Gerador de Analisador Hyacc LR (1) X. Chen 1, D. Pager 1

1 Departamento de Informação e Ciência da Computação, Universidade do Havaí em Manoa, Honolulu, HI, EUA

Resumo -

O custo de espaço e tempo da geração do analisador LR é alto. Geradores de analisador LR (1) robustos e eficazes são raro de encontrar. Este trabalho empregou o canônico de Knutz algoritmo, método geral prático de Pager, rastreamento de faixa algoritmo e outros algoritmos relevantes, implementaram um gerador de analisador eficiente, prático e de código aberto Hyacc em ANSI C, que suporta LR completo (0) / LALR (1) / LR (1) e LR parcial (k) e é compatível com Yacc e Bison em formato de entrada e interface com o usuário da linha de comandos. nesse papel apresentamos o Hyacc e fornecemos uma breve visão geral arquitetura, mecanismo de análise, tabela de armazenamento, precedência e manipulação de associatividade, manipulação de erros, estruturas de dados, desempenho e uso. Palavras-chave: Hyacc, LR (1), Gerador de analisador, Compilador, Ferramenta de software

1 - Introdução

O algoritmo canônico LR (k) [1] proposto por Knuth em 1965 é um poderoso algoritmo de geração de analisador para gramáticas livres de contexto. Foi potencialmente exponencial no tempo e espaço para ser de uso prático. Alternativas ao LR (k) incluem o algoritmo LALR (1) usado em geradores de analisador como Yacc e mais tarde Bison, e o algoritmo LL usado por geradores de analisadores, como ANTLR. No entanto, LALR e LL não são tão poderosos quanto LR. Bom gerador de analisador LR (k) permanece escasso, mesmo no caso k = 1. Este trabalho desenvolveu o Hyacc, um eficiente e código aberto prático LR completo (0) / LALR (1) / LR (1) e parcial Ferramenta de geração de analisador LR (k) em ANSI C. É compatível com Yacc e Bison. Os algoritmos LR (1) empregados são baseado em 1) o algoritmo canônico de Knuth [1], 2) o algoritmo de rastreamento de faixa de Pager [2] [3], o que reduz analisando o tamanho da máquina dividindo a partir de uma análise LALR (1) máquina que contém conflitos de redução a redução e 3) a método geral prático de Pager [4], que reduz a análise tamanho da máquina mesclando estados compatíveis de uma análise máquina obtida pelo método de Knuth. O algoritmo LR (0) usado no Hyacc é o tradicional. O algoritmo LALR (1) usado no Hyacc é baseado na primeira fase do rastreamento de faixa algoritmo. LR (0) e LALR (1) são implementados porque O algoritmo de rastreamento de pista de Pager depende desses como o primeiro degrau. O algoritmo LR (k) é chamado de empurrar a borda algoritmo [5] baseado na aplicação recursiva do rastreamento de faixa processo e trabalha para uma subclasse de gramáticas LR (k). Como um otimização lateral, a Hyacc também implementou a unidade algoritmo de eliminação da produção de Pager [6] e seus extensão [5].

2

2.1 - O gerador Hyacc Parser visão global

Hyacc é pronunciado como "HiYacc". É um eficiente e gerador de analisador prático escrito do zero em ANSI C, e é fácil de portar para outras plataformas. O Hyacc é de código aberto.
A versão 0.9 foi lançada em janeiro de 2008 [7]. Versão 0.95 foi lançado em abril de 2009. A versão 0.97 foi lançada em Janeiro de 2011.

Hyacc é liberado sob a licença GPL. Mas o Arquivo do mecanismo de análise LR (1) hyaccpar e arquivo do mecanismo de análise LR (k) O hyaccpark está sob a licença BSD para que o analisador geradores criados pela Hyacc podem ser usados tanto em código aberto e software proprietário. Isso aborda os direitos autorais problema que Richard Stallman discutiu em “Condições para Usando o bisonte ”de seus manuais do bisonte [8] [9]. Os algoritmos empregados pelo Hyacc estão listados na introdução.

Hyacc é compatível com Yacc e Bison em seu arquivo de entrada formato, manipulação gramatical ambígua e manipulação de erros. Essas diretrizes da Yacc e Bison são implementadas em Hyacc:% token,% à esquerda,% à direita,% de expectativa,% de início,% de prec. Hyacc pode ser usado junto com o analisador lexical Lex. isto pode gerar informações avançadas de depuração na geração do analisador processo e armazene-os em um arquivo de log para revisão.

