
Analísadores sintáticos: conflitos e ambigüidades¹

Silvio Luiz Bragatto Boss

Mestrando em Informática pela Universidade Federal do Paraná (UFPR),
bacharel em Análise de Sistemas pela Universidade Estadual do Centro-
Oeste (Unicentro).

E-mail: silvioboss@yahoo.com.br

Sandra Mara Guse Scós Venske

Mestre em Informática pela Universidade Federal do Paraná (UFPR),
docente do Departamento de Ciência da Computação da Universidade
Estadual do Centro-Oeste (Unicentro).

E-mail: ssvenske@unicentro.br

RESUMO

Analísador Sintático (parser) é um programa que recebe a descrição formal de uma gramática (de acordo com uma linguagem) e fornece como saída um código-fonte que reconhecerá cadeias (sentenças) válidas para a gramática (linguagem). Em alguns casos de reconhecimento de cadeias por gramáticas, ocorrem conflitos e problemas de ambigüidade. Essas gramáticas representam um problema para a construção de compiladores, que determinam o código a ser gerado para uma instrução, examinando sua árvore sintática. Assim, se a estrutura de uma linguagem tiver mais de uma árvore possível, o significado dessa árvore não poderá ser determinado de maneira única. Neste trabalho, avaliaram-se 10 analisadores sintáticos, com o objetivo de analisá-los focando na forma com que eles trabalham ou tratam conflitos e ocorrência de ambigüidade no processo de reconhecimento de cadeias. Desta forma, pode-se distinguir dentre os analisadores sintáticos analisados quais os métodos de tratamento mais utilizados e eficazes.

Palavras-chave: Análise sintática. Gerador de analisador sintático. Ambigüidade. Compiladores. Gramática.

ABSTRACT

Parser is a program that receives formal description of a grammar (in accordance with a language) and outputs a source code that will recognize valid strings (sentences) for a grammar (language). In some cases of strings recognition for grammars, can occurs conflicts and ambiguity problems. These grammars represent a problem for the compilers construction that determines the code to be generated for one instruction, examining its syntactic tree. Thus, if the language structure will have more than a possible tree, the meaning of this tree could not be determined in unique way. In this work, it is evaluated 10 parsers with the objective of analyse them; emphasizing the form they work or treat conflicts and ambiguity occurrence in the strings recognition process. This way, it can be distinguished among analysed parsers which treatment methods are most used and work efficiently.

Keywords: Syntactic analysis. Parser generator. Ambiguity. Compilers. Grammar.

¹ Esse trabalho foi parcialmente apoiado pelo CNPq/Brasil.

INTRODUÇÃO

Um analisador sintático (*parser*) é um programa que recebe a descrição formal de uma gramática (de acordo com uma linguagem) e fornece como saída um código-fonte que reconhecerá cadeias (sentenças) válidas para a gramática (linguagem). Alguns exemplos de geradores de analisadores sintáticos mais conhecidos são o *Yacc* e o *Bison*.

Em alguns casos de reconhecimento de cadeias por gramáticas, ocorrem conflitos e problemas de ambigüidade. Os conflitos podem aparecer como ambigüidade de ações quando da construção da tabela de análise sintática. Os problemas também podem ser encontrados diretamente na gramática, que pode apresentar ambigüidade inerente.

Neste trabalho, avaliou-se a documentação de 10 analisadores sintáticos com licença livre. O objetivo foi examiná-los focando a forma com que eles verificam ou tratam conflitos e ocorrência de ambigüidade no processo de reconhecimento de cadeias.

Optou-se pela análise de ferramentas livres por terem um uso maior na comunidade científica e por disponibilizarem o acesso à sua documentação e código fonte.

“Em alguns casos de reconhecimento de cadeias por gramáticas, ocorrem conflitos e problemas de ambigüidade.”

Este artigo está organizado como segue. A seção 2 apresenta conceitos inerentes ao trabalho. A seção 3 mostra os resultados encontrados com as ferramentas analisadas. A seção 4 sumariza os resultados e apresenta uma comparação entre eles. As conclusões são apresentadas na seção 5.

