Reconhecedor Léxico com Autômato Finito Determinístico (AFD)

Autores: João Leonardo Comiran Figueiró e Marco Aurélio Lunardi
Instituição: Universidade Federal da Fronteira Sul - Campus Chapecó
Curso: Ciência da Computação
Disciplina: Construção de Compiladores

Resumo

Este trabalho apresenta o desenvolvimento de um analisador léxico baseado em Autômatos Finitos Determinísticos (AFDs), aplicado ao reconhecimento de palavras-chave da linguagem SQL. A aplicação lê sentenças a partir de um arquivo de entrada e gera como saída a fita de transições e uma Tabela de Símbolos com os rótulos correspondentes. São discutidos os fundamentos teóricos, as decisões de projeto, a construção do AFD e os resultados obtidos a partir de testes com diferentes cadeias de entrada.

---

1. Introdução

Reconhecedores léxicos são componentes fundamentais na construção de compiladores. Sua principal função é realizar a análise léxica, ou seja, identificar tokens válidos em uma cadeia de entrada, classificando-os conforme regras definidas pela linguagem alvo. Esses reconhecedores têm diversas aplicações, como em compiladores, interpretadores e processadores de texto.

O presente trabalho tem como objetivo implementar um reconhecedor léxico simples baseado em um AFD construído previamente como projeto na disciplina de Linguagens Formais e Autômatos (LFA). O sistema deve processar entradas textuais com palavras reservadas separadas por espaço, reconhecer os tokens válidos da linguagem simulada e identificar erros léxicos. Os resultados são apresentados por meio de uma fita de saída e de uma Tabela de Símbolos detalhada.

---

2. Referencial Teórico

A análise léxica é a primeira etapa de um compilador. Nela, a entrada (tipicamente um código-fonte) é lida e segmentada em unidades léxicas chamadas *tokens*. Essa tarefa é normalmente executada por meio de autômatos finitos, sendo os determinísticos (AFDs) os mais comuns pela sua eficiência.

Um AFD é uma quíntupla $(Q, \Sigma, \delta, q_0, F)$, onde:

- **Q**: conjunto finito de estados

- **Σ**: alfabeto de entrada
- **δ**: função de transição δ: Q × Σ → Q
- **q_0**: estado inicial
- **F**: conjunto de estados finais (de aceitação)

Tokens são definidos a partir de expressões regulares, e essas expressões podem ser convertidas em autômatos equivalentes por algoritmos clássicos. No projeto em questão, essas conversões e a construção do AFD foram feitas manualmente.

---

3. Implementação e Resultados

3.1 Tokens e Decisões de Projeto

Os tokens definidos para a linguagem simulada incluem:

- Palavras reservadas: `select`, `from`, `where`, `create`, `case`, `var`, `op`, `then`, `table`
- Qualquer palavra não reconhecida é classificada como erro léxico.

Foi construída a tabela de transições para cada token, ocasionando em um AFND (Autômato Finito não Determinístico), pois o estado inicial `q0` possui transições com o mesmo símbolo que levam a diferentes caminhos:

- Com **`c`**, pode ir para `q16` (→ `create`) ou `q25` (→ `case`)
- Com **`t`**, pode ir para `q33` (→ `then`) ou `q37` (→ `table`)
- Com **`w`**, pode ir para `q11` (→ `where`) ou `q29` (→ `when`)

Essas múltiplas transições a partir do mesmo estado e com o mesmo símbolo caracterizam o indeterminismo presente no autômato.

Já no AFD, essas situações foram resolvidas criando estados compostos que unificam os caminhos possíveis para os símbolos com indeterminismo:

- Com **`c`**, vai para o novo estado `q16_q25`, que agrupa os caminhos de `q16` (→ `create`) e `q25` (→ `case`)
- Com **`t`**, vai para o estado `q33_q37`, que representa os caminhos de `q33` (→ `then`) e `q37` (→ `table`)

- Com `w`, vai para o estado `q11_q29`, combinando os caminhos de `q11` (→ `where`) e `q29` (→ `when`)

O autômato foi projetado com base nesses tokens, garantindo que cada cadeia fosse aceita apenas se seguisse exatamente a sequência de caracteres correspondente.

3.2 Implementação

A implementação foi feita em Python. O AFD foi representado como um dicionário de transições e os estados finais foram definidos em uma lista. O programa:

- Lê um arquivo `.txt` contendo as palavras separadas por espaço ou quebras de linha;
- Para cada token, aplica as transições do AFD;
- Se chega a um estado final, o token é reconhecido como válido;
- Caso contrário, o token recebe o rótulo `ERRO`.

A saída do programa consiste em:

- **Fita de saída**: lista dos estados finais alcançados (ou 'X' para erro);
- **Tabela de Símbolos**: contendo linha, identificador e rótulo.

3.3 Validação

Testes foram realizados com entradas contendo apenas palavras válidas, apenas palavras inválidas e uma mistura de ambas. Os resultados mostraram que o analisador reconhece corretamente os tokens válidos e marca como erro qualquer palavra fora da linguagem.

---

4. Conclusões

O trabalho desenvolveu com sucesso um reconhecedor léxico baseado em AFD capaz de identificar palavras reservadas previamente definidas. A aplicação cumpre os requisitos principais: reconhecimento léxico, geração de fita de saída e construção de tabela de símbolos.

Durante a implementação, algumas dificuldades incluíram:

- Representar estados múltiplos de forma legível (ex: `{q11_q29}`)

- Transformar o AFND para AFD

Como perspectiva futura, sugere-se:

- Implementar reconhecimento de identificadores genéricos e números;
- Aumentar a quantidade de tokens reconhecidos e implementar reconhecimento sintático