Linguagens Formais e Compiladores - INE5421

Gustavo Kundlatsch e Paola Abel

25 de Novembro de 2019

1 Introdução

Esse trabalho foi realizado para a disciplina Linguagem Formais e Compiladores (INE5421), do curso de Ciência da Computação da Universidade Federal de Santa Catarina. Ele consiste da implementação de diversos algoritmos visto ao longo da disciplina, que se aplicam a autômatos finitos, gramáticas e expressões regulares. Esse relatório contém detalhes sobre a arquitetura utilizada para produzir o programa, detalhes da implementação, uma explicação detalhada de como usar o software e também como podem ser executados os testes.

2 Arquitetura

O trabalho foi realizado na linguagem Python, sem o uso de interface gráfica, apenas interface por linha de comando. A arquitetura geral da implementação utiliza o conceito de modos de terminal para separar os algoritmos. Existem cinco modos de terminal: o terminal base, o de autômatos, o de gramáticas regulares (ou simplesmente gramáticas), o de gramáticas livre de contexto (CFG) e o de expressões regulares. Ao inciar o programa, o usuário deve digitar o nome do modo que deseja utilizar (automata, grammar, cfg ou regex, respectivamente). A partir disso, novos comandos podem ser usados para carregar estruturas salvas anteriormente, modificá-las, executar os algoritmos implementados e salvar os objetos.

A principal estrutura de dado utilizada no projeto foi o dicionário. O dicionário do Python funciona como um mapa. Dado uma chave, ele aponta para um valor específico. Esse valor pode ser qualquer coisa, como um inteiro, uma lista ou uma função. Os dicionários foram utilizados para implementar o controle do terminal, redirecionando o input do usuário para funções específicas, e também para registrar as transições das tabelas de autômatos e as produções das gramáticas, tanto regular quanto livre de contexto.

3 Formalização

Os diferentes mecanismos de formalização tiveram implementações diferentes. Nessa seção, falaremos de cada um, tentando explicar as decisões tomadas no de-

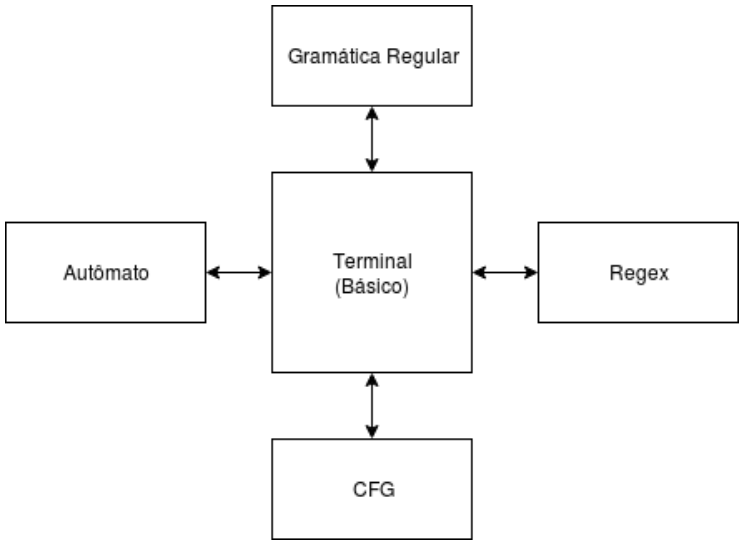

Figura 1: Diagrama da arquitetura dos terminais

senvolvimento. Cada classe possui o atributo nome, para simplificar o processo de diferenciação entre os objetos, e facilitando o processo de busca e salvamento de arquivos em disco.

3.1 Autômato Finito

A classe do autômato finito, presente em `/regular/automata.py` possui três atributos principais além do nome: transições, estado inicial e estados finais. As transições são um dicionário que mapeia de um estado (cujo nome é descrito por uma letra maiúscula) para uma lista de transições desse estado. Essa lista de transições é uma lista de tuplas, onde o primeiro elemento é aquele pelo qual ele transita, e o segundo é uma lista de possíveis estados para o qual ele está indo. É necessário uma lista ao invés de apenas um elemento para podermos representar não determinismo. O estado inicial é apenas uma string, que guarda o nome do estado que foi definido como inicial. Os estados finais são representados por uma lista, que guarda o nome de cada um dos estados de aceitação.

3.2 Gramáticas

Tanto as gramáticas regulares quanto as gramáticas livre de contexto são representadas da mesma maneira, mas possuem jeitos diferentes de tratar a inserção de novas produções e também possuem métodos diferentes. Ambas as classes se encontram no arquivo `/regular/grammar.py`. As gramáticas possuem apenas o nome, uma string para a produção inicial, e um dicionário para as produções. Esse dicionário possui o lado esquerdo da produção como chave e uma lista de

produções como valores, representando os "ou" das produções.

Nas gramáticas, algumas restrições foram criadas para garantir que as produções que o usuário insere são de fato válidas para uma gramática livre de contexto ou para uma gramática regular.

