Necessidade de prova de correção?

Sabemos que o processo algoritmico de resolução de problemas envolve os seguintes passos básicos:

1. Entender o problema a ser resolvido
2. Criar um algoritmo para resolver o problema
3. Codificar o algoritmo em alguma linguagem de programação
4. Testar o código usando alguns casos típicos para os dados de entrada

Se o passo 4 indicar erros, então provavelmente o algoritmo não é adequado e devemos voltar ao passo 2 para refazer o algoritmo. No caso de algoritmos longos e complicados, voltar ao passo 2 pode significar perda de tempo e desperdicio de recursos preciosos quando foram executados os passos 2, 3 e 4.

Pior, note que mesmo se os testes do passo 4 não indicarem erros, ainda *não temos garantia* de que o algoritmo esteja correto. E os outros casos de teste que não foram tentados? Nada podemos dizer sobre eles. A ausência de erros no passo 4 apenas aumenta nossa confiança de que o algoritmo esteja realmente correto, mas *não garante* essa propriedade de forma irretorquível.

Seria possível, antes de embarcar nos passos 3 e 4, obtermos uma garantia de que o algoritmo está correto, i.e., que ele resolve adequadamente o problema para *todas* as instâncias *válidas* dos dados de entrada?

A resposta é "sim", mas a um certo custo. Para concretizar isso, ou seja obter uma *garantia* de correção doalgoritmo, a única saída viável seria apresentar uma *prova formal*, como a prova de um teorema em matemática, de que o algoritmo realmente funciona.

Dificuldades

Para apresentar um argumento de correção realmente formal e à prova de dúvidas, seria necessário:

1. Precisão e rigor ao descrever o *texto* do algoritmo. Precisaríamos usar alguma linguagem formal para isso, como a linguagem da matemática.
2. Precisão e rigor ao descrever o *comportamento* do algoritmo.

   Não confundir com o item anterior. O item 1 fala do texto estático do algoritmo, i.e., do programa que o descreve. Esse item fala do que acontece dinamicamente quando o algoritmo executa.
3. Precisão e rigor para descrever os *dados* de entrada e de saída

Temos como alcançar essas metas? A resposta é positiva, com muitas variações para acomodar os inúmeros tipos de programas que se pode escrever. É uma tarefa difícil, que só deve ser tentada para trechos críticos do código.

Por partes:

1. O texto do programa, corretamente escrito numa linguagem de programação, já é uma expressão precisa e formal. Teremos mais experiência sobre isso quando descrevermos a sintaxe de linguagens de programação em detalhes.

2. Para o comportamento do algoritmo, devemos especificar uma semântica adequada para cada instrução executável da linguagem de programação. A semântica descreve, num modelo matemático, qual o resultado de se executar cada instrução básica da linguagem. Um pouco mais de detalhes sobre isso logo adiante.
3. Para os dados, normalmente usa-se algum tipo de linguagem lógica, ou linguagem de especificação de dados. Uma expressão lógica diz, precisamente, quais conjuntos de dados de entrada são válidos. E outra expressão lógica diz que tipo de propriedade os dados de saída devem satisfazer.

Vamos colocar esses itens todos juntos. Seja X uma sequência de valores para as variáveis de entrada de um algoritmo ALG, que desenvolvemos. Seja EE(X) a expressão lógica que valida os dados de entrada em função de X. Isto é, se EE(X) resultar em "verdadeiro", então os valores armazenados nas variáveis da sequência X representam dados de entrada válidos. Seja Y uma sequ6encia de valores para as variáveis de saída do algoritmo e seja ES(Y) a expressão lógica que verifica se os dados de saída realmente são uma solução do problema. Finalmente, seja A(.) a função que descreve o comportamento do algoritmo, i.e. A(X) representa a sequência de valores de saída do algoritmo quando este executa sobre valores de entrada dados na sequência X. Então a correção do algoritmo corresponderia a provarmos um teorema da forma:

```
[Se EE(X) é "verdadeiro"] e [Z=A(X)]
então ES(Z) é "verdadeiro"
```

Para exemplificar, e sem maiores cuidados, considere um programa que usa N e X como variáveis de entrada, Y como variável auxiliar, e R como variável de saída. Todas as variáveis armazenarão apenas números inteiros.

