RESUMO

As definições recursivas aparecem como uma forma natural de traduzir um programa, existindo actualmente métodos poderosos e simples de provar a sua correcção. são no entanto pouco eficientes. A sua transformação num programa iterativo equivalente produzirá um programa que corresponde ao duplo objectivo de correcção e eficiência. Apresentamos um método de transformação que se baseia num princípio de generalização análogo aos utilizados em inteligência Artificial.

.1- INTRODUÇÃO

A década que agora findou marca o Início e o desenvolvimento de novas metodologias (formais) de programação. O custo proibitivo dos programas, devido sobretudo a sua correcção e manutenção. levou os informáticos a por em causa os "métodos" tradicionais de construção e validação e a procurar desenvolver, pare a sua substituição, metodologias formais. Foi DIJKSTRA que iniciou este trabalho. A sua frase (12) "testar um programa pode servir para provar que possui um erro, mas nunca que está correcto veio por em causa os testes como método de validação de programas. Do mesmo modo a sua frase (11) "a qualidade de um programador é inversamente proporcional ao número de goto's' que utiliza" chamou a atenção para a necessidade de manter o programa-texto (estático) como imagem do programa-executado (dinâmico), mediante a redução do número de estruturas de controle utilizadas, Dijkstra propôs então como solução e uso exclusivo de estruturas de controle concatenação, selecção, e iteração. As suas observações constituíram a base de uma nova metodologia do programação apelidada "Programação Estruturada": o programa á escrito de modo descendente, pela decomposição do problema inicial em sub-problemas (mais simples), Com esta metodologia, construção a prova de um programa são empreendidas simultaneamente. tendo-se para isso desenvolvido novos métodos de prova (método das asserções indutivas - FLOYD/HOARE (13.15). Todavia, alguns problemas surgem: (1) por um lado, a construção/prova apresenta a dificuldade de criação dos chamados invariantes de anel, como adiante se verá, (2) por outro lado, as estruturas do controle utilizadas vierem a revelar-se demasiadamente restritivas, originando por vezes programas de difícil leitura, ao contrário do que se pretendia, (3) finalmente, tendo sido constituído e desenvolvido em torno do problema da correcção. relegou a questão da eficiência pare segundo plano. com todas as consequências negativas que esse facto acarreta.

Por tudo isso começarem a surgir um pouco por toda a parte tentativas de desenvolvimento de outras metodologias. As dificuldades decorrentes do uso exclusivo da selecção e da iteração estão na origem de um estudo intensivo sobre os estruturas de controle (ver por exemplo ASCHCROFT (6) FLOYD/KNJTH (18) PETERSON/KASAMI/TOKURA (21) ou ainda ARSAC/RUGGIU (5). Como resultado. surgem vários trabalhos teóricos sobre a potência relativo das estruturas de controle, sobre a equivalência de programas cuja sistematização originará a Teoria dos Esquemas de Programa.

Igualmente importantes foram os trabalhos sobre a possibilidade de se obterem programas simultaneamente correctos e eficientes. Foi KNUTH (16) quem, pela primeira vez, propôs a transformação de programas como metodologia de programação, possibilitando assim a obtenção do duplo objectivo (correcção e eficiência)acima indicado. A ideia de base á a seguinte: escrever primeiramente fim programa correcto, usando para isso uma metodologia formal (programação estruturada) sem cuidar do aspecto da eficiência, em seguida transformar o programa, preservando a sua correcção, e tendo como objectivo central a busca da eficiência.

Na linha de Knuth, varios foram os trabalhos feitos (ARSAC (1.2.3) AUSLANDER/STRONG (7) GERHART (14) LOVEMAN (19) STANDISH (22) WEGBREIT (23) advogando a transformação de programas como método prático de construção de programas. enquanto outros (BURSTALL/DARLINGTON (9)) extraindo do estudo teórico prático das transformações resultados importantes que motivaram o aparecimento de sistemas(semi) automáticos de síntese de programas.

O método que vamos descrever situa-se dentro da Área transformacional e baseia-se nas ideias de

ARSAC/KODRATOFF (4) estudadas por Costa (10) . A aceitação do princípio de superação dos objectivos de correcção e eficiência, base de qualquer método transformacional, tem como corolário a a adopção das definições recursivas como ponto de partida para as transformações optimizantes e isto pelo menos, por duas ordens de razões: primeiramente. elas aparecem como um modo natural e (quase) directo de um problema em segundo lugar. Existem actualmente poderosos métodos de avaliação de programas recursivos (indução estrutural BURSTALL (8). indução computacional MANNA (20). As duas conjugadas tornam relativamente rápido a obtenção de um programa provadamente correcto e de simples leitura. As transformações farão o resto.

