A condição Q é chamada a **precondição** para o **programa** P e a condição R é a **pós-condição**.
- Na notação de Hoare, o quantificador universal não aparece explicitamente; fica subentendido.

Asserções

- Ao invés de ter apenas um predicado inicial e um final, um programa, ou segmento de programa, é dividido em proposições individuais s_i, com predicados inseridos entre proposições, além do início e do fim.
- Esses predicados são, também, chamados de **asserções**, pois afirmam o que deve ser verdadeiro sobre as variáveis do programa naquele ponto.

Asserções

- Temos, então,

$$\{Q\}, s_0, \{R_1\}, s_1, \{R_2\}, \ldots, s_{n-1}, \{R\}$$

onde $Q, R_1, R_2, \ldots, R_n = R$ são asserções.

- As asserções intermediárias são obtidas, muitas vezes, de trás para frente, a partir de R.

Axioma de Atribuição

- Suponha que uma proposição s_i é uma declaração de atribuição da forma $x = e$, isto é, a variável x toma o valor e, onde e é uma determinada expressão.
- A tripla de Hoare para demonstrar a correção dessa proposição tem a forma

$$\{R_i\}\, x = e\, \{R_{i+1}\}$$

- Para que isso seja válido, as asserções de R_i e R_{i+1} têm que estar relacionadas de uma forma especial.

Exemplo 41

Considere as seguintes declarações de atribuição, junto com a precondição e a pós-condição dadas:

$\{x - 1 > 0\}$
$x = x - 1$
$\{x > 0\}$

Para todo x, se $x - 1 > 0$ antes da execução da proposição (note que isso significa que $x > 1$), então, após a redução do valor de x por 1, teremos $x > 0$. Portanto,

$$\{x - 1 > 0\}\ x = x - 1\ \{x > 0\}$$

é válida. ❖

Axioma de Atribuição

- No exemplo 41, raciocinamos até obter um caminho para verificar a validade de uma fbf representada por uma tripla de Hoare.
- O ponto da lógica de predicados é permitir a determinação da validade de um modo mais mecânico, através da aplicação de regras de inferência.
- Afinal de contas, não queremos “raciocinar até obter um caminho” analisando um programa inteiro até nos convencer que está correto; o programador já fez isso ao escrever o programa!

Axioma de Atribuição

- A regra de inferência apropriada para declarações de atribuição é o **axioma de atribuição**:

De	Podemos Deduzir	Nome da Regra	Restrições sobre o Uso
	$\{R_i\}s_i\{R_{i+1}\}$	Axioma de atribuição	1. s_i tem a forma $x = e$. 2. R_i é R_{i+1} com x substituído por e em todos os lugares.

- Ele diz que, se a precondição e a pós-condição estão relacionadas de forma apropriada, então a tripla de Hoare pode ser inserida em qualquer lugar em uma sequência de demonstração sem que tenha que ser inferida por alguma etapa anterior na sequência.
- Isso faz com que a tripla de Hoare, para uma declaração de atribuição, tenha um papel semelhante a uma hipótese nas nossas demonstrações anteriores.

Exemplo 42

Para o caso do Exemplo 41,

$\{x - 1 > 0\}$
$\quad x = x - 1$
$\{x > 0\}$

1. s_i tem a forma $x = e$.
2. R_i é R_{i+1} com x substituído por e em todos os lugares.

a tripla

$$\{x - 1 > 0\}\ x = x - 1\ \{x > 0\}$$

é válida pelo axioma de atribuição. A pós-condição é

$$x > 0$$

Substituindo x por $x - 1$ na pós-condição, obtemos

$$x - 1 > 0 \qquad \text{ou} \qquad x > 1$$

que é a precondição. Aqui, não tivemos que pensar; apenas verificamos que o axioma de atribuição foi seguido. ❖

1. s_i tem a forma $x = e$.
2. R_i é R_{i+1} com x substituído por e em todos os lugares.

