4
Heurísticas para a Aplicação do Cálculo de Hoare Automaticamente

4.1
Introdução

No sistema dedutivo da lógica clássica chamado de dedução natural [46] é possível normalizar uma prova via aplicação de sucessivas reduções. O sistema de dedução natural é composto por regras de eliminação e de introdução para cada conectivo e quantificador. Mais ainda, não possui axiomas, somente regras de inferência, e se utiliza do mecanismo de descarga de hipótese, que é muito utilizado informalmente em provas de teoremas na prática matemática [30, 31].

Na dedução natural, a prova normal é de tal forma que, havendo regras de eliminação, todas ocorrem antes de qualquer regra de introdução no ramo, e, havendo regras de introdução, todas ocorrem depois de qualquer regra de eliminação. Assim, a forma dos ramos em uma prova normal é sempre como ilustrada na figura abaixo [30, 31, 39]:

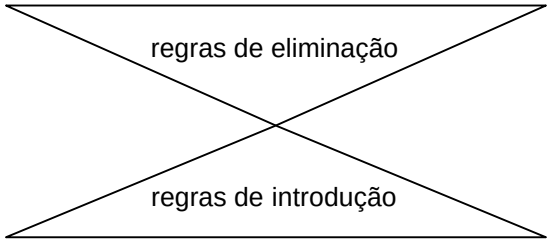

Figura 5 - Forma dos ramos em uma prova normal

Um ramo pode ser definido da seguinte maneira [30, 31]:

Definição de Ramo da Árvore de Prova:

Considerando-se o fragmento da lógica proposicional clássica que contém apenas o absurdo ($\perp$) e a implicação ($\rightarrow$), uma seqüência de ocorrência de fórmulas δ_1, ..., δ_n em uma dedução Π (ou seja, Π é uma árvore de dedução de uma certa fórmula α a partir de um certo conjunto Δ de hipóteses) é um ramo da árvore de prova se e somente se (supondo que $i \in \{ 1, ..., n - 1 \}$):

- δ_1 é uma ocorrência de fórmula que não é conclusão de nenhuma regra em Π;
- δ_{i+1} é conclusão de uma regra de inferência de $\rightarrow$ -introdução, $\perp$-clássico ou $\perp$-intuicionista e δ_i é sua premissa ou é conclusão de uma $\rightarrow$ -eliminação no qual δ_i é a premissa maior dessa regra; e
- δ_n ou é conclusão da prova, ou premissa menor de uma regra de $\rightarrow$ -eliminação.

A definição acima pode ser facilmente estendida para a lógica clássica com todos os conectivos e seus quantificadores.

Uma prova normal pode ser produzida diretamente utilizando-se heurísticas. Portanto, o fato desta lógica possuir normalização facilita a construção (automática) de provas, sendo que essa conclusão vale para qualquer lógica que possua normalização [31].

O objetivo deste capítulo é aplicar essa idéia às provas de correção de programas. Ou seja, propõe-se a criação de padrões para a geração de provas de forma a diminuir o espaço de busca e, portanto, construi-las de forma mais eficiente. Isso inclui a tentativa de se gerar provas com as sentenças a serem demonstradas válidas por construção, evitando o uso de um provador de teoremas para prová-las, se o programa satisfizer sua especificação dada por sua pré-condição e sua pós-condição [2].

Deseja-se conseguir uma prova somente com sentenças demonstráveis sem o uso de um provador de teoremas, já que, em geral, a demonstração destas é ineficiente e tentar fazê-las para sentenças inválidas pode levar o provador a entrar em *loop*. Afinal, saber se uma sentença em lógica de primeira ordem é válida ou não é um processo indecidível. Isso não seria um problema se fossem utilizadas apenas teorias de tipos de dados decidíveis, como a álgebra booleana. Esse, porém, não é o caso dos números inteiros, por exemplo. Mais ainda, sabe-se que é possível gerar muitas provas formais de programas respeitando a sintaxe do cálculo de Hoare, mas que possuem sentenças não-demonstráveis, as quais, obviamente, não são provas corretas.

Portanto, dado um programa com a pós-condição derivada de sua pré-condição depois de sua execução, um verificador formal de programas pode dar diferentes provas como resposta, com seus conjuntos de sentenças onde todas elas podem ser demonstráveis ou não, se não há a garantia de que todas as sentenças são demonstráveis. As figuras 11 e 28 são exemplos claros disso. Em ambas as figuras, as provas são sintaticamente corretas, mas a primeira prova em cada figura possui uma sentença inválida.

Devido a este fato, quer-se analisar heurísticas eficientes (e comprová-las na prática) para um verificador formal de programas utilizando o cálculo de Hoare de modo que ele gere o menor número possível de provas diferentes (de preferência só uma), o qual deve, de alguma forma, sempre que o programa tiver o comportamento desejado, funcionar de tal maneira que sempre haja uma prova correta.

Com as heurísticas apresentadas a seguir, consegue-se construir um corretor de programas que só gera uma prova como saída. Se o programa satisfizer sua especificação, esta só terá sentenças válidas. Caso contrário, pelo menos uma delas não será demonstrável.