Ométodo recebe o nome de indutivo porque surge inicialmente como resposta as dificuldades do método indutivo do DIJKSTRA. A ligação entre os dois métodos leva-nos a apresentar primeiramente o de Dijkstra, o que faremos mediante a apresentação de um exemplo, simples, e da sua generalização.

2 - PROGRAMAÇÃO RECORRENTE

2.1- Exemplo

Suponhamos que pretendemos um programa que ordene, de forma crescente, um vector A de dimensão n.

Formalmente, podemos definir o problema por meio de predicados; de entrada (E) e de saída (S) seguintes:

P(A) significa que o vector B é uma permutação, do vector A. Para resolver o problema, vamos supor que num dado momento do cálculo possuímos uma solução parcial na forma de um vector A = P (A) subdividido em dois subvectores esquerdo (A' B) a direito (A' d) e tal que A' b está ordenado. Isto é (usando predicados)

Graficamente teremos a situação da figura 1.

É evidente que se por meio de um processo qualquer conseguirmos aumentar a dimensão do A'b estaremos mais próximos da solução do nosso problema inicial. A diferença entre k e n dá-nos uma ideia da distancia d (=k-n) a que nos encontramos da solução. Formalmente, podemos exprimir tal facto pela asserção

que significa que. se o predicado Q for verdadeiro e se d=O. Então o predicado S também será verdadeiro.

Resta-nos pois agora. partindo do princípio de que Q é verdadeiro definir uma estratégia que permita Ir aumentando a dimensão de A' b A estratégia utilizada deverá, por um lado manter o predicado Q verdadeiro e, por outro, diminuir estritamente a distancia d à solução. Uma possível hipótese de estratégia é a seguinte:

(1)- Se a dimensão de A'b for inferior a n comparar. da esquerda e para a direita. o primeiro elemento de A' d com os elementos de A' e'

(2)- Se esse elemento for maior do que todos os elementos de A' e começar o processo em (1) considerando agora o elemento comparado como pertencente a A' e'

(3)- Se existir um elemento a'(j) do A' e maior do que o primeiro elemento do A' d então:

(a)- Salvaguardar esse elemento:

b) - Deslocar todos os elementos de Ab a partir de j uma casa para a direita:

c) - Inserir o elemento salvaguardando na posição j

d) - Recomeçar em (1) com A'b incrementado de uma unidade.

Definido deste modo o algoritmo iterativo. facilmente o podemos traduzir num programa iterativo, Usando uma linguagem de tipo Algol:

Entre chavetas indicamos as asserções que são verdadeiras quando a execução do programa se encontra nosso ponto. A primeiro asserção (Q A d < 0) derivo das nossas hipóteses de partida. ou seja. que existe um subvector esquerdo do dimensão k totalmente ordenado. Mas como obter essa situação? Duma maneira geral. isso equivale a resolver. pelo mesmo método, um novo problema materializado pelo mesmo predicado do entrada E e tendo como predicado de saída Q. Sendo um subproblema do problema inicial é mais simples que este último. No nosso caso trate-se de obter um subvector esquerdo totalmente ordenado. Ume solução trivial consiste em fazer k:-1 . Acrescentando esta instrução ao início do programa anterior obtemos o programa final desejado.

2.: - Generalização

Se tentarmos generalizar o processo de resolução anterior, verificamos que este consiste em encontrar um algoritmo iterativo para um problema. materializado no predicado de saída S. mediante a sua decomposição num outro predicado Q(designado por invariante de anel) e numa distancia d, tais que

QA d= O - --) S.

A resolução do subproblema Q é feita aplicando a mesma metodologia. Esquematicamente, poderemos traduzir este método por:

Encontrar uma solução que valide Q

Transformar a solução parcial preservando Q e diminuindo estritamente d:

is (Q A d > 0) fait

Reflectindo sobre o método. verificamos que:

(1)- É fundamental a invenção (1) do invariante do anel Q, que funciona como uma verdadeira hipótese indutiva,
(2)- É necessário explicitar uma grandeza Inteira d que traduza a distancia a que nos encontramos da solução

(3)- Uma vez (1) e (2) resolvidos. é necessário definir uma estratégia de resolução do modo a que se mantenha o invariante, e que a distancia á solução diminui estritamente)

(4)- Após (1), (2) e (3) é ainda necessário resolver o subproblema Q

(5) - A adopção de predicados permite efectuar a prova da correcção parcial do programas

(6)- A existência da distancia d. que descresce estritamente a cada passagem pelo anel, permite completar a prova da correcção do programa (correcção total):

(7)- Finalmente. refira-se ainda que construção e prova do programa se processam conjuntamente.

Tudo o que até aqui foi dito é susceptível de ciar a ideia de que este método é simples de aplicar. Na pratica tal não e verifica. Ocupando o ponto (1) - escolha do variante - o papel central do método, a sua invenção, pois de invenção se trata nem sempre é trivial. Mas, mesmo após a sua descoberta. como definir a estratégia que nos permita aproximarmo-nos da solução ? será essa estratégia óptima?

