LPSBACK — Um Exercício de Construção e Verificação de um Algoritmo por Sucessivas Transformações

J. A. Legatheaux Martins
CIUNL

RESUMO: A propósito de um problema concreto introduz-se um curioso algoritmo cuja compreensão e verificação é conseguida por transformações sucessivas. Começa-se por desenhar uma primeira versão ineficiente, mas correcta, em seguida por sucessivas transformações e enriquecendo as propriedades verificadas em cada uma delas, chega-se à versão final e respectiva verificação.

ABSTRACT: Motivated by a concrete problem we introduce a curious algorithm whose understanding and verification is achieved by sucessive transformations. We start by building a first, inefficient but correct version and then, by sucessive transformations and enrichement of the verified properties, we arrive to a final and correct version.

1. Introdução e notações

Durante a construção de dois programas, [1], [2], que tem como resultado impressões com efeitos especiais, como a reescrita de caracteres e edição de caracteres sublinhados e com acentos, houve necessidade de virtualizar a impressora utilizada.

Decidiu-se então que os referidos programas emitiriam caracteres para uma impressora virtual que executasse a operação de puxar a cabeça atrás sempre que recebesse o caracter «backspace».

Quando uma impressora deste tipo não está disponível é produzido um ficheiro a partir do qual o programa aqui discutido reproduziria na impressora disponível o efeito equivalente.

Trata-se da simulação de uma rede de processos em que o último programa consome caracteres de um dado produtor e produz sequências de caracteres (convenientemente transformadas) que envia para a impressora.

LPSBACK é este último programa. LPSBACK recebe um texto contendo eventualmente «backspaces» e decompõe cada uma das suas linhas (até um máximo de três) que imprime em seguida sobrepostas.

LPSBACK permite reproduzir efeitos que exijam até três caracteres sobrepostos na mesma posição. Para além deste limite os efeitos são desprezados; no entanto, o algoritmo utilizado pode ser facilmente generalizado até um número de sobreposições arbitrário.

A construção deste programa baseia-se no algoritmo que trata uma dada linha decompondo-a em três linhas a imprimir. Este algoritmo tal como foi por nós concebido é bastante elegante e dos mais eficientes para realizar esta tarefa. A sua verificação é um exercício interessante pois só a mesma o permite compreender na totalidade.

Para verificar o algoritmo aplica-se informalmente o método assercional desenvolvido por Floyd e Hoare, [3].

Para verificar o algoritmo final, verifica-se primeiro uma sua versão ineficiente mas correcta, de seguida por sucessivas transformações e enriquecendo as propriedades verificadas em cada uma delas, chega-se à versão final e respectiva verificação.

O problema do tratamento de uma linha pode ser assim enunciado:

Dada uma sequência I de caracteres <ci, ..., cN> com Ø <= N <= Max e cujo iesimo caracter passaremos a designar por I[i].

Sabendo que para todo o 1 <= i <= N, I[i] pertence a um conjunto de caracteres contendo o espaço, as letras, os digitos, os símbolos especiais e «backspace», mas do qual não fazem parte nenhum caracter de controlo incluindo o caracter «tab».

Obter 3 sequências de Max caracteres cuja impressão sobreposta reproduza o efeito gráfico de a impressora executar a impressão do caracter «backspace» puxando a cabeça atrás. A cabeça da impressora passará a partir de agora a ser designada por «carro da impressora».

Uma sequência I cuja correcta reprodução exigisse mais do que três sobreposições tem por resultado apenas três sequências, nada se especificando sobre quais os caracteres a desprezar nem sobre o facto de alguns ou eventualmente todos esses três caracteres serem brancos.

Consideremos as seguintes definições:

C — o conjunto dos caracteres que podem estar presentes na sequência I.

Branco — o caracter «space».

Backspace — o caracter «backspace».

I [i] — o iesimo caracter de sequência I.

L1, L2, L3 — as sequências de Max caracteres a obter como resultado.

Max — o número máximo de caracteres em I.