Problema Prático 31

De acordo com o axioma de atribuição, qual deveria ser a precondição no segmento de programa a seguir?

{precondição}
$x = x - 2$
$\{x = y\}$ ❖

Lembrete: para usar o axioma da atribuição, trabalhe de baixo para cima.

31. $x - 2 = y$ ou $x = y + 2$.

Axioma de Atribuição

- Como o axioma de atribuição nos diz que forma deve ter uma precondição baseado na forma pós-condição, a demonstração de correção começa, frequentemente, com a pós-condição desejada no final e trabalha de trás para frente (ou de baixo para cima), considerando qual deve ser a forma das asserções anteriores de acordo com o axioma de atribuição.
- Uma vez determinada qual deve ser a primeira asserção, verifica-se, então, se essa é, realmente, uma asserção verdadeira.

1. s_i tem a forma $x = e$.
2. R_i é R_{i+1} com x substituído por e em todos os lugares.

Exemplo 43

Verifique a correção do segmento de programa a seguir, que troca os valores de x e y:

temp = x
$x = y$
y = temp

No início desse segmento de programa, x e y têm certos valores. Assim, podemos expressar essa pré-condição de fato como sendo $x = a$ e $y = b$. A pós-condição desejada é, então, $x = b$ e $y = a$. Usando o axioma de atribuição, podemos começar nossa análise pela pós-condição para encontrar as asserções anteriores (leia de baixo para cima).

$\{y = b, x = a\}$
temp = x
$\{y = b,$ temp $= a\}$
$x = y$
$\{x = b,$ temp $= a\}$
y = temp
$\{x = b, y = a\}$

A primeira asserção está de acordo com a precondição; o axioma de atribuição, aplicado repetidamente, nos garante que o segmento de programa está correto. ❖

Problema Prático 32

Verifique a correção do segmento de programa a seguir, com a precondição e a pós-condição dadas:

$\{x = 3\}$
$y = 4$
$z = x + y$
$\{z = 7\}$

32. Trabalhando de trás para a frente a partir da pós-condição e usando o axioma de atribuição, temos

$\{x + 4 = 7\}$
$y = 4$
$\{x + y = 7\}$
$z = x + y$
$\{z = 7\}$

A primeira atribuição, $x + 4 = 7$, é equivalente à pré-condição $x = 3$. O axioma de atribuição, aplicado duas vezes, prova que o segmento de programa está correto.

Axioma de Atribuição

- Algumas vezes a precondição necessária é trivialmente verdadeira, como ilustrada no próximo exemplo.

1. s_i tem a forma $x = e$.
2. R_i é R_{i+1} com x substituído por e em todos os lugares.

Exemplo 44

Verifique a correção do segmento de programa a seguir, que calcula $y = x - 4$.

$y = x$
$y = y - 4$

Aqui a pós-condição desejada é $y = x - 4$. Usando o axioma de atribuição a partir da pós-condição, obtemos (novamente, leia de baixo para cima)

$\{x - 4 = x - 4\}$
$y = x$
$\{y - 4 = x - 4\}$
$y = y - 4$
$\{y = x - 4\}$

A precondição é sempre verdadeira; portanto, pelo axioma de atribuição, cada asserção sucessiva, incluindo a pós-condição, é verdadeira. ❖

A Regra Condicional

- Uma **proposição** ou **declaração condicional** é uma proposição, em um programa, da forma

if condição B **then**

P_1

else

P_2

end if

- Quando essa proposição é executada, a condição B, que é verdadeira ou falsa, é calculada. Se B for verdadeira, o segmento P_1 é executado, mas, se B for falsa, o segmento P_2 é executado.

A Regra Condicional

- A regra condicional de inferência, mostrada na tabela abaixo, determina quando uma tripla de Hoare

$$\{Q\}s_i\{R\}$$

pode ser inserida em uma sequência de demonstração, onde s_i é uma declaração condicional.