GRAMÁTICAS E AMBIGÜIDADE

Esta seção apresenta algumas definições sobre os temas utilizados neste trabalho. Inicialmente, são abordados os conceitos referentes a gramáticas livres de contexto e ambigüidade. A seguir, definições sobre a técnica de *backtracking* são abordadas.

Gramáticas livres de contexto

Uma gramática é composta por regras de produção ou regras de reescrita, por meio das quais é possível obter todos os elementos de uma linguagem a partir do símbolo inicial. Uma gramática (AHO; UILLMAN, 1986) é uma quádrupla, sendo definida como $G = (N, T, P, S)$, onde:

- N é um alfabeto de símbolos auxiliares chamados de não-terminais que denotam cadeias de caracteres. Neste artigo, símbolos não-terminais serão representados por letras maiúsculas;
- T é o alfabeto no qual a linguagem é definida, cujos elementos são os símbolos terminais a partir dos quais as cadeias são formadas. Neste artigo, símbolos terminais serão representados por letras minúsculas;
- P é o conjunto de regras ou de produções da forma $A \rightarrow w$, onde $A \in N$ e $w \in (N \cup T)^*$ que especificam de que maneira os símbolos, os não-terminais e terminais podem ser combinados para formar as cadeias;
- S é um símbolo não-terminal distinto, chamado símbolo inicial, a partir do qual toda derivação é iniciada, com $S \in N$.

Um gerador para um analisador sintático é uma ferramenta que lê uma especificação de gramática e converte-a em um programa (como exemplo, Java ou C) que possa reconhecer se um determinado texto pertence a uma gramática especificada.

Ambigüidade de gramática

Uma gramática é ambígua quando, a partir dela, uma cadeia pode dar origem a pelo menos duas árvores sintáticas distintas ou, equivalentemente, se possuir mais de uma derivação mais à esquerda ou mais à direita. Essas gramáticas representam um problema para a construção de compiladores, na qual determinam o código a ser gerado para uma instrução examinando sua árvore sintática. Assim, se a estrutura de uma linguagem tiver mais de uma árvore possível, o significado dessa árvore não poderá ser determinado de maneira única. Este problema pode ser representado em linguagens de programação como Pascal (BRAND, 2002). Considerando a entrada da sentença “*array[1..10] of integer*” a escala 1..10 pode ser representada

por duas maneiras diferentes:

- qualquer número real 1. seguido por outro número real .10;
- como inteiro 1 seguido por um operador .. seguido por um inteiro 10.

Para encontrar uma solução correta o analisador deve saber o processo de declaração de um *array* em Pascal.

Uma gramática para expressões pode ser dada por suas regras:

$$E \rightarrow E + E \mid E - E \mid E / E \mid E * E \mid id$$

$$id \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

A cadeia $9 - 5 + 1 * 2$ pode ter mais de uma derivação a partir do símbolo inicial *E* e suas árvores são mostradas na figura 1.

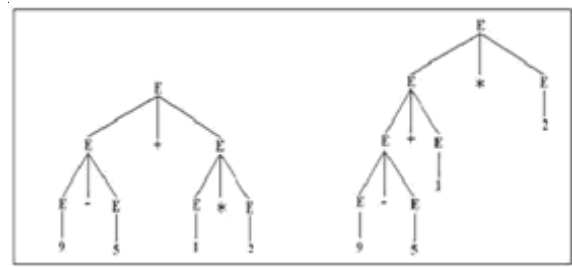


FIGURA 1 - Árvores de derivação para a cadeia $9 - 5 + 1 * 2$

De maneira geral, o mecanismo de inambigüação para linguagens livres de contexto é um procedimento que escolhe de uma gama de possíveis árvores de análise sintática para uma sentença a mais apropriada de acordo com alguns critérios.

Citam-se como algumas regras de inambigüação (KLINT; VISSER, 1994):

- regras de inambigüação: preferência para o *casamento* mais longe, preferência por palavras-chave em vez dos identificadores;
- regras de desambigüação para gramáticas livres de contexto: precedência entre relações de operadores, recursividade das regras, ambigüidade disjuntiva e conjuntiva;
- regra de desambigüação para semântica estática: tipos ou regras de dependência de declarações.