Se especificado, o analisador gerado pode gravar a análise etapas em um arquivo, o que facilita a depuração e teste. Também pode gerar um arquivo de entrada Graphviz para o máquina de análise. Com esta entrada, o Graphviz pode desenhar um imagem da máquina de análise.

2.2 - Arquitetura

O Hyacc obtém primeiro as opções da opção de linha de comando, depois lê do arquivo de entrada gramatical. Em seguida, ele cria o máquina de análise de acordo com diferentes algoritmos como especificado pelas opções da linha de comandos. Então escreve o analisador gerado para y.tab.c e, opcionalmente, y.output e y.gviz. y.tab.c é o analisador com a máquina de análise armazenada em matrizes. y.output contém todos os tipos de informações necessárias pelo desenvolvedor do compilador para entender o analisador processo de geração e a máquina de análise. y.gviz pode ser usado como arquivo de entrada para o Graphviz para gerar um gráfico do máquina de análise.

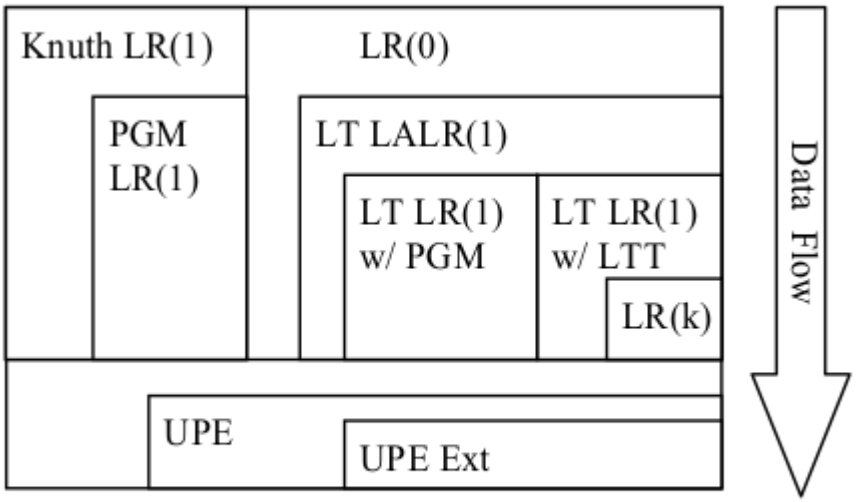

Fig. 1. Relationship of algorithms from the point of view of grammar processing [1]

A Fig. 1 mostra como os algoritmos usados no Hyacc são estruturado do ponto de vista do processamento gramatical. As gramáticas de entrada podem ser processadas pelo caminho de mesclagem no esquerda, primeiro pelo algoritmo canônico de Knuth e parar por aqui, ou ser processado posteriormente pelo algoritmo PGM de Pager.

As gramáticas de entrada também podem ser processadas pela divisão caminho à direita. Primeiro, a máquina de análise LR (0) é gerado. Em seguida, a máquina de análise LALR (1) é gerada

pela primeira fase do algoritmo de rastreamento de faixa. Se reduzir reduzir conflitos, isso não é uma gramática LALR (1) e a segunda fase do rastreamento de faixa é aplicada para gerar o Máquina de análise LR (1). Existem dois métodos para a segunda fase de rastreamento de faixa. Um é baseado no PGM método [4], o outro é baseado no método da tabela de faixas [10]. Se LR (1) não puder resolver todos os conflitos, pode ser um LR (k) gramática e o processo LR (k) é aplicado.

A máquina de análise LR (1) gerada pode conter unidade produções que podem ser eliminadas com a aplicação do UPE algoritmo e sua extensão. A figura 2 mostra a relação dos algoritmos do ponto de vista da implementação, ou seja, como um algoritmo é com base no outro.