- A tripla de Hoare é inferida de duas outras triplas de Hoare.
  - Uma dela diz que se Q e B forem verdadeiras e o segmento de programa P_1 é executado, então R é válida;
  - A outra diz que se Q é verdadeira, B é falsa e P_2 é executado, então R é válida.
- Isso diz que tem que se demonstrar a correção de cada ramificação da declaração condicional.

De	Podemos Deduzir	Nome da Regra	Restrições sobre o Uso

$\{Q \wedge B\}\ P_1\ \{R\}$
$\{Q \wedge B'\}\ P_2\ \{R\}$
	$\{Q\}\ s_i\ \{R\}$	Condicional	s_i tem a forma
			if condição B **then**
			P_1
			else
			P_2
			end if

Exemplo 45

De	Podemos Deduzir	Restrições sobre o Uso

$\{Q \wedge B\}\ P_1\ \{R\}$
$\{Q \wedge B'\}\ P_2\ \{R\}$
	$\{Q\}\ s_i\ \{R\}$	s_i tem a forma
		if condição B **then**
		P_1
		else
		P_2
		end if

Verifique a correção do segmento de programa a seguir, com as precondição e pós-condição dadas.

$\{n = 5\}$
if $n >= 10$ **then**
$y = 100$
else
$y = n + 1$
end if
$\{y = 6\}$

A precondição aqui é $n = 5$ e a condição B a ser avaliada é $n \geq 10$. Para aplicar a regra condicional, precisamos provar, primeiro, que

$$\{n = 5 \text{ e } n \geq 10\}\ y = 100\ \{y = 6\}$$

é válida. Lembre-se de que isso é um condicional que será verdadeiro, pois seu antecedente, $n = 5$ e $n \geq 10$, é falso. Também precisamos mostrar que

Exemplo 45

Também precisamos mostrar que

$$\{n = 5 \text{ e } n < 10\}\ y = n + 1\ \{y = 6\}$$

é válida. Partindo da pós-condição e usando o axioma de atribuição, obtemos

$\{n + 1 = 6 \text{ ou } n = 5\}$
$y = n + 1$
$\{y = 6\}$

Logo,

$$\{n = 5\}\ y = n + 1\ \{y = 6\}$$

é verdadeira pelo axioma de atribuição e, portanto,

$$\{n = 5 \text{ e } n < 10\}\ y = n + 1\ \{y = 6\}$$

também é verdadeira, já que a condição $n < 10$ não adiciona nada de novo na asserção. A regra condicional nos permite concluir que o segmento de programa está correto. ❖

Problema Prático 33

De	Podemos Deduzir	Restrições sobre o Uso

$\{Q \wedge B\}\ P_1\ \{R\}$
$\{Q \wedge B'\}\ P_2\ \{R\}$
	$\{Q\}\ s_i\ \{R\}$	s_i tem a forma
		if condição B **then**
		P_1
		else
		P_2
		end if

Verifique a correção do segmento de programa a seguir com a precondição e a pós-condição dadas:

$\{x = 4\}$
if $x < 5$ **then**
$y = x - 1$
else
$y = 7$
end if
$\{y = 3\}$

33. Pelo axioma de atribuição,

$\{x = 4\}\ y = x - 1\ \{y = 3\}$

é verdade e, portanto,

$\{x = 4 \text{ e } x < 5\}\ y = x - 1\ \{y = 3\}$

também é verdade. Além disso,

$\{x = 4 \text{ e } x \geq 5\}\ y = 7\ \{y = 3\}$

é verdade porque o antecedente é falso. O segmento de programa está correto pela regra condicional de inferência.

Exemplo 46

De	Podemos Deduzir	Restrições sobre o Uso

$\{Q \wedge B\}\ P_1\ \{R\}$
$\{Q \wedge B'\}\ P_2\ \{R\}$
	$\{Q\}\ s_i\ \{R\}$	s_i tem a forma
		if condição B **then**
		P_1
		else
		P_2
		end if

Verifique a correção do segmento de programa a seguir, que calcula o valor máximo, max(x, y), entre dois valores distintos x e y.