“Um gerador para um analisador sintático é uma ferramenta que lê uma especificação de gramática e converte-a em um programa que possa reconhecer se um determinado texto pertence à uma gramática especificada.”

Essa gramática possui dois problemas de ambigüidade: dupla recursividade de regras e precedência de operadores (AHO; UILLMAN, 1986).

Para eliminar a ambigüidade da gramática, elimina-se inicialmente a dupla recursividade de suas regras deixando-a com apenas uma recursividade por regra. Para isto utiliza-se o método conhecido como regra de eliminação de ambigüidade (LOUDEN, 2004), recurso empregado para corrigir a ambigüidade sem alterar a linguagem da gramática, acrescentando-se um novo não-terminal a esta. Podem-se criar várias combinações de regras, algumas recursivas à esquerda, outras à direita. O problema da precedência entre operadores de mesmo nível pode ser resolvido selecionando-se a recursividade à esquerda para operadores com precedência à esquerda e usando-se recursividade à direita caso o operador tenha precedência à direita.

A gramática não ambígua correspondente é mostrada a seguir.

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T / F \mid T * F \mid F$$

$$F \rightarrow id$$

Ambigüidade em analisadores sintáticos (conflitos)

Desambigüação também pode ser definida em termos da ação de um analisador sintático (KLINT; VISSER 1994), ou pode ser entendida independentemente do método de análise usado. Uma das distinções que deve ser feita quando se trata de ambigüidade está entre o conflito que existe em um analisador sintático (descendente, ascendente) e uma gramática ambígua.

A forma mais utilizada para a execução da análise sintática é por meio das tabelas de análise sintática. Essas tabelas possuem tanto símbolos terminais quanto não-terminais, que servirão de guia para o analisador sintático escolher qual a ação e por qual regra de produção seguir durante uma derivação. Os analisadores sintáticos ascendentes (*bottom-up*) são conhecidos como métodos de empilhar e reduzir. Duas ações são possíveis em um analisador sintático ascendente (AHO; UILLMAN, 1986):

- empilhar: pode ocorrer quando existe uma transição

com um terminal a partir do estado corrente (o estado da cadeia de entrada);

- reduzir: quando existe um item completo $B \rightarrow \beta^1$ no estado corrente.

Com isso podem surgir problemas, com relação à execução das ações. São eles:

1. Conflito empilhar/reduzir: o analisador sintático não consegue decidir se empilha o próximo símbolo da entrada, ou se reduz para uma regra já disponível.
2. Conflito reduzir/reduzir: o analisador sintático pode realizar uma redução para duas regras distintas.

“Uma das distinções que deve ser feita quando se trata de ambigüidade está entre o conflito que existe em um analisador sintático (descendente, ascendente) e uma gramática ambígua.”

Nesses analisadores a tabela de análise sintática é construída com base no autômato finito determinístico de itens. Com isso, uma gramática só poderá ser executada por um determinado tipo de analisador sintático se na construção da tabela de análise sintática, não ocorrerem conflitos (empilhar/reduzir ou reduzir/reduzir), ou seja, para cada linha da tabela só é permitida uma única ação.

Processo de *Backtracking*

O processo de *Backtracking* (MCNAUGHTON, 1993) ocorre quando, ao se analisar sintaticamente uma cadeia, tem-se mais de um caminho para completar a árvore de análise sintática. O procedimento a ser feito é ou iniciar o processo novamente ou voltar até algum estágio anterior na construção da árvore.

RESULTADOS

Nesta seção estão relacionados alguns analisadores sintáticos e a forma com que eles tratam ou respondem à ocorrência de ambigüidade, tanto com relação a gramáticas ambíguas quanto à ocorrência de conflitos.

Yacc

Yacc (*Yet Another Compiler Compiler*) (JOHNSON, 2006) é um gerador de analisador sintático escrito em C. Ele usa gramáticas LALR (1) com regras de desambigüação. O *Yacc* detecta os conflitos de ambigüidade (empilhar/reduzir ou reduzir/reduzir) e mesmo assim gera um analisador sintático. A escolha que o *Yacc* faz em uma dada situação é chamada regra de desambigüação. Geralmente, o *Yacc* utiliza duas regras:

- Regra 1: em um conflito empilhar/reduzir, o padrão é optar pelo empilhar.
- Regra 2: em um conflito reduzir/reduzir, o padrão é optar pelo reduzir utilizando a primeira regra da entrada.