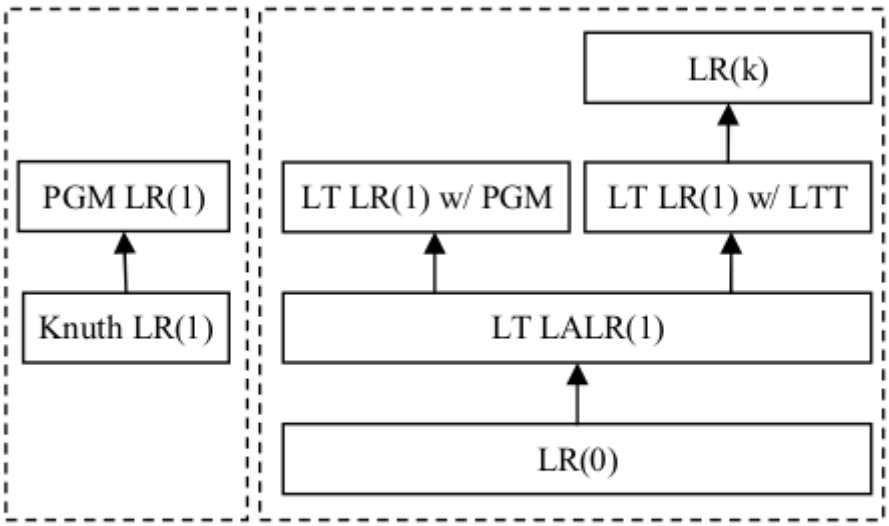

Fig. 2. Relationship of algorithms from the point of view of implementation

2.3

O mecanismo de análise LR (1) Semelhante ao arquivo yaccpar do Yacc, o arquivo hyaccpar é o mecanismo de análise do Hyacc. O processo de geração do analisador incorpora a tabela de análise no hyaccpar. Como o hyaccpar O mecanismo de análise LR (1) funciona é mostrado no Algoritmo 1.

Algorithm 1: The hyaccpar LR(1) parse engine algorithm.[2]

```
1  Initialization:
2  push state 0 onto state_stack;

3  while next token is not EOF do {
4    S ← current state;
5    L ← next token/lookahead;
6    A ← action of(S, L) in parsing table;
7    if A is shift then {
8      push target state on state_stack,
9        pop lookahead symbol;
10     update S and L;
11   } else if A is reduce then {
12     output code for this reduction;
13     r1 ← LHS symbol of reduction A;
14     r2 ← RHS symbol count of A;
15     pop r2 states from state_stack,
16       update current state S;
17     A_tmp ← action for (S, r1);
18     push target goto state A_tmp to
         state_stack;
19   } else if A is accept then {
20     if next token is EOF then {
21       is valid accept, exit;
22     } else {
23       is error, error recovery or exit;
24     }
25   } else {
26     is error, do error recovery;
27   }
28 }
```

No Algoritmo 1, uma pilha de estados é usada para acompanhar o status atual de atravessar a máquina de estado. O parâmetro 'S' ou estado atual é o estado na parte superior da pilha de estado. O parâmetro 'L' ou lookahead é o símbolo usado para decidir a próxima ação do estado atual. o parâmetro "A" ou ação é a ação a ser tomada e é encontrada por verificando a entrada da tabela de análise (S, L). "Denota operação de atribuição. Esse mecanismo de análise é semelhante ao usado em Yacc, mas há variação nos detalhes, como como a tabela de análise de armazenamento, conforme discutido na próxima seção.

TABLE I. STORAGE ARRAYS FOR THE PARSING MACHINE IN HYACC PARSE ENGINE.

Array name	Explanation
yyfs[]	List the default reduction for each state. If a state has no default reduction, its entry is 0. Array size = *n*.
yyrowoffset[]	The offset of parsing table rows in arrays yyptblact[] and yyptbltok[]. Array size = *n*.
yyptblact[]	Destination state of an action (shift/goto/reduce/accept). If yyptblact[i] is positive, action is 'shift/goto', If yyptblact[i] is negative, action is 'reduce', If yyptblact[i] is 0, action is 'accept'. If yyptblact[i] is -10000000, labels array end. Array size = *p*.
yyptbltok[]	The token for an action. If yyptbltok[i] is positive, token is terminal, If yyptbltok[i] is negative, token is non-terminal. If yyptbltok[i] is -10000001, is place holder for a row. If yyptbltok[i] is -10000000, labels array end. Array size = *p*.
yyr1[]	If the LHS symbol of rule i is a non-terminal, and its index among non-terminals (in the order of appearance in the grammar rules) is *x*, then yyr1[i] = -*x*. If the LHS symbol of rule i is a terminal and its token value is *t*, then yyr1[i] = *t*. Note yyr1[0] is a placeholder and not used. Note this is different from yyr1[] of Yacc or Bison, which only have non-terminals on the LHS of its rules, so the LHS symbol is always a non-terminal, and yyr1[i] = x, where x is defined the same as above. Array size = *r*.
yyr2[]	Same as Yacc yyr2[]. Let x[i] be the number of RHS symbols of rule i, then yyr2[i] = x[i] << 1 + y[i], where y[i] = 1 if production i has associated semantic code, y[i] = 0 otherwise. Note yyr2[0] is a placeholder and not used. This array is used to generate semantic actions. Array size = *r*.
yynts[]	List of non-terminals. This is used only in debug mode. Array size = number of non-terminals + 1.
yytoks[]	List of tokens (terminals). This is used only in debug mode. Array size = number of terminals + 1.
yyreds[]	List of the reductions. Note this does not include the augmented rule. This is used only in debug mode. Array size = *r*.

