Aplicação do Problema da Gramática Mínima ao Projeto do Aspecto Sintático de Linguagens de Programação

Ivone Penque Matsuno[1] e Ricardo Luis de Azevedo da Rocha[1]

[1] Departamento de Engenharia de Computação e Sistemas Digitais – PCS
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav.3, nº.158 - 05508-900 – São Paulo – SP – Brasil

{ivone.matsuno,luis.rocha}@poli.usp.br

Abstract. *This paper introduces a proposal of method for grammar inference of programming languages based in source-code examples. The strategy proposed in this method uses the solution of the smallest grammar problem for conversion of text representation into grammars that form the base of an adaptive model, which generalizes the respective resulting grammars from positive and negative examples.*

Resumo. *Este artigo apresenta uma proposta de método de inferência de gramáticas de linguagens de programação baseado em exemplos de código-fonte. A estratégia proposta no método utiliza a solução do problema da gramática mínima para a conversão da representação de textos em gramáticas, que formam a base de um modelo adaptativo o qual generaliza as respectivas gramáticas resultantes a partir de exemplos positivos e negativos.*

1. Introdução

A inferência de uma gramática é uma tarefa que tem sido estudada na literatura através de diversos modelos teóricos distintos, e com propósitos diversos. Um exemplo disso é o chamado "problema da gramática mínima", cujo objetivo é encontrar a menor gramática para um determinado conjunto de palavras em um texto [Charikar et al 2002]. Outro exemplo é o de inferir a gramática regular através de um conjunto de palavras, resultantes de expressões regulares [JoséNeto 1998]. As diferenças entre os objetivos acabam por conduzir a estratégias e métodos distintos para o tratamento de cada caso. Neste trabalho propõe-se um método de inferir a gramática de uma linguagem de programação. Este método é baseado na estratégia de se encontrar, usando o problema da gramática mínima, uma formulação adequada para a composição dos átomos da linguagem, e, a partir disso e da exposição a exemplos positivos e negativos da linguagem de programação, encontrar uma gramática adequada para a linguagem.

Através de exemplos fornecidos de código-fonte em uma linguagem de programação, o objetivo é, então, propor um modelo para inferir uma gramática que represente a sintaxe da linguagem em questão.

Nesta pesquisa o objetivo é inferir uma gramática para uma linguagem de programação, entretanto, pode-se vislumbrar outras aplicações dentro da Teoria de Linguagens, como, por exemplo, o estudo de linguagens naturais, ou ainda, estudar a

representação sintática de aspectos usualmente associados à chamada semântica estática de linguagens de programação [JoséNeto 1998].

A primeira questão a ser tratada é como representar uma seqüência de caracteres de um código-fonte em uma gramática. Considerando esse aspecto, utilizou-se a aplicação das soluções propostas para o problema de geração da gramática mínima. Em uma definição simples, uma gramática é mínima quando gera somente uma única sentença. Aplicando-se tais soluções para esse problema obtém-se, assim, a representação de um código-fonte por uma gramática.

As principais aplicações para o problema da gramática mínima são a compressão de dados e o reconhecimento de padrões, portanto, a maioria das soluções propostas para esse problema baseia-se na repetição de símbolos da sentença para a geração da gramática. Devido ao fato de que, em geral, um código-fonte possui diversos tipos de repetição para referenciar suas estruturas de controle de fluxo, identificadores, expressões, etc, a aplicação direta do método de geração da gramática mínima permite identificar alguns agrupamentos especiais, como por exemplo, as palavras reservadas da linguagem. Entretanto, uma mesma estrutura pode ser representada diferentemente em uma gramática apenas por apresentar valores numéricos e identificadores distintos. Pode, ainda, ocorrer o contrário, sendo a repetição muito freqüente, diversas estruturas são representadas da mesma forma na gramática.

Antes de realizar a conversão do código-fonte em gramática, realiza-se a análise léxica do mesmo, para que a gramática mínima gerada se aproxime da gramática da linguagem de programação. Aplicando-se o mecanismo de conversão com pequenas modificações para que atue sobre os átomos da linguagem, os resultados gerados tornam-se mais relevantes para a inferência da gramática da linguagem de programação.

Além do tratamento da representação do código-fonte, é necessário um processo que possa relacionar as diversas gramáticas geradas, identificando os ciclos sintáticos existentes e distinguindo quando ocorre um novo ciclo ou quando este já foi representado. Para que os resultados se aproximem de uma gramática mais adequada para a linguagem utiliza-se de mais uma etapa, contendo também exemplos de código-fonte que não estão corretos. Esta etapa do método de inferência de uma gramática para uma linguagem de programação, baseia-se no treinamento de um modelo adaptativo semelhante ao proposto em [José Neto 1998].

A figura 1 apresenta as etapas da proposta, das quais as duas primeiras estão implementadas e a última em desenvolvimento.