Yacc sempre relata ao usuário o número de conflitos empilhar/reduzir e reduzir/reduzir resolvidos pelas regras 1 e 2. *Yacc* pode ser empregado para criar analisadores sintáticos, usando C e C++.

“O *Yacc* detecta os conflitos de ambigüidade (empilhar/reduzir ou reduzir/reduzir) e mesmo assim gera um analisador sintático.”

Yacc resolve o problema de gramáticas ambíguas (como precedência e associatividade de operadores), com o uso de palavras-chave: %left, %right, ou %nonassoc. A palavra-chave %prec muda o nível da precedência associado com uma regra gramatical particular, isso porque alguns operadores (unário ou binário) têm a mesma representação simbólica, mas com diferentes precedências.

Bison

Bison (DONNELLY; STALLMAN, 2006) é um gerador de analisador sintático que trabalha com gramáticas LALR (1), compatível com o *Yacc*. O analisador sintático gerado é em C e trabalha sobre Win32, MSDOS, Linux e outros sistemas operacionais.

Em termos de tratamento de ambigüidade e conflitos, basicamente, para cada estado da máquina, o analisador verifica o próximo *token* para decidir se a ação deve

¹ Item usado pela gramática para uma ação de redução.

carregar o *token* ou reduzi-lo por alguma regra. Ocorrem conflitos empilhar/reduzir ou reduzir/reduzir se o *Bison* não pode decidir qual regra deve ser escolhida. Assim, a precedência de regras permite resolver esses conflitos.

Quando existe um conflito empilhar/reduzir, *Bison* compara a precedência do *token* a ser deslocado com a precedência da regra a ser reduzida. Aquela que tiver maior precedência será escolhida.

O *Bison* resolve um conflito reduzir/reduzir escolhendo a regra que aparece primeiro na gramática. Todo conflito reduzir/reduzir, segundo Donnelly e Stallman (2006) deve ser estudado e eliminado.

Grammatica

Grammatica (CEDERBERG, 2006) é um gerador de analisador sintático para as linguagens C# e Java, que usa como base gramáticas LL (k), podendo solucionar algumas ocorrências de ambigüidade verificando alguns *tokens* à frente. Quando essa técnica não for possível aplicar, a ambigüidade será detectada pelo *Grammatica* e informada para o usuário, que deve então reescrever a gramática.

Lemon

O analisador sintático *Lemon* (HIPPEL, 2006) é um gerador para a linguagem C. É do tipo LALR (1), mais rápido que os analisadores sintáticos *Yacc* e *Bison*, mas não é compatível com eles. O *Lemon* faz o tratamento da ambigüidade exatamente como o *Yacc* e o *Bison*.

Assim como nos geradores de analisador sintático *Yacc* e *Bison*, o *Lemon* permite uma medida de controle sobre a resolução do conflito do analisador sintático usando regras de precedência. O valor da precedência pode ser atribuído para qualquer símbolo terminal por meio de palavras-chave como: %left, %right ou %nonassoc.

A seguir é descrita a forma com que o *Lemon* resolve os conflitos.

Conflitos empilhar/reduzir são resolvidos como segue:

- se o símbolo a ser deslocado ou na regra a ser reduzida falta a informação da precedência, então resolve em favor da ação empilhar, mas reporta um conflito;
- se a precedência do símbolo a ser deslocado é maior

que a precedência da regra a ser reduzida, então resolve em favor da ação empilhar. Nenhum conflito é relatado;

- se a precedência do símbolo a ser deslocado é menor que a precedência da regra a ser reduzida, então resolve em favor da ação reduzir. Nenhum conflito é relatado;
- se os precedentes são os mesmos e o símbolo do deslocamento é direito-associativo, então resolve em favor da ação empilhar. Nenhum conflito é relatado;
- se os precedentes são os mesmos e o símbolo do deslocamento é esquerdo-associativo, então resolve em favor da ação reduzir. Nenhum conflito é relatado;
- caso contrário, resolve o conflito em favor da ação de empilhar e reporta o conflito.