3.3 Expressão Regular

A expressão regular é a formalização mais simples de todas, pois possui apenas o nome e a string que representa a expressão.

4 Implementação

A implementação dos algoritmos seguiu os passos apresentados tanto pelo material disponibilizado pela professora (slides), quanto os passos retirados diretamente do livro texto da disciplina (Compilers: Principles, Techniques, and Tools, 2nd Edition, Alfred V. Aho). Implementações adicionais foram realizadas para garantir a manipulação dos dados.

4.1 Salvamento de Arquivos

Para salvar os objetos produzidos pelo usuário e pelos algoritmos executados, foi utilizada a biblioteca Pickle do Python, que permite salvar em texto o segmento de memória que representa um objeto arbitrário, e depois lê-lo novamente. Portanto, os objetos salvos (autômatos, gramáticas e expressões regulares) não são legíveis para pessoas. Entretanto, é possível utilizar o programa para ler o arquivo em que está salvo o item que se deseja ler, e utilizar o comando `print` após carregá-lo para visualizar. Os arquivos salvos ficam na pasta `/terminal/saves`.

4.2 Leitura de Arquivos

A leitura de arquivos permite tanto ler os objetos gerados pelo Pickle quanto ler de arquivos de texto produzidos pelo usuário. Para isso, basta criar um arquivo no formato `nome.classe.txt`, onde *nome* será o nome do objeto criado e *classe* o tipo dele, podendo ser automata, regGrammar, cfg ou regex. Para fazer a leitura desses arquivos eles precisam ser colocados no na pasta `/terminal/input`. A formatação necessária para cada um pode ser encontrada nos exemplos presentes na pasta, mas é bastante simples.

4.3 Algoritmos

Todos os algoritmos necessários foram implementados dentro das próprias classes das formalizações. Alguns afetam o próprio objeto, e outras alteram um terceiro.

4.3.1 Conversão de AFND para AFD

Esse algoritmo é aplicado sobre o próprio autômato. Primeiro, foi criado uma função para verificar se o autômato já é determinístico. Caso seja, ele retorna a si próprio. Caso contrário, ele cria um novo autômato com o mesmo nome mas com uma adição de "dfa" no final. Depois, realiza o processo de verificar e adicionar os estados determinísticos e calcular os novos estados a partir dos não determinísticos.

4.3.2 Conversão de AFD para GR e GR para AFD

Na realidade, esse são dois algoritmos. O de AFD para GR está na classe do autômato, e vai retornar uma nova gramática regular. Para isso, ele necessita que o autômato esteja determinizado, para então criar as produções de acordo para a própria tabela. O de GR para AFD possui um processo bastante similar, passando as produções para a forma de tabela, para por fim calcular os estados de aceitação e adicioná-los ao autômato final que será retornado.

4.4 Reconhecimento de Sentença em AF

O reconhecimento de sentença precisa ser realizado tanto para AFD quanto para AFND. Para AFD é um processo bastante simples, uma vez que basta verificar se existe alguma transição pelo símbolo atual que está sendo olhado, caso sim, basta passar para o estado que ele leva e avançar para o próximo símbolo da sentença. Para AFND, é preciso olhar todas as possibilidades de transição em um mesmo estado, tanto pelo mesmo símbolo quanto por épsilon, e realizar o backtracking.

4.4.1 Minimização de AFD

O processo de minimização de um AFD é dividido em três algoritmos: remover os inalcançáveis, os estados mortos e os equivalentes. Os inalcançáveis podem ser removidos fazendo uma análise de quais estados são marcados ao realizar todas as transições, aqueles que não foram marcados são inalcançáveis e podem ser removidos. Os estados mortos são aqueles a partir dos quais nunca será possível atingir um estado de aceitação. Para removê-los, basta partir dos estados de aceitação e ir marcando os estados em um processo contrário. Por fim, os equivalentes são removidos analisando cada uma das transições, tanto de entrada quanto de saída, de cada um dos estados.

4.4.2 União e Intersecção

Esses dois algoritmos possuem uma implementação bastante simples, que constitui basicamente da automatização desses processos já estudados em sala. A união utiliza o método não determinístico, para depois utilizar o próprio algoritmo implementado de determinização para encontrar o autômato da união final.

4.4.3 Conversão de ER para AFD

Para essa conversão, que utiliza o algoritmo do Aho, primeiro necessita que a árvore da ER seja produzida. A implementação desse algoritmo encontra-se no caminho `/utils/sintax_tree.py`. Após isso, são calculados os firsts e os follows, que por fim darão origem ao autômato já minimizado.

4.4.4 Transformação de GLC para forma normal de Chomsky

Assim como a minimização, a transformação em forma normal de Chomsky depende de etapas anteriores. Primeiro, é preciso remover estados inalcançáveis e depois improdutivos, de maneira bastante similar aos autômatos. Depois disso, é preciso remover produções unitárias, puxando as produções para a produção que gera a unitária desnecessária. Por fim, aplicamos as regras para construir uma gramática na forma normal de Chomsky, expandindo a gramática através da criação de novas produções, nomeadas com numerais após o não terminal que a originou.