$\{x \neq y\}$
if $x >= y$ **then**
max $= x$
else
max $= y$
end if

A pós-condição desejada reflete a definição do máximo, ($x > y$ e max $= x$) ou ($x < y$ e max $= y$). Os dois condicionais a serem demonstrados são

$$\{x \neq y \text{ e } x \geq y\} \text{ max} = x \{(x > y \text{ e max} = x) \text{ ou } (x < y \text{ e max} = y)\}$$

e

$$\{x \neq y \text{ e } x < y\} \text{ max} = y \{(x > y \text{ e max} = x) \text{ ou } (x < y \text{ e max} = y)\}$$

Exemplo 46

De	Podemos Deduzir	Restrições sobre o Uso

$\{Q \wedge B\}\ P_1\ \{R\}$
$\{Q \wedge B'\}\ P_2\ \{R\}$
	$\{Q\}\ s_i\ \{R\}$	s_i tem a forma
		if condição B **then**
		P_1
		else
		P_2
		end if

Usando o axioma de atribuição no primeiro caso (substituindo max na pós-condição por x) nos daria a pre-condição

$$(x > y \wedge x = x) \vee (x < y \wedge x = y)$$

Como a segunda disjunção é sempre falsa, isso é equivalente a

$$(x > y \wedge x = x)$$

que, por sua vez, é equivalente a

$$x > y \qquad \text{ou} \qquad x \neq y \text{ e } x \geq y$$

O segundo condicional é demonstrado de maneira análoga.

A Regra Condicional

- Como vimos, a demonstração de correção envolve muitos detalhes.
- É uma ferramenta difícil de aplicar para programas grandes já existentes. Em geral, é mais fácil provar a correção enquanto o programa está sendo desenvolvido.
- De fato, a lista de asserções, do princípio ao fim, especificam o comportamento desejado do programa e podem ser usadas cedo no projeto.
- Além disso, as asserções servem como documentação valiosa depois do programa completo.

Exercício 1

Obs.: Nos Exercícios 2, 3, 5 e 8, * denota multiplicação.

★1. De acordo com o axioma de atribuição, qual é a precondição para o segmento de programa a seguir?

{precondição}
$x = x + 1$
$\{x = y - 1\}$

*1. x + 1 = y - 1, or x = y - 2

Exercício 2

2. De acordo com o axioma de atribuição, qual é a precondição para o segmento de programa a seguir?

{precondição}
$x = 2 * x$
$\{x > y\}$

2. 2x > y, or x > y/2

Exercício 3

3. Verifique a correção do segmento de programa a seguir com a precondição e a pós-condição indicadas.

$\{x = 1\}$
$y = x + 3$
$y = 2 * y$
$\{y = 8\}$

3. Working backwards from the postcondition using the assignment rule,

$\{x + 3 = 4\}$
$y = x + 3$
$\{2y = 8 \quad \text{or} \quad y = 4\}$
$y = 2*y$
$\{y = 8\}$

The first assertion, $x + 3 = 4$, is equivalent to the precondition $x = 1$. The assignment rule, applied twice, proves the program segment correct.

Exercício 4

4. Verifique a correção do segmento de programa a seguir com a precondição e a pós-condição indicadas.

$\{x > 0\}$
$y = x + 2$
$z = y + 1$
$\{z > 3\}$

4. Working backwards from the postcondition using the assignment rule,

$\{x + 2 > 2\}$
$y = x + 2$
$\{y + 1 > 3 \quad \text{or} \quad y > 2\}$
$z = y + 1$
$\{z > 3\}$

The first assertion, $x + 2 > 2$, is equivalent to the precondition $x > 0$. The assignment rule, applied twice, proves the program segment correct.