Conflitos reduzir/reduzir são resolvidos como segue:

- se na regra a ser reduzida falta a informação, então resolve em favor da regra que aparece em primeiro na gramática e reporta o conflito;
- se ambas as regras têm precedência e a precedência é diferente, então resolve em favor da regra com maior precedência e não reporta o conflito;
- caso contrário, resolve o conflito reduzindo-se pela regra que aparece primeiramente na gramática e reporta o conflito.

Coco/R

O gerador de compiladores *Coco/R* (MOSENBOCK; WOB; LOBERBAUER, 2006) trabalha com gramáticas de atributos de uma linguagem fonte e gera um analisador sintático para ela. Há versões para C#, Java, C++, Delphi e Modula-2. O analisador sintático usa o algoritmo LL(1) e conflitos de ambigüidade são solucionados pela verificação de vários símbolos à frente ou por meio de verificação semântica. Dessa forma, pode-se dizer que o analisador sintático *Coco/R* aceita gramáticas LL(k) para um valor qualquer de k.

Accent

O analisador sintático *Accent* (*A Compiler Compiler for the Entire Class of Context free Grammars*) (SCHROER, 2006) pode processar todas as gramáticas livres de contexto sem nenhuma restrição, isto inclui também gramáticas

ambíguas. O usuário pode controlar essas ambigüidades mediante notações. O *Accent* é um gerador de analisador sintático baseado em uma combinação dos algoritmos de Earley e LL (EARLEY, 1970; AHO; ULLMAN, 1986). O analisador sintático é escrito em C e gera código para C, sendo o compilador C usado para compilar o código e criar o programa objeto.

O *Accent* evita os problemas de análise sintática LALR (conflitos empilhar/reduzir e reduzir/reduzir) e análise sintática LL (restrições com relação a regras recursivas à esquerda).

Esse analisador faz uso de duas classes distintas de ambigüidade, que são chamadas de ambigüidade disjuntiva e conjuntiva, as quais serão descritas a seguir, conforme Schroer (2006).

Ambigüidade disjuntiva

Em caso de ambigüidade disjuntiva um mesmo pedaço de texto pode ser processado por diferentes alternativas para um mesmo não-terminal. Na gramática:

$$\begin{aligned} M &\rightarrow A \mid B \\ A &\rightarrow x \text{ (saída a)} \\ B &\rightarrow x \text{ (saída b)} \end{aligned}$$

A entrada x pode produzir a *saída a* ou a *saída b*. Isto porque ambas as alternativas para M (A e B) podem ser aplicadas. Se mais de uma alternativa puder ser aplicada, *Accent* selecionará a última. Isto é, no exemplo, B é selecionado e b será a saída.

O usuário pode controlar a seleção das alternativas. Cada alternativa tem uma determinada prioridade. Em caso de conflitos, uma alternativa com maior prioridade é selecionada. A prioridade padrão para uma alternativa é um número, contando as regras e começando com 1. Uma prioridade pode ser definida por uma notação %prio N , no final das regras, sendo N a prioridade desta. Por exemplo:

$$\begin{aligned} M &\rightarrow A \%prio2 \\ M &\rightarrow B \%prio1 \end{aligned}$$

A primeira alternativa tem maior prioridade e será selecionada nos casos de conflitos.

Ambigüidade conjuntiva

Considerando:

$$\begin{aligned} M &\rightarrow A B \\ A &\rightarrow x \text{ (saída short A)} \\ A &\rightarrow xx \text{ (saída long A)} \\ B &\rightarrow x \text{ (saída short B)} \\ B &\rightarrow xx \text{ (saída long B)} \end{aligned}$$

A entrada xxx pode ser dividida em duas maneiras diferentes para combinar o lado direito de M : um único x reconhecido como A seguido por um par xx reconhecido como B , ou por um par xx reconhecido como A seguido por um único x reconhecido como B . Neste exemplo a saída é:

short A
long B

No segundo caso a saída seria:

long A
short B

O *Accent* resolve tal ambigüidade selecionando uma derivação mais curta para a última escolha. Com isso, a produção B será escolhida para o conflito. Entretanto, assim como na ambigüidade disjuntiva o usuário pode selecionar uma opção diferente. As notações %short ou %long podem preceder a derivação. A definição $M \rightarrow A \%short B$ resulta em

long A
short B

A definição $M \rightarrow A \%short B$ resulta em

short A
long B

Gold

O gerador de analisador sintático *Gold* (COOK, 2006) é um moderno analisador sintático *bottom-up*, que utiliza o algoritmo LALR para análise das linguagens e um autômato finito determinístico para identificar diferentes classes de

tokens. *Gold* é o acrônimo para *Grammar Oriented Language Developer*.

