MOITRA,A; MUDUR,S. P; NARWEKAR,A. W. Desing and analysis of a hiphenation procedure. Software - Practice and Experience, 9: 325-337, Sep. 1979.

MURAKAMI,T. K.; JUNIOR, A, B. PAGE - Sistema de edição de página. Trabalho de Graduação em Tecnologia de Computação. São José dos Campos, ITA, dez. 1981.

UM MÉTODO DE COMPACTAÇÃO DE TABELAS LR(1)

Roberto da Silva Bigonha

Mariza Andrade da Silva Bigonha

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais
Caixa Postal 702
30.000 - Belo Horizonte - MG

SUMÁRIO

Um método de compactação de tabelas LR(1) baseado em propriedades dos reconhecedores LR(1) é apresentado e discutido. O método proposto basea-se no esquema de listas de Aho e Ullman e visa reduzir o espaço necessário às tabelas sem comprometer o seu tempo de acesso.

1. INTRODUÇÃO

Uma tabela SLR(1), LALR(1) e possivelmente LR(1) [Spector 81, Pager 77] para uma gramática como a do PASCAL possuem tipicamente, cerca de 300 linhas (estados) e 100 colunas (terminais e não-terminais).

Tomando 300 estados e 100 produções como valores típicos, pelo menos 11 bits seriam necessários para codificar cada entrada da tabela. Consequentemente, uma tabela 300X100 consumiria pelo menos 330.000 bits, ou seja, cerca de 40 KBytes.

Na prática, a memória usada seria ainda maior devido ao fato de o tamanho de cada entrada não ser necessariamente múltiplo (ou divisor) da menor unidade de memória endereçável na maior parte dos computadores. Entretanto, tabelas LR(1) são, em geral, esparsas, isto é, grande parte de suas entradas são vazias (entradas indicativas de erro). A implementação direta dessas tabelas como matrizes causaria um enorme desperdício de memória. Claramente, algum método eficiente de codificação de matrizes esparsas é necessário.

Este trabalho apresenta uma técnica de compactação de tabelas LR(1) que permite uma redução substancial no consumo de memória sem comprometer o tempo de acesso às tabelas.

2. ESTRUTURA DO ANALISADOR LR(1)

Um reconhecedor LR(1) é constituído por um algoritmo básico, uma pilha e uma tabela. A pilha é usada para armazenar estados do reconhecedor, sendo o estado corrente aquele que estiver no seu topo. A tabela LR(1) em sua forma original é uma matriz retangular, onde os índices de linhas são nomes de estado, usualmente inteiros e os índices das colunas são terminais e não-terminais da gramática que deu origem a tabela. Terminais e não-terminais também são geralmente representados como números inteiros para facilitar indexação em implementação reais.

Por razões técnicas, as tabelas LR(1) são, em geral, divididas em duas partes, denominadas ACTION e GOTO respectivamente. Na parte ACTION os índices de colunas representam terminais e na parte GOTO não-terminais.

Cada entrada na sub-tabela ACTION pode ter um dos quatro valores:

1. SHIFT s, onde s é o número de um estado. Este valor denota a ação "empilhar o estado s".
2. REDUCE p, onde p é o número de uma produção da forma $A \rightarrow \omega$, sendo A não-terminal e ω uma sequência de símbolos gramaticais. REDUCE p denota a ação "reduzir segundo a p-ésima produção".
3. ACCEPT.
4. ERROR (são valores em branco na tabela).

Cada entrada na parte GOTO da tabela LR(1) contém o número de um estado ou o valor ERROR (branco).