4.2
Fortalecimentos e Enfraquecimentos Somente para os Axiomas

Para diminuir o espaço de busca na geração de provas, algo que pode ser feito é deixar para fazer o fortalecimento da pré-condição e o enfraquecimento da pós-condição somente nos axiomas, ou seja, somente nas regras dos comandos **skip** e de atribuição. As únicas exceções são para comandos **while**, para transformar a pré-condição atual no invariante e a pós-condição atual na conjunção do invariante com a negação da condição do **while**, e para o caso da correção de um comando **if** quando sua pós-condição é desconhecida no tratamento dos ramos **then** e **else** (ou só o **then**). Para provar que isso é possível, é necessário mostrar, para cada um dos comandos estudados, que é possível “adiar” os enfraquecimentos e os fortalecimentos. Ou seja, se houver uma prova com um enfraquecimento ou um fortalecimento em uma regra que não seja um axioma nem um comando **while**, haverá uma equivalente a esta que os apresente apenas nos axiomas e na regra do **while** (no caso do **if** é possível subir o enfraquecimento da pós-condição a um axioma quando este comando já foi corrigido e esta asserção já foi determinada, porém isso não é

feito porque se perde em eficiência). Resumindo, para cada prova existe uma outra com o enfraquecimento (ou fortalecimento) postergado e com as mesmas pré e pós-condições. Obviamente, é necessário que os enfraquecimentos e fortalecimentos sejam previamente conhecidos para que possam ser postergados.

A seguir, estão as provas para cada um dos casos:

$$\dfrac{P \rightarrow R \qquad \dfrac{\{ R \}\ C_1\ \{ S \} \quad \{ S \}\ C_2\ \{ Q \}}{\{ R \}\ C_1 ; C_2\ \{ Q \}}}{\{ P \}\ C_1 ; C_2\ \{ Q \}}$$

$$\dfrac{P \rightarrow R \quad \{ R \}\ C_1\ \{ S \} \qquad \dfrac{\{ P \}\ C_1\ \{ S \} \qquad \{ S \}\ C_2\ \{ Q \}}{\{ P \}\ C_1 ; C_2\ \{ Q \}}}{\{ P \}\ C_1 ; C_2\ \{ Q \}}$$

Figura 6 - Fortalecimento da pré-condição para uma seqüência de comandos

$$\dfrac{\dfrac{\{ P \}\ C_1\ \{ S \} \quad \{ S \}\ C_2\ \{ R \}}{\{ P \}\ C_1 ; C_2\ \{ R \}} \qquad R \rightarrow Q}{\{ P \}\ C_1 ; C_2\ \{ Q \}}$$

$$\dfrac{\{ P \}\ C_1\ \{ S \} \qquad \dfrac{\{ S \}\ C_2\ \{ R \} \quad R \rightarrow Q}{\{ S \}\ C_2\ \{ Q \}}}{\{ P \}\ C_1 ; C_2\ \{ Q \}}$$

Figura 7 - Enfraquecimento da pós-condição para uma seqüência de comandos

$$\dfrac{P \rightarrow R \qquad \dfrac{\{ R \wedge B \}\ C_1\ \{ Q \} \quad \{ R \wedge \neg B \}\ C_2\ \{ Q \}}{\{ R \}\ \texttt{if B then } C_1 \texttt{ else } C_2 \texttt{ fi } \{ Q \}}}{\{ P \}\ \texttt{if B then } C_1 \texttt{ else } C_2 \texttt{ fi } \{ Q \}}$$

$$\dfrac{\dfrac{P \wedge B \rightarrow R \wedge B \quad \{ R \wedge B \}\ C_1\ \{ Q \}}{\{ P \wedge B \}\ C_1\ \{ Q \}} \qquad \dfrac{P \wedge \neg B \rightarrow R \wedge \neg B \quad \{ R \wedge \neg B \}\ C_2\ \{ Q \}}{\{ P \wedge \neg B \}\ C_2\ \{ Q \}}}{\{ P \}\ \texttt{if B then } C_1 \texttt{ else } C_2 \texttt{ fi } \{ Q \}}$$

Figura 8 - Fortalecimento da pré-condição para um comando **if-then-else**

$$\frac{\{ P \wedge B \}\ C_1\ \{ R \} \quad \{ P \wedge \neg B \}\ C_2\ \{ R \}}{\{ P \}\ \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi } \{ R \} \qquad R \rightarrow Q}{\{ P \}\ \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi } \{ Q \}}$$

$$\frac{\{ P \wedge B \}\ C_1\ \{ R \} \quad R \rightarrow Q \qquad \{ P \wedge \neg B \}\ C_2\ \{ R \} \quad R \rightarrow Q}{\dfrac{\{ P \wedge B \}\ C_1\ \{ Q \} \qquad \{ P \wedge \neg B \}\ C_2\ \{ Q \}}{\{ P \}\ \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi } \{ Q \}}}$$

Figura 9 - Enfraquecimento da pós-condição para um comando **if-then-else**

Nas provas acima, a regra do **if-then** foi omitida porque pode ser reduzida à do **if-then-else**. Note-se, ainda, que para cada par de provas de correção, os trechos de programa a serem corrigidos recursivamente foram os mesmos, como deveria ocorrer. Por fim, na figura 8, é fácil provar que, se **P** ® **R**, então **P** Ù **B** ® **R** Ù **B** e **P** Ù Ø**B** ® **R** Ù Ø**B**.