2.4 - Armazenando a tabela de análise

2.4.1 - Mesas de armazenamento

A seguir, são descritas as matrizes usadas no hyaccpar para armazenar a tabela de análise. Deixe a tabela de análise ter n linhas (estados) em colunas (número de terminais e não terminais). Assumindo existem r regras (incluindo a regra aumentada) e a regra o número de entradas não vazias na tabela de análise é p. Mesa 1 lista todas as matrizes de armazenamento e explica seu uso.

2.4.2 – Complexidade

Suponha que no estado i haja um token j, podemos descobrir se um ação existe olhando a tabela yyptbltok de yyptbltok [yyrowoffset [i]] para yyptbltok [yyrowoffset [i + 1] -1]:

i) Se yyptbltok [k] == j, yyptblact [k] é a ação associada;
ii) Se yyptblact [k]> 0, esta é uma ação 'shift / goto';
iii) Se yyptblact [k] <0, for uma redução, use yyr1 e yyr2
para encontrar o número de estados para pop e o próximo estado para ir;
iv) Se yyptblact [k] == 0, é uma ação de 'aceitação', que é
válido quando j é o fim de uma sequência de entrada.

O espaço usado pelo armazenamento é: 2n + 2p + 3r + (m +2) Na maioria dos casos, a tabela de análise é uma matriz esparsa. Em geral, 2n + 2p + 3r + (m + 2) <n * m. Durante o tempo utilizado, o principal fator é ao pesquisar através da matriz yyptbltok da yyptbltok [yyrowoffset [i]] para yyptbltok [yyrowoffset [i + 1] -1]. Agora é pesquisa linear e leva O (n) tempo. Isso pode ser feito mais rapidamente com a pesquisa binária, isso é possível se os terminais e os não terminais forem classificados alfabeticamente. Então a complexidade do tempo será O (ln (n)). Pode ser feita de modo que a complexidade do tempo seja O (1), usando o método de deslocamento duplo que armazena toda a linha de Cada estado. Isso exigiria mais espaço embora.

2.4.3 – Exemplo

Um exemplo é dado para demonstrar o uso desses matrizes para representar a tabela de análise. Dada a gramática G1:

E → E + T | T
T → T * a | a

A tabela de análise é mostrada na Tabela 2. Aqui, a análise A tabela possui n = 8 linhas, m = 6 colunas e r = 5 regras (incluindo a regra aumentada). As matrizes de armazenamento reais em o mecanismo de análise hyaccpar é mostrado na Tabela 3. A matriz yyfs [] lista a redução padrão para cada estado: o estado 3 tem redução padrão na regra 4 e o estado 7 tem redução padrão na regra 3.

A matriz yyrowoffset [] define o deslocamento da tabela de análise linhas nas matrizes yyptblact [] e yyptbltok []. Por exemplo, a linha 1 inicia no deslocamento 0, a linha 2 inicia no deslocamento 3. Matriz yyptblact [] é o estado de destino de uma ação. A primeira entrada é 97, que pode ser vista na matriz yytoks []. A segunda entrada é 1, que significa E. não terminal. como vemos na tabela de análise, a entrada (0, a) tem a ação s3, A entrada (0, E) possui a ação g1, portanto, em yyptblact [] vemos correspondentemente, a primeira entrada é 3 e a segunda entrada é 1. Entrada -10000000 no yyptblact [] e no yyptbltok [] rotula o final da matriz. A entrada 0 no yyptblact [] rotula o aceite ação. A entrada 0 no yyptbltok [] representa o final do token marcador $. A entrada -10000001 nos rótulos yyptbltok [] que este O estado (linha na tabela de análise) não tem outras ações, mas o redução padrão. -10000001 é apenas um valor fictício que é nunca usado e servidores como espaço reservado, então yyrowoffset [] pode ter um valor correspondente para esta linha.

As entradas da matriz yyr1 [] e da matriz yyr2 [] são definidas como na Tabela 1, e é fácil ver a correspondência do valores.