N — o número de caracteres em I.

L1[i], L2[i], L3[i] — o iesimo caracter das sequências L1, L2, L3 respectivamente.

Read (c) — a operação que permite «tomar o próximo» caracter da sequência I.

:= — a operação de afectação no «sentido normal».

Carro — o mecanismo de impressão da impressora, ou melhor, o número de ordem da coluna na qual vai ser impresso o próximo caracter.

2. Primeira versão do algoritmo

No sentido de clarificar o significado da frase «cuja impressão sobreposta reproduza o efeito de a impressora executar o Backspace puxando o carro atrás» começaremos por construir um modelo matemático do movimento do carro.

Seja carro a posição da cabeça (número de ordem da coluna sobre a qual a mesma está pressupondo a numeração da esquerda para a direita e começando em 1 — trata-se de uma posição virtual).

Como é sabido, no nosso caso: 1 <= Carro <= Max + 1.

Seja seguinte (carro, c) a função que devolve o novo valor de carro depois de se «imprimir» um caracter c pertencente a C.

Esta função é completamente definida pela seguinte equação:

EQUAÇÃO 1:

Se Carro = 1 e c = Backspace → seguinte (Carro, c) = 1.

Se carro > 1 e c = Backspace → seguinte (Carro, c) = Carro − 1.

Se c =/= Backspace → seguinte (Carro, c) = Carro + 1.

Esta relação tem de ser mantida invariante pelo nosso algoritmo.

O algiritmo tem a forma:

```
Carro: = 1;
L1: = L2: = L3: = <Branco, ..., Branco>; isto é Max Brancos.
Enquanto houver caracteres em I Fazer
  read (c);
  tratar (c)
Fimfazer;
```

Em que a operação Tratar (c) tem a forma:

```
Se c = Backspace Então
  Se Carro > 1 Então Carro: = Carro − 1 Fimse
Senão
  Se existe uma sequência L1, L2 ou L3 tal que
                Li [Carro] = branco
    Então
      Li [Carro]: = c;
      Carro: = Carro + 1
    Senão
      Carro: = Carro + 1
    Fimse
Fimse;
```

Nota: Li pode tomar um dos valores L1, L2, ou L3. Fim de nota.

A ideia fundamental consiste no seguinte:

Se a equação 1 é o modelo matemático do movimento do Carro então qualquer algoritmo de tratamento de I tem de:

a) Conservar o modelo do movimento do Carro invariante.

b) Qualquer caracter =/= de Backspace tem de ser colocado em qualquer uma das sequências L1, L2 ou L3 mas sempre na posição do Carro antes de esta ser actualizada (ou eventualmente desprezado).

PROPOSIÇÃO 1: O algoritmo apresentado verefica estas duas propriedades.

DEMONSTRAÇÃO DA PROPOSIÇÃO 1:

A demonstração é feita por indução.

Se a sequência I é vazia o algoritmo tem como resultado três sequências de Max caracteres Branco o que verefica *a*) e *b*).

Se foi tratado o caracter I [i] de I e até aí *a*) e *b*) são verificadas então o tratamento do caracter I [i + 1], que designamos por c, mantem *a*) e *b*) como invariantes.

Suponhamos que após o tratamento de I [i] Carro tem o valor K e L1, L2 e L3 são sequências de caracteres que verificam *b*), o tratamento de c provoca as seguintes alterações:

Se c = Backspace, Carro é actualizado de acordo com a equação 1, logo *a*) mantem-se, L1 L2 e L3 não são alteradas, logo *b*) mantem-se.

Se c =/= Backspace e existe uma sequência Li tal que Li [k] = Branco c é colocado na Kesima posição de Li, o que verifica *b*) e o Carro toma o valor K + 1 o que verifica *a*). Se não existe uma sequência Li naquelas condições, c é desprezado e o Carro toma o valor K + 1 o que conserva *a*) e *b*).

Nota: desprezar o caracter é equivalente a colocá-lo numa sequência L4 ou L5 ... que nunca seriam impressas. Fim de nota.

