Universidade de São Paulo

PCS5730: Projeto e Técnicas de Construção de Compiladores

There and back again: Convertendo autômatos para gramáticas

Lucas Virgili

1º Semestre de 2014

Sumário

1 Resumo

Neste trabalho, propomos o desenvolvimento de um conjunto de ferramentas que será usado para obter uma gramática equivalente a um autômato, representado em forma de matriz.

2 Introdução

É bastante simples, a partir de uma gramática livre de contexto (as gramáticas do tipo 2 na hierarquia de Chomsky[1]) definida em notação de Wirth, por exemplo, obter-se um autômato que reconhece uma sentença escrita na gramática como correta ou incorreta[2].

Não só, Kleene[3] mostrou a equivalência entre as linguagens aceitas por autômatos finitos e expressões regulares. Não só, uma linguagem aceita por um autômato finito não deterministico pode ser representada por uma expressão regular.

Assim, é possível, pelo menos teoricamente, obter-se a partir de um autômato uma expressão regular equivalente, ou seja, que aceite as mesmas sentenças corretas.

Também sabemos que as linguagens geradas pelas gramáticas do tipo 3 na hierarquia de Chomsky definem as expressões regulares. Logo, a partir de uma expressão regular podemos criar uma gramática que a gere. Claro que não necessariamente a melhor gramática, mas uma equivalente.

O objetivo deste trabalho foi propor um conjunto de ferramentas que, dado um autômato, gera uma gramática equivalente a ele.

3 Expressões regulares, autômatos e gramáticas

3.1 De autômatos para expressões regulares

Diversos métodos existem para obter-se uma expressão regular equivalente a um autômato finito determinístico[4, 5].

Infelizmente, o problema de se obter uma expressão regular mínima é tanto PESPAÇO-completo como NP-completo[6].

De forma geral, o tamanho da expressão está relacionado à ordem na qual os estados do autômato são examinados. Logo, heurísticas são utilizadas para obter-se uma expressão de tamanho aceitável em tempo e espaço razoáveis.

3.1.1 Obtendo a expressão regular - Método baseado no fecho de Kleene

Sejam $\{q_1, q_2, \ldots, q_k\}$ os estados do autômato.

A ideia é usar um tipo de "indução": seja $R^k_{i,j}$ a expressão regular para as entradas indo do estado i ao estado j, usando somente os k primeiros estados do autômato.

Para cada par i, j, suponha que $R_{i,j}$ represente a expressão de q_i até q_j, mas sem usar o estado q_k. Agora, se pudermos usar o estado q_k, podemos montar o novo R:

$$R^k_{i,j} = R^{k-1}_{i,j} + R^{k-1}_{i,k}.R^{k-1*}_{k,k}.R^{k-1}_{k,j} \tag{1}$$

Isso quer dizer que, para irmos de i a j, podemos usar o que já sabíamos ou ir de i até o estado k, ficar em *loop* em k e depois irmos de k até o estado j.

O algoritmo, em "pseudo python":

```
# Automato Sigma:
# n estados
# R matriz

# Inicializacao:
for i in range(1, n):
    for j in range(1, n):
        if i == j:
            R[i][j][0] = epsilon
        else:
            R[i][j][0] = nada
        for terminal in Sigma:
            if transicao(i, terminal, j):
                R[i][j][0] = R[i][j][0] + terminal

# Inducao
for k in range(1, n):
    for i in range(1, n):
        for j in range(1, n):
            R[i][j][k] = R[i][j][k-1] + R[i][k][k-1]
            . estrela_kleene(R[k][k][k-1]) . R[k][j][k-1]

# Regex
inicio = find_inicio(Sigma)
for i in range(1, n):
    if is_final(i):
        regex = regex + R[inicio][i][n]
```

Esse algoritmo é claramente exponencial, tanto em espaçøquanto tempo. Contudo, ele demonstra a possibilidade "prática" da conversão.

3.1.2 Obtendo a expressão regular - *State elimination*

Análises experimentais feitas durante esse trabalho mostraram que a heurística conhecima como *state elimination*[7] gera expressões regulares bastante pequenas em tempo aceitável quando comparada com diversas outras heurísticas propostas em [8]. Portanto ela foi utilizada na nossa ferramenta.

O algoritmo de remoção de estado está descrito a seguir. Primeiro precisamos normalizar o autômato, isto é, o estado inicial não tem transições para ele, há apenas um estado final e este estado não tem transições saindo dele. É trivial obter um autômato normalizado a partir de um não normalizado, basta adicionar dois estados "falsos" representando os estados iniciais e finais e adicionar as transições em vazio adequadas.

Seja Q o conjunto de estados do autômato, i o estado inicial e f o final. Também denotamos por $\delta q, a$ o conjunto $\{p \in Q | (q, a, p) \in \delta\}$, onde δ são as transições, a é um símbolo e $q \in Q$.

1. **Normalização**
2. **Eliminação de estado**
   (a) Se $Q = \{i, f\}$, a expressão procurada é α_{if} e o algoritmo termina. Caso contrário vá para o próximo passo.
   (b) Escolha um $q \in Q \backslash \{i, f\}$ e elime q do autômato, que passa a ter $Q \backslash \{q\}$ como conjunto de estados. Para cada estado q_1, q_2 nesse novo conjunto de estados faça:

$$\delta'(q_1, q_2) = \alpha_{q1q2} + \alpha_{q1q}\alpha_{qq} * \alpha_{qq2}$$

   Em seguida, retorne para o passo (a).