A figura 1 mostra um exemplo de tabela LR(1), onde a, b, d, e, f são terminais; A, B, C são não-terminais e $ um símbolo especial que marca o fim da entrada; sk denota a ação SHIFT k; rp a ação REDUCE p, sendo p o número da produção e ACC representa ACCEPT.

	a	b	d	e	f	$	A	B	C
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Figura 1: Tabela LR(1)

A figura 2 apresenta o algoritmo básico do analisador, o qual supõe que a tabela LR(1) esteja armazenada como duas matrizes retangulares, ACTION e GOTO. O procedimento lex retorna a cada chamada o próximo símbolo da entrada na forma de um par (u, v), onde u identifica o símbolo e v denota o seu valor quando aplicável.

```
var analisa: lógico
    u      : terminal;
    v      : valor;
    ACTION : arranjo [estado, terminal] de ação;
    GOTO   : arranjo [estado, não_terminal] de estado;
    SIN    : arranjo [0..MAX] de estado;
    topo   : 0..MAX;
    k, s   : estado;
    A      : não_terminal;
    p      : número_de_produção;
início
    topo := 0; SIN[topo] := estado_inicial;
    analisa := verdadeiro;
    ...preenche tabelas ACTION e GOTO...
    lex(u, v); s := estado_inicial;
    enquanto analisa repita
      caso ACTION[s,u]
        seja SHIFT k : topo := topo+1; SIN[topo] := k;
                       s := k; lex (u,v)
        seja REDUCE p: A := lado esquerdo da produção p;
                       topo:=topo-tamanho_lado direito de p;
                       s := GOTO[SIN[topo],A];
                       topo := topo + 1; SIN[topo] := s
        seja ACCEPT  : analisa := falso;
        seja ERROR   : chama rotina de recuperação de erro
      fim
    fim
fim
```

Figura 2: Algoritmo básico do reconhecedor LR(1)

3. CODIFICAÇÃO DE TABELAS LR(1)

3.1. TÉCNICAS DE REPRESENTAÇÃO DE MATRIZES ESPARSAS

O problema a ser resolvido é a formulação de um método que possibilite representar uma matriz esparsa A(i,j) de forma a usar um mínimo de memória sem comprometer o tempo de acesso.

Uma possibilidade seria a utilização de uma tabela "hash", onde a chave de acesso seria o par (i,j). Outro candidato é uma representação em anel conforme descrito por Knuth [Knuth 73].

Estas técnicas são certamente exequíveis. No entanto, o uso de propriedades intrínsecas às tabelas LR(1) pode conduzir a melhores resultados. Por exemplo, o esquema de codificação proposto por Aho e Ullman [Aho 77] adota esta abordagem.

3.2. MÉTODO DE AHO & ULLMAN

Aho e Ullman [Aho 77] sugerem um método que possibilita uma redução no consumo de memória superior a 90% em relação à representação direta da matriz LR(1). O método proposto fundamenta-se na observação de que a grande maioria das entradas de uma tabela LR(1) representa a ação ERROR e no fato de os reconhecedores LR(1) possuírem a propriedade do prefixo correto |Aho 72|.

Um método de análise sintática é dito ter essa propriedade se todo erro de sintaxe for detectado assim que a sequência de terminais e não-terminais que estão implícita ou explicitamente na pilha do analisador e o próximo símbolo da entrada não constituírem o prefixo de alguma forma sentencial da linguagem em análises.

A rigor, reconhecedores SLR(1) e LALR(1) não possuem esta propriedade porque após o LR(1) haver indicado um erro de sintaxe em uma dada entrada, estes métodos ainda podem efetuar algumas reduções no conteúdo da pilha. Entretanto, pode-se mostrar que, tanto no SLR(1) como no LALR(1), o símbolo da entrada que causa o LR(1) reportar o erro nunca será empilhado. Em outras palavras, a condição de erro na entrada de um analisador LR(1) não é removida se reduções no conteúdo da pilha ocorrerem. O primeiro estado de leitura no qual o analisador entrar após estas reduções será sempre capaz de detectar a condição de erro.

Assim, ao custo de dificultar um pouco o algoritmo de recuperação de erro pode-se eliminar totalmente a previsão de condições de erro em estados do analisador que contêm ações de redução. A ação de redução mais frequente numa dada linha pode ser escolhida sempre que as demais não forem aplicáveis para um dado símbolo da entrada. Além disso, nos estados em que somente existem ações do tipo SHIFT k, para algum estado k, a ação ERROR pode ser codificada apenas uma vez, sendo selecionada somente quando nenhuma outra for aplicável.