Dado que Max (finito) é o limite para o comprimento de I, não é necessário demonstrar que o algoritmo termina.

FIM DA DEMONSTRAÇÃO DA PROPOSIÇÃO 1

O algoritmo não é determinista visto, que nada se pode afirmar sobre quais os caracteres desprezados, nem em qual das sequências L1, L2 ou L3 os outros são colocados. Apenas se garante que numa dada posição são impressos três caracteres alguns dos quais brancos.

3. Segunda versão do algoritmo

Em LPSBACK não é tratada apenas uma sequência I (linha), mas uma sequência de sequências I. Para cada uma delas é necessário aplicar de novo o algoritmo 1.

No sentido de optimizar o processo será conveniente evitar a impressão de Brancos à direita e dispensar tanto quanto possível a impressão de sequências apenas constituídas por brancos.

Assim é conveniente introduzir as variáveis Topo 1, Topo 2 e Topo 3 que contem em cada momento a posição na sequência L1, L2 e L3 respectivamente, do caracter ratirado de I mais à direita.

Por outro lado convém fazer uma escolha determinista de quais as sequêncijas onde os caracteres devem ser colocados previligeando a ordem L1, L2, L3.

A nova versão do algoritmo é:

```
Carro: = 1; Topo1: = Topo2: = Topo3: = Ø;
L1: = L2: = L3: = <Branco, ..., Branco>;
Enquanto houver caracteres em I Fazer
  read (c);
  tratar (c)
Fimfazer;
```

Com Tratar (c):

```
Se c = Backspace Então
  Se Carro < 1 Então Carro: = Carro − 1 Fimfazer
Senão
  Se L1 [Carro] = Branco Então
      L1 [Carro]: = c;
      Se Topo1 < Carro Então Topo 1: = Carro Fimse;
```

```
        Carro: = Carro + 1
  Senão se L2 [Carro] = Branco Então
        L2 [Carro]: = c;
        SE Topo2 < Carro então Topo 2: = Carro Fim
se;
        Carro: = Carro + 1
  Senão se L3 [Carro] = Branco Então
        L3 [Carro]: = c;
        Se Topo3 < Carro Então Topo3: = Carro Fim se;
        Carro: = Carro + 1
  Senão
        Carro: = Carro + 1
        Fim se
  Fim se;
```

Este algoritmo é o mesmo que o anterior a menos do caracter determinista da escolha da linha onde colocar o caracter e da introdução das variáveis Topoi. As propriedades verificadas são as mesmas que o algoritmo anterior enriquecidas com a seguinte:

PROPOSIÇÃO 2:

Durante toda a execução do algoritmo 2 as seguintes relações são mantidas invariantes:

a) Topo1 >= Ø e Topo2 >= Ø e Topo3 >= Ø.
b) Em qualquer das sequências Li se existe um K tal que Li[K] =/= Branco então K >= Topoi.

DEMONSTRAÇÃO DA PROPOSIÇÃO 2:

Em primeiro lugar é necessário ter em atenção que em I podem existir caracteres Branco logo, à esquerda de Topoi podem existir Brancos em Li. O que a proposição garante é que não existem à direita caracteres não brancos.

A demonstração é por indução. Dispensa-se a demonstração da terminação dado o caracter finito do comprimento de I.

Inicialmente Topoi = Ø, Topo2 = Ø, Topo3 = Ø e L1 = L2 = L3 = <Branco, ..., Branco>, logo *a*) e *b*) são verificadas.

Se após o tratamento de I[i] *a*) e *b*) são verificadas, seja c o caracter I[i+1] e K a posição do Carro após o tratamento de I[i].

Se c = Backspace, L1, L2, L3, Topo1, Topo2 e Topo3 não são alterados e o conjunto continua a verificar a hipótese. Se c =/= Backspace suponhamos que L1[K] = Branco. Neste caso c é afectado a L1[K] e Topo1 é actualizado. Se Topo1 era >= K o seu valor mantém-se e *a*) e *b*) continuam a verificar-se, senão Topo1 toma o valor K e a hipótese continua a verificar-se. Se L1[K] =/= Branco L1 e Topo1 não são alterados o Topo1 >= K por hipótese indutiva.