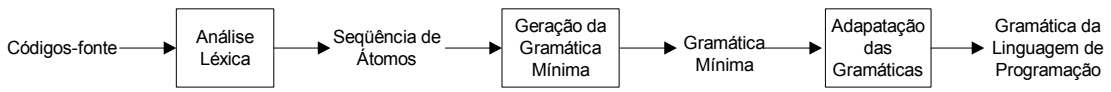


Figura 1 - Etapas do método proposto

Este artigo está dividido em cinco seções. Na seção 2 são apresentados o método de geração da gramática mínima e um exemplo ilustrativo. O relacionamento entre a gramática mínima e a gramática da linguagem de programação é apresentado na seção 3. O método proposto é apresentado na seção 4. Por fim, na seção 5, encontram-se as conclusões e propostas de trabalhos futuros desta pesquisa.

As principais contribuições desta pesquisa são a apresentação de uma proposta para inferência gramatical baseada em dispositivos adaptativos e o uso do problema da gramática mínima como base para obter um conjunto de gramáticas.

2. O Problema da Gramática Mínima

O problema da gramática mínima, conforme descrito na introdução deste artigo, pode ser entendido como a busca pela menor gramática capaz de gerar uma sentença. Seja x uma sentença sobre um alfabeto, tal que $|x| = n$, o problema da gramática mínima é gerar uma gramática G_x, em que o tamanho de G_x seja menor que a sentença x. O tamanho da gramática será considerado pelo total de símbolos à direita de suas produções [Kieffier 2000, Nevill 1997]. O problema da geração da gramática mínima tem sido abordado por diferentes linhas de construção e de tipos de gramáticas, como por exemplo, as soluções apresentadas em [Kieffer 2000], [Charikar et al 2002], e [Ryterr 2002].

Em [Charikar et al 2002] este problema é estudado, e uma solução para ele é apresentada. A solução baseia-se na construção de um algoritmo que constrói uma gramática para cada sentença, buscando a de menor complexidade. No caso utiliza-se uma versão, baseada em gramáticas, da medida de complexidade de Kolmogorov definida sobre uma máquina de Turing. Esse algoritmo baseia-se na repetição dos símbolos no texto de entrada e nas propriedades de Gramáticas Binárias Balanceadas G_{bb}. Essa solução e respectiva implementação são utilizadas em uma das etapas no método de inferência proposto seguindo [Kessey 2003].

2.1 O Algoritmo

A gramática inicial $G_x = (N, T, P, S_0)$ tem os conjuntos vazios. O conjunto de produções de G_x é o único conjunto que será alterado diretamente, o qual será representado no método de geração por uma lista ativa de produções. Ao fim do processo, os demais conjuntos serão derivados a partir dessa lista.

Dado um texto x para gerar a gramática mínima G_x, a primeira etapa consiste em obter agrupamentos de símbolos que se repetem. Para tanto, é utilizado o algoritmo de compressão LZ77 [Ziv 1977], [Ziv 1978]. Esse algoritmo obtém uma lista L_z de pares ordenados, (p_1, l_1), ... , (p_n, l_n), em que p_i indica a posição inicial da repetição no texto e l_i o comprimento da subcadeia. Cada par é uma subcadeia w_i do texto e a concatenação das subcadeias gera o texto original. Em particular, quando p_i é igual a zero, a subcadeia é representada por um único símbolo na posição l_i do texto. A primeira subcadeia w_1, necessariamente é um desses casos e, portanto, será inserida diretamente na lista ativa. Desta forma tem-se: $P=\{S_0 \rightarrow S_1, S_1 \rightarrow w_1\}$.

Cada w_i da lista L_z deverá ser derivado pelas produções da gramática. Essas produções serão construídas de forma ascendente (*bottom-up*) através de operações para adicionar um não-terminal na árvore de derivação, para que esta derive (tenha a expansão) para um par, uma seqüência, ou ainda uma subcadeia, baseando-se nas propriedades de Gramáticas Binárias Balanceadas [Charikar et al 2002].

Desta forma, para cada subcadeia w_i deverão existir uma ou mais produções para gerá-la. Logo, o conjunto de produções P será modificado através da inserção de novas regras de produção, ou da alteração ou remoção das regras já existentes. Conseqüentemente, o conjunto de não-terminais N também será redefinido. Para definir

quais serão as produções necessárias para gerar w_i serão avaliados o conjunto de produções já existente e a árvore de derivação construída em paralelo.

Ao analisar cada w_i haverá dois casos possíveis: w_i não é gerado pelas produções de G_x; ou w_i pode ser gerado por uma ou mais produções que já existem em G_x. No primeiro caso, uma nova regra de produção $S_k \rightarrow w_i$ é criada e w_i é representada na árvore de derivação.