Neste esquema de codificação, a sub-tabela ACTION é representada através de um conjunto de listas, uma para cada linha da tabela. Os elementos dessas listas são pares da forma (COLUMN, ACT), exceto o último par de cada lista que tem a forma (ANY, ACT). O campo COLUMN representa um dado terminal, ANY designa todos os terminais que não aparecem nos demais pares da mesma lista e ACT descreve a ação do reconhecedor associada ao par. O elemento (ANY, ACT) deve ser utilizado em substituição a todas as ocorrências da ação mais frequente de cada linha. Em geral nos estados só de leitura, o último par é (ANY, ERROR) e naqueles que contêm ações de redução o último par tem a forma (ANY, REDUCE p), onde REDUCE p denota a ação mais frequente na linha considerada.

Para a codificação da subtabela GOTO é mais conveniente associar a cada não-terminal da gramática listas de pares da forma (ROW, NEXT), onde ROW é o número de um estado (topo da pilha) e NEXT o número do próximo estado.

Similar às listas de ACTION, o último par em cada lista pode ter a forma (ANY, NEXT) em substituição ao maior conjunto de pares contendo um mesmo valor no campo NEXT. Note que as entradas ERROR da tabela GOTO nunca são consultadas pelo analisador e portanto podem ser totalmente ignoradas [Aho 77].

A economia de memória que se obtém neste esquema de codificação advém principalmente do fato de que, na prática, cerca de 90% das entradas na tabela são ações ERROR e que ações de redução numa mesma linha da matriz LR(1), referem-se com muita frequência a uma mesma produção. Por conseguinte, grande parte das linhas que contém ações de redução se resumem a apenas um par da forma (ANY, REDUCE p). Além do mais, pode-se ter um ganho extra de espaço eliminando-se listas repetidas.

Apesar de o espaço adicional gasto no armazenamento de apontadores que associam listas a estados e a não-terminais e dos índices de linhas e colunas presentes nos elementos das listas, a redução observada no consumo de memória tem sido superior a 90% do espaço gasto pela matriz original [Aho 77].

A principal desvantagem deste esquema em relação ao método direto diz respeito ao tempo de acesso a tabela codificada. Cada transição do analisador envolve um endereçamento indireto para se obter o endereço da lista associada ao estado (ou ao não-terminal) seguido de uma pesquisa sequencial dentro da lista. Entretanto, na prática as listas são pequenas. Além disto, o acréscimo no tempo de acesso à tabela representa apenas uma pequena fração do tempo total de compilação, sendo portanto justificável em vista da economia de espaço obtido.

4. O MÉTODO PROPOSTO

O método aqui proposto para compactação de tabelas LR(1) basea-se no esquema Aho e Ullman e tem como meta reduzir ainda mais o consumo de memória sem degradar o tempo de acesso às tabelas.

O primeiro passo neste sentido consiste em eliminar os apontadores que associam estados a listas de pares. No método

direto, estados são usados como índices de linha da matriz LR(1). Consequentemente, é desejável que os números dos estados sejam inteiros consecutivos a partir de um dado valor base, usualmente zero. Com o uso de listas, esta exigência pode ser relaxada, importando unicamente que a partir do número de um estado qualquer, o endereço da lista que lhe está associada possa ser determinado. Ora, por construção, a cada estado corresponde uma única lista e se for assegurado que a cada lista corresponde apenas um estado, endereços de listas podem ser usados, para identificar estados univocamente. Esta garantia pode ser facilmente obtida se listas repetidas não forem eliminadas da representação compactadas da matriz.

Além do fato de a eliminação dos apontadores acima representar uma economia substancial em espaço, existe um ganho real em tempo de acesso às listas, já que um nível de endereçamento indireto às listas foi abolido.

Contudo, a redução mais expressiva no consumo de memória advém de uma propriedade inerente aos reconhecedores LR(1). Em um reconhecedor LR(1) todas as transições para um dado estado têm o mesmo rótulo. Esta propriedade, que pode ser facilmente inferida a partir da definição dos conjuntos GOTO [Aho 77], possibilita a remoção dos rótulos das transições, isto é, dos pares da forma (COLUMN, SHIFT k) para associá-los aos estados destinos. Ou seja, uma transição passa a ser identificada apenas pelo número do estado destino o qual fornece o rótulo. Considerando que, em geral, os estados de um analisador LR(1) são alcançados por mais de um caminho, o ganho em espaço é substancial.

