Descrição do DFA do Lexer da Linguagem RPN

Grupo 4 - Aitor Eler Lucas

Descrição Geral

Este documento descreve o **Autômato Finito Determinístico (DFA)** que modela o comportamento do *lexer* para uma linguagem RPN personalizada. Cada estado representa uma etapa na identificação de tokens válidos, e cada transição corresponde à leitura de um caractere.

Estados Finais

Os seguintes estados representam tokens reconhecidos e são estados de aceitação (círculos duplos no grafo):

Estado	**Token Reconhecido**	**Tipo**
`ParenOpen`	Parêntese esquerdo `(`	PAREN_OPEN
`ParenClose`	Parêntese direito `)`	PAREN_CLOSE
`Operator`	Operadores aritméticos `+ - * / % ^ |`	ARITHMETIC_OP
`Newline`	Quebra de linha (\n)	NEWLINE
`Whitespace`	Espaço em branco	*ignorado
`CompStart`	`< > = !`	COMPARISON_OP
`EqualAfterComp`	`<= >= == !=`	COMPARISON_OP
`Digit`	Número inteiro	NUMBER
`DotInNumber`	Número com ponto (`12.`)	NUMBER
`DigitAfterDot`	Número decimal (`12.34`)	NUMBER
`Minus`	Menos	ARITHMETIC_OP
`Ident`	Identificador genérico	IDENTIFIER
`Error`	Caractere inválido	ERROR
`EndComment`	Fim de comentário após `#`	*ignorado

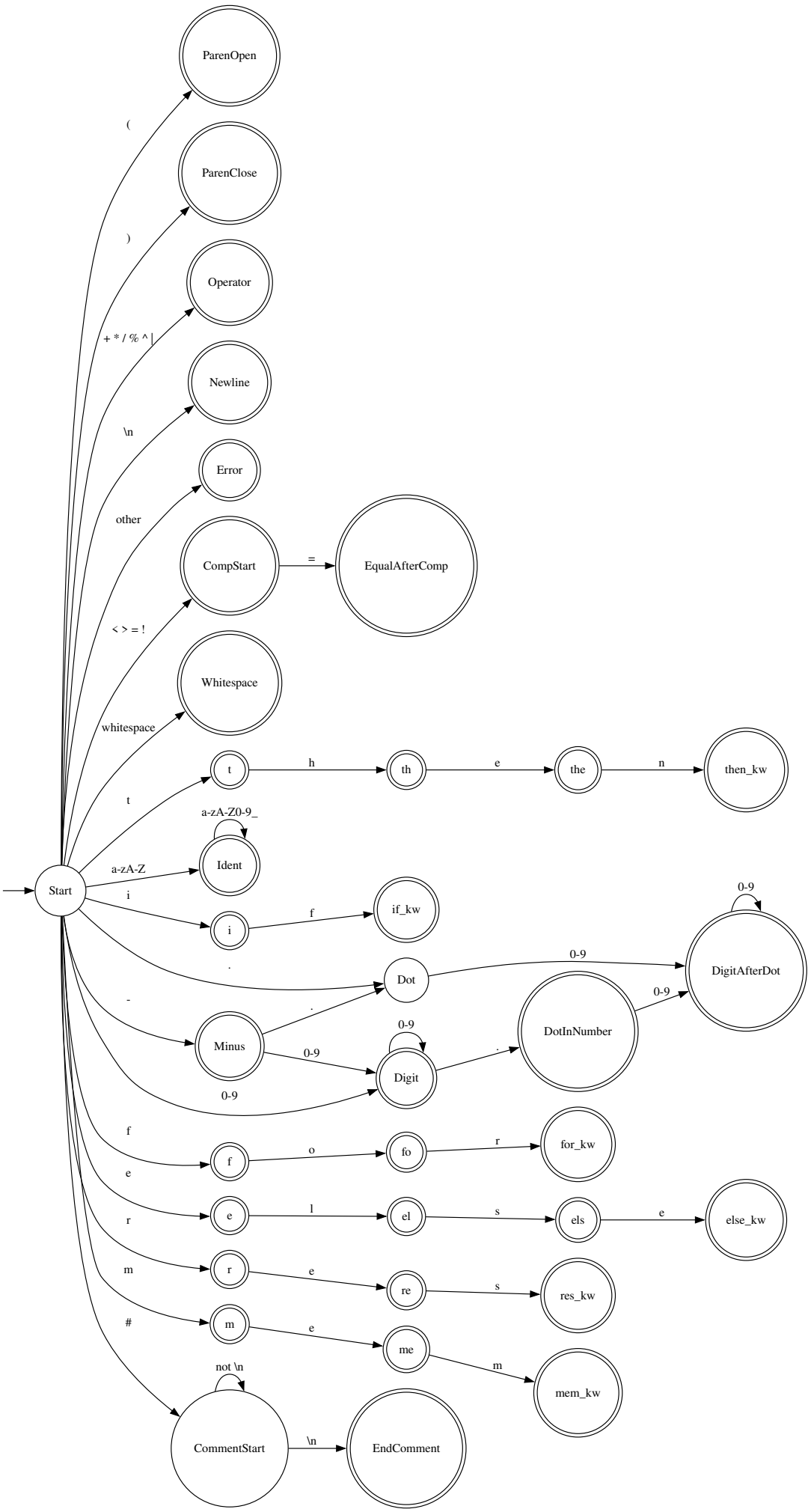

Figura 1: Diagrama do DFA do Lexer

Reconhecimento de Palavras-chave

O DFA distingue palavras-chave específicas por transições sequenciais específicas:

- `if`: Start → i → if_kw
- `then`: Start → t → th → the → then_kw
- `else`: Start → e → el → els → else_kw
- `for`: Start → f → fo → for_kw
- `res`: Start → r → re → res_kw
- `mem`: Start → m → me → mem_kw

Caso essas sequências não sejam completas, elas caem no estado `Ident`, representando identificadores comuns.

Reconhecimento de Números

O DFA reconhece diferentes tipos de números:

- **Inteiros**: `Start` → `Digit` → `Digit...`
- **Decimais**:
  - Começando com ponto: `Start` → `Dot` → `DigitAfterDot...`
  - Com ponto após dígitos: `Digit` → `DotInNumber` → `DigitAfterDot...`
- **Negativos**: `Start` → `Minus` → `Digit` ou `Dot`

Comentários e Espaços

- Comentários iniciam com `#` e permanecem em `CommentStart` até `\n`, momento em que vão para `EndComment`.
- Quebras de linha são tratados separadamente como token distinto.
- Espaços sao ignorados.

Tratamento de Operadores de Comparação

A leitura de símbolos como `<`, `>`, `=`, `!` leva ao estado `CompStart`. Se for seguido por `=`, a transição vai para `EqualAfterComp`, reconhecendo operadores compostos como `<=`, `>=`, `==` ou `!=`.

Tratamento de Erros

Qualquer caractere não pertencente às categorias anteriores resulta em transição para o estado `Error`. Isso permite que o analisador continue mesmo após encontrar tokens inválidos, permitindo diagnósticos mais detalhados.