No segundo caso, também existem duas possibilidades:

- ***w_i* pode ser gerado por uma única regra de produção X_i:**, nesse caso, representa-se a subcadeia na árvore de derivação através da operação para adicionar uma subcadeia em G_{bb}. Essa operação pode alterar os símbolos terminais e acrescentar uma ou mais produções em G_{bb}; entretanto, em G_x, será inserida no fim da lista ativa uma única produção: $M_i \rightarrow X_i$.
- ***w_i* é gerado por mais do que uma produção:** nesse caso, deve-se encontrar a seqüência de não-terminais da lista ativa que contém a expansão w_i . A seqüência de não-terminais é da forma $X_i \ldots X_t$. O primeiro e o último não-terminais da seqüência, X_i e X_t, são compostos por duas partes: uma que não pertence a w_i ($X_{iprefixo}$,$X_{tsufixo}$), e outra que pertence($X_{isufixo}$, $X_{tprefixo}$).Essas devem ser separadas para que a seqüência tenha expansão exatamente para w_i. Gerando as seguintes produções: $M_i \rightarrow X_i \ldots X_t$, a qual substituirá a seqüência $X_i \ldots X_t$. na lista ativa e $N_i \rightarrow X_{isufixo} \ldots X_{tprefixo}$ .que será inserida no fim da lista.

Ao final dessas operações, G_x deverá ter seus conjuntos definidos para que gerem somente a sentença do código-fonte *x*. A árvore de derivação não é gerada a partir de G_x e sim das propriedades de balanceamento. O exemplo a seguir resume esse método de geração da gramática mínima.

2.2. Exemplo de Geração da Gramática Mínima.

Para exemplificar o método para a geração da gramática mínima qualquer texto poderia ser escolhido. Se o texto não possui repetições de seus símbolos, nenhum agrupamento será identificado e a gramática terá uma única produção da forma $S \rightarrow x$. Para visualizar a conversão desejada em nossa proposta, o texto *x* é o trecho do código-fonte em uma linguagem de programação qualquer. Enfatiza-se que na geração da gramática mínima, segundo o algoritmo de Charikar, nada se tem sobre a linguagem de programação, não se conhece seus átomos, nem sua sintaxe.

Embora os trechos de código-fonte nas figuras deste artigo apresentem-se com os caracteres de controle, esses são eliminados para a geração da gramática mínima. Por exemplo, o código-fonte da figura 2 é analisado da seguinte forma: "*ifx>2thenx:=x+1ify=2theny:=xifz=3theny:=z+y*". Para esse exemplo apresenta-se todas as etapas do método de geração. Para os demais exemplos neste artigo serão apresentados e comentados apenas os resultados finais.

```
if x>2 then x:=x+1
if y=2 then y:=x
if z=3 then y:=z+y
```

Figura 2 – Exemplo de código-fonte 1

A primeira etapa é realizar o agrupamento dos símbolos pelas repetições dos mesmos através da aplicação do algoritmo LZ77. Com base na repetição dos símbolos, os agrupamentos obtidos foram: *if*, *2then*, *:=x*, *theny:=*.

Inicialmente a gramática mínima G_x tem apenas um símbolo inicial S_0. Os primeiros lexemas identificados possuem comprimento um e não são repetidos. Tratando esses símbolos a partir do primeiro caso ($|w_i| = 1$), cada novo símbolo é inserido no fim da lista ativa. Para cada símbolo também é criado um não-terminal com expansão (derivação) para esse símbolo. Cada não-terminal deverá produzir uma árvore de derivação com as propriedades de G_{bb}. Entretanto, neste caso, por terem expansão para um único símbolo as árvores não são criadas.

$S_0 \rightarrow S_1\ S_2\ S_3\ \ldots\ S_{15}$ (expansão de S_0 é if x>2 then x:=x+1)
$S_1 \rightarrow$ i
$S_2 \rightarrow$ f
. . .
$S_{15} \rightarrow$ 1

Figura 3 – Gramática G_x com símbolos não-terminais com expansão para um único símbolo.

Ao analisar a próxima subcadeia, w = *if*, será utilizado o segundo caso ($|w_i| > 1$), pois esse lexema já está contido na expansão de um ou mais símbolos da lista ativa. O algoritmo obtém as seguintes listas de não-terminais:

- Lista de não-terminais com expansão para w (B)
- Lista de não-terminais antes de w (X)
- Lista de não-terminais após de w (Y)

Desta forma obtém-se as seguintes listas: $B_1 = S_1S_2$, $X_1 = \varepsilon$, $Y_1 = S_3S_4...S_{15}$. É necessário criar um não-terminal M_1 com expansão B_1, através da operação para adicionar uma seqüência de símbolos não-terminais e então criar a árvore de derivação para M_1 como apresentado na figura 4.

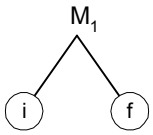


Figura 4 – Árvore de derivação gerada para um símbolo não-terminal através da operação para adicionar seqüência.

Após criar M_1 a seqüência de símbolos B_1 deverá ser substituída na lista ativa por M_1, desta forma as produções da gramática G_x são como na figura a seguir.

