

Compiladores

Análise lexical (1)

Expressões Regulares

Estrutura de um compilador

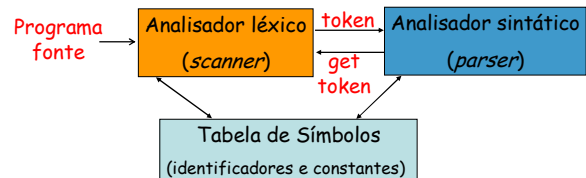


Plano da aula

1. Motivação do uso de Expressões Regulares e definições
 - Linguagens, tokens, lexemas...
2. Regras de formação e exemplos de E.R.
 - Operações
 - Exemplo de uso no Linux: grep.
3. Autômatos Finitos e E.R.
 - Definição e exemplos.
 - Construção de um autômato reconhecedor de E.R.

Análise lexical

- identifica as principais seqüências de caracteres que constituem unidades léxicas (*tokens*)
- começa a construção da tabela de símbolos
- retorna mensagens no caso de erro



Motivação para análise lexical

- **Projeto simples:**
 - separação da análise léxica da sintática faz com que seja possível simplificar cada etapa
- **Eficiência:**
 - em separado, o analisador léxico pode ser mais eficiente
- **Portabilidade:**
 - restrições referentes ao dispositivo podem ser encapsuladas dentro do analisador léxico

Vocabulário básico

- **Lexema:**
 - Conjunto de caracteres no programa fonte
- **Tokens:**
 - Palavras-chaves, operadores, identificadores, constantes, literais, cadeias e símbolos
 - Símbolos terminais na gramática
 - Remete ao analisador sintático!
- **Padrão:**
 - Regra que descreve o conjunto de lexemas que podem representar um *token*

Exemplos: Tokens, Padrões, Lexemas

TOKEN	LEXEMAS	PADRÃO
const	const	const
if	if	if
relation	<, <=, =, <>, >, >=	< ou <= ou = ou <> ou ...
id	pi, contador, D2	letra seguida por letras ou dígitos
num	3.1416, 0, 6.0E23	constante numérica
literal	"compiladores"	caracteres entre aspas

Como definir padrões ?

- Enumerar os valores possíveis:
 - <, <=, =, <>, >, >=
- Usar conceitos de linguagens formais
- 4 categorias de linguagens (classificação de Chomsky):
 - linguagens regulares - tipo 3
 - Autômato de estados finitos
 - linguagens livres de contexto – 2
 - Autômato com pilha
 - linguagens sensíveis ao contexto - tipo 1
 - Máquina de Turing com fita limitada
 - recursivamente enumeráveis - tipo 0
 - Máquina de Turing

No contexto de LP, as linguagens são representadas por gramáticas ou por autômatos que as reconhecem.

Linguagens Formais

- **Símbolo:**
 - entidade abstrata. Ex: letras, dígitos, caracteres especiais, etc.
- **Alfabeto (Σ):**
 - Um conjunto finito de símbolos. Ex: $\Sigma = \{0, 1\}$
- **String (cadeia, frase):**
 - Uma sequência finita de símbolos de um determinado alfabeto
- **String vazio (ϵ):**
 - string com 0 símbolos
- **Linguagem Formal (Σ^*):**
 - Conjunto de todos os possíveis strings de um alfabeto

Expressão Regular (ER)

- A definição de E.R. é **recursiva**:
 - Uma ER "básica" é definida por literais
 - ER complexas são definidas pela combinação de ER básicas.
- Regras para formação de palavras válidas
 - **Concatenação:**
 - xy (x seguido de y)
 - **Alternação:**
 - $x | y$ (x ou y)
 - **Repetição(0):**
 - x^* (x repetido 0 ou mais vezes)
 - **Repetição(1):**
 - x^+ (x repetido 1 ou mais vezes)

Precedências em Expressões Regulares

- Operador unário $^*, ^+$
- Concatenação
- Operador $|$
- Usa-se parênteses em casos ambíguos.
- Exemplo:
 - $(a) | ((b)^*(c))$ é equivalente a $a | b^*c$
 - $a(b|c)d$ é diferente de $ab|cd$

Exemplos de E. R.