Para o caso do **while** (na figura a seguir), supondo **Q** ¹ **P** e **R** ¹ **(P** Ù Ø**B)**, sendo **P** o invariante do *loop* (que nesta pesquisa é fornecido pelo programador), deve-se fazer o enfraquecimento da pós-condição e o fortalecimento da pré-condição para provar a corretude desse comando como mostrado na figura abaixo. Afinal, para que seja possível aplicar a regra do **while** (regra (6) da seção 3.3), é preciso que **P** seja sua pré-condição e **(P** Ù Ø**B)** sua pós-condição. No caso de uma ou ambas as desigualdades acima citadas serem falsas em termos sintáticos, é só não se fazer o fortalecimento ou o enfraquecimento correspondente.

$$\frac{Q \rightarrow P \qquad \dfrac{\{ P \}\ \text{while } B \text{ do } C \text{ od } \{ P \wedge \neg B \} \quad (P \wedge \neg B) \rightarrow R}{\{ P \}\ \text{while } B \text{ do } C \text{ od } \{ R \}}}{\{ Q \}\ \text{while } B \text{ do } C \text{ od } \{ R \}}$$

Figura 10 - Colocando o invariante na pré-condição e na pós-condição

4.3
Fortalecer a Pré-Condição antes de Enfraquecer a Pós-Condição

Ao se tentar fortalecer uma pré-condição, é necessário, por definição, descobrir um conseqüente para uma implicação, que obviamente é mais fraco do que o antecedente. Já o contrário acontece ao se tentar enfraquecer uma pós-condição. De acordo com experimentos feitos utilizando-se uma versão preliminar do corretor de programas descrito no próximo capítulo, ficou claro que é mais fácil achar um conseqüente do que um antecedente de uma implicação, porque o primeiro é mais fraco do que o segundo.

Um exemplo bem simples para este caso é a prova do seguinte: **{ true } x := E { x = E }**, onde **E** não depende de **x**. Neste caso, se o enfraquecimento da pós-condição for feito, como mostrado na primeira prova da figura abaixo, gera-se uma sentença que não é demonstrável, a qual é **true** ® **(x = E)**. Já se o fortalecimento da pré-condição for feito, como exibido na segunda prova, gera-se uma sentença que é um teorema, a qual é **true** ® **(E = E)**, já que o antecedente e o conseqüente dessa implicação são equivalentes.

$$\frac{\{\ true\ \}\ x := E\ \{\ true\ \} \quad true \rightarrow (\ x = E\ )}{\{\ true\ \}\ x := E\ \{\ x = E\ \}}$$

$$\frac{true \rightarrow (\ E = E\ ) \quad \{\ E = E\ )\ x := E\ \{\ x = E\ \}}{\{\ true\ \}\ x := E\ \{\ x = E\ \}}$$

Figura 11 - Fortalecer a pré-condição antes de enfraquecer a pós-condição

Em vários casos diferentes, notou-se comportamento semelhante ao do caso acima, ou seja, o fortalecimento da pré-condição gerou sentenças demonstráveis, porém o enfraquecimento da pós-condição gerou sentenças inválidas. Na correção das atribuições, como descrito na seção abaixo, note-se que as heurísticas sempre fazem o fortalecimento da pré-condição por causa da constatação empírica aqui apresentada.

Uma exceção é quando se corrige um comando onde a variável a receber o valor da atribuição é a posição de um vetor e a pré-condição e a pós-condição possuem informações com relação a todo o vetor e não apenas à posição que está sendo modificada. Nesses casos, como será visto adiante (na seção 4.7), em geral, deve-se fazer os dois, começando-se pelo enfraquecimento da pós-condição.

No caso do comando **skip**, é indiferente fortalecer ou enfraquecer para provar sua corretude. Logo, ambas as provas abaixo são equivalentes. e, portanto, ao implementar um corretor de programas é necessário apenas seguir um dos dois caminhos, diminuindo assim a árvore de busca.

$$\frac{\{ P \}\ \texttt{skip}\ \{ P \} \quad P \rightarrow Q}{\{ P \}\ \texttt{skip}\ \{ Q \}} \qquad \frac{P \rightarrow Q \quad \{ Q \}\ \texttt{skip}\ \{ Q \}}{\{ P \}\ \texttt{skip}\ \{ Q \}}$$

Figura 12 - Equivalência de provas do comando **skip**

4.4
Fazer a Pós-Condição de uma Atribuição ser a Mais Forte Possível

É interessante tentar fazer com que a pós-condição de uma atribuição seja a mais forte possível pois, desse modo, sempre se terá o máximo de informações para a pós-condição do comando que a contém (um **if** ou um **while**) ou para a pré-condição do comando seguinte. Isso ainda garantirá que, se o programa satisfizer a sua especificação, dada pela sua pré-condição e pela sua pós-condição, os enfraquecimentos das pós-condições sempre gerarão sentenças demonstráveis. Contudo, tal fato só é interessante se a pós-condição não for conhecida (algo que acontece quando há seqüências de comandos), pois não é necessário se provar algo mais forte do que o desejado pelo usuário.