4 Ferramenta para obter a gramática a partir de um autômato

Neste trabalho, desenvolemos duas ferramentas: a primeira, dada uma matriz representando um autômato, computa a representação do autômato em expressão regular; a segunda converte uma expressão regular em uma gramática na notação escolhida pelo usuário. Logo, utilizando as duas ferramentas em um *pipeline*, obtemos uma gramática equivalente ao autômato. A figura 1 representa o processo.

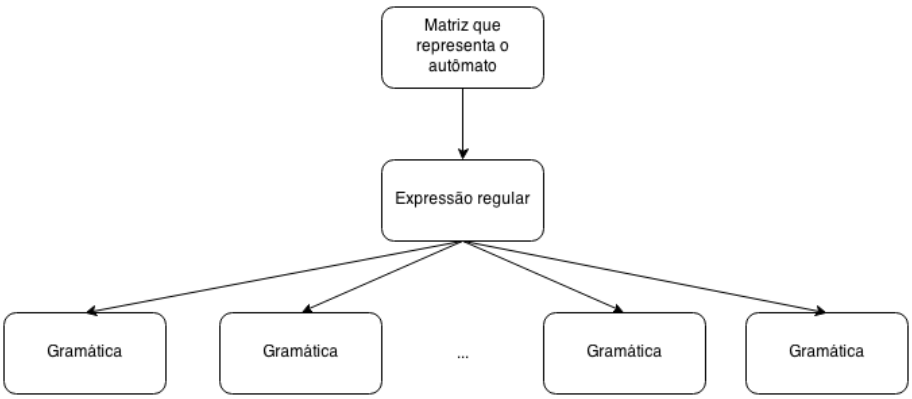

Figura 1: Otenção de gramáticas a partir do autômato

4.1 Do autômato para a expressão regular

O programa "aut2regex.py" recebe como parâmetro um arquivo que representa um autômato e imprime uma expressão regular equivalente ao autômato.

O arquivo de entrada deve seguir o seguinte formato:

- "#" é um comentário
- "@DFA" ou "@NFA" declara um novo autômato e o seu tipo. Deve ser seguido pela lista de estados finais
- Se for um NFA, pode haver um "*" seguido da lista de estados iniciais
- Um "$" opcional seguido da lista de símbolos do alfabeto
- A lista das transições, que deve ser da seguinte forma:
  - "estado" "símbolo" "novo estado"
- Uma transição em vazio é denotada por "@epsilon" no caso de um NFA

Por exemplo, o autômato representado na figura 2 tem o seguinte arquivo de entrada:

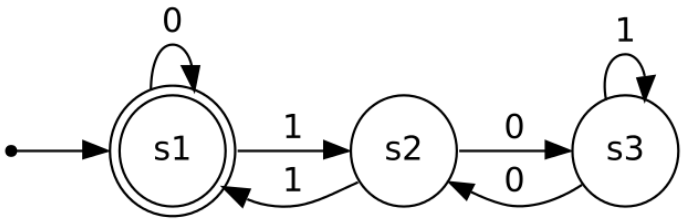

Figura 2: Autômato exemplo

```
@DFA 0
0 1 1
0 0 0
1 1 0
1 0 2
2 1 2
2 0 1
```

O programa também pode ser usado sozinho para gerar expressões regulares equivalentes.

4.2 Da expressão para a gramática

Usando uma pilha e analisando a expressão é simples obter uma gramática equivalente em notação de Wirth[1].

O programa "regex2grammar.py" converte uma expressão regular passada como parâmetro para uma gramática e a imprime na saída padrão.

[1] A única que eu tive tempo de implementar.

4.3 A união dos dois

Seguindo a filosofia do UNIX, as duas ferramentas acima podem ser unidas através de um pipe e, a partir do arquivo que representa o autômato, imprimir a gramática equivalente:

```
python aut2regex.py arquivo_do_automato | python regex2grammar.py
```

5 Conclusão

Referências

[1] Noam Chomsky, *Three models for the description of language*, IRE Transactions on Information Theory, (2): 113-24, 1956

[2] Dick Grune and Ceriel J. H. Jacobs, *Parsing Techniques: A Practical Guide*, Springer, 2007

[3] S. C. Kleene, *Representation of events in nerve nets and finite automata*, Automata Studies, 3-41, Princeton University, 1956

[4] K. Ellul, B. Krawetz, J. Shallit e M. Wang, *Regular expressions: New results and open problems*, J. Aut., Lang. and Combin., 10(4): 407-37, 2005

[5] H. Gruber and M. Holzer, *Provably shorter regular expressions from deterministic finite automata*, Proceedings of the 12th International Conference on Implementation and Application of Automata, (5462): 188-97, Springer, 2008

[6] T. Jiang and B. Ravikumar, *Minimal NFA problems are hard* SIAM Journal of Computation, 1117-41, 1993

[7] J. A. Brzozowsk and E. J. McCluskey Jr., *Signal flow graph techniques for sequential circuit state diagrams*, IEEE Trans. on Electronic Computers, EC-12(2): 67-76, 1963

[8] Nelma Moreira, Davide Nabais e Rogério Reis, *State Elimination Ordering Strategies: Some Experimental Results*, 12th International Workshop on Descriptional Complexity of Formal systems, 139-48, 2010