· Com o se pode supor pelas questões apontadas. este método exige um grande esforço intelectual. bem como um conhecimento profundo do domínio do problema, por parte do programador.

O método transformacional que apresentamos permite ultrapassar essas dificuldades. fornecendo um processo mecanizável de obtenção de um programa Iterativo com a mesma forma do esquema anterior.

3 - MÉTODO TRANSFORMACIONAL

Como indicamos na introdução, este método baseia-se na utilização das definições recursivas. Deriva-se uma definição recursiva do problema a resolver e prova-se a sua correcção. Transforma-se de seguida essa definição num algoritmo iterativo mais eficiente preservando a correcção. Velemos com alguns exemplos como utiliza o método.

3.1- Primeiro Esquema

Exemplo

Suponhamos que se pretende um programa para o calculo da função factorial. Esta função pode ser definida recursivamente por:

fact(n) = si n= 0 alors 1 sinon fact(n-1) *n is (3.1.1)

Provemos. usando o método da indução estrutural, que esta função assim definida calcula efectivamente a função factorial.

caso de base: n=O

Se n=0 teremos pela definição (3.1.1) factor (0)=1 que por definição é igual a factorial do zero.

hipótese indutiva: para 0.1.2,3.,,,, m-t fact(n) calcula a função factorial.

Se n ¹0, fact(n) = fact(n-1) *n. Mas por hipótese fact (n-1)=(n-1) Logo fact(n) = (n-1): *n = n:.

Uma vez provada a sua correcção, vejamos como se podem resolver as quatro primeiras questões

indicadas em 2.2. de modo a obtermos o programa iterativo equivalente.

iii)- obtenção da invariante
Para se obter a expressão equivalente ao invariante utiliza-se uma técnica de generalização Substituem-se todas as constantes e subexpressões do apelo recursivo da função por variáveis. ou seja:
fact(n) = fact(u) * v (3.1.2)

(ii)- Obtenção das transformações preservando a hipótese indutiva.

Esta etapa é realizada em três fases que se baseiam na aplicação, por esta ordem, de: (1) substituição, (2) aplicação de propriedades dos operadores, (3) unificação. Concretamente. começa-se por:

(1)substituir a chamada da função fact (n) em (3.1.2) pelo corpo correspondente dado pelo membro direito de (3.1.1).

fact(n) = (si u = 0 alors 1 sinon fact(u-1) *u) * v
(3.1.3)

(2) aplicar as propriedades dos operadores (resta caso si alors sinon e multiplicação) de modo a transformar o apelo recursivo do membro direito de (3.1.3) numa expressão com a forma do membro direito de (3.1.2),
Fact(n)= si u=0 alors q * ',

sinon (fact(u-1) * u) * v
Fact(n) = si u=O alors v (3.1.4)

sinon fact(u-1) * (u *v)

(1 é elemento neutro da multiplicação e esta última a associativa).

(3) comparando fact(u) * v e fact(u-1) * (u *v). verificamos que possuem a mesma forma, pois é possível passar da primeira para a segunda substituindo as variáveis u e v por, respectivamente. (u-1)e (u * v). Diz-se então que dois termos unificam.

Iii Resolução do problema inicial
A obtenção do invariante baseia-se essencialmente na existência de um elemento neutro para a multiplicação. Fazenda u=1 e v=1 estabelece-se a condição:
fact(n) =fact(u) *v

(iv)- distancia à solução

É dada pelo valor de u. Quando u=o teremos a solução desejada, sendo o resultado dado por v pois u=O implica fact(n) v.

Partindo destes resultados. podemos escrever equivalente a (3.1.1), com a forma dada pelo esquema geral de 2.2

v: =1 u: =n

faire (fact(n) = fact(u) * v u = O)
si u= 0 alors fact(n) = fact(u)*

si non
v= v* u
u=u-1

Antes de passarmos a um esquema abstracto impõe-se três comentários. Em primeiro lugar. a ordem pela qual são efectuadas as alterações das variáveis u e v, após o alternante sinon . nao é arbitrário: o valor de v deve ser modificado antas de o mesmo acontecer a u. Em segundo lugar. utilizou-se uma variável suplementar para armazenar o resultado. Finalmente. o segundo membro de (3.1. 2) possui 2 variáveis enquanto o primeiro apenas uma. Isto significa que. na realidade, a função fact é calculada mediante a definição de uma função auxiliar g(u.v) a fact(u) *v com duas variáveis. Teremos então fact(n) - g(n,1) com

Generalização

Generalizamos a metodologia atras descrita. passando ao esquema abstracto de uma função definida recursivamente,

f(x) = si c(x) alors a(x)
sinon h(f(b(x)),d(x)) (3.1.5)

