ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

PCS3556 – Lógica Computacional

CAIO VINICIUS SOARES AMARAL - NUSP: 10706083

THALES AUGUSTO SOUTO RODRIGUEZ - NUSP: 10706110

GUSTAVO PRIETO – NUSP 4581945

PROF. DR. Ricardo Rocha

Exercício de Programação 4

São Paulo

Introdução

O objetivo deste exercício programa é implementar um algoritmo de reconhecimento de linguagens livres de contexto, a partir da gramática em forma normal de Chomsky, usando programação dinâmica.

Para tal, o exercício possui duas partes: na primeira um programa feito a partir das funções desenvolvidas no EP2, capaz de transformar uma gramática livre de contexto qualquer em sua forma normal de Chomsky, e na segunda parte um algoritmo baseado em programação dinâmica que reconheça se uma cadeia pertence ou não à gramática na forma normal.

O repositório do projeto pode ser encontrado em: https://github.com/master0022/ep4

Algoritmo

Antes de tudo, é importante ressaltar os 4 passos principais para converter uma gramática para a norma de Chomsky:

1. Remoção de regras de produção nulas (A->Ɛ)
2. Remoção de regras de produção unitárias de não-terminais (A->B)
3. Remoção de regras de produção “grandes” (A -> mais de 2 variáveis)
4. Acréscimo de regras de produção de terminais, para substituir regras de produção mistas (A-> aB)

Para fazer isto, o código foi separado em 5 módulos, cada um sendo responsável por um passo, e o módulo CNF (“Conversion to Normal Form”) implementando os 4 passos e retornando o resultado final.

O funcionamento de modo geral de cada módulo consta a seguir:

RemoveNulo

Este módulo é responsável por remover regras de produção nulas do tipo (A->Ɛ).

Para fazer isto, ele considera dois casos possíveis:

Caso 1) A produção nula é uma das produções de um não-terminal.

Neste caso, temos uma produção do tipo A -> B | C | DE ... | Ɛ

Quando removermos esta regra, é necessário alterar todas as outras regras usando o seguinte algoritmo:

Algoritmo para substituir A->Ɛ:

Para cada regra,

Caso a produção gere uma string que contenha “A”

Substituia “A” por Ɛ, de todas as formas possíveis (ABA-> BA | AB | B)

Acrescente a string original (ABA)

Essas regras são adicionadas e analisamos a próxima regra.

Caso 2) A produção nula é a única produção de um não-terminal.

Neste caso, temos A-> Ɛ e apenas isto.

Não é necessário manter “A” na lista de não-terminais, então podemos deletá-lo. Em todas as regras de produção, “A” é simplesmente trocado por Ɛ.

RemoveUnitario

Este módulo é responsável pela remoção de regras de produção unitárias de não-terminais (do tipo A->B)

Neste caso, o algoritmo é bem simples. Basta procurarmos uma regra deste tipo não-terminal -> não-terminal, como A->B, e então fazemos o seguinte:

1. Procuramos todas as regras de produção em que B gera algo.
2. Acrescentamos estas regras a A.
3. Removemos regras duplicadas ou A->A.

RemoveMultiplo

Este módulo trabalha removendo regras de produção “grandes” (A -> mais de 2 variáveis)

Ele funciona da seguinte forma:

1. Buscamos uma regra de produção que gere mais do que 2 variáveis.
2. Definimos o padrão a ser alterado como as 2 primeiras variáveis da regra encontrada (Exemplo S->aABCD, padrão = aA)
3. Definimos uma letra não-terminal que não esteja em uso como NovaLetra.
4. Acrescentamos a regra NovaLetra -> Padrão à gramatica, e substituímos o padrão encontrado em todas as regras **que tenham mais de duas variáveis**.
5. Isso é repetido até não serem encontrados mais regras que gerem mais de 2 variáveis.

AcrescentaTerminais

Este módulo funciona de forma similar ao **RemoveMultiplo**, e serve para ajustar regras de produção da forma A->aA ou A->Aa. Ele vai acrescentar novas regras que geram terminais para chegarmos na forma normal de Chomsky.

1. Procuramos regras que gerem 2 variáveis.
2. Caso uma das variáveis seja um terminal, ele se torna o Padrão que vamos trocar.
3. Definimos uma letra nova não-terminal que não esteja em uso, NovaLetra.
4. Acrescentamos a regra NovaLetra -> Padrão à gramática.
5. Substituímos o Padrão nas regras em que ocorre.
6. Isto é repetido até não haverem mais regras que não estejam na forma nomal de Chomsky.

Casos de teste

Para rodar os casos de teste criados, basta entrar na pasta do projeto elixir e executar

mix test

Ou, caso seja desejado usar as funções com casos novos, elas estão disponíveis na pasta /lib.

Para executá-las, basta usar:

Mix run /lib/[nome_do_modulo].ex

Os casos de teste gerados foram os seguintes, e podem ser encontrados em /test/**all_test.exs**:

Teste 1

Testamos o módulo **RemoveNulo**.

Testamos duas gramáticas similares (1 e 2), mas que caem em casos diferentes do módulo.

Também usamos, na gramática 3, o mesmo exemplo que o que foi usado no vídeo preparatório para a aula.

Gramática 1:

P -> ABC | bCC

A -> aAA | BB

B -> ε | asd

C -> ABC | b

Gramática 2:

P -> ABC | bCC

A -> aAA | BB

B -> ε

C -> ABC | b

Gramática 3:

S->ASA | ab,

A -> B|S,

B->b|ε

Teste 2

Testamos o módulo **RemoveUnitario**

Testamos duas gramáticas similares (1 e 2), mas que caem em casos diferentes do módulo. G

Gramática 1:

P -> AC | C | bCC

A -> aAA | aA | a

C -> AC | C | b

Gramática 2 (Do exemplo visto no vídeo preparatório da matéria):

S -> ASA | aB | a | AS | SA

A -> B | S

B -> b

Teste 3

Testamos o módulo **RemoveMultiplo**

Testamos duas gramáticas similares (1 e 2), mas que caem em casos diferentes do módulo. G

Gramática 1:

P -> AC | bCC | b

A -> aAA | aA | a

C -> AC | b

Gramática 2 (Do exemplo visto no vídeo preparatório da matéria):

S -> ASA | aB | a | AS | SA

A -> b | ASA | aB | a | AS | SA

B -> b

Teste 4

Testamos o módulo **AcrescentaTerminais**

Testamos duas gramáticas similares (1 e 2), mas que caem em casos diferentes do módulo. G

Gramática 1:

P -> AC | DC | b

A -> BA | aA | a

C -> AC | b

B -> aA

D -> bC

Gramática 2 (Do exemplo visto no vídeo preparatório da matéria):

A -> b | CA | aB | a | AS | SA

C -> AS

S -> CA | aB | a | AS | SA

B -> b

Teste 5

Testamos o módulo **CNF**

Testamos duas gramáticas similares (1 e 2), mas que caem em casos diferentes do módulo. G

Gramática 1(Do exemplo visto no vídeo preparatório da matéria)::

S -> ASA | aB

A -> B | S

B -> b | ε

Resultado final obtido (Que confere com o visto em aula)

A -> b | CA | AS | SA | DB | a

B -> b

C -> AS

D -> a

S -> CA | AS | SA | DB | a