$S_0 \rightarrow M_1\ S_3 \ldots S_{15}$ (expansão de S_0 é *if x>2 then x:=x+1*)
$M_1 \rightarrow S_1S_2$ (expansão *if*)
$S_1 \rightarrow$ i
$S_2 \rightarrow$ f
. . .
$S_{15} \rightarrow$ 1

Figura 5 – 1ª etapa:G_x após criação de um símbolo não-terminal com expansão para uma das subcadeias já existentes em G_x

Após separar o prefixo e o sufixo da expansão B que não fazem parte de *w*, nesse caso $B_{prefixo}$ e B_{sufixo} são nulos, é criado apenas um não-terminal N_1 com expansão M_1, que será adicionado ao fim da produção inicial de G_x, que resulta em alterações na gramática que são representadas na figura 6.a. Como $N_1=M_1$, tem-se as simplificações na figura 6.b.

(a)

$S_0 \rightarrow M_1\ S_3 \ldots S_{15}\ N_1$
(expansão de S_0 é *if x>2 then x:=x+1*)
$N_1 \rightarrow B_{1prefixo}\ M_1\ B_{1prefixo}$
$M_1 \rightarrow S_1S_2$ (expansão *if*)
$B_{1prefixo} \rightarrow \varepsilon$
$B_{1sufixo} \rightarrow \varepsilon$
$S_1 \rightarrow$ i
. . .
$S_{15} \rightarrow$ 1

(b)

$S_0 \rightarrow M_1\ S_3 \ldots S_{15}\ M_1$
(expansão de S_0 é if x>2 then *x:=x+1 if*)
$M_1 \rightarrow S_1S_2$ (expansão *if*)
$S_1 \rightarrow$ i
$S_2 \rightarrow$ f
. . .
$S_{15} \rightarrow$ 1

Figura 6 – 2ª etapa:G_x após criação de um símbolo não-terminal com expansão para uma das subcadeias já existentes em G_x

Este processo é repetido criando-se novas produções e suas respectivas árvores de derivação. Para tornar mais claro o exemplo, ilustra-se a seguir a etapa em que o método de geração separa os símbolos com expansão *2then* e *:=x* para obter *then* e *:=*. Antes de analisar a subcadeia *theny:=* a gramática G_x e as árvores de derivação encontram-se com a seguinte configuração:

$S_0 \rightarrow M_1\ S_3\ S_4\ M_2\ S_{10}\ M_3\ S_{14}\ S_{15}\ M_1\ S_{16}\ S_{17}\ M_2\ S_{18}\ M_3\ M_1\ S_{19} \ldots S_{21}$
(expansão de S_0 é *if x>2 then x:=x+1 if y=2 then y:=x if z >3*)
$M_1 \rightarrow S_1S_2$ (expansão *if*)
$M_2 \rightarrow S_5 \ldots S_9$ (expansão *2then*)
$M_3 \rightarrow S_{11} \ldots S_{13}$ (expansão *:=x*)
$S_1 \rightarrow$ i , $S_2 \rightarrow$ f , . . .,$S_{15} \rightarrow 1$,$S_{16} \rightarrow$ y,
$S_{17} \rightarrow$ =,$S_{18} \rightarrow$ y,$S_{19} \rightarrow$ z,$S_{20} \rightarrow$ =, $S_{21} \rightarrow 3$

Figura 7 - G_x após a análise dos 25 primeiros lexemas

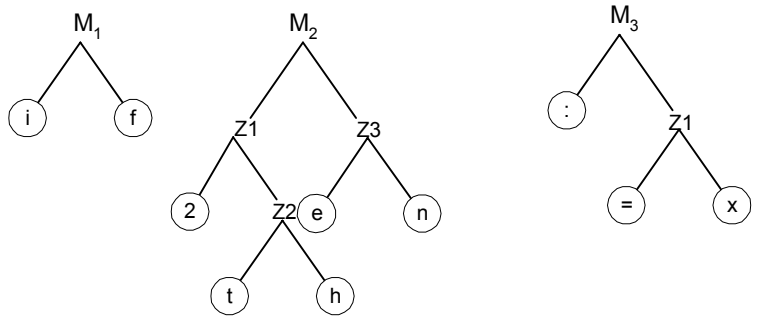


Figura 8 – Árvore de derivação dos símbolos não-terminais de G_x da gramática da figura 7

Ao analisar o lexema *theny:=* tem-se a expansão das seguintes listas de não-terminais:

- $B_4 = \mathbf{M_2\ S_{18}\ M_3}$
- $M_4 \rightarrow \mathbf{M_2\ S_{18}\ M_3}$
- $B_{4prefixo}$ = *then* ($X_4 \rightarrow$ *then*)
- $M_2 \rightarrow 2X_4$
- $B_{4sufixo}$ = *:=* ($Y_4 \rightarrow$ *:=*)
- $M_3 \rightarrow Y_4 x$
- $N_4 \rightarrow X_4\ S_{18}\ Y_4$

Essas produções serão alteradas ou incluídas em G_x como na figura 9.