A representação proposta para matrizes LR(1) consiste, basicamente, em um vetor, aqui denominado LR, onde as listas contendo ações do analisador são armazenadas linearmente. O primeiro elemento de cada lista indica o símbolo que rotula transições para o estado associado. Os demais elementos são números ou endereços dos estados sucessores.

Por razões de uniformidade de tratamento, as ações ERROR, ACCEPT e REDUCE p são tratadas como transições para estados especiais. A ação ACCEPT corresponde a uma transição sob a marca $, para o estado final F, cuja lista contém apenas um elemento, o $. Todas as transições ERROR têm como destino um mesmo estado E. A prática demonstra que, na maior parte dos casos, a ação ERROR é a mais frequente nas linhas da matriz LR(1). Assim, em consonância com as idéias de Aho e Ullman, as transições para E seriam, com grande frequência, codificadas apenas uma vez no final de cada lista. Com objetivo de manter o algoritmo do analisador simples, escolheu-se a opção de sempre codificar uma transição para E no final de toda lista que não contiver ações de redução. Desta forma transições para E, somente são selecionadas quando nenhuma outra for aplicável. A lista associada ao estado especial E possui apenas um elemento, o qual sempre deve conter o próximo símbolo u da entrada. Assim, transições para E funciona como "sentinelas" em uma pesquisa sequencial rápida [KNUTH 73].

Suponha que k seja o estado corrente do analisador, i.e., s=k e que o próximo símbolo da entrada seja u. O estado sucessor de k é determinado pelo seguinte trecho de programa:

```
i := s + 1; s := LR[i];
enquanto u <> LR[s] repita
   i := i + 1; s := LR[i]
fim
```

Se, na saída do comando de repetição, s = E então um erro de sintaxe na entrada foi detectado, do contrário s contém o estado sucessor.

As ações da forma REDUCE p são tratadas como transições para "estados de redução" onde estão codificados os rótulos das transições e os números das produções envolvidas. Cada produção da gramática corresponde a um ou mais estados de redução.

Conforme discutido anteriormente, as ações ERROR são completamente eliminadas das linhas que contém ações de redução. A ação de redução mais frequente deve ser escolhida caso nenhuma das outras seja aplicável.

O trecho de programa acima pressupõe a existência de um estado "sentinela" (último da lista) cujo endereço é fixo e conhecido. Por conseguinte, estados contendo ações de redução também devem ser codificados conforme o mesmo padrão. Para tal, criou-se um outro estado especial, denominado R, que funciona de maneira análoga ao E, exceto que, à saída do comando de repetição do algoritmo acima, se s = R então o estado sucessor é na verdade aquele que precede R na lista pesquisada, ou seja, o comando:

```
se s = R então s := LR[i-1] fim
```

deve ser executado em algum ponto após a saída do elo de pesquisa para garantir a correção do valor em s.

Além disto, LR[R] deve ser feito conter sempre o próximo símbolo de entrada. O elemento antecessor a R em cada lista, representa sempre a ação de redução mais frequente, codificada apenas uma vez. O seu rótulo, que não é relevante, pode ser escolhido arbitrariamente dentre os vários possíveis. Esta representação nada mais é do que uma lista de pares (COLUMN, REDUCE p) que no esquema de Aho e Ullman estariam associados a cada estado k. A diferença é que, no presente método, na lista de k, têm-se apenas os endereços dos pares, os quais são armazenados em outras posições de LR. Note que cada par (COLUMN, REDUCE p) só precisa ser armazenado uma vez no presente método.

O tipo de ação, i.e., SHIFT k, REDUCE p, ERROR ou ACCEPT, associada a cada transição é determinado pelo analisador implicitamente através do endereço do estado destino da transição escolhida. São cinco os tipos de estado nesta codificação: os estados normais, ou seja, aqueles que correspondem a linhas da matriz LR(1) original, o estado de erro E, o final F, os de redução e o estado especial R. Estes estados são organizados no vetor LR da maneira indicada na figura 3.