Exercício 5

***5.** Verifique a correção do segmento de programa a seguir que calcula $y = x(x - 1)$.

$y = x - 1$
$y = x * y$

5. A pós-condição desejada é $y = x(x - 1)$. Trabalhando a partir da pós-condição, usando o axioma de atribuição, obtemos

$\{x(x - 1) = x(x - 1)\}$
$y = x - 1$
$\{xy = x(x - 1)\}$
$y = x * y$
$\{y = x(x - 1)\}$

Como a precondição é sempre verdadeira, cada proposição subseqüente também é verdadeira, incluindo a pós-condição.

Exercício 6

6. Verifique a correção do segmento de programa a seguir que calcula $y = 2x + 1$.

$y = x$
$y = y + y$
$y = y + 1$

6. The desired postcondition is $y = 2x + 1$. Working back from the postcondition, using the assignment rule, gives

$\{x + x = 2x\}$
$y = x$
$\{y + y = 2x\}$
$y = y + y$
$\{y + 1 = 2x + 1 \text{ or } y = 2x\}$
$y = y + 1$
$\{y = 2x + 1\}$

Because the precondition is always true, so is each subsequent assertion, including the postcondition.

Exercício 7

De	Podemos Deduzir	Restrições sobre o Uso

$\{Q \wedge B\}\ P_1\ \{R\}$
$\{Q \wedge B'\}\ P_2\ \{R\}$
	$\{Q\}\ s_i\ \{R\}$	s_i tem a forma
		if condição B **then**
		P_1
		else
		P_2
		end if

★7. Verifique a correção do segmento de programa a seguir com a precondição e a pós-condição indicadas.

$\{y = 0\}$
if $y < 5$ **then**
$y = y + 1$
else
$y = 5$
end if
$\{y = 1\}$

7. Os dois condicionais que têm que ser provados são

$\{y = 0 \text{ e } y < 5\}\ y = y + 1\ \{y = 1\}$

e

$\{y = 0 \text{ e } y \geq 5\}\ y = 5\ \{y = 1\}$

O primeiro condicional é válido porque

$\{y = 0\}\ y = y + 1\ \{y = 1\}$

é verdadeiro pelo axioma de atribuição, e o segundo é verdadeiro porque o antecedente é falso. O segmento de programa está correto pela regra condicional de inferência.

Exercício 8

8. Verifique a correção do segmento de programa a seguir com a precondição e a pós-condição indicadas.

$\{x = 7\}$
if $x <= 0$ **then**
  $y = x$
else
  $y = 2 * x$
end if
$\{y = 14\}$

8. The two implications to prove are

$\{x = 7 \text{ and } x \leq 0\}\ y = x\ \{y = 14\}$
and
$\{x = 7 \text{ and } x > 0\}\ y = 2*x\ \{y = 14\}$

The first is true because the antecedent is false, and the second is true because

$\{x = 7\}\ y = 2*x\ \{y = 14\}$

holds by the assignment rule. The program segment is correct by the conditional rule.

Exercício 9

9. Verifique a correção do seguinte segmento de programa, que calcula o mínimo, min(x, y), entre dois valores distintos x e y.

$\{x \neq y\}$
if $x <= y$ **then**
  min = x
else
  min = y
end if

9. The desired postcondition follows from the definition of minimum: (x < y and min = x) or (x > y and min = y). The two implications to prove are

{x ≠ y and x ≤ y} min = x {(x < y and min = x) or (x > y and min = y)}
{x ≠ y and x > y} min = y {(x < y and min = x) or (x > y and min = y)}

Using the assignment rule on the first implication gives the precondition

(x < y and x = x) or (x > y and x = y)

Because the second disjunct is false, this is equivalent to (x < y and x = x) or (x < y) or (x ≠ y and x ≤ y).

Using the assignment rule on the second implication gives the precondition

(x < y and y = x) or (x > y and y = y)

which is equivalent to (x > y) or (x ≠ y and x > y).

The program segment is correct by the conditional rule.

Exercício 10

10. Verifique a correção do seguinte segmento de programa, que calcula o valor absoluto de x, $|x|$, de um número não-nulo x.