Para o caso em que se sequências escolhidas são L2 ou L3 a demonstração é semelhante. Para o caso em que nenhuma delas é alterada a hipótese continua a verificar-se.

FIM DA DEMONSTRAÇÃO DA PROPOSIÇÃO 2

4. Terceira e última versão do algoritmo

Nesta fase aplicaremos uma nova transformação ao algoritmo. Como foi incialmente afirmado 1 <= Carro <= Max + 1.

Sendo o caso em que Carro toma o valo Max + 1 o caso limite que marca a posição do carro após o tratamento de uma sequência I de comprimento Max e sem nenhum Backspace. Dado que as sequências L1, L2 e L3 são inicialmente preenchidas a Brancos sempre que Carro > Topoi então necessariamente Li[Carro] = Branco.

Repare-se que Carro > Topoi => Li[Carro] = Branco mas a implicação inversa não é verdadeira. Este aspecto tem influência sobre os caracteres desprezados e será discutida no ponto seguinte.

Tomando em atenção aquela afirmação podemos substituir o teste Li[Carro] = Branco por Carro > Topoi em toda a acção tratar (c).

Se esta substituição for realizada as instruções:

```
Se Topoi > Carro Então Topoi: = Carro Fimse;
```

podem ser substituídos Topoi: = Carro pois Carro > Topoi foi garantido pelo teste anterior. A nova versão do algoritmo é pois:

..... idêntico ao da versão anterior.....

```
Com Tratar (c):

Se c = Backspace Então
      Se Carro > 1 Então Carro: = Carro – 1 Fimse
Senão
      Se Carro > Topo1 Então
            Li [Carro]: = c;
            Topo1: = Carro;
            Carro: = Carro + 1
      Senão se Carro > Topo2 Então
            L2 [Carro]: = c;
            Topo2: = Carro;
            Carro: = Carro + 1
      Senão se Carro > Topo3 então
            L3 [Carro]: = c;
            Topo3: = Carro;
            Carro: = Carro + 1
      Senão
            Carro: = Carro + 1
      Fimse
Fimse;
```

O novo algoritmo está verificado e podem demonstrar as seguintes duas propriedades adicionais.

PROPOSIÇÃO 3:

Topo1 > = Topo2 > = Topo3 > = 0 é um invariante de Tratar (c).

DEMONSTRAÇÃO DA PROPOSIÇÃO 3:

A demonstração também é por indução.
Inicialmente Topo1 = Topo2 = Topo3 = 0 logo verifica-se.
Se quando se vai tratar o caracter I[i] de I a hipótese se verifica analisemos se a mesma se mantem após o tratamento de I[i] que designaremos por c.
Se c = Backspace os Topoi's não são alterados e portanto mantêm-se.
Se c =/= Backspace e Carro > Topo1, Topo1 toma o valor de Carro, logo é incrementado e continua a manter-se. Se Carro < = Topo1 e Carro > Topo2 então Topo2 é alterado tomando o valor de Carro logo Topo1 < = Carro = Topo2. Como pela hipótese Topo2 > = Topo3 e Topo2 foi incrementado Topo2 > = Topo3 continua a manter-se.
Se Carro > = Topo1 e Carro > = Topo2 e Carro < Topo3 é semelhante.
Se Carro > = Topoi, estes não são alterados e o invariante é conservado.

FIM DA DEMONSTRAÇÃO DA PROPOSIÇÃO 3.

COROLÁRIO DA PROPOSIÇÃO 3:

Se no fim da iteração Topo1 = 0 Então necessariamente Topo2 = Topo3 = 0. Se no fim da iteração Topo1 =/= 0 e Topo2 = 0 então necessariamente Topo3 = 0.
A demonstração é imediata.
Este corolário permite optimizar a impressão das sequências evitando testes inúteis.