$S_0 \rightarrow M_1\ S_3\ S_4\ M_2\ S_{10}\ M_3\ S_{14}\ S_{15}\ M_1\ S_{16}\ S_{17}\ \mathbf{M_4}\ M_1\ S_{19}\ldots S_{21}\mathbf{N_4}$
(expansão de S_0 é *if x>2 then x:=x+1 if y=2 then y:=x if z >3 then y:=*)
$M_1 \rightarrow S_1S_2$ (expansão *if*)
$M_2 \rightarrow S_5X_4$ (expansão *2then*)
$M_3 \rightarrow Y_4S_{13}$ (expansão *:=x*)
$M_4 \rightarrow M_2S_{18}M_3$ (expansão *2theny:=x*)
$N_4 \rightarrow X_4\ S_{18}\ Y_4$ (expansão *theny:=*)
$X_4 \rightarrow S_6\ S_7\ S_8\ S_9$ (expansão *then*)
$Y_4 \rightarrow S_{11}\ S_{12}$ (expansão *:=*)

Figura 9 – Alterações em G_x após análise do lexema *theny:=*

Ao fim do processo a gramática G_x encontra-se da seguinte forma:

$S_0 \rightarrow M_1\ S_3\ S_4\ M_2\ S_{10}\ M_3\ S_{14}\ S_{15}\ M_1\ S_{16}\ S_{17}\ \mathbf{M_4}\ M_1\ S_{19}\ldots S_{21}N_4\ S_{22}\ S_{23}\ S_{24}$
(expansão de S_0 é *if x>2 then x:=x+1 if y=2 then y:=x if z >3 then y:=z+y*)
$M_1 \rightarrow S_1S_2$ (expansão *if*)
$M_2 \rightarrow S_5X_4$ (expansão *2then*)
$M_3 \rightarrow Y_4S_{13}$ (expansão *:=x*)
$M_4 \rightarrow M_2S_{18}M_3$ (expansão *2theny:=x*)
$N_4 \rightarrow X_4\ S_{18}\ Y_4$ (expansão *theny:=*)
$X_4 \rightarrow S_6\ S_7\ S_8\ S_9$ (expansão *then*)
$Y_4 \rightarrow S_{11}S_{12}$ (expansão *:=*)

Figura 10 – Gramática G_x ao final do processo

Para finalizar, a gramática mínima G_x gerada deve ser simplificada realizando-se as seguintes substituições:

- Substituir os símbolos não-terminais que têm expansão de comprimento igual a um pelo próprio símbolo terminal.
- Substituir os símbolos não-terminais que têm como regra de produção uma substituição por apenas um único símbolo não-terminal, que por sua vez também é substituído por um único não-terminal, isto é, encontrar o fecho transitivo de regras de produção cuja substituição seja de um não-terminal por outro, como, por exemplo, A→B e B→C. Neste caso substituir por A→C.

Após essas substituições, G_x apresenta as produções apresentadas na figura 11.

$S_0 \rightarrow M_1$ x> M_2 x M_3 +1 M_1 y=M_4 M_1 y z=3 N_4 z+y
(expansão de S_0 é *if x>2 then x:=x+1 if y=2 then y:=x if z =3 then y:=z+y*)
$M_1 \rightarrow$ *if*
$M_2 \rightarrow$ *2* X_4
$M_3 \rightarrow Y_4$ *x*
$M_4 \rightarrow M_2$ *y* M_3
$N_4 \rightarrow X_4$ *y* Y_4
$X_4 \rightarrow$ *then*
$Y_4 \rightarrow$ *:=*

Figura 11 – G_x final com simplificações

3. Relação entre Gramática Mínima e Indução de Gramática para Linguagem de Programação

Ao aplicar a solução proposta por Charikar [Charikar et al 2002] em diversos exemplos de código-fonte de uma linguagem de programação, a fim de encontrar sua respectiva gramática mínima, observou-se que, dependendo dos exemplos, os agrupamentos e palavras especiais da linguagem em questão poderiam ser definidos ou não na gramática gerada. Isso pode ser observado nas figuras 12 e 13, em que são apresentados o código-fonte e a gramática mínima correspondente.