O conjunto de *lookahead* é usado pela construção do algoritmo LALR para determinar quando reduzir por uma regra. Quando uma configuração está completa - (por exemplo, o cursor está após o último símbolo), o algoritmo LALR reduz a regra para cada símbolo no conjunto. Esta informação é armazenada enquanto uma série de ações de redução acontece no estado LALR. Quando um símbolo é lido pelo algoritmo LALR, observa o símbolo no estado atual e então executa a ação associada.

A partir da versão 2.4, o analisador sintático *Gold* fixa automaticamente conflitos empilhar/reduzir pela redução quando o conflito ocorrer. Este é o mesmo comportamento do compilador *Yacc*. Caso o erro seja causado pela ambigüidade da gramática, o *Gold* sugere que esta seja modificada para resolver o conflito.

Ratatosk

Ratatosk (MOGENSEN, 2006) é um gerador de analisador sintático SLR em *Gof* (uma variante Haskell) que gera analisadores sintáticos para uma sintaxe livre de contexto. O gerador de analisador sintático escolherá a técnica de *backtracking* sobre o conflito (empilhar/reduzir e reduzir/reduzir). Com isso uma linguagem SLR poderá ser analisada, com o custo extra de tempo de execução.

Os analisadores sintáticos gerados funcionarão mesmo se existirem conflitos na tabela de análise sintática. Será selecionada a técnica de *backtracking* sobre as possíveis ações do analisador sintático. Primeiramente, ele tenta a ação empilhar, então tenta qualquer ação de redução pela produção na ordem em que ela aparece na gramática.

LLgen

Como *LLgen* (JACOBS, 2006) gera um analisador sintático descendente recursivo sem *backtracking*, ele é capaz de determinar o que fazer em qualquer situação baseado no símbolo corrente de entrada.

Infelizmente, não pode ser feito para todas as gramáticas. O *LLgen* pode ser usado para criar reconhedores para Pascal, C e Modula-2 e foi escrito em C.

Uma das causas para os conflitos pode ser a ambigüidade da gramática, ou ainda porque a gramática pode requerer

uma maior complexidade do analisador sintático que o *LLgen* pode construir. Os conflitos podem ser resolvidos pela reescrita da gramática ou pelos tratamentos clássicos de conflitos sendo eles:

- conflito pode ser resolvido adicionando uma condição *if* em frente da primeira produção em que existe o conflito. Isso consiste em adicionar `%if` seguido por uma expressão em C entre parênteses. *LLgen* avaliará essa expressão sempre que o *token* for encontrado no momento em que o conflito for detectado, dessa forma o conflito será resolvido dinamicamente. Se o valor de expressão não for zero, a primeira produção do conflito será escolhida, senão uma das restantes será escolhida;
- uma outra alternativa para resolver o conflito é o uso de palavras-chave como: `%prefer` ou `%avoid`. A palavra chave `%prefer` é equivalente ao comportamento para `%if (1)`. A palavra chave `%avoid` é equivalente para `%if (0)`.

Um conflito repetitivo pode ser resolvido adicionando-se uma condição *while* à direita. Essa condição consiste em `%while` seguido por uma expressão em C entre parênteses.

Yappy

Yappy (*Yet Another LR(1) Parser Generator for Python*) (REIS; MOREIRA 2006) fornece um analisador léxico e um gerador de analisador sintático LR para aplicações *Python*. *Yappy* constrói tabelas de análise sintática SLR, LR(1) e LALR (1). Algumas gramáticas ambíguas podem ser manipuladas se a informação de propriedade e associatividade é fornecida. O *Yappy* também pode ser útil para o ensino de técnicas de análise sintática LR.