- 00 : representa o conjunto $\{00\}$
- $(0 | 1)$: representa o conjunto $\{0, 1\}$
- $(0 | 1)(0 | 1)$: representa o conjunto $\{00, 01, 10, 11\}$
- 0^* : $\{\epsilon, 0, 00, 000, 0000, 00000, \dots\}$
- $(0 | 1)^*$: representa todos os strings de 0's e 1's
- $(0 | 1)^* 00 (0 | 1)^*$: representa todos os strings de 0's e 1's com pelo menos dois 0's consecutivos
- Todos os strings de 0's e 1's começando por 1 e não tendo dois 0's consecutivos ?

Propriedades Algébricas

$r s$	$s r$	comutativa
$r (s t)$	$(r s) t$	associativa
$(rs)t$	$r(st)$	concatenação associativa
$r(s t)$	$rs rt$	concatenação distributiva
$(s t)r$	$st sr$	concatenação distributiva
ϵr	r	ϵ é identidade da concatenação
$r \epsilon$	r	ϵ é identidade da concatenação
r^*	$(r \epsilon)^*$	relação entre $*$ e ϵ
r^{**}	r^*	$*$ é idempotente

Definições regulares

- Associar nomes a expressões regulares
- Seja Σ um alfabeto de símbolos básicos
 - Definição regular
 - $d_1 \rightarrow r_1$
 - $d_2 \rightarrow r_2$
 - $\vdots$
 - $d_n \rightarrow r_n$
 - d_i são nomes distintos
 - r_i expressão sobre os símbolos $\Sigma \cup \{d_1, d_2, \dots, d_n\}$

Exemplo: o token 'num'

- Representação de números sem sinal :
5280, 39.37, 6.336E4, 1.894E-4

$\text{num} \rightarrow \text{digitos fracao_opc expoente_opc}$
 $\text{digito} \rightarrow 0 | 1 | \dots | 9$
 $\text{digitos} \rightarrow \text{digito digito}^*$
 $\text{fracao_opc} \rightarrow \text{.digitos} | \epsilon$
 $\text{expoente_opc} \rightarrow (\text{E} (+ | - | \epsilon) \text{digitos}) | \epsilon$

Outras notações e simplificações

- E^+ é equivalente a EE^*
 - E é presente pelo menos uma vez.
- $E^?$ é equivalente a $\epsilon | E$
 - E é presente zero ou uma vez.
- $[xy]$ é equivalente a $x|y$
 - Sintaxe de sed/grep
- $[a-z]$ é equivalente a $a|b|c|\dots|z$
 - $[0-9]$
 - $[A-Z]$
 - $[^a-l]$: negação de $[a-l]$
- $\wedge \dots \$$ delimitam uma linha.
 - Obs: estamos introduzindo uma noção de "contexto"!

Exercícios

- Descrever as linguagens seguintes:

- $(a| \epsilon)(b|ba)^*$
- $0^*10^*10^*10^*$

Expressões Regulares comuns em compiladores

```

if      → if
then    → then
else    → else
relop   → < | <= | = | > | >= | <=
idf      → letra (letra | digito)*
num      → digito*(.digito)?(E(+|-)?digito)?
space    → ('\' | \" | \t)+
    
```

Problema seguinte: reconhecer se um lexema está de acordo com uma E. R.:
Será que "tmp_1" é um idf?