2.5 - Lidando com precedência e associatividade

A maneira como o Hyacc lida com precedência e associatividade é a mesma que Yacc e Bison. Por padrão, em um mudar / reduzir conflitos, mudança é escolhida; em uma redução / redução conflito, a redução cuja regra aparece primeiro no gramática é escolhida. Mas isso pode não ser o que o usuário deseja. Portanto,% left,% right e% nonassoc são usados para declarar tokens e especifique precedência e associatividade personalizadas.

2.6 - Tratamento de erros

O tratamento de erros é o mesmo que no Yacc. Houveram queixas abundantes sobre o esquema de recuperação de erros de Yacc. Estamos nos concentrando nos algoritmos LR (1) em vez de melhor recuperação de erros. Também queremos manter a compatibilidade com Yacc e Bison. Por essas razões, mantemos o caminho que Yacc lida com erros.

2.7 - Estruturas de dados

Uma tabela de símbolos é implementada pela tabela de hash e usa encadeamento aberto para armazenar elementos em uma lista vinculada a cada balde. A tabela de símbolos é usada para obter O (1) desempenho para muitas operações. Todos os símbolos usados em a gramática é armazenada como um nó nesta tabela de símbolos. Cada O nó também contém outras informações sobre cada símbolo. Essas informações são calculadas no momento da análise do arquivo de gramática e armazenado para uso posterior.

TABLE 3. STORAGE TABLES IN HYACC LR(1) PARSE ENGINE FOR GRAMMAR G1

Stat
0
1
2
3
4
5
6
7

```
#define YYCONST const
typedef int yytabelem;

static YYCONST yytabelem yyfs[] = {0, 0,
0, -4, 0, 0, 0, -3};

static YYCONST yytabelem yyptbltok[] = {
97, -1, -2, 0, 43, 0, 43, 42, -10000001, 97,
-2, 97, 0, 43, 42, -10000001, -10000000};

static YYCONST yytabelem yyptblact[] = {
3, 1, 2, 0, 4, -2, -2, 5, -4, 3,
6, 7, -1, -1, 5, -3, -10000000};

static YYCONST yytabelem yyrowoffset[] = {
0, 3, 5, 8, 9, 11, 12, 15, 16};

static YYCONST yytabelem yyr1[] = {
     0,    -1,    -1,    -2,    -2};
static YYCONST yytabelem yyr2[] = {
     0,     6,     2,     6,     2};

#ifdef YYDEBUG

typedef struct {char *t_name; int t_val;}
yytoktype;

yytoktype yynts[] = {
  "E", -1,
  "T", -2,
  "-unknown-", 1  /* ends search */
};
yytoktype yytoks[] = {
  "a", 97,
  "+", 43,
  "*", 42,
  "-unknown-", -1  /* ends search */
};
char * yyreds[] = {
  "-no such reduction-"
  "E : 'E' '+' 'T'",
  "E : 'T'",
  "T : 'T' '*' 'a'",
  "T : 'a'",
};
#endif /* YYDEBUG */
```

Lista vinculada, matrizes estáticas, matrizes dinâmicas e hash tabelas são usadas onde apropriado. Às vezes, vários dados estruturas são usadas para o mesmo objeto e qual usar depende da circunstância particular. Na tabela de análise, as linhas indexam os estados (por exemplo, linha 1 representa ações do estado 1), e as colunas representam os símbolos lookahead (terminais e não terminais) em que turno / ir / reduzir / aceitar ações ocorrem. O tabela de análise é implementada como um número inteiro unidimensional array. Cada entrada [linha, col] é acessada como entrada [linha * tamanho da coluna + col]. Na tabela de análise, números positivos são para 'shift', números negativos são para 'reduzir', -10000000 é para "aceitar" e 0 é para "erro". Não há limite de tamanho para nenhuma estrutura de dados. Eles podem crescer até consumir toda a memória. No entanto, Hyacc define artificialmente um limite superior de 512 caracteres para o comprimento máximo de um símbolo.

2.8 - Atuação

O desempenho do Hyacc é comparado com outros LR (1) geradores de analisador. Menhir [11] e MSTA [12] são ambos implementado de maneira muito eficiente e robusta. Tabela 4 e A Tabela 5 mostra a comparação do tempo de execução dos três geradores de analisador de gramática C ++ e C. As velocidades são semelhante. O MSTA, implementado em C ++, é uma opção útil para usuários do setor. Não usa LR de espaço reduzido (1) algoritmos, porém, sempre resulta em maior análise máquinas Menhir usa o algoritmo PGM de Pager, mas é implementado em Caml, que não é tão popular na indústria. Portanto, o Hyacc deve ser uma escolha favorável.