Nesta heurística, há quatro casos distintos que devem ser analisados, que são:

- `{ P } x := E { Q }`
- `{ P } x := E( x ) { Q }`
- `{ P( x ) } x := E { Q }`

❖ { P(x) } x := E(x) { Q }

Nos casos onde a pré-condição é da forma **P**, a mesma não depende da variável **x**. Já naqueles onde a forma é **P(x)**, a pré-condição contém a variável **x**. O mesmo vale para as expressões **E** e **E(x)** que são atribuídas a **x**. **Q** é a pós-condição da atribuição ainda não conhecida, que será estudada nas próximas subseções. Se fosse conhecida, ela seria aplicada à heurística explicada na seção seguinte. Em ambas as seções, é suposto que a linguagem de programação utilizada não possui vetores, porque para eles estas heurísticas não podem ser diretamente aplicadas, como será mostrado na seção 4.7.

A seguir serão estudados cada um dos casos detalhadamente:

4.4.1
{ P } x := E { Q }

Neste caso, como **P** não depende de **x**, tem-se que a pós-condição mais forte **Q = (P** Ù **x = E)**. Portanto, supondo que **P** seja diferente de **true** e de **E = E**, a prova de correção é a seguinte:

$$\frac{P \rightarrow (P \wedge E = E) \quad \{ P \wedge E = E \}\ x := E\ \{ P \wedge x = E \}}{\{ P \}\ x := E\ \{ P \wedge x = E \}}$$

Figura 13 - Prova de **{ P } x := E { P** Ù **x = E }**

Como exemplo, para a atribuição **x := a**, onde **P = (y = 0)**, a prova de correção é a seguinte:

$$\frac{y = 0 \rightarrow (y = 0 \wedge a = a) \quad \{ y = 0 \wedge a = a \}\ x := a\ \{ y = 0 \wedge x = a \}}{\{ y = 0 \}\ x := a\ \{ y = 0 \text{ and } x = a \}}$$

Figura 14 - Prova de **{ y = 0 } x := a { y = 0 and x = a }**

Já se **P = true**, a prova da figura 13 ficaria assim:

$$\frac{true \rightarrow E = E \quad \{ E = E \}\ x := E\ \{ x = E \}}{\{ true \}\ x := E\ \{ x = E \}}$$

Figura 15 - Prova de **{ true } x := E { x = E }**

Se no exemplo dado **P = true**, obter-se-ia **Q = (x = a)** simplesmente. Por fim, se **P = (E = E)**, seria possível omitir o fortalecimento da pré-condição da prova da figura anterior.

4.4.2
{ P } x := E(x) { Q }

Como **P** não possui informações sobre **x**, não se pode saber o novo valor de **x** depois da atribuição, tendo em vista que **E(x)** depende de **x**, supondo-se que **E(x)** seja uma expressão que tenha passado por uma simplificação levando em conta os valores das outras variáveis que porventura apareçam em **E(x)**. Portanto, a pós-condição mais forte na maioria dos casos é simplesmente **P**. Logo, a prova de correção fica simplesmente assim:

$$\frac{}{\{ P \}\ x := E(x)\ \{ P \}}$$

Figura 16 - Prova de **{ P } x := E(x) { P }**

Se **E(x)** não tiver passado por uma simplificação, pode ser que nela apareçam operações cujo resultado não dependa de **x**. Um exemplo é **E(x) = 0 * x** ou **E(x) = x** Ú **true**, sendo **x** uma variável inteira e booleana

respectivamente. Quando isso acontecer, a heurística a ser aplicada deve ser aquela para o caso **{ P } x := E { Q }**.

Em algumas situações, pode ser que não se saiba o valor de **x** na pós-condição. Porém, talvez seja possível saber alguma propriedade sobre o mesmo. Por exemplo, seja a atribuição **x := x * x**. Supondo-se **P = true**, tem-se que **Q = (x ³ 0)**. Neste caso, a prova de correção seria a seguinte:

$$\frac{true \rightarrow (x * x \geq 0) \quad \{ x * x \geq 0 \}\ x := x * x\ \{ x \geq 0 \}}{\{ true \}\ x := x * x\ \{ x \geq 0 \}}$$

Figura 17 - Prova de **{ true } x := x * x { x ³ 0 }**

Por isso, quando for necessário corrigir uma atribuição nessa situação, pode-se assumir que a variável **x** já possuía um valor anterior à atribuição, o qual não é conhecido. E depois da execução do comando, seu valor atual dependerá do antigo. Como é existente, utiliza-se o quantificador existencial para capturar essa idéia. Assim, tem-se que **Q = P** Ù $**y (x = E(y))**, assumindo que **y** não ocorra livre em **E**. Logo,

$$\frac{P \rightarrow R \quad \{ R \}\ x := E(x)\ \{ P \wedge \exists y (x = E(y)) \}}{\{ P \}\ x := E(x)\ \{ P \wedge \exists y (x = E(y)) \}}$$

onde $R = P \wedge \exists y (E(x) = E(y))$

Figura 18 - Prova de **{ P } x := E(x) { P** Ù $**y (x = E(y)) }**

Obviamente, a pós-condição da prova acima pode ser enfraquecida para se tornar igual a das duas provas dadas acima.

