Capítulo 9: Linguagens sensíveis ao contexto e autômatos linearmente limitados.

José Lucas Rangel

9.1 - Introdução.

Como já vimos anteriormente, a classe das linguagens sensíveis ao contexto (lsc) é uma classe intermediária entre a classe das linguagens recursivas e a classe das linguagens livres de contexto (llc). Já vimos anteriormente que a classe das llc está propriamente contida na classe das lsc's: como aplicação do Lema do Bombeamento, provamos que alguns exemplos de linguagens geradas por gramáticas sensíveis ao contexto não são llc's.

Neste capítulo, queremos apresentar os autômatos linearmente limitados, uma classe de máquinas de Turing cuja operação é restringida, de forma a aceitar apenas as lsc's. Adicionalmente, queremos mostrar que a inclusão da classe das lsc's na classe das linguagens recursivas é própria, através de um exemplo de linguagem recursiva que não é uma lsc.

9.2 - Autômatos linearmente limitados.

Um autômato linearmente limitado (all) é uma máquina de Turing não determinística $A = <K, \Gamma, \Sigma, \delta, i, F>$, que satisfaz certas restrições:

- dois símbolos especiais, um marcador esquerdo [e um marcador direito], são incluídos em Γ;
- quando [é lido, a máquina não pode se mover para a esquerda, nem escrever um símbolo distinto de [;
- quando] é lido, a máquina não pode se mover para a direita, nem escrever um símbolo distinto de].
- [e] não podem ser escritos em nenhuma outra situação.

Supondo que a fita de um all contém inicialmente uma sequência da forma [x], estas restrições fazem com que o all não possa abandonar a região entre os dois marcadores [e], nem apagá-los ou movê-los. Portanto, todas as computações de um all devem ser feitas no espaço que originalmente contém sua entrada x.

Como já vimos anteriormente, é possível definir uma fita com k trilhas, e, num trecho da fita de comprimento n, podemos armazenar kn símbolos. Esta é a origem do nome "linearmente limitado": o all pode manipular em sua fita informação com tamanho limitado por uma função linear do tamanho de sua entrada.

Definimos configurações e transições para um all da mesma forma que para uma mT. Entretanto, a configuração inicial associada à entrada $x \in \Sigma^*$ é i [x]. Definimos a linguagem de um all A, por

$$L(A) = \{ x \in \Sigma^* \mid i [x] \vdash\!\!\!\!-^* [\alpha f \beta], \text{ com } f \in F \}$$

Exemplo: Vamos descrever informalmente um all A que aceita a lsc $\{ x x \mid x \in \Sigma^* \}$, sendo $\Sigma = \{ a, b \}$.

Inicialmente, a fita de A contém [y]. Em seguida, A marca o primeiro símbolo de y, e o primeiro símbolo da segunda metade de y, que verifica ser igual ao

primeiro. (A posição inicial da segunda metade é escolhida de forma não determinística.) A seguir, o processo se repete com os demais símbolos das duas metades, marcando o primeiro símbolo não marcado da primeira metade e o primeiro símbolo não marcado (igual) da segunda metade, até que nenhum símbolo não marcado reste na fita.

Atingido este ponto, A verificou que o ponto escolhido como início da segunda metade era o correto, porque havia um número igual de símbolos nas duas metades, e além disso, que os símbolos correspondentes das duas metades eram iguais em todos os casos. Portanto, a cadeia de entrada y pertence à linguagem, e pode ser aceita por A, que passa para um estado final, onde pára.

Exercício: Construa um all que aceite a linguagem $\{x\ x \mid x \in \Sigma^*\}$, onde $\Sigma = \{a, b\}$, em todos os detalhes.

9.3 - All's e lsc's

Vamos agora provar dois teoremas que mostram que a classe das linguagens aceitas por all's não determinísticos e a classe das linguagens sensíveis ao contexto (lsc's) são idênticas.

Teorema 9.1: Toda lsc é reconhecida por um all não determinístico.

Dem.: Seja L uma lsc, e G uma gsc que gera L. Vamos mostrar como construir um all A, que aceita L(G). Para verificar que $x \in L(G)$, A simula uma derivação $S \Rightarrow^* x$ em sua fita.

Primeiro, observamos que, se G tem uma regra $S \rightarrow \varepsilon$, A deve aceitar a entrada vazia, identificada pelo conteúdo de fita []. Este caso deve ser tratado especialmente, porque a derivação $S \Rightarrow \varepsilon$ não pode ser simulada sem um espaço de comprimento pelo menos $|S| = 1$.