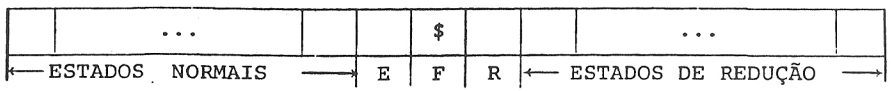

Figura 3: Vetor LR

Nesta organização, os estados k cujos endereços são menores que E representam ações da forma SHIFT k; aqueles de endereços maior que R designam ações REDUCE p onde p = LR[k + 1]. O estado final e o de erro têm endereços fixos, F e E respectivamente.

Para completar o quadro, existe ainda a tabela PROD que fornece o código do não-terminal do lado esquerdo (LE) e o comprimento do lado direito (TAMANHO) de cada produção. Note que a codificação acima inclui também a parte GOTO da tabela LR(1). Isto foi obtido, permitindo-se que não-terminais fossem rótulos de transição. Em outras palavras, neste método, a parte ACTION da tabela LR(1) é estendida de modo a incorporar a parte GOTO, onde os números de estados k são substituídos por ações da forma SHIFT k. Na verdade, a tabela GOTO havia sido criada apenas por razões de eficiência. Originalmente, os autômatos a partir dos quais tabelas LR(1) são construídas, possuem transições tanto sob terminais como não-terminais.

Esta mudança na forma da matriz LR(1) implica numa pequena alteração no algoritmo no analisador com relação à semântica da ação REDUCE p. Originalmente, uma ação da forma REDUCE p, onde p é uma produção A → ω, causa a remoção de m (comprimento de ω) elementos da pilha de estados seguida de uma consulta a tabela GOTO, dado o estado do topo corrente da pilha e o não-terminal A. A tabela GOTO fornece o próximo estado a ser empilhado.

Este mecanismo equivale a desempilhar m elementos e a partir do estado que aparecer no topo, executar uma transição sob o símbolo A, conforme está detalhado no seguinte trecho de programa:

```
A := PROD[p].LE;
topo := topo - PROD[p].TAMANHO;
i := SIN[topo] + 1; s := LR[i];
enquanto A <> LR[s] repita
  i := i + 1; s := LR[i]
fim;
topo := topo + 1; SIN[topo] := s
```

Note que, devido ao fato de as entradas ERROR da tabela GOTO nunca serem consultadas, tem-se a garantia que uma transição sob o símbolo em A existe para o estado que aparece no topo após o desempilhamento. Por conseguinte, a pesquisa sempre termina com sucesso.

Para finalizar, deve-se ressaltar que em consequência da fusão das tabelas ACTION e GOTO, os códigos de terminais e não-terminais devem ser disjuntos.

O algoritmo completo do analisador sintático LR(1) modificado é apresentado na figura 4. A figura 5 ilustra a codificação da tabela da figura 1.


```
var analisa: lógico;
    u: 0..imax;
    v: valor;
    LR: arranjo [0..imax] de 0..imax;
    topo: 0.. max;
    SIN: arranjo [0..max] de 0..imax;
    s,i: 0..imax;
    A: 0..imax;
início
    topo := 0; s := estado-inicial; sin[topo] := s;
    analisa := verdadeiro;
    ..... preenche tabela LR
    lex (u,v); LR[E] := u; LR[R] := u;
    enquanto analisa repita
      i := s + 1; s := LR[i];
      enquanto u <> LR[s] repita
        i := i + 1; s := LR[i]
      fim;
      se s >= R então -- REDUCE p --
          se s = R então s = LR[i - 1] fim
          p := LR[s + 1]; A := PROD[p].LE;
          topo := topo - PROD[p].TAMANHO;
          i := SIN[topo] + 1; s := LR[i];
          enquanto A <> LR[s] repita
            i := i + 1; s := LR[i]
          fim;
          topo := topo + 1; SIN[topo] := s

      ou s < E então -- SHIFT s
          topo := topo + 1; sin[topo] := s
          lex(u,v); LR[E] := u; LR[R] := u;
      ou s = E então recupera do erro; -- ERROR
      senão analisa := falso           -- ACCEPT
      fim
    fim
fim
```

