Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Ciências da Computação

Relatório de Projeto

Bernardo Gums Brack
Gabriel de Vargas Coelho
Rafael Francisco Réus

Florianópolis,
28 de junho de 2023

Sumário

Implementação

A linguagem escolhida para a implementação do trabalho foi Python, na sua versão 3, devido a sua flexibilidade e também maior familiaridade. Foram utilizadas 3 estruturas de dados principais no trabalho, todas elas utilizando o conceito de classes, são elas: State, AF e GR.

A classe State representa um estado de um autômato finito, e possui dois atributos: *id*, para identificação; e *transitions*, que é um dicionário - estrutura chave-valor - que armazena as transições de um dado estado. O atributo *transitions* é um dicionário em que a chave é uma *string*, que representa o símbolo da transição, e o valor é uma lista de State, que são os alvos da transição. Se optou por utilizar uma lista de transições para poder representar autômatos finitos não determinísticos.

A classe AF representa autômatos finitos, tanto os determinísticos quanto ou não determinísticos. Ela possui 4 atributos: *alphabet*, que representa o alfabeto da linguagem e é do tipo lista de *string*; *states*, que representa os estados do autômato, e é do tipo lista de *State*; *initialState*, que representa o estado inicial do autômato, e é do tipo *State*; e *finalStates*, que representa os estados finais do autômato, e é do tipo lista de *State*.

A classe Table representa a tabela de Análise Sintática Preditiva LL(1). Ademais, ela possui um único atributo: *table*, que é do tipo Dict[str, Dict[str, str]], além da função __str__, que formata-a para uma tabela quando transformada em string. Sua utilização é de suma importância para o reconhecimento de sentença em AP usando LL(1) e, obviamente, para a visualização da tabela (com o propósito de depuração). Para a formatação, é necessária a instalação da biblioteca *tabulate* com: *pip install tabulate.*

Além disso, as classes presentes no arquivo *“Regex.py”* foram utilizadas com o propósito de moldar a *SyntaxTree,* a qual é composta por um conjunto de nodos (armazenados em dicionário) e por métodos de construção tal como o *postorderNullableFirstposLastposFollowpos* o qual é responsável pela adequação dos conjuntos LastPos FirstPos, pela verificação de possíveis nós anuláveis e a submissão dos *id’s* dos nós (utilizado posteriormente para a criação do AFD). Nesse sentido, foi instanciada uma nova classe de AF para que pudéssemos obter os

estados por recursividade (DFAWithTree é composto por um ou mais objetos da classe StateOfDFAWithTree). O algoritmo utilizado segue fielmente ao apresentado na obra Compiladores -Princípios, Técnicas e Ferramentas - Aho, Sethi e Ullman.

Por fim, a classe GR representa gramáticas, tanto as regulares quanto as livres de contexto. Ela possui 4 atributos: *terminals*, que representa os símbolos terminais da gramática, e é do tipo lista de *string*; *nTerminals*, que representa os símbolos não-terminais, e é do tipo lista de *string*; *initial*, que representa o símbolo inicial da gramática, e é do tipo *string*; e *productions*, que representa as produções da gramática, e é do tipo dicionário em que a chave é do tipo *string* e representa a cabeça de produção, e o valor é do tipo lista de *string* e representa o corpo de produção de uma cabeça. Existem ainda dois outros atributos usados internamente pela classe que são: *_first* e *_follow*, que são do tipo conjunto de *string* e representam o conjunto *first* e *follow* de uma gramática. A visualização aprofundada de métodos (como: *removeLeftRecursion, removeDirectNonDeterminism,* etc*…)* está presente nos repositórios disponibilizados.

Dependências do projeto

Para a formatação adequada, em linha de comando, da tabela LL(1), foi utilizada a biblioteca Tabulate. Para instalar ela use o comando: pip install tabulate.

Observação: em sistemas Linux o pip é instalado juntamente com o Python, assim, se já possuir Python em sua máquina não é necessário instalá-lo previamente; já em sistemas Windows e MacOS a instalação do pip pode ser independente. Para instalar o pip pode-se seguir as instruções que se encontram nesse site: https://pip.pypa.io/en/stable/cli/pip_install/.

Entrada dos algoritmos

O presente trabalho implementa diversos algoritmos, e a maioria (exceto a Conversão de ER para AFD, que possui a entrada por meio do *console*/terminal) deles demandam a entrada de dados na forma de arquivos de texto, e esses arquivos devem seguir algumas regras para que possam ser corretamente lidos e interpretados pelo programa. Abaixo estão listadas as regras de formação do arquivo.