Para considerar as demais entradas $x = x_1\ x_2\ \ldots\ x_n$, A inicialmente divide sua fita em duas trilhas, copiando x na primeira trilha, e preparando a segunda para simular uma derivação de x. Para representar as duas trilhas, vamos usar pares de símbolos, de forma que

$$(a_1, b_1)\ (a_2, b_2)\ \ldots\ (a_n, b_n)$$

representa uma fita com duas trilhas, que contêm, respectivamente, $a_1a_2\ldots a_n$ e $b_1b_2\ldots b_n$.

Inicialmente, a fita de A contém

$$[\ x_1\ x_2\ \ldots\ x_n\ ].$$

Dividindo a fita em duas trilhas, temos

$$[\ (x_1, S)\ (x_2, \Diamond)\ \ldots\ (x_n, \Diamond)\ ]$$

Na segunda trilha, que contém inicialmente $S\Diamond\ldots\Diamond$, A simula uma derivação em G, sem alterar o conteúdo da primeira trilha, escolhendo as regras de forma não determinística. Supondo-se, naturalmente, que $x \in L$, uma derivação de x pode ser simulada, e ao final da simulação, a fita conterá duas cópias de x, uma em cada trilha:

$$[\ (x_1, x_1)\ (x_2, x_2)\ \ldots\ (x_n, x_n)\ ]$$

Neste ponto, A verifica que o conteúdo da primeira trilha é idêntico ao da segunda, e passa para um estado final.

A idéia central da demonstração é a de que, como G é uma gsc, as formas sentenciais intermediárias entre S e x, na derivação $S \Rightarrow^* x$, tem todas comprimento menor ou igual ao de x, e podem portanto ser escritas na segunda trilha.

Nota: a demonstração acima usa o fato de que o all A é não-determinístico de forma essencial, e essa hipótese não pode ser retirada, pelo menos de forma simples. Na demonstração, apenas uma forma sentencial é construída a cada vez, sendo a escolha da regra e da maneira de sua aplicação feita de forma não-determinística. A verificação final de que a cadeia x foi obtida, verifica a correção da escolha. (Escolhas erradas não poderiam derivar x, e portanto não levariam à aceitação.)

No caso de um all determinístico, entretanto, isto não seria possível, e seria necessário, em princípio, examinar todas as formas sentenciais possíveis, até o comprimento n da entrada, como foi feito no algoritmo usado para demonstrar que todas as linguagens sensíveis ao contexto são conjuntos recursivos. Não é possível, entretanto, considerar todas as formas sentenciais simultaneamente por causa da restrição de espaço, porque o número de formas sentenciais de comprimento menor ou igual a n pode ser exponencial em n. Uma outra alternativa seria tratar uma forma sentencial por vez, e anotar apenas o caminho percorrido durante a derivação da forma sentencial corrente (que regras foram aplicadas em que pontos, por exemplo), de forma a recuar (*backtrack*) posteriormente, se a cadeia x não for atingida, para voltar e considerar as alternativas restantes. Isto também não é possível, pois o comprimento do caminho também pode ser exponencial em n.

Como esta demonstração não pode ser adaptada para o caso determinístico, pelo menos de forma simples, e nenhuma outra demonstração alternativa foi descoberta, o problema da equivalência das classes das linguagens aceitas por all's determinísticos e all's não determinísticos é um problema ainda em aberto.

Teorema 9.2: Toda linguagem aceita por um all é uma lsc.

Dem.: Seja A um all (não determinístico). Para aceitar uma entrada x, A realiza uma computação

$$i\ [\ x\ ] \vdash\!\!\!-^*\ \alpha\ f\ \beta$$

onde i é o estado inicial, e f um dos estado finais de A. Vamos construir uma gsc G que simula esta computação em cada derivação. Note, entretanto, que uma derivação de x em G deve partir do símbolo inicial S, e terminar com x, e que a computação acima só pode ser iniciada após a definição de x. Usando símbolos não terminais da forma (a, b, c, d), podemos simular quatro "trilhas":

- a primeira trilha guarda uma cópia de x, que não será alterada;
- a segunda guarda outra cópia, que será usada na simulação;
- a terceira guarda o estado corrente e a posição da cabeça, durante a simulação;
- a quarta tem as marcas que indicam as extremidades da fita.