Já se **P = true**, tem-se que:

$$\frac{true \rightarrow R \quad \{ R \}\ x := E(x)\ \{ \exists y\ (x = E(y)) \}}{\{ true \}\ x := E(x)\ \{ \exists y\ (x = E(y)) \}}$$

onde $R = \exists y\ (E(x) = E(y))$

Figura 19 - Prova de **{ true } x := E(x) { $y (x = E(y)) }**

Aplicando-se esta heurística ao exemplo da figura 18, tem-se que:

$$\frac{true \rightarrow R \quad \{ R \}\ x := x * x\ \{ \exists y\ (x = y * y) \}}{\{ true \}\ x := x * x\ \{ \exists y\ (x = y * y) \}}$$

onde $R = \exists y\ (x * x = y * y)$

Figura 20 - Prova de **{ true } x := x * x { $y (x = y * y) }**

Claramente, tem-se que **$y (x = y * y) ® x ³ 0**. Por fim, se **P = $y (E(x) = E(y))**, seria possível omitir o fortalecimento da pré-condição da prova da figura anterior.

4.4.3
{ P(x) } x := E { Q }

Supondo-se que **P(x)** não possua contradições e que o valor de **x** antes da atribuição possa ser encontrado analisando-se a pré-condição, pode-se dizer que **Q = (P(a) Ù x = E)**, chamando-se esse valor de **x** de **a** e considerando-se que **P(x)** não contenha apenas informações sobre o valor de **x** encontrado:

$$\frac{P(x) \rightarrow R \quad \{ R \}\ x := E\ \{ P(a) \wedge x = E \}}{\{ P(x) \}\ x := E\ \{ P(a) \wedge x = E \}}$$

onde $R = P(a) \wedge E = E$

Figura 21 - Prova de **{ P(x) } x := E { P(a) Ù x = E }**

Como exemplo, para a atribuição **x := a**, onde **P = (x = b Ù y = x)**, a prova de correção é a seguinte, de onde é omitido o termo **y = y** ou **b = b** que aparecem em **R** e em **Q** ao se fazer a substituição do valor de **x** (algo que é feito na implementação da heurística):

$$\frac{x = b \wedge y = x \rightarrow R \quad \{ R \}\ x := a\ \{ y = b \wedge x = a \}}{\{ x = b \wedge y = x \}\ x := a\ \{ y = b \wedge x = a \}}$$

onde $R = (y = b \wedge a = a)$

Figura 22 - Prova de **{ x = b Ù y = x } x := a { y = b Ù x = a }**

Se **P(x)** possuir apenas informações sobre o valor de **x**, a prova da figura 21 poderá ser simplificada, omitindo-se os **P(a)**'s encontrados, ficando desta forma:

$$\frac{P(x) \rightarrow E = E \quad \{ E = E \}\ x := E\ \{ x = E \}}{\{ P(x) \}\ x := E\ \{ x = E \}}$$

Figura 23 - Prova de **{ P(x) } x := E { x = E }**

Se no exemplo dado **P = (x = b)**, obter-se-á **Q = (x = a)** simplesmente. Já se o valor de **x** antes da atribuição não puder ser encontrado analisando-se a pré-condição ou for muito difícil achá-lo, pode-se fazer algo semelhante ao feito na seção anterior, ou seja, introduzir o existencial, já que **x** possui um valor inicial que não é conhecido. Deve-se lembrar que a variável ligada introduzida não deve ser livre em **P**. Então, tem-se que:

$$\frac{P(x) \rightarrow R \quad \{ R \}\ x := E\ \{ \exists a\ P(a) \wedge x = E \}}{\{ P(x) \}\ x := E\ \{ \exists a\ P(a) \wedge x = E \}}$$

onde $R = \exists a\ P(a) \wedge (E = E)$

Figura 24 - Prova de **{ P(x) } x := E { $a P(a) Ù x = E }**

Como exemplo, para a atribuição **x := a**, onde **P = x = b**, a prova de correção é a seguinte, a qual é claramente correta:

$$\frac{x = b \rightarrow R \quad \{ R \}\ x := a\ \{ \exists c\ (c = b) \wedge (x = a) \}}{\{ x = b \}\ x := a\ \{ \exists c\ (c = b) \wedge (x = a) \}}$$

onde $R = \exists c\ (c = b) \wedge (a = a)$

Figura 25 - Prova de **{ x = b } x := a { $c (c = b) Ù (x = a) }**

4.4.4
{ P(x) } x := E(x) { Q }

Na prova abaixo, as suposições feitas são as mesmas das da primeira figura da situação anterior (figura 21), com a restrição de que **a** não possui variáveis ligadas, sendo que neste caso, **Q = (P(a) Ù x = E(a))**:

