

Linguagens Formais: de Inferência Recursiva à Derivação Mais a Esquerda

Giovani Facchini
giovani@exatas.unisinos.br

21 de novembro de 2003

Resumo

Este trabalho apresenta um programa feito na linguagem de programação C que recebe como entrada as regras de produção de uma Gramática Livre de Contexto e uma inferência recursiva para uma dada string da linguagem. O programa analisará esta entrada de dados e posteriormente irá gerar uma derivação mais a esquerda para a gramática.

Keywords: GLC, derivação, inferência recursiva

1 Introdução

As Gramáticas Livres de Contexto (GLC) são formas mais poderosas de gerar linguagens do que as linguagens regulares, pois estas sofrem bastante limitações em certos tipos de representações como, por exemplo, palavras palíndromas.

Este trabalho visa apresentar um programa que gere uma derivação mais a esquerda a partir de uma inferência recursiva, com isso conseguimos mostrar, de maneira informal ou prática como podemos fazer essa conversão que é provada formalmente. Esta é apenas uma entre as diversas conversões possíveis.

Este artigo está organizado da seguinte forma: na Seção 2 é discutida a metodologia empregada na resolução do problema. A Seção seguinte apresenta o algoritmo e o código. Finalmente, a Seção 4 tece comentários sobre as conclusões.

2 Metodologia

Para realizar o programa, foi utilizada a linguagem de programação C e um algoritmo para a realização da derivação. O programa trata o arquivo de entrada contendo as regras de produção e a inferência recursiva. O arquivo de entrada

```

E -> I
E -> E+E
E -> E*E
E -> (E)
I -> a
I -> b
I -> Ia
I -> Ib
I -> IO
I -> I1

a      I   5   -
b      I   6   -
b0     I   9   2
b00    I   9   3
a      E   1   1
b00    E   1   4
a+b00      E   2   5 6
(a+b00)    E   4   7
a*(a+b00) E   3   5 8

```

Figura 1: Formato do arquivo de entrada

terá uma aparência como a da figura 1, onde primeiro teremos as regras de produção da gramática depois teremos uma linha em branco e a seguir a inferência recursiva propriamente dita.

O arquivo de saída será gerado pelo programa e terá a derivação mais a esquerda de acordo com as regras da inferência. Um exemplo de um arquivo gerado é mostrado na figura 2, este arquivo conterá cada passo da derivação mais a esquerda. A cada derivação iremos utilizar \Rightarrow para representar a transição para o próximo passo.

A figura 3 mostra as equivalências que temos nas gramáticas livre de contexto, ou seja, com ela podemos ver que para chegar a derivação mais a esquerda a partir de uma inferência recursiva necessitamos, primeiramente, da árvore. Com essa árvore podemos fazer um caminhamento pré-fixe e então obter a derivação mais a esquerda.

$$\begin{aligned}
E &\Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow a * (E) \Rightarrow \\
a * (E + E) &\Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow \\
a * (a + I) &\Rightarrow a * (a + IO) \Rightarrow \\
a * (a + IOO) &\Rightarrow a * (a + b00) \Rightarrow
\end{aligned}$$

Figura 2: Formato do arquivo de saída

3 Algoritmo

O programa principal trata primeiramente os arquivos de entrada, ou seja, faz o *parsing* da entrada. Cada linha que representa uma regra de produção é armazenada em uma estrutura que tem o formato descrito na figura 4, onde *var* indica a variável e *producao* a sua respectiva produção.

O próximo passo é fazer a leitura da inferência recursiva e esta será guardada em uma estrutura com o formato da figura 5. Nesta *string* é o campo onde fica a criação, *var* indica a variável utilizada pra gerar a string, *producao* é o número de regra de produção utilizada para a criação da string e *usado1* e *usado2* contém o número das strings utilizadas para criar a string atual.

Depois de feita a leitura dos dados de entrada é utilizado um algoritmo recursivo na forma de um caminhamento pré-fixo em árvore. Para o algoritmo é passado a última linha da Inferência recursiva e então ele vai percorrendo as strings *usado1* e *usado2* para gerar as string (utilizando as regras de produção). Toda vez que a função recursiva é chamada a função *muda_string* atualiza a variável *saida* que é utilizada para manter a atual string de saída que representa a derivação naquele momento. Quando a função *muda_string* faz as modificações na string de saída ela chama a função *imprime_saida* que irá colocar a nova string no arquivo de saída pra gerar a derivação.

4 Conclusão

Este trabalho mostra como é possível, a partir das regras de produção de uma GLC e sua respectiva inferência recursiva para a criação de uma string da gramática, como podemos convertê-la em uma derivação mais a esquerda. Esse passo necessita que se passe pela forma de árvore. Essa passagem para a derivação foi, algoritmicamente, simples, pois ela é feita diretamente.

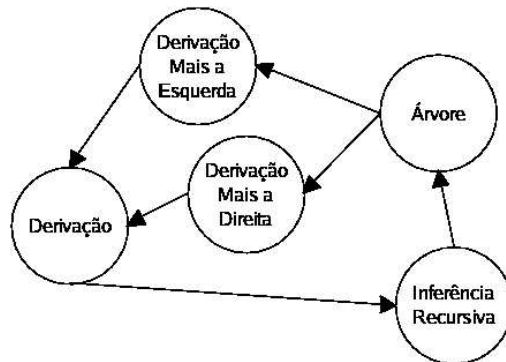


Figura 3: Equivalências

```

} typedef struct producao {
    char var[5];
    char produzido[10];
} producao_type;

```

Figura 4: Estrutura das Produções

```

typedef struct inferencia {
    char string[40];
    char var[5];
    int producao;
    int usado1;
    int usado2;
} inferencia_type;

```

Figura 5: Estrutura da Inferência

Olhando para o código percebemos a complexidade criada pelo uso da linguagem C. Essa complexidade obscurece o código e faz com que a interpretação de partes sem importância para o algoritmo definitivo sejam exaustivamente observadas para uma melhor compreensão. E por esse motivo que existem linguagens de mais alto nível, como Java, que permitem uma leitura de código muito mais “livre de sujeiras” devido a parsers e alocações desnecessárias. Com essas podemos criar programas mais legíveis com um alto nível de abstração.

Como trabalhos futuros esperamos utilizar a linguagem Java para desenvolvimento e pode ser criado programas que façam todas as equivalências permitidas pelas Gramáticas Livre do Contexto.