No exemplo da figura 12, os lexemas obtidos podem ser associados às palavras reservadas da linguagem, ou operador especial. Entretanto, o método não representa a seqüência *"A...B...C...D...E"* que se repete separadamente como expansão de um novo não-terminal.

```
for i:=1 to 2 step 1 do
  for j:= 3 to 4 step 5 do
        m:=i+j
```

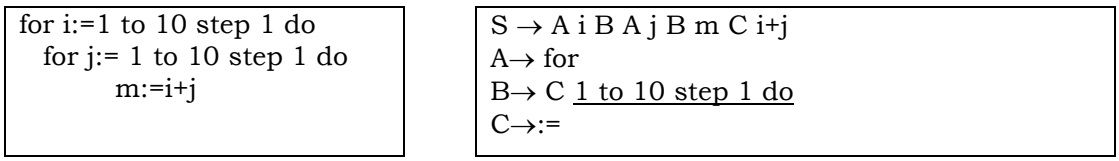
(a) Código-fonte 2

S → A i B 1 C 2 D 1 E A j B 3 C 4 D 5 E m B
i+j
A→for
B→:=
C→ to
D→ step
E→do

(b) Gramática Mínima para código-fonte 2

Figura 12 – Exemplo para uma linguagem de programação

Já no exemplo da figura 13, o método inclui os valores que se repetem no agrupamento dos caracteres, formando lexemas que não são necessariamente palavras reservadas.



(a) Código-fonte 3 (b) Gramática Mínima para código-fonte 3

Figura 13 – Exemplo da figura 12 com pequenas modificações

Ao gerar uma gramática para um código-fonte em uma determinada linguagem de programação, pouco se pode associar automaticamente da gramática mínima gerada à gramática da linguagem de programação. O que não se constitui em nenhum absurdo, afinal o método de Charikar não tem esse objetivo.

Entretanto, os resultados obtidos através da aplicação direta do algoritmo de geração da gramática mínima sem restrições e sem as ampliações utilizadas nesta seção, apresentam-se como exemplo de gramáticas que podem ser úteis na aproximação da gramática da linguagem de programação. Para tanto, é necessário complementar o método com algum mecanismo que possa generalizar as gramáticas mínimas obtidas.

4. Proposta de Método de Indução de Gramáticas para Linguagens de Programação Baseado em Exemplos Gerados pelo Método de Geração da Gramática Mínima

Os resultados obtidos na seção anterior mostram que é possível utilizar o algoritmo de Charikar com o intuito de obter uma gramática adequada para uma linguagem de programação. Entretanto, a gramática não poderá ser obtida diretamente, conforme ilustrado anteriormente.

As dificuldades enfrentadas na geração de uma gramática adequada, preferencialmente a mais próxima possível da gramática original para a linguagem sob estudo, residem na generalização dos casos estudados. Dois aspectos observados anteriormente são que a disposição do texto-fonte, e a ordem de exposição do algoritmo aos exemplos em estudo, influenciam fortemente no resultado do algoritmo.

Estas observações conduziram ao método aqui proposto, e, como primeiro resultado, é necessário obter os átomos, isto é, utilizar o algoritmo de geração da gramática mínima aos átomos, e não ao texto-fonte diretamente. O objetivo desta mudança é propiciar um grau maior de generalização desde o início do método.

O outro aspecto tem seu reflexo no uso da gramática de uma linguagem de programação, que deve produzir somente programas sintaticamente corretos (ou, no processo inverso, utilizando a gramática para produzir um dispositivo reconhecedor para a linguagem). Isto sugere que a exposição aos exemplos deve contemplar também casos negativos.

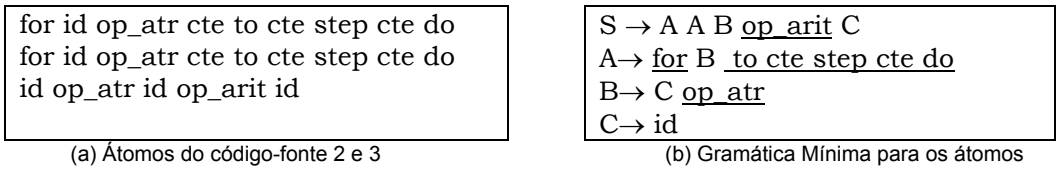
4.1 Geração da Gramática Mínima para os Átomos da Linguagem de Programação

Com o intuito de tornar as gramáticas mínimas geradas exemplos mais significativos no processo de indução de gramáticas para linguagens de programação, propõe-se, antes de iniciar o processo de geração da gramática mínima, a realização da análise léxica do código-fonte. Desta forma, uma seqüência de átomos da linguagem é analisada, evitando com que diversos identificadores e valores numéricos distintos dificultassem a generalização da gramática.

Na figura 14 é apresentada a gramática mínima gerada para os átomos gerados a partir da analise léxica dos exemplos das figuras 12 e 13. Observa-se que embora os códigos sejam diferentes, os átomos obtidos para ambos os códigos são os mesmos.Para esses exemplos, a classificação dos átomos da linguagem é dada pela tabela 1.