$$\frac{P(x) \rightarrow R \quad \{ R \}\ x := E(x)\ \{ P(a) \wedge x = E(a) \}}{\{ P(x) \}\ x := E(x)\ \{ P(a) \wedge x = E(a) \}}$$

onde $R = P(a) \wedge (E(x) = E(a))$

Figura 26 - Prova de **{ P(x) } x := E(x) { P(a)** Ù **x = E(a) }**

Como exemplo, para a atribuição **x := a + x**, onde **P = (x = b** Ù **y = x)**, as duas provas de correção abaixo podem ser resultantes, de onde também são omitidos o termo **y = y** ou **b = b** que aparecem em **R** e em **Q** ao se fazer a substituição do valor de **x**:

$$\frac{x = b \wedge y = x \rightarrow R \quad \{ R \}\ x := a + x\ \{ y = b \wedge x = a + z \}}{\{ x = b \wedge y = x \}\ x := a + x\ \{ y = b \wedge x = a + z \}}$$

onde $R = (y = b \wedge a + x = a + z)$ e $z = b$ ou $z = y$

Figura 27 - Provas de **{ x = b** Ù **y = x } x := a + x { y = b** Ù **x = a + y / b }**

Já se **P(x)** possuir apenas informações sobre o valor de **x**, tem-se que:

$$\frac{P(x) \rightarrow R \quad \{ R \}\ x := E(x)\ \{ x = E(a) \}}{\{ P(x) \}\ x := E(x)\ \{ x = E(a) \}}$$

onde $R = E(x) = E(a)$

Figura 28 - Prova de **{ P(x) } x := E(x) { x = E(a) }**

Se no exemplo dado **P = (x = b)**, somente uma prova será obtida, com **Q = (x = a + b)** simplesmente. Assim como na seção anterior, se o valor de **x** antes da atribuição não puder ser encontrado analisando-se a pré-condição ou for muito difícil achá-lo, novamente introduz-se o quantificador existencial, ficando a prova assim:

$$\frac{P(x) \rightarrow R \quad \{ R \}\ x := E(x)\ \{ \exists a\ (P(a) \wedge x = E(a)) \}}{\{ P(x) \}\ x := E(x)\ \{ \exists a\ (P(a) \wedge x = E(a)) \}}$$

onde $R = \exists a\ (P(a) \wedge E(x) = E(a))$

Figura 29 - Prova de **{ P(x) } x := E(x) { $a (P(a) Ù x = E(a)) }**

Como exemplo, para a atribuição **x := a + x**, onde **P = x = b**, a prova de correção é a seguinte:

$$\frac{x = b \rightarrow R \quad \{ R \}\ x := a + x\ \{ \exists c\ (c = b \wedge x = a + c) \}}{\{ x = b \}\ x := a + x\ \{ \exists c\ (c = b \wedge x = a + c) \}}$$

onde $R = \exists c\ (c = b \wedge a + x = a + c)$

Figura 30 - Prova de **{ x = b } x := a + x { $c (c = b Ù a + x = a + c) }**

Note-se que nas últimas três situações de correção de atribuições, apresentadas nas três seções anteriores, não se consegue, de forma geral, uma pós-condição enxuta, ou seja, aquela que seria obtida caso a verificação formal fosse feita manualmente. Se assim fosse, em muitas situações não seria necessário o uso do quantificador existencial e a pós-condição não conteria informações redundantes desnecessariamente. Todavia, para sempre se obter uma pós-condição semelhante a obtida por um usuário, seria necessário que o

corretor tivesse um módulo aritmético completo e consistente, algo que não é possível [4, 21].

Um problema da abordagem adotada é que as sentenças e as asserções encontradas podem ficar muito maiores do que as que seriam obtidas se a prova fosse feita manualmente, o que dificultará a verificação das primeiras no provador de teoremas. Por outro lado, a geração da prova de correção de programas, a menos da verificação dessas sentenças, é feita automaticamente.

4.5
Começar de Trás para Frente

Sempre que se tem uma seqüência de comandos que deve ser verificada, a prova desta é a prova de correção de duas seqüências de comandos, onde a pós-condição da primeira é a pré-condição da segunda. E pode-se começar a prova de todo o conjunto começando-se tanto a partir do primeiro grupo de comandos quanto do segundo.

Como sempre é possível saber a pós-condição mais forte a partir das atribuições, é fácil começar a prova da seqüência a partir do primeiro comando e provar os demais na ordem em que se encontram. Ou seja, dada uma pré-condição e um comando, quer-se determinar a pós-condição deste comando que será a pré-condição do que se segue. Porém, como em muitas vezes isso vá fazer as asserções e as sentenças ficarem muito grandes graças à introdução do existencial, talvez seja melhor começar do último comando e seguir provando a correção dos comandos até se chegar ao primeiro deles. Isto é, dada uma pós-condição e um comando, deseja-se determinar a pré-condição deste comando que será a pós-condição do anterior a esse. Além disso, ao começar de trás pra frente na atribuição, é necessário somente fazer substituições e renomeações de variáveis (apenas se existirem variáveis ligadas) e não serão provadas propriedades mais fortes do que o desejado.

