Construção de aplicação para geração de Autômato Finito Determinístico

Eduardo R. Barcaroli
Fernando S. Magnabosco

Resumo:
Objetivo: criar uma solução para resolver indeterminismos relacionados a geração de autômatos finitos. **Método:** desenvolvimento de uma aplicação na linguagem Python onde o usuário entra um arquivo com a relação de tokens e/ou GRs de uma linguagem hipotética e o sistema tem como saída um Autômato Finito Determinístico (AFD), livre de épsilon transições, e mínimo sem a aplicação de classes de equivalência entre estados.

Introdução:

O Autômato Finito (AF) é o tipo mais simples de reconhecedor de linguagens. Ele é usado como reconhecedor de padrões em processamento de textos e também como analisador léxico de linguagens. Tem um importante papel na área da computação, pois eles têm a capacidade de realizar várias tarefas simples de compilação, a simulação de um AF requer uma quantidade finita de memória e existem vários teoremas e algoritmos para construir e simplificar autômatos finitos para vários propósitos.

Um Autômato Finito Determinístico (AFD) é uma máquina de estados finita que aceita ou rejeita cadeias de símbolos gerando um único ramo de computação para cada cadeia de entrada. AFDs são utilizados para modelar softwares que validam entradas de usuário tal como um e-mail em um servidor de correio eletrônico, reconhecem exatamente o conjunto de Linguagens Regulares que são, dentre outras coisas, úteis para a realização de análise léxica e reconhecimento de padrões.

Referencial teórico básico:

É necessário alguns conhecimentos básicos da disciplina de Linguagens Formais e Autômatos para a compreensão de conceitos apresentados neste trabalho, tais como

- Alfabeto: conjunto finito de símbolos ou caracteres.
- Palavra: cadeia de caracteres.
- Token: palavra reservada.
- Concatenação: junção de dois ou mais símbolos.

Desenvolvimento:

- **Criação de tabelas:**

```
self.table = [[[] for _ in self.alphabet] for _ in self.productions]

for (p, production) in enumerate(self.productions):
    self.pHash.update({production.left: p})

    for (s, symbol) in enumerate(self.alphabet):
        for rule in production.rules:
            if symbol == rule.terminal:
                if rule.non_terminal is not None:
                    self.table[p][s].append(
                        rule.non_terminal)
                if self.HAS_EPSILON is False \
                        and rule.non_terminal == EPSILONSTATE:
                    self.HAS_EPSILON = True
```

- **Resolução das indeterminações:**

```
def determinize(self):
    self.nextNT = len(self.productions)

    needToRepeat = True

    while needToRepeat:  # while the automata is not deterministic
        needToRepeat = False
        toMerge = set()
        for production in self.table:
            for rule in production:
                if len(rule) >= 2:  # if the rule has more than one symbol
                    rule.sort()
                    hash = self.pHash.get(tuple(rule))
                    # if it has already been hashed, just update to the
                    # new production
                    if hash:
                        rule = hash
                    # else, add it to the list of rules to be merged
                    # into a new production
                    elif EPSILONSTATE not in rule:
                        toMerge.add(tuple(rule))
                        needToRepeat = True

        for production in toMerge:  # create the new productions

            self.pHash.update({production: self.nextNT})
            isFinal = False
            rules = set()

            for symbol in production:
                hash = self.pHash.get(symbol)

                if hash is None:
                    continue
                if self.productions[hash].is_final:
                    isFinal = True
                rules.update(self.productions[hash].rules)

            # Update the productions
            self.productions.append(Production(
                list(production), rules, isFinal))

            # Update the table
            newRow: list = [[] for _ in self.alphabet]
            for(s, symbol) in enumerate(self.alphabet):
                for rule in rules:
                    if symbol == rule.terminal:
                        if rule.non_terminal is not None:
                            newRow[s].append(rule.non_terminal)
            self.table.append(newRow)
            self.nextNT += 1
```

Conclusão:

Concluímos que com o desenvolvimento dessa aplicação podemos ter a solução de um AFD com muito mais facilidade e precisão, pois ao longo do semestre ministrado pelo professor Bráulio, tivemos conhecimentos adquiridos que facilitaram muito a resolução de problemas desse tipo, tais como a introdução da matéria de LFA e alguns conceitos mais avançados dentro do assunto.

Referências:

- http://www.dsc.ufcg.edu.br/~pet/jornal/junho2014/materias/recapitulando.html#:~:text=Aut%C3%B4matos%20Finitos%20Determin%C3%ADsticos%20(AFD),para%20cada%20cadeia%20de%20entrada.
- https://moodle-academico.uffs.edu.br/pluginfile.php/691540/mod_resource/content/1/APOSTILA-Roberto-Scheffel.pdf