Tabela 1 – Classificação dos Átomos da Linguagem

Classe Átomo	Descrição	Sigla Átomo
identificador	letra.(letra | dígitos)*	id
constante	(dígitos)*	cte
op_atribuição	:=	op_atr
op_artmético	+ | - | * | / | ^	op_arit
for	for	for
to	to	to
step	step	step
do	do	do

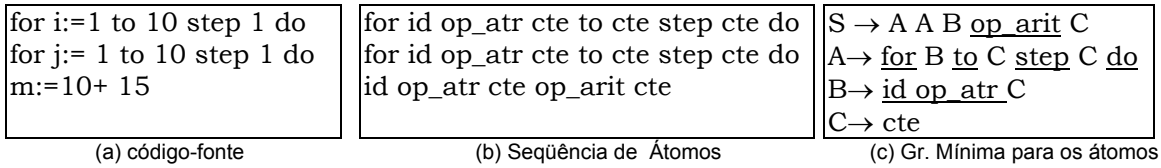


(a) Átomos do código-fonte 2 e 3 (b) Gramática Mínima para os átomos

Figura 14 – Gramática mínima obtida através dos átomos da linguagem de programação

Através dessa pequena modificação, as gramáticas geradas tornaram-se mais abrangentes, permitindo que exemplos de código-fonte distintos na escrita, mas que apresentam o mesmo significado sintático, sejam tratados da mesma forma.

No exemplo da figura 15, o código-fonte da figura 13 é apresentado com uma pequena modificação que influencia o resultado obtido:



(a) código-fonte (b) Seqüência de Átomos (c) Gr. Mínima para os átomos

Figura 15 – Gramática mínima para os átomos da linguagem de programação

Esta primeira modificação permitiu um grau de abrangência e generalidade maior do que o observado anteriormente, quando foi utilizado o algoritmo de Charikar diretamente ao código-fonte. Entretanto, este resultado ainda não é o suficiente, já que ainda não contempla todas as possíveis variações do código fonte, conforme observado na figura 15.

Um outro questionamento diz respeito aos programas com texto-fonte sintaticamente incorreto. O algoritmo utilizado pelo método ainda não é capaz de distinguir um exemplo correto (positivo) de um errado (negativo), não sendo, portanto, capaz de inferir se determinado texto-fonte errado for gerado pela gramática.

4.2 Estratégia de Consolidação da Gramática Resultante através de Exemplos

A generalização e a garantia de que a gramática inferida gerará somente programas corretos poderá ser garantida através da introdução de mais uma etapa no método. Esta última etapa deverá trabalhar com as possíveis gramáticas induzidas até então, e, através da exposição a exemplos positivos e negativos, efetuar as últimas alterações na gramática.

A proposta para esta etapa é utilizar um dispositivo adaptativo para induzir a gramática resultante. A base para a proposta é o algoritmo apresentado em [JoséNeto, 1998], que utiliza o autômato adaptativo como base para a indução de dispositivos reconhecedores de linguagens regulares (autômato finito).

Neste método a etapa final (ainda não implementada) tem como base para a indução da gramática resultante da gramática adaptativa. Na próxima seção apresenta-se brevemente o conceito de dispositivo adaptativo geral.

4.2.1. Dispositivos Adaptativos

Dispositivos formais adaptativos são definidos como aqueles cujo comportamento varia dinamicamente ao longo de sua operação, como reação à recepção de determinados estímulos em sua entrada. Em dispositivos adaptativos, essa reação faz parte das características do mesmo, e deve ocorrer espontaneamente, sem a interferência do operador ou de outros agentes externos.

Para incorporar tais características em um dispositivo adaptativo, cujo comportamento seja completamente descrito por um conjunto de regras, deve-se modificar correspondentemente o conjunto de regras que o definem. Para tornar essas modificações viáveis, os dispositivos devem antecipar todas as possíveis alterações que seu comportamento possa sofrer. Assim, sempre que ocorrer alguma situação que exija modificação do conjunto de regras de operação, o dispositivo estará apto a fazê-la. Note-se que não há necessidade de que todas essas alterações estejam integralmente definidas a priori, embora se exija que o dispositivo esteja apto a determiná-las em qualquer situação em que as modificações possam fazer-se necessárias.

Os dispositivos adaptativos devem ser criados de tal modo que sejam capazes de detectar situações nas quais não lhes seja possível dar continuidade à sua operação sem antes alterar seu comportamento, e, em resposta a tal necessidade, promover todas as alterações necessárias antes de prosseguir. O artigo [JoséNeto 2001] apresenta todos os detalhes do formalismo que define os dispositivos adaptativos baseados em regras que o presente trabalho utiliza.

4.3. Situação do Método Proposto

A figura 1 apresenta as etapas do método proposto neste artigo. Definiu-se um tratamento para o código-fonte que pode ser subdividido em duas etapas: (a) Extração dos átomos da linguagem através de um analisador léxico, com duas possibilidades de escolha, a primeira, quando se conhecem os átomos e deseja-se informar o analisador, e a segunda, quando nada se conhece a priori da linguagem, e o analisador léxico é bastante simplificado, de tal forma que seu único objetivo é determinar e identificar as seqüências numéricas e textuais; (b) Geração de gramáticas através do algoritmo de Charikar para os casos de exemplo. Estas etapas compreendem exatamente as duas primeiras etapas da figura 1.