1. A primeira linha do arquivo identifica se aquele arquivo é de um autômato finito ou de uma gramática. Para arquivos de autômatos a primeira linha deve conter a entrada AF, já para gramáticas deve conter a entrada GR.
2. Para arquivos do tipo AF:
    a. A segunda linha representa o alfabeto da linguagem, cada símbolo do alfabeto deve ser unitário e separado de outros símbolos por vírgula e sem o uso de espaço em branco, exemplo: a,b.
    b.
    c. A terceira linha representa os estados do autômato, cada símbolo deve ser unitário e separado de outros símbolos por vírgula e sem o uso de espaço em branco, exemplo: A,B.
    d. A quarta linha representa o estado inicial do autômato, que deve ser um símbolo unitário e que faça parte da lista de estados do autômato.
    e. A quinta linha representa os estados finais do autômato, cada símbolo deve ser unitário e separado de outros símbolos por vírgula e sem o uso de espaço em branco, exemplo: A
    f. As linhas subsequentes do arquivo representam as transições do autômato, e seguem o seguinte formato: <estado atual> <símbolo de transição> <estado alvo>, exemplo: A a B. Cada linha deve conter apenas uma transição.
3. Para arquivos do tipo GR:
    a. A segunda linha representa os símbolos terminais da gramática, cada símbolo do alfabeto deve ser unitário e separado de outros símbolos por vírgula e sem o uso de espaço em branco, exemplo: a,b.
    b. A terceira linha representa os símbolos não-terminais, cada símbolo deve ser unitário e separado de outros símbolos por vírgula e sem o uso de espaço em branco, exemplo: A,B.
    c. A quarta linha representa a produção inicial da gramática, que deve ser um símbolo unitário e que faça parte da lista de símbolos não-terminais.

   d. As linhas subsequentes do arquivo representam as produções da gramática, e seguem o seguinte formato: <símbolo não-terminal> [<mistura de símbolos terminais e não terminais>], exemplo: A aB B AB (na notação usada em aula: A -> aB | B | AB)
4. Para arquivos do tipo reconhecimento de sentenças em uma LL(1):
   a. O segundo parâmetro de entrada pelo terminal é o mesmo que o utilizado nas do tipo GR (o arquivo com suas descrições).
   b. O terceiro parâmetro por sua vez define a sentença que se deseja verificar.
5. Transições por épsilon ou produções vazias devem ser representadas pelo símbolo &.

Operações

Abaixo há uma lista de operações implementadas pelo programa, as entradas e saídas esperadas e a forma de rodar cada uma delas. Vale salientar que quando a saída é um arquivo de texto, ele se encontrará na pasta *results* e seu nome será composto de um número baseado na data atual em segundos, e seu tipo, que pode ser AF ou GR, exemplo: results/1687110088-af.txt. A nomeação dos arquivos de saída é feita dessa forma para que o último arquivo gerado seja o último arquivo listado na ordenação dos arquivos do diretório em ordem alfanumérica.

Função	Comando	Entrada	Saída
Conversão de AFND para AFD	python3 main.py AF-det <nome do arquivo>	Arquivo de texto do tipo AF	Arquivo de texto do tipo AF
Conversão de AFND para GR	python3 main.py AF-GR <nome do arquivo>	Arquivo de texto do tipo AF	Arquivo de texto do tipo GR
Conversão de GR para AFND	python3 main.py GR-AF <nome do arquivo>	Arquivo de texto do tipo GR	Arquivo de texto do tipo AF
Minimização de	python3 main.py	Arquivo de texto	Arquivo de texto

AFD	AF-min <nome do arquivo>	do tipo AF	do tipo AF
União de AFs	python3 main.py AF-union <nome do arquivo 1> <nome do arquivo 2>	2 arquivos de texto do tipo AF	Arquivo de texto do tipo AF
Interseção de AFs	python3 main.py AF-intersection <nome do arquivo 1> <nome do arquivo 2>	2 arquivos de texto do tipo AF	Arquivo de texto do tipo AF
Reconhecimento de sentença em AF	python3 main.py AF-test <nome do arquivo> “<palavra>”	Arquivo de texto do tipo AF e uma palavra a ser testada (coloque a palavra entre aspas para garantir que o terminal não interprete a entrada incorretamente)	Informação no terminal indicando se a palavra pertence ou não a linguagem
Conversão de ER para AFD	python3 main.py ER-AFD	Número de símbolos, suas definições e a expressão regular (as concatenações devem ser explícitas, exemplo: a*b deve ser escrito como a*.b; o *ou* é representado pelo sinal + ao invés de |)	Informações do autômato gerado e um arquivo de texto do tipo AF
Checar determinismo de GLC	python3 main.py GLC-isDet <nome do arquivo>	Arquivo de texto do tipo GR	Informação no terminal indicando se a gramática está determinizada ou não
Fatoração de GLC	python3 main.py GLC-det <nome	Arquivo de texto do tipo GR	Arquivo de texto do tipo GR

	do arquivo>		
Criação da tabela LL(1)	python3 main.py LL1-table <nome do arquivo>	Arquivo de texto do tipo GR	Informação da tabela no terminal
Reconhecimento de sentença em AP usando LL(1)	python3 main.py AP-test <nome do arquivo> “<palavra>”	Arquivo de texto do tipo GR e uma palavra a ser testada (coloque a palavra entre aspas para garantir que o terminal não interprete a entrada incorretamente)	Informação no terminal indicando se a palavra pertence ou não a linguagem