A expressão EE(N,X) = [(N≥0) e (X ∈ Z)] indicaria que N é inteiro não negativo e X é um inteiro qualquer (aqui o símbolo Z é usado para designar o conjunto dos inteiros). Assim, EE(3, 5) resulta em "verdadeiro". Já E(-2,8) resultaria em "falso".

A expressão ES(R) =[(R=2k) e (k ∈ Z)] indica que R é um número inteiro par. Então ES(5) resulta em "falso".

A semântica das instruções da linguagem poderia ser especificada através dos efeitos de cada instrução sobre os conteúdos das variáveis. No caso, todas as variáveis usadas pelo programa são N, X, Y e R. Então formamos a sequência (n,x,y,r), onde as letras minúsculas indicam os conteúdos das variáveis. Nesse caso, temos uma quádrupla de números inteiros. Uma instrução simples tal como

```
X = 3
```

causaria uma transformação da memória na forma,

$$(n,x,y,z) \longrightarrow (n,3,y,z)$$

refletindo a atribuição para X. Outras instruções básicas mais complicadas teriam efeitos mais complexos, mas a idéia é a mesma.

Para verificar a correção do programa, teríamos que mostrar que,

Se [EE(n,x) é "verdadeiro"] e [(n,x,y,r) $\underrightarrow{\text{ação}}$ (n', x', y', r')],

então [ES(r') é "verdadeiro"]

onde a seta marcada com "ação" indica a seguinte assertiva AA:

"se o programa P for iniciado com as variáveis contendo os valores indicados por (n,x,y,z), e se P termina, então, no final da execução de P, as variáveis conterão os valores indicados por (n',x', y', r') ".

Não precisaremos entrar em maiores detalhes sobre provas de correção de programas. O assunto é vasto e detalhes estão fora de escopo dessas notas introdutórias.

Terminação

Sabemos que uma das propriedades fundamentais de um algoritmo é a terminação. Ou seja, quando iniciado com um conjunto de dados de entrada válidos, o algoritmo sempre deve terminar sua execução.

Por outro lado, note que a afirmativa AA, acima, evita cuidadosamente a questão da terminação. Só podemos afirmar a conclusão *se* o programa terminar. Nada podemos concluir a respeito da terminação. De fato, a assertiva não garante nada sobre a terminação. O programa pode, muito bem, jamais terminar para qualquer conjunto de dados de entrada, inclusive os válidos. Um programa desse tipo seria inútil e não seria um algoritmo legítimo. No entanto, para esse programa, a assertiva AA seria (vacuamente) válida (!) uma vez que não é jamais falsificada.

Para evitar essas situações, é importante exibir também uma prova de terminação para o programa. Quando a prova de terminação é incluída, diz-se que temos uma prova de *correção total* do programa. Quando a prova de terminação não é incluída, temos uma prova de *correção parcial* do programa.

Construir uma prova de terminação, porém, é uma tarefa bem mais complexa que obter uma prova de correção parcial. Geralmente, envolve uma prova indutiva de alguma espécie.

No caso do algoritmo para calcular o máximo divisor comum de dois inteiros positivos, visto anteriormente, construímos uma prova indutiva de terminação.

Não entraremos em maiores detalhes.

Outras dificuldades

Em casos mais ambiciosos, e mais reais, surgem ainda outras dificuldades. Por exemplo, o algoritmo pode lidar diretamente com a noção de tempo real, i.e., tempo físico. Um caso típico seria de uma instrução básica na forma

"se sensor 17 indicar pelo menos 3,15 graus dentro do intervalo de tempo dos próximos 5 segundos, atue sobre o sensor 3"

Note que a instrução faz referência ao tempo real de 5 segundos e de 3 segundos. Esse tipo de instrução deve ser expressa numa linguagem de programação especializada, que contenha uma variável cujo conteúdo seria o tempo decorrido.

Ou então podemos estar criando programas que naturalmente não param. Tal é o caso de sistemas operacionais e de uma multitude de programas que executam sob controle desses sistemas operacionais.

Ainda uma outra possibilidade, surgida com o advento de linguagens de programação voltadas para objetos (tais como C#, C++ e Java), seria a presença de instruções primitivas importantes que lidam com esses objetos. Precisaríamos especificar a semântica, i.e. o efeito, da execução dessas instuções, o que seria uma tarefa bem mais complexa.

Em todos esses casos, as provas de correção tornam-se consideravelmente mais elaboradas.