A resolução de conflito dá-se da seguinte forma: se uma gramática for ambígua, a ação de conflitos do analisador sintático será gerada. O analisador sintático testa o tipo de conflito: se o conflito for de ambigüidade na gramática, apenas a informação da precedência e de associatividade será usada para a resolução de conflito empilhar/reduzir. Caso contrário, o conflito será resolvido de maneira padrão (como o analisador sintático *Yacc*):

- no caso de conflito empilhar/reduzir, se a informação de precedência/associatividade está disponível, tenta

usá-la; senão o conflito é resolvido em favor do empilhar;

- no caso de conflito reduzir/reduzir, a primeira regra listada será escolhida.

DISCUSSÃO

Nesta seção são agrupados os resultados do levantamento de ferramentas, resumizando e comparando as informações sobre os analisadores sintáticos examinados. O quadro 1 apresenta um comparativo entre eles.

Pode-se perceber que a maioria dos analisadores sintáticos pesquisados usa o algoritmo LALR.

O gerador de analisador sintático *Yacc* é uma referência para os outros projetos de geradores. A maior parte dos analisadores pesquisados utiliza, baseia-se ou pelo menos menciona as técnicas empregadas pelo *Yacc*.

Percebe-se que os analisadores sintáticos têm implementado ou a forma padrão de tratamento de ambigüidade ou esse tratamento é feito por meio de uma técnica que possibilita a ação em conjunto com o usuário do analisador sintático. Este deve incluir na gramática de entrada especificações relacionadas à precedência, associatividade ou outro tipo de informação relacionada pelo uso de palavras-chave predefinidas.

	Tipo	Linguagem	Conflitos	Ambigüidade de Gramática
Yacc	LALR(1)	Escrito em C, gera para C e C++.	Detecta.	Resolve o problema da ambigüidade de forma padrão ou com o uso de palavras-chave que especificam as precedências de suas regras.
Bison	LALR(1)	Gera para C.	Detecta.	Resolve o problema da ambigüidade de forma padrão ou com o uso de palavras-chave que especificam as precedências de suas regras.
Grammatica	LL(k)	Gera para C# e Java.	Detecta e reporta ao usuário.	Problemas com gramáticas ambíguas são reportados ao usuário.
Lemon	LALR(1)	Gera para C.	Detecta e resolve os conflitos.	Resolve o problema da ambigüidade de forma padrão ou com o uso de palavras-chave que especificam as precedências de suas regras.
Coco/R	LL(1)	Gera para C#, Java, C++, Delphi e Modula-2.	Detecta e resolve os conflitos.	Resolve o problema por verificação de vários símbolos à frente.
Accent	Earley e LL	Escrito em C e gera para C.	Detecta e resolve os conflitos.	Resolve ambigüidades com duas classes de regras (disjuntiva e conjuntiva) que especificam as prioridades das regras.
Gold	LALR(1)	Não encontrado.	Detecta e resolve os conflitos.	Resolve o problema da ambigüidade de forma padrão e sugere modificação na gramática para resolver o problema.
Ratatosk	SLR	Escrito em Gofer, gera para Gofer.	Detecta e resolve os conflitos.	Tenta uma ação empilhar, depois qualquer ação de redução pela produção na ordem em que ela aparece na gramática.
LLgen	LL	Escrito em C, e gera para Pascal, C e Modula-2.	Detecta e resolve os conflitos.	Resolve o problema da ambigüidade com o uso de palavras-chave que especificam as precedências de suas regras.
Yappy	SLR, LR(1) LALR(1)	Gera para, Python.	Detecta e resolve os conflitos.	Resolve o problema da ambigüidade com o uso de palavras-chave que especificam as precedências de suas regras.

QUADRO 1 - Comparativo entre os analisadores sintáticos pesquisados

**“O gerador de analisador sintático *Yacc*
é uma referência para os outros
projetos de geradores.”**

CONCLUSÕES

A ambigüidade é, de maneira geral, um problema para o processo de geração de um analisador sintático. As gramáticas com essa característica (seja inerente ou pela geração de uma tabela de análise sintática com conflitos) representam um problema para a construção de compiladores. Se a estrutura da linguagem para a qual o compilador gerará o código tiver mais de uma árvore possível, o significado dessa árvore não poderá ser determinado de maneira única.