É simples começar a prova de trás para frente nos comandos **skip** (vide figura 12) e **while** (é só saber o invariante, que é dado pelo usuário), e nas atribuições (se a pós-condição é conhecida, a determinação da pré-condição é obtida com a substituição de variáveis como foi mostrada na regra de atribuição na seção 3.3).

Porém, para as regras do **if** isso não é trivial, já que a pós-condição deste comando pode depender do teste do mesmo, ou seja, a asserção que vale após

a execução do **if** depende do resultado da avaliação do teste do mesmo. Portanto, pode ser necessário saber se a pré-condição deste comando satisfaz este teste ou não, o que pode ser muito difícil de se saber através da pós-condição e dos comandos nos ramos **then** e **else** (ou só **then**) do **if**. Note-se que se a pós-condição do **if** for conhecida, os ramos do mesmo podem ser avaliados de trás para frente, mas não o comando todo propriamente dito. Na figura abaixo isso fica claro para o caso do **if-then-else** (para o **if-then** a explicação é semelhante):

$$\frac{\dfrac{P \wedge B \rightarrow R \quad \{ R \}\ C_1\ \{ Q \} \qquad P \wedge \neg B \rightarrow S \quad \{ S \}\ C_2\ \{ Q \}}{\{ P \wedge B \}\ C_1\ \{ Q \} \qquad \{ P \wedge \neg B \}\ C_2\ \{ Q \}}}{\{ P \}\ \texttt{if B then } C_1 \texttt{ else } C_2 \texttt{ fi } \{ Q \}}$$

Figura 31 - Corrigir os ramos do **if** de trás para frente

Quando se corrige cada ramo de trás para frente, pode se chegar a duas pré-condições diferentes, como é possível se observar acima. Porém, no caso geral, a partir de **R** e **S** não é possível descobrir **P** sem o uso de um provador de teoremas, justamente porque não dá para se saber se **B** vale ou não antes da execução do **if** sem a utilização do mesmo.

4.6
Correção do Comando if Quando a Pós-Condição é Desconhecida

Quando se corrige os ramos de um comando **if-then-else** na ordem em que foram escritos, algo que ocorre quando não se sabe a sua pós-condição, é que esta pode ter dois valores possíveis dependendo da avaliação do teste, pois a sua execução pode seguir por dois ramos diferentes.

Quando **P** permite avaliar **B**, já foi dito que a pré-condição de um dos ramos possuirá uma contradição. De acordo com as heurísticas apresentadas para a correção dos axiomas, essa será levada até o fim do ramo, já que em nenhum momento a contradição será eliminada. Logo, pode-se dizer que a pós-

condição do **if** será a disjunção da de cada ramo. Como uma delas será falsa, e tem-se que **A** Ú ∧ º **A**, chega-se ao resultado desejado.

Já quando **P** não permite avaliar **B**, tem-se que, de fato, a disjunção das pós-condições é o que vale para o **if**. Nas figura abaixo mostra-se as provas para os casos **if-then-else** e para o **if-then** (obtido fazendo o ramo do else ser o **skip**):

$$\dfrac{\dfrac{\{ P \wedge B \}\ C_1\ \{ Q \}\quad Q \rightarrow Q \vee R \qquad \{ P \wedge \neg B \}\ C_2\ \{ R \}\quad R \rightarrow Q \vee R}{\{ P \wedge B \}\ C_1\ \{ Q \vee R \} \qquad \{ P \wedge \neg B \}\ C_2\ \{ Q \vee R \}}}{\{ P \}\ \texttt{if B then } C_1 \texttt{ else } C_2 \texttt{ fi } \{ Q \vee R \}}$$

Figura 32 - Correção do **if-then-else** para deteminar sua pós-condição

$$\dfrac{\dfrac{\{ P \wedge B \}\ C\ \{ Q \}\quad Q \rightarrow Q \vee (P \wedge \neg B)}{\{ P \wedge B \}\ C\ \{ Q \vee (P \wedge \neg B) \}} \qquad P \wedge \neg B \rightarrow Q \vee (P \wedge \neg B)}{\{ P \}\ \texttt{if B then C fi } \{ Q \vee (P \wedge \neg B) \}}$$

Figura 33 - Correção do **if-then** para deteminar sua pós-condição

No topo de cada prova pode-se verificar que há enfraquecimentos. São esses enfraquecimentos que não serão levados até os axiomas, pois a prova pode ser facilmente construída desta maneira. Afinal, a vantagem de tal heurística é apenas minimizar o espaço de busca durante a construção das provas, para não se ficar tentando fazer enfraquecimentos e fortalecimentos para qualquer regra. Sendo que nesse caso tal vantagem não existe.