Supomos que o operador h possui um elemento neutro à direita e que é associativo, isto é
3x: Vx h(x.X) x
Yx,v.z r(h(x,y),z) = h(x,h(y.z))

Se a eficiência do programa obtido depende da existência destas(e /ou doutras) propriedades. a sua inexistência não torna impossível a desrecursivaçao. Veja-se, para o estudo deste caso, ARSAC/KODRATOFF ou COSTA .

i)- Generalização

O invariante obtém -se generalizando h(f(b(e)), d(x)), resultando

ftx) - htf(u),v) (3.1.6)

(ii)- Transformação

-substituição
f(x) - h(si c(u) alors a(u) sinon h(f(b(.u)). d(i)),v) (3.1.7)

-aplicação de propriedades

f(x) - si c(u) alors h(a(u).v) sinon h(h(f(b(u)). D(u)),v) devido as propriedades de si alors sinons
1 E

f(x) si c(u) alors h(a(u),v) sinon h(f(b(u)), h((d(u).v))

devido ao facto de h ser associativo.

-unificação

Possível mediante as substituições v:-(d(u),v) e u=b(u) por esta ordem.

(iii)- Estabelecimento da situação inicial

Como gx:Vx h(x,«) - x basta fazer u-x e v:-«,

(lv)- Condição de saída do anel e resultado

A condição de saída é dada por c(u). sendo o resultado igual a ti(a!u) vi.

Resumindo estes resultados num programa iterativo tipo Algol:

u:-x, v:-x:

3.2- Segundo Esquema

Esquema abstracto

Uma vez exposto o método nas suas ligações com a propagação indutiva, surge naturalmente a ideia de tentar aplicar os seus princípios

(generalização, sequência de substituição. aplicação de propriedade
Neste caso, a expressão generalizada terá que englobar as duas chamadas recursivas que aparecem no segundo ei alors sinon. Porque h 1 H2o primeiro passo do método consistira na transformação de h 1 e h2 em expressões equivalentes, mas que possuam a mesma composição de funções hi. Para tal. supomos que h1 e h2 possuem ambas um elemento neutro a direita, isto e.,

Obtemos então

Estas duas expressões facilmente se podem generalizar em
Fx = h1(h2(f(u),v, w)(3.2.3)
Este processo designa-se por busca do "menor generalizado" podendo ser estendido a um numero qualquer, finito,, de funções h1.

Uma vez obtido o menor generalizado, o processo desenrola-se de forma em tudo análoga ao indicado em 3.1, com a diferença de que agora existe mais do que um apelo recursivo e tratar. Omitindo o tratamento das condicionais si alors sinon, obtemos de (3.2.3). por substituição, duas expressões

A unificação das expressões (3.2.4) o (3.2.3) apenas sera possível mediante a sua transformação por aplicação de propriedades dos operadores h1 e h2 e/OU da sua composição t1h h e h1 h2 h2

4.3 - O problema da palavra

Duas outras questões teóricas se colocam ainda, ligadas a tentativa de unificação mediante o uso de propriedades dos operadores: a unificação é sempre possível? É arbitraria a ordem pela qual se aplicam as propriedades?

Estas questões constituem um caso particular do problema conhecido sob o nome de problema da palavra que se pode enunciar do seguinte modo: dadas duas palavras a e b e um conjunto de relações R existira sempre um processo computacional que. num número finito de passos, nos diga se a se pode transformar em B mediante o uso das relações R (ou vice versa)?

Assim formulado. este problema é irresolúvel. Existem, no entanto casos particulares em que ele tem solução: Knuth e Bendix demonstram que. se as relações R formarem um sistema de reescrita noetheriano (todas as palavras tem pelo menos uma forma normal) e confluente(se a forma existir é única). existe um algoritmo que resolve o problema. No nosso cano não é transparente que as nossas relações constituam um sistema de reescrita, Isto é que apenas sejam usadas num único sentido. A possibilidade de serem usadas nos dois sentidos introduz a possibilidade de ciclos infinitos na computação ao mesmo tempo que impede a definição de uma ordem coerente sobre as palavras. E uma questão que permanece ainda em aberto, pelo que, as duas questões acima colocadas, continuam ainda sem resposta definitiva

4.4 - Implementação

Não sendo nossa intenção tratar neste artigo as questões téoricas que o método coloca. limitamo-nos a aflora-los muito superficialmente (correndo com isto o risco de as tornar incompreensíveis e isto porque a sua eventual resolução tem Implicações ao nível da Implementação. Se existir um algoritmo de resolução para O(nosso)problema da palavra, o método torna-se bastante poderoso e rápido sendo um excelente auxiliar para a construção de programas iterativos de uma classe de funções relativamente extensa. Caso não exista solução. a implementação será essencialmente combinatória, logo menos eficaz do ponto de vista espaço/tempo.