A derivação se dá em várias fases:

$$\text{fase 1: } S \Rightarrow^* (x_1, x_1, i, [\,])\ (x_2, x_2, \Diamond, \Diamond)\ \ldots\ (x_n, x_n, \Diamond,])$$

Nesta primeira fase, são geradas as duas cópias de $x=x_1x_2\ldots x_n$, e é preparada a configuração inicial, com o estado i na primeira posição.

fase 2: $(x_1, x_1, i, [])\ (x_2, x_2, \Diamond, \Diamond) \ldots (x_n, x_n, \Diamond,])$
$\Rightarrow^* (x_1, z_1, \Diamond, [])\ (x_2, z_2, \Diamond, \Diamond) \ldots (x_m, z_m, f, \Diamond) \ldots (x_n, z_n, \Diamond,])$

Na segunda fase, é feita a simulação de A até que uma configuração final $\alpha f \beta$ seja obtida, com $\alpha = z_1 z_2 \ldots z_{m-1}$ e $\beta = z_m z_{m+1} \ldots z_n$.

fase 3: $(x_1, z_1, \Diamond, [])\ (x_2, z_2, \Diamond, \Diamond) \ldots (x_m, z_m, f, \Diamond) \ldots (x_n, z_n, \Diamond,]) \Rightarrow^* x_1 x_2 \ldots x_n$

Na última fase, a partir dos símbolos em que o estado (a terceira componente) é final, é feita a substituição, de forma a deixar apenas os símbolos da primeira "trilha".

Os detalhes da construção da gramática ficam como exercício para o leitor.

9.4 - Uma linguagem recursiva que não é sensível ao contexto.

Vamos agora apresentar uma linguagem L, definida a seguir, que é recursiva, mas não é uma lsc. Para definir L, vamos inicialmente supor uma enumeração das gsc's, G_0, G_1, G_2, …, limitadas ao caso em que o alfabeto dos terminais contém apenas 0 e 1. Como feito anteriormente com máquinas de Turing, essa enumeração pode ser baseada na codificação das gramáticas em cadeias em um alfabeto adequado.

Apenas para fixar as idéias, podemos supor que os símbolos nãoterminais são codificados como n, n|, n||, …, sendo o nãoterminal inicial indicado apenas por n. Dessa forma, para codificar uma gramática $G = < N, \Sigma, P, S >$, não há necessidade de incluir na codificação de uma gramática o conjunto Σ de seus terminais (0 e 1 podem ser representados simplesmente por 0 e 1), o conjunto de nãoterminais N (sempre representados por n, n|, n||, …, ou o símbolo inicial S (codificado como n). Basta assim apenas codificar as regras da gramática, que podem ser separadas por #.

Com isso, o alfabeto pode ser $\Delta = \{0, 1, n, |, \rightarrow, \#\}$. Como fizemos na enumeração das máquinas de Turing, a enumeração deve eliminar as cadeias de Δ^* que não correspondem a gsc's, e substituí-las por alguma gsc, por exemplo ε, que representa a gsc "vazia", que não tem nenhuma regra, e que gera a linguagem vazia $\varnothing$.

Para cada gramática G_i, é possível construir uma mT que sempre pára e que reconhece a linguagem de G_i. Seja M_i a mT construída a partir de G_i. Portanto, a enumeração M_0, M_1, M_2, … inclui mT's que aceitam todas as lsc's.

Defina a linguagem L por

$L = \{ x_i \mid M_i \text{ não aceita } x_i \}$

Fato: L é recursiva.

Dem.: Note que, por construção, todas as máquinas M_i aceitam conjuntos recursivos, e param para todas suas entradas. L é aceita por uma mT M que

- a partir de sua entrada x, M determina i tal que $x=x_i$, e constrói uma representação de M_i.
- simula M_i com entrada x_i.
- M aceita $x=x_i$ se e somente se M_i não aceita x_i, ou seja, se M_i, com entrada x_i, pára em um estado que não é final.

Fato: L não é uma lsc.

Dem.: Por contradição. Suponha que L é uma lsc. Neste caso, L tem uma gsc $G=G_i$, e é aceita pela máquina M_i: $L = L(G_i) = L(M_i)$. Mas então

$x_i \in L \Leftrightarrow M_i$ não aceita $x_i \Leftrightarrow x_i \notin L(M_i) \Leftrightarrow x_i \notin L$,

estabelecendo uma contradição. Concluímos, portanto, que L não é uma lsc.

As duas propriedades acima estabelecem que a classe das linguagens sensíveis ao contexto está propriamente contida na classe das linguagens recursivas.

(julho 99)