FIM DO COROLÁRIO.

PROPOSIÇÃO 4:

Topo1 só varia por incrementos de 1 durante a iteração.
A demonstração é imediata e decorre do facto de Carro só variar por incrementos de 1 logo quado Carro > Topo1 necessariamente Carro = Topo1 + 1.
Esta proposição completa os resultados que permitem optimizar a inicialização das sequências Li. Assim a sequência L1 não precisa de ser inicializada e as sequências L2 e L3 são preenchidas com Brancos apenas até aos Topos respectivos anteriores (inicialmente o Topoi anterior é 0).

FIM DA PROPOSIÇÃO 4.

5. Conclusões

O algoritmo aqui discutido é a parte central de LPSBACK. Digamos que os resultados e proposições são utilizados de tal forma que LPSBACK está construido sobre os mesmos.

Para Max é tomado um valor suficientemente grande para evitar todos os testes sobre este parâmetro já que o mesmo não influencia em nada a eficiência do algoritmo excepto no que toca à inicializacão pela primeira vez de L2 e L3.

As sequências Li são representadas em 3 vectores.

O algoritmo pode ser expandido até ao número de linhas sobrepostas que se desejar e permite, na forma em que foi inserido em LPSBACK, que o utilizador especifique que apenas pretende que um número de sobreposições inferior a três seja utilizado.

Para terminar a discussão do algoritmo seria necessário deduzir um predicado que em função de I permitisse concluir quais os caracteres I[i], 1 < = i < = N que seriam impressos.

A dedução deste predicado é complexa; no entanto, uma análise intuitiva permite concluir que as versões 1 e 2 do algoritmo garantem que nenhum caracter de I é desprezado desde que I não contenha sequências cuja reprodução implique mais de 3 sobreposições.

A transformação efectuada da segunda para a terceira versão impõe uma condição suplementar à anterior: I não deve conter movimentos superfluos do carro, isto é, movimentos não optimizados ou para colocar Brancos em cima de outros caracteres. Nomeadamente se I obedecer a uma das seguintes duas filosofias:

a) Sempre que é necessário sobrepor 2 caracteres c1 e c2 então a sub-sequência < c1, Backspace, c2> pertence a I.

b) Sempre que é necessário sobrepor caracteres envia-se uma linha seguida de tantos Backspaces como os necessários para fazer o Carro voltar à posição 1, seguidos de outra linha com as sobreposições e assim sucessivamente. Esta segunda filosofia tem em geral movimentos inúteis.

Então é garantido que sob o ponto de vista dos caracteres a desprezar todas as versões são equivalentes.

O método de construção e verificação aqui utilizado parece-nos ser o mais adequado já que qualquer tentativa de construir e verificar «de uma só vez» este algoritmo se nos afigura muito difícil.

6. Agradecimentos

Ao meu colega Luís Monteiro são devidos agradecimentos quer pela sua útil e atenta crítica deste documento, quer pelas suas sugestões no sentido de melhorar o algoritmo final.

LPSBACK e todo o trabalho aqui referido enquadra-se no projecto 3PES sobre Metodologia da Programação da Linha de Acção I do Centro de Informática da Universidade Nova de Lisboa suportado pelo INIC.

7. Referências

[1] José A. Legatheaux Martins, ESCRIBA — UM FORMATADOR DE TEXTO — LIÇÕES DE UMA EXPERIÊNCIA. Relatório interno do CIUNL — CIUNL 4/82. Lisboa. Março de 1982.

[2] José A. Legatheaux Martins. NICE — UMA EXPERIÊNCIA DE APLICAÇÃO DA TEORIA DOS AUTOMATOS À PROGRAMAÇÃO. Relatório interno do CIUNL. CIUNL — 5/82. Lisboa. Março de 1982.

[3] C. A. R. Hoare. An axiomatic basis for computer programming. CACM 10/1969 — pág. 576.