$\{x \neq 0\}$
if $x >= 0$ **then**
  abs $= x$
else
  abs $= -x$
end if

10. The desired postcondition follows from the definition of absolute value for a nonzero number: (x > 0 and abs = x) or (x < 0 and abs = -x). The two implications to prove are

{x ≠ 0 and x ≥ 0} abs = x {(x > 0 and abs = x) or (x < 0 and abs = -x)}
{x ≠ 0 and x < 0} abs = -x {(x > 0 and abs = x) or (x < 0 and abs = -x)}

Using the assignment rule on the first implication gives the precondition

(x > 0 and x = x) or (x < 0 and x = -x)

Because the second disjunct is false, this is equivalent to (x > 0 and x = x) or (x > 0) or (x ≠ 0 and x ≥ 0).

Using the assignment rule on the second implication gives the precondition

(x > 0 and -x = x) or (x < 0 and -x = -x)

which is equivalent to (x < 0) or (x ≠ 0 and x < 0).

The program segment is correct by the conditional rule.

Exercício 11

11. Verifique a correção do segmento de programa a seguir, com as asserções dadas.

$\{z = 3\}$
$x = z + 1$
$y = x + 2$
$\{y = 6\}$
if $y > 0$ **then**
$z = y + 1$
else
$z = 2 * y$
end if
$\{z = 7\}$

11. For the top section of the program, we can work backwards from the postcondition using the assignment rule twice:

$\{z + 1 = 4 \text{ or } z = 3\}$
$x = z + 1;$
$\{x + 2 = 6 \text{ or } x = 4\}$
$y = x + 2;$
$\{y = 6\}$

which agrees with the given precondition. For the bottom section of the program, we use the conditional rule to prove

$\{y = 6\}$
if $(y > 0)$
$z = y + 1;$
else
$z = 2*y;$
$\{z = 7\}$

One implication is

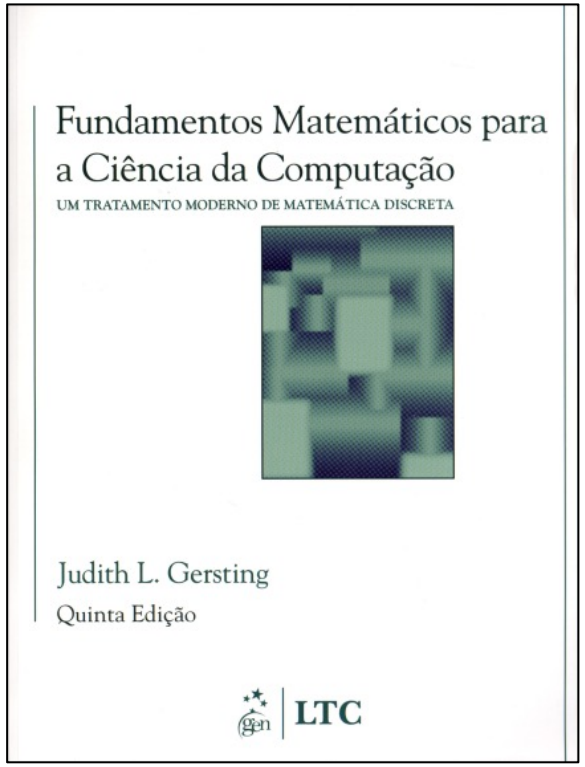
$\{y = 6 \wedge y > 0\}\ z = y + 1\ \{z = 7\}$

which is true by the assignment rule. The other implication is

$\{y = 6 \wedge y \leq 0\}\ z = 2*y\ \{z = 7\}$

which is true because the antecedent is false.

Referência Bibliográfica



GERSTING, Judith L.; IÓRIO, Valéria de Magalhães, Fundamentos matemáticos para a ciência da computação: um tratamento moderno de matemática discreta. 5 ed. Rio de Janeiro, RJ: LTC, 2004, 597 p. ISBN 978-85-216-1422-7.