Figura 4: Analisador LR(1) modificado

5. AVALIAÇÃO DO MÉTODO

A eficácia do método de compactação de tabelas LR(1) apresentado depende da existência das seguintes condições para cada tabela:

1. A relação número de transições/número de estados é razoavelmente maior que 1. Quanto maior esta relação mais eficaz é o método dado que representa-se o rótulo de cada transição apenas uma vez.
2. A relação número de linhas repetidas/número total de linhas é menor que (a/m.c), onde a é o espaço necessário para se armazenar um apontador, m é o número médio de entradas por linha e c espaço necessário para uma entrada no método de Aho e Ullman. Em linguagens do ponto do PASCAL tem-se os valores tí

picos a = 2, m = 10 e c = 3. Assim, para estas linguagens, o espaço necessário para os apontadores de listas do método de Aho e Ullman só é compensado pela eliminação de listas repetidas se a relação acima for maior que 1/15.

3. ERROR é sempre a entrada mais frequente em cada linha.

Nos testes realizados, (1) e (3) foram satisfeitas permitindo-se representar matrizes em apenas 4% do espaço original.

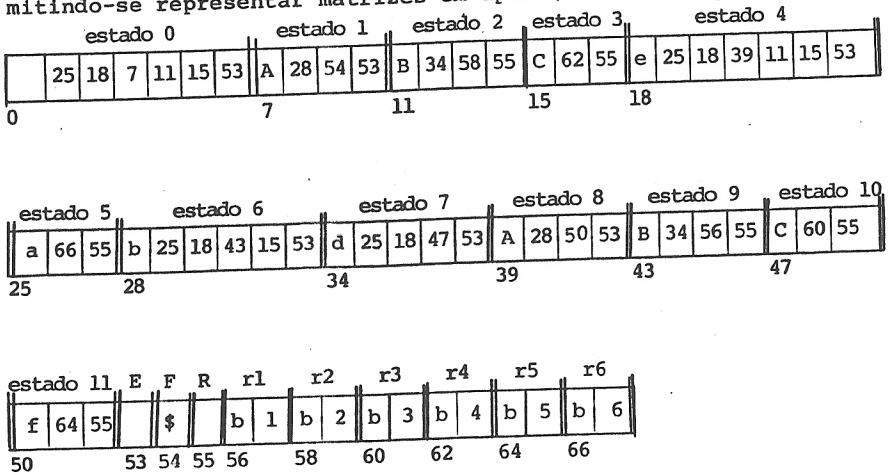

Figura 5: Codificação no vetor LR.

6. CONCLUSÃO

O método de compactação de tabelas apresentado baseia-se em duas propriedades dos reconhecedores LR(1). A primeira é a propriedade do prefixo correto e a segunda é o fato de todas as transições para um dado estado terem o mesmo rótulo.

A eficácia do método depende de algumas características das tabelas LR(1). O fator que mais pesa quando comparado com o difundido método de Aho e Ullman é a questão de eliminação de linhas repetidas. O método proposto não permite que linhas repetidas seja eliminadas. Sua grande fonte de economia de espaço está na codificação dos rótulos de transições juntos aos estados destinos.

7. REFERÊNCIAS

1. Aho,A.V. and Ullman,J.D.,The Theory of Parsing Translation and Compiling, Prentice-Hall,INC,1972.

2. Aho,A.V., and Ullman,J.D., Principles of Compiler Design, Addison-Wesley Publishing Company Co. (1973).

3. Knuth,D., The Art of Computer Programming, 2nd Edition, Addison-Wesley Publishing Co., (1973).

4. Spector,David, "Full LR(1) Parser Generation",ACM Sigplan Notices, Vol 16, Nº 8, August 1981.

5. Pager,David, "A Practical General Method for Constructing LR(k) Parsers", Acta Informatica 7, 249-268 (1977).