4.7 Problemas com Vetores

As heurísticas mostradas acima fazem a prova de correção de programa automaticamente, exceto pela verificação de sentenças. Entretanto, sabe-se que elas são válidas se o programa satisfizer a sua especificação dada pela sua pré-

condição e pela sua pós-condição. Portanto, a prova será correta a não ser que o programa não satisfaça a mesma.

Contudo, quando vetores são incluídos na linguagem utilizada pelo corretor de programas, isso não pode mais ser feito automaticamente. Infelizmente as duas últimas heurísticas mostradas acima não funcionam adequadamente com vetores. O problema ocorre quando a variável que receberá o novo valor na atribuição é a posição de um vetor.

O que acontece é que as propriedades que descrevem vetores são geralmente relacionadas com todos os seus elementos, e não a alguns específicos. Logo, um provador de teoremas ou um ser humano são necessários para fazer certas inferências relacionando índices, a posição sendo modificada e as propriedades referenciando o vetor inteiro.

Quando corrigido de trás para frente, a pré-condição será igual à pós-condição se a asserção da pós-condição diz respeito a todo vetor. Por exemplo, na figura abaixo, a prova não pode ser começada pela pós-condição porque a pré-condição será idêntica a ela, o que gerará uma sentença inválida: "**j (j < i ® a[j] = j)** ® "**j (j < i + 1 ® a[j] = j)**. As maneiras errada e correta de se fazer a prova são, respectivamente:

$$\frac{P \rightarrow Q \quad \{ Q \}\ a[\ i\ ] := i\ \{ Q \}}{\{ P \}\ a[\ i\ ] := i\ \{ Q \}}$$

$$\frac{P \rightarrow R \qquad \dfrac{\{ R \}\ a[\ i\ ] := i\ \{ P \text{ and } a[\ i\ ] = i \} \quad P \text{ and } a[\ i\ ] = i \rightarrow Q}{\{ R \}\ a[\ i\ ] := i\ \{ Q \}}}{\{ P \}\ a[\ i\ ] := i\ \{ Q \}}$$

onde $P = \forall j\ (\ j < i \rightarrow a[\ j\ ] = j\ )$,
$Q = \forall j\ (\ j < i + 1 \rightarrow a[\ j\ ] = j\ )$ e $R = (\ P \text{ and } i = i\ )$

Figura 34 - Prova que não pode ter a correção iniciada de trás para frente

Ainda, quando se tenta achar a pós-condição mais forte de uma atribuição, a inserção de existenciais pode causar problemas, porque a propriedade da posição do vetor sendo modificada não pode ser facilmente encontrada na pré-condição. Adicionalmente, se esta for descoberta, a nova pré-condição, que é

mais fraca do que a antiga, é difícil de ser obtida, já que inferências com os índices devem ser feitas.

4.8
Diminuição do Tamanho das Sentenças e Asserções

Como anteriormente foi explicado, a inclusão do existencial pode tornar as sentenças e asserções muito grandes. Além disso, algumas vezes o próprio usuário pode entrar informações repetidas na pré-condição ou na pós-condição, como por exemplo **true and true**, o que também contribui para o crescimento das sentenças a serem demonstradas e das asserções. Portanto, parece ser interessante simplificá-las sempre que possível a cada fortalecimento ou enfraquecimento.

Supondo cada asserção como um conjunto de conjunções, ou seja, uma asserção **P** sendo sempre da forma $\mathbf{P_1 \wedge ... \wedge P_n}$, onde cada $\mathbf{P_i}$ é uma sentença lógica, pode-se remover um desses $\mathbf{P_i}$'s sempre que ele for o **true** ou se $\exists \mathbf{j\ (\ i \neq j \rightarrow P_i = P_j\ )}$.

Ainda, se **P** possuir um termo com um quantificador que não possua variáveis livres, ou seja, variáveis que correspondam àquelas presentes no programa, o mesmo pode ser removido do conjunto de conjunções.

Também, supondo-se que um termo com quantificador seja um conjunto de conjunções, o tipo de simplificação citado acima pode ser feito recursivamente.

Em outras situações, um dos termos do conjunto de conjunções é redundante, por haver um outro que engloba seu significado. Um exemplo seria se $\mathbf{x \geq 0}$ e $\mathbf{x > 0}$ fossem termos de **P**. Obviamente, $\mathbf{x \geq 0}$ é uma informação que já está presente no conjunto, podendo ser retirada. Porém, para se fazer este tipo de simplificação, faz-se necessário o uso de um provador de teoremas.

Finalizando, seria possível simplificar uma asserção se uma contradição fosse encontrada, transformando-a em **false**. Pode-se também diminuir sentenças e asserções considerando-se as propriedades da aritmética e a lógica de primeira ordem. Todavia, novamente seria necessário o uso de um provador de teoremas.

De acordo com o que foi dito nos últimos dois parágrafos, fica evidente que não vale a pena implementar um simplificador de sentenças e asserções. Afinal, no caso geral, as simplificações a serem feitas se encaixariam no que foi

abordado neles. As exceções, basicamente, são quando o usuário entra informações redundantes nas asserções ou quando o programa possui computações desnecessárias, tais como a expressão **true** no teste de **if**.