TABLE 4. RUNNING TIME (SEC) COMPARISON OF MENHIR, MSTA AND HYACC ON C++ GRAMMAR.

	Knuth LR(1)	PG MLR(1)	LALR(1)
Menhir	1.97	1.48	N/A
MSTA	5.32	N/A	1.17
Hyacc	3.53	1.78	1.10

TABLE 5. RUNNING TIME (SEC) COMPARISON OF MENHIR, MSTA AND HYACC ON C GRAMMAR.

	Knuth LR(1)	PG MLR(1)	LALR(1)
Menhir	1.64	0.56	N/A
MSTA	0.92	N/A	0.13
Hyacc	1.05	0.42	0.19

2.9 - Uso

Na página inicial do Hyforc [7], sourceforge.net O usuário pode baixar os pacotes de origem para unix / linux e windows e o pacote binário para windows. Todos instruções de instalação e uso estão disponíveis no arquivo leia-me incluído. É muito fácil de usar, especialmente para usuários familiarizados com Yacc e / ou Bison. Hyacc é um utilitário de linha de comando. Para iniciar o hyacc, use: "Hyacc input_file.y [-bcCdDghKlmnoOPQRStvV]". O arquivo de gramática de entrada input_file.y tem o mesmo formato dos usado por Yacc / Bison. Os significados de algumas das opções de linha de comando são brevemente apresentados aqui. '-B' especifica o prefixo a ser usado para todos os nomes de arquivos de saída hyacc. O padrão é y.tab.c, como em Yacc. Se '-c' for especificado, nenhum arquivo analisador (y.tab.c e y.tab.h) será gerado. É usado quando o usuário deseja apenas use as opções -v e -D para analisar a gramática e verificar o arquivo de log y.output. '-D' é usado com um número de 0 a 15 (por exemplo, -D7) para especificar os detalhes a serem incluídos no arquivo de log y.output durante o processo de geração do analisador. "-G" diz que um arquivo de entrada Graphviz deve ser gerado. "-S" significa aplicar o algoritmo LR (0). "-R" aplica

LALR (1) algoritmo. '-Oi', onde i = 0 a 3 aplica o canônico de Knuth algoritmo eo método geral prático com diferentes otimizações. '-P' aplica o algoritmo de rastreamento de faixa com base no método geral prático. '-Q' aplica o rastreamento de faixa algoritmo baseado no método da tabela de faixas. '-K' aplica o Algoritmo LR (k). '-M' mostra a página de manual. Para obter mais uso do gerador de analisador Hyacc, usuários interessados podem consultar o manual de uso do Hyacc.

3 - Trabalho relatado

O método geral prático de Pager foi implementado em alguns outros geradores de analisador. Alguns exemplos são LR (1979, em Fortran 66, em Lawrence Laboratório Nacional de Livermore) [13], LRSYS (1985, em Pascal, no Laboratório Nacional Lawrence Livermore) [14], LALR (1988, no MACRO-11 sob RSX-11) [15], Menhir (2004, em Caml) [11] e o módulo Python Parsing (2007, em Python) [16]. O algoritmo de rastreamento de pista foi implementado por Pager (1970, em Assembly sob OS 360) [3]. Mas nenhum outro implementação disponível é conhecida. Para outros geradores de analisador LR (1), o analisador Muskox generator (1994) [17] implementou a LR de Spector (1) algoritmo [18] [19]. O MSTA (2002) [12] fez uma divisão abordagem, mas o detalhe é desconhecido. Produtos comerciais include Yacc ++ (LR (1) foi adicionado por volta de 1990, usando abordagem de divisão baseada livremente no algoritmo de Spector) [20] [21] e Dr. Parse (detalhe desconhecido) [22]. Mais recentemente Foi proposto um algoritmo IELR (1) [23] [24] para fornecer Solução LR (1) para gramáticas não LR (1) com especificações para resolver conflitos, e os autores o implementaram como um extensão de Bison.

4 - Conclusões

Neste trabalho, investigamos a geração de analisadores LR (1) algoritmos e implementou um gerador de analisador Hyacc, que suporta LR (0) / LALR (1) / LR (1) e LR parcial (k). isto foi lançado para a comunidade de código aberto. O uso Hyacc é altamente semelhante ao amplamente utilizado LALR (1) geradores de analisadores Yacc e Bison, o que facilita aprender e usar. Hyacc é único em sua ampla gama de algoritmos cobertura, eficiência, portabilidade, usabilidade e disponibilidade.