4.4.5 Eliminação de Recursão a Esquerda

A recursão a esquerda acontece quando o símbolo mais a esquerda de uma produção é um não terminal. Isso pode levar ao analisador sintático entrar em looping infinito, dependendo da implementação utilizada. Para evitar isso, usamos o algoritmo do Aho, que cria novas produções conforme for necessário, para jogar a produção desses não terminais para a direita.

4.4.6 Fatoração

A fatoração é o processo utilizado para remover não determinismo, ou seja, produções que possuem um mesmo prefixo. Para removê-las, basta criar uma nova produção, com um novo não terminal, mantendo apenas uma produção pelo prefixo e que leva para esse novo não terminal. Nele, são criados "ou" entre as produções que antes geravam ambiguidade, mas sem os prefixos.

4.4.7 Reconhecimento de Sentenças em AP

O algoritmo escolhido para essa questão foi o LL(1), que calcula os FIRST e FOLLOW de cada um dos símbolos da gramática, e depois constrói a respectiva tabela. Não é verificado se a gramática é LL(1), portanto o algoritmo pode falhar nesse aspecto.

5 Modo de Uso

Nessa seção apresentaremos os comandos disponíveis em cada um dos modos do terminal. Para acessar um modo, basta escrever seu nome no terminal base (inicial).

5.1 automata

Comandos sem parâmetro:

q: retorna ao terminal base.

des: desativa o autômato atual.

print: mostra na tela o autômato ativo.

rm_start: remove o estado inicial do autômato.

to_dfa: converte o autômato para determinístico.

to_RG: transforma o autômato em gramática regular.

Comandos com um parâmetro, onde esse parâmetro é o nome do autômato especificado:

act: ativa o autômato.

create: cria um autômato (não é salvo ao criar).

delete: deleta o autômato.

save: salva o autômato.

Comandos com um ou mais parâmetros:

add_state: o primeiro argumento é o nome do estado, o segundo é uma lista de estados destino.

rm_state: remove o estado especificado.

add_transition: adiciona uma nova transição.

rm_transition: similar ao add_transition, mas remove.

set_start: escolhe o estado inicial do autômato.

set_end: adiciona o estado especificado como estado de aceitação.

rm_end: remove o estado especificado da lista de estados de aceitação.

compute: computa a cadeia passada como parâmtro.

5.2 grammar

Comandos sem parâmetro:

q: retorna ao terminal base.

des: desativa a gramática atual.

print: mostra na tela a gramática ativo.

remove_start_symbol: remove o símbolo inicial da gramática.

to_fa: transforma a gramática em autômato finito.

Comandos com um parâmetro, onde esse parâmetro é o nome da gramática especificado:

act: ativa a gramática.

act-file: lê a gramática a partir de um arquivo de texto.

create: cria uma gramática (não é salvo ao criar).

delete: deleta a gramática.

save: salva a gramática.

Comandos com um ou mais parâmetros:

set_start_symbol: seleciona o símbolo especificado como inicial da gramática.

add_production: adiciona a produção a gramática. O primeiro parâmetro é o lado esquerdo da produção, o segundo é o lado direito, que deve ser separado por vírgula e sem espaço.

remove_productions: análago ao add_production, mas remove a produção.

5.3 CFG

Comandos sem parâmetro:

q: retorna ao terminal base.

des: desativa a gramática atual.

print: mostra na tela a gramática ativo.

remove_start_symbol: remove o símbolo inicial da gramática.

chomsky: converte a gramática para a forma normal de chomsky.

remove_recursion: remove a recursão a esquerda da gramática.

remove_non_determinism: fatora a gramática para remover o não determinismo.

Comandos com um parâmetro, onde esse parâmetro é o nome da gramática especificado:

act: ativa a gramática.

act-file: lê a gramática a partir de um arquivo de texto.

create: cria uma gramática (não é salvo ao criar).

delete: deleta a gramática.

save: salva a gramática.

Comandos com um ou mais parâmetros:

set_start_symbol: seleciona o símbolo especificado como inicial da gramática.

add_production: adiciona a produção a gramática. O primeiro parâmetro é o lado esquerdo da produção, o segundo é o lado direito, que deve ser separado por vírgula e sem espaço.

remove_productions: análago ao add_production, mas remove a produção.

6 Testes

Os testes foram implementados de maneira bastante simples, importando os módulos que estão contidos no pacote **regular**. Os testes estão listas de acordo com a letra da questão, e o input deles é realizado através da leitura dos arquivos da pasta input. Para rodar algum teste, para usar o comando `python + nomeDoTeste.py`. Não é feita verificação automática do resultado, essa verificação foi realizada no papel para os valores originais testados.