A etapa final, que ainda não foi completamente implementada, procura relacionar as diversas gramáticas geradas, não apenas identificando os ciclos sintáticos existentes e distinguindo quando ocorre um novo ciclo ou quando ele já foi representado, como também identifica casos incorretos. Esta última etapa do método de inferência de uma gramática para uma linguagem de programação baseia-se no treinamento de um modelo adaptativo, semelhante ao proposto em [José Neto 1998].

Portanto, a visão geral desta última etapa é a de um algoritmo capaz de percorrer os modelos de gramática existentes para cada caso, e procurar por não-terminais já utilizados anteriormente na derivação de uma sentença. Caso sejam encontrados, há a indicação de existência de um ciclo. Desta maneira, pode-se promover a alteração do conjunto de regras da gramática resultante final, inserindo este ciclo. Caso não haja ciclos, insere-se a regra que gera a menor seqüência de derivação.

Atualmente a proposta encontra-se em fase de implementação da terceira etapa, para posterior integração às etapas anteriores. Procedendo-se depois disso à fase de avaliações, re-especificações e melhorias.

5. Conclusão

Um primeiro resultado desta pesquisa é propiciar o estabelecimento de uma aplicação direta do problema da gramática mínima ao projeto de linguagens de programação em geral.

Outro resultado importante é relativo ao estudo dos algoritmos utilizados para o problema da gramática mínima. Visou-se, também, a encontrar uma formulação mais adequada ao problema da inferência de gramáticas para linguagens de programação, que, conforme mostrado nesta pesquisa, necessita de uma etapa anterior à aplicação do algoritmo em si, e outra etapa posterior para melhor generalizar a gramática, e garantir o funcionamento somente para os programas corretos.

As duas primeiras etapas foram implementadas, e alguns exemplos foram utilizados neste trabalho para ilustrar o método. A terceira e última etapa ainda não está completa, embora a sua formulação geral já seja conhecida, e baseada em um algoritmo já utilizado anteriormente [JoséNeto 1998]. Com isso, é possível abstrair um pouco mais e inferir manualmente algumas pequenas gramáticas. Esta pesquisa está na fase de implementação da terceira etapa, para que se tenha uma visão mais completa e definitiva do método proposto.

Pretende-se utilizar o método também em outras aplicações dentro da Teoria de Linguagens, como, por exemplo, o estudo de linguagens naturais, ou ainda, estudar a representação sintática de aspectos usualmente associados à chamada semântica estática de linguagens de programação [JoséNeto 1998].

Outro aspecto em estudo é o modelo que serve de base para a indução da gramática resultante. Atualmente, está em uso a gramática adaptativa, mas pretende-se utilizar, como experiência comparativa, outros dispositivos adaptativos, como, por exemplo, a árvore de decisão adaptativa [Pistori 2002].

Referências

[Charikar et al 2002] Charikar, M., Lehman, E., Liu, D., Panigrahy, R., Prabhakaran, M., Rasala, A., Sahai, A., Shelat, A., "Approximating The-Smallest Grammar: Kolmogorov Complexity In Natural Models", STOC 2002.

[JoséNeto 2001] José Neto, J. "Adaptive Rule-Driven Devices - General Formulation and Case Study." Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Vol.2494, Pretoria, South Africa, July 23-25, pp. 234-250 2001.

[JoséNeto 1998] José Neto, J., e Iwai, M. K., "Adaptative Automata for Syntax Learning.", Anais Conferência Latino Americana de Informática – CLEI 1998 MEMORIAS. pp. 135-149, Quito, Equador, 1998.

[Kessey 2003] Keesey, J., "CFG Based File Compression", http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Fall03/Keesey/index.shtml ?report.shtml, Dezembro,2003.

[Kieffer 2000] Kieffer, J.C. e Yang, E. H., "Grammar Based Codes: A New Class of Universal Lossless Source Codes", IEEE Transactions on Information Theory, 46(3):737-754, 2000.

[Nevill 1997] Nevill-Manning, C.G., WITTEN I.H., Compression and explanation using hierarchical grammars., The Computer Journal ,40(2/3):103-116,1997.

[Pistori 2002] Pistori, H., e José Neto, J., "AdapTree - Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas**"**. Anais Conferência Latino Americana de Informática - CLEI 2002. Montevideo, Uruguai, Novembro, 2002.

[Rytter 2002] RYTTER, W., "Application of Lempel –Ziv Factorization to the Approximation of Grammar-Based Compression", in Combinatorial Pattern Matching,Lecture Notes in Computer Science,Vol.2373, pp.20 –31, Springer,Berlin, June 2002,.

[Ziv 1977] Ziv, J., Lempel, A., "A Universal Algorithm for Sequential Data Compression.", IEEE Transactions on Information Theory ,IT-23(3):337-343,1977.

[Ziv 1978] Ziv, J., Lempel, A., "Compression Of Individual Sequences Via Variable-Rate Coding"., IEEE Transactions on Information Theory ,IT-24:530-536,1978.