Definição de Autômato Finito

- **Autômato finito M sobre um alfabeto Σ** é uma 5-upla $(K, \Sigma, \delta, e_0, F)$, onde:
 - K é o conjunto finito de estados
 - Σ é o alfabeto dos símbolos da linguagem
 - $\delta : K \times \Sigma \rightarrow K$ é a função de transição de estados
 - e_0 é o estado inicial
 - F é o conjunto de estados finais
- δ é uma **função parcial**, pois não precisa estar definida para todos os pares $K \times \Sigma$
- **Interesse**: há equivalência entre AEF e ER

Exemplo de autômato finito

$M = (K, \Sigma, \delta, e_0, F)$
 $\Sigma = (d, .)$
 $K = (e_0, e_1, e_2, e_3)$
 $F = (e_1, e_3)$

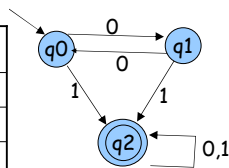
$\delta(e_0, d) = e_1$
 $\delta(e_1, d) = e_1$
 $\delta(e_1, .) = e_2$
 $\delta(e_2, d) = e_3$
 $\delta(e_3, d) = e_3$

	d	.
e_0	e_1	
e_1	e_1	e_2
e_2	e_3	
e_3	e_3	

Autômato que reconhece números inteiros e reais

Exemplo de autômato finito

	Estado Atual	0	1
Inicial	q0	q1	q2
	q1	q0	q2
Final	q2	q2	q2



Qual expressão regular corresponde a este autômato ?

Dois tipos de autômatos: AFD e AFND

- Um **Autômato Finito Não-Determinístico (AFND)**
 - Tem um conjunto de estados S
 - Uma entrada definida como um string s de um alfabeto Σ
 - Uma função de transição $(S, \Sigma) \rightarrow S^+$
 - Um estado de partida
 - Um conjunto de estados finais (de aceitação)
- Um **Autômato Finito Determinístico**
 - É um AFND
 - Não tem **ϵ -transição**
 - **No máximo uma** transição saindo de um estado com rótulo um dado símbolo.

Da ER ao Autômato (AFND)

- Construção de Thompson
 - Cada ER “básica” se traduz em AFND
 - Pode-se agregar os AFND conforme se agregam as ERs.
 - Segue as regras de definição de uma ER:
 - ERs básicas
 - Alternativa, repetição, concatenação...
 - Cada AFND tem exatamente um estado de partida e um estado final.

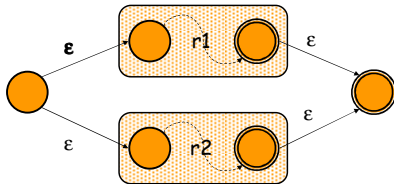
Reconhecedores básicos

- AFND para reconhecer ϵ
- AFND para reconhecer um símbolo x



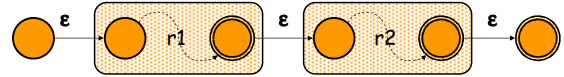
Reconhecedor de alternativa

- AFND que reconhece a alternativa $(r1|r2)$



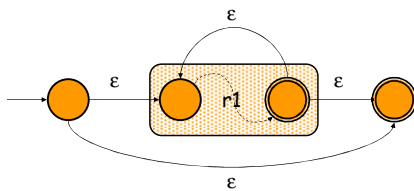
Reconhecedor de concatenação

- A partir de dois AFNDs que reconhecem $r1$ e $r2$, pode-se reconhecer $r1r2$ assim:



AFND reconhecedor de r^*

- A partir do AFND que reconhece r , pode-se criar um AFND que reconhece r^*



Exemplo:

- Construir um AFND que reconheça $(a|b)^*abb$

Do AFND ao AFD

- Problema: as ϵ -transições e as transições múltiplas atrapalham!
 - Não há como determinar, simplesmente, a partir de um símbolo de entrada, qual estado atingir.
- É muito mais eficiente trabalhar com AFDs para reconhecer uma linguagem.
 - Por outro lado, é mais complicado obter o AF a partir da ER!
- Solução: transformar automaticamente um AFND em AFD!
- ... Será o assunto da **próxima** aula!
- ... E terá mais: o FLEX.

Referências

- Livro do Dragão, cap. 3
 - Tokens e uso de ERs: 3.3
 - Autômatos: 3.6
 - Construção de Thompson: 3.7
- Série Didática:
 - Compiladores: Cap. 2
 - Linguagens Formais (Blauth): Cap. 3