Editor de linguagens regulares

Projeto feito por Gabriel Müller e Juliana Pinheiro para disciplina de Linguagens Formais e Compiladores na UFSC. Feito na linguagem Python3 com o framework gráfico PyQt5.

Instalação

É necessário instalar PyQt5 para Python 3. Em sistemas Ubuntu / Debian:

```
sudo apt install python3-pyqt5
```

Para executar, na pasta do projeto:

```
python3 main.py
```

Funcionamento

Autômatos (`class NFA`)

O trabalho usa extensivamente as estruturas de dados da linguagem para representações formais. As transições de um autômato são um `dict` (mapeamento) de estados para outros mapeamentos, de símbolo para um conjunto de estados. Além disso, há um conjunto de estados de aceitação e um estado inicial. Os estados são representado por strings.

```
transitions = {
'qo': {'a': {'q1', ' 'q2'}, 'b': {'q1'}},
'q1': {'a': {'q0'}},
'q2': {}
 }
```

Expressões regulares (`class Regex`)

Utiliza-se o algoritmo de Simone para converter a árvore sintática de ER para um AFD. As semânticas dos operadores são representados for funções `lambda` . Uma diferença é o uso de semânticas de subida e descida também para o operador '+':

```
down = \
{'|': lambda node: {Move(node.left, DOWN), Move(node.right, DOWN)},
'.': lambda node: {Move(node.left, DOWN)},
'?': lambda node: {Move(node.left, DOWN), Move(node.right, UP)},
'+': lambda node: {Move(node.left, DOWN)},
'*': lambda node: {Move(node.left, DOWN), Move(node.right, UP)}}

up = \
{'|': or_up_semantics,
'.': lambda node: {Move(node.right, DOWN)},
'?': lambda node: {Move(node.right, UP)},
'+': lambda node: {Move(node.left, DOWN), Move(node.right, UP)},
'*': lambda node: {Move(node.left, DOWN), Move(node.right, UP)}}

semantics = {DOWN: down, UP: up}

leaf_up = lambda node: {Move(node.right, UP)}
```

A `class Node` representa nodos de uma árvore, enquanto a `class Move` representa um `Node` em conjunto com uma direção (subida ou descida). Há detecção de loops infinitos em casos como `(a | (b|c)*)*`, quando a árvore costurada apresenta ciclos entre nodos operadores.

Gramáticas Regulares (`class RegularGrammar`)

O programa permite criação de uma gramática, edição de gramática, exportar e importar (.txt), e operações de união, concatenação e fechamento. A edição de gramáticas na interface gráfica se dá apenas pela definição de suas produções, sem necessidade de definir alfabeto, Vn e Vt. Gramáticas também podem ser obtidas através da conversão de autômatos, bem como podem ser convertidas para autômatos.

A classe `RegularGrammar` é definida por um símbolo inicial e um dicionário de conjuntos representando as produções. Segue um exemplo de como as produções de uma gramática G são representadas:

```
G: P = { S -> aA | ϵ
         A -> aA | a }

productions = {
    "S": {"aA", "&"},
    "A": {"aA", "a"}
}
```

Operações

Foram implementas as seguintes operações em Autômatos Finitos:

1. Complemento
2. União
3. Diferença
4. Intersecção
5. Reverso
6. Minimização
7. Determinização
8. Transformação em AFD Completo

E em Gramáticas Regulares:

1. Concatenação
2. União
3. Fechamento

A maior parte das operações entre autômatos ou gramáticas são feitas por algoritmos vistos em aula. Em especial, a eliminação de estados equivalentes na minimização usa o algoritmo de preenchimento de tabela (*table-filling algorithm*).

Table-filling algorithm

Base: Se p é um estado final e q não é um estado final, então o par $\{p,q\}$ é equivalente (ou não-distinguível).

Indução: Sejam p e q estados tais que para alguma entrada a, $r=\delta(p, a)$ e $s=\delta(q, a)$ são pares equivalentes conhecidos. Então $\{p,q\}$ é um par conhecido. A razão para essa regra é que deve existir alguma entrada w que diferencie r de s; isto é, uma entrada que apenas $\delta(r, w)$ ou $\delta(s, w)$ seja final. Então a entrada aw deve diferenciar p de q, visto que $\delta(p, aw)$ e $\delta(q, aw)$ é o mesmo par de estados que $\delta(r, w)$ e $\delta(s, w)$.

Fonte: Hopcroft, John E.; Ullman, Jeffrey D. (1979), "Chapter 3", Introduction to Automata Theory, Languages, and Computation.

```
Entrada: AFD Completo

Saída: AFD Mínimo

Passo 1: Encontre todos os pares de estados (Qi, Qj) não necessariamente conecta
```

```
Passo 2: Marque todos os pares (Qi, Qj) no AFD tal que Qi ∈ F e Qj ∉ F ou Qj ∈ F

Passo 3: Repita até não ser possível marcar mais estados:
         Se existe um par não marcado (Qi, Qj), marque-o se o par {δ (Qi, A), δ

Passo 4: Combine os pares não marcados (Qi, Qj), transformando-os em um único es
```

Análise (`parser.py`)

O parsing de expressões regulares e gramáticas regulares é feito respectivamente por `parse(string)` e `parse_rg(string)` . O parser ignora espaços e símbolos explícitos de concatenação (’.’). Há detecção de erro para vários casos de expressões inválidas.