Dessa forma, o tratamento para a ambigüidade é foco quando se elabora um analisador sintático.

Por meio do estudo e observações de algumas das ferramentas para geração de analisador sintático existente, pode-se perceber que o poder computacional dos analisadores sintáticos se restringe aos conflitos ocasionados pela gramática de entrada, ou seja, um analisador sintático é mais poderoso quando ele tem um número menor de gramáticas que ocasionam conflitos durante a construção da tabela de análise sintática. Portanto, este é um ponto que merece atenção por parte dos desenvolvedores.

REFERÊNCIAS

AHO, A.; SETHI, R.; ULLMAN, J. D. **Compilers**: principles, techniques, and tools. Reading Massachusetts: Addison-Wesley, 1986.

BRAND, Mark et al. Disambiguation filters for scannerless generalized LR parsers. In: CONFERENCE ON COMPUTATIONAL COMPLEXITY, 2002, [S. l.]. **Anais...** [S. l.: s. n.], 2002. p. 143-158.

CEDERBERG, Per. **Grammatica**: parser generator. Disponível em: <<http://grammatica.percederberg.net/doc/release/manual/>>. Acesso em: 5 ago. 2006.

COOK, Devin D. **Gold parsing system**. Disponível em: <<http://www.devincook.com/goldparser/>>. Acesso em: 10 ago. 2006.

Este estudo proporcionou aos pesquisadores um aumento do entendimento da relação entre os analisadores sintáticos, os conflitos ocasionados por estes e a desambigüação. Esta pesquisa auxiliou também na compreensão de todo o problema que a ambigüidade acarreta, e sua interação com o formalismo de definição de gramáticas.

Das dez ferramentas consideradas, a maioria utiliza o algoritmo de análise sintática LALR e o analisador sintático *Yacc* como referência. Percebeu-se, também, que os analisadores sintáticos têm implementadas basicamente duas formas de tratamento para conflitos ou ambigüidades. Uma delas é a forma padrão para conflitos empilhar/reduzir e reduzir/reduzir. Outra forma de tratamento é a ação em conjunto com o usuário do analisador sintático. O usuário deve incluir na gramática de entrada as especificações com o uso de palavras-chave predefinidas para tratar a ambigüidade da gramática.

“Se a estrutura da linguagem para a qual o compilador gerará o código tiver mais de uma árvore possível, o significado dessa árvore não poderá ser determinado de maneira única.”

Como objeto de trabalhos futuros pretende-se ampliar o número de ferramentas analisadas, incluindo ferramentas comerciais ou livres.

DONNELLY, Charles; STALLMAN, Richard. **Bison**. Disponível em: <<http://www.gnu.org/software/bison/manual/>>. Acesso em: 4 maio 2006.

EARLEY, Jay. Efficient context-free parsing algorithm. **Communications of ACM**. New York, v. 13, p. 94-102, fev. 1970.

HIPP, Wyrick. **The lemon parser generator**. Disponível em: <<http://www.hwaci.com/sw/lemon/lemon.html>>. Acesso em: 1 set. 2006.

JACOBS, Cerial. **Llgen, an extended ll(1) parser generator**. Disponível em: <<http://tack.sourceforge.net/doc/LLgen.html>>. Acesso em: 9 set. 2006.

JOHNSON, Stephen C. **Yacc: yet another compiler compiler**. Disponível em: <<http://dinosaur.compilertools.net/yacc/index.html>>. Acesso em: 12 set. 2006.

KLINT, P.; VISSER, E. **Using filters for the disambiguation of context-free grammars**. [S.l. : s.n.], 1994.

LOUDEN, K. **Compiladores: princípios e práticas**. São Paulo: Thomson Learning, 2004.

MCNAUGHTON, Robert. **Elementary computability, formal languages, and automata**. Kansas: Z B Publishing Industries, 1993.

MOGENSEN, Torben. **Ratatosk, a parser generator for gofer**. Disponível em: <<ftp://ftp.diku.dk/diku/users/torbenm/Ratatosk.tar.Z>>. Acesso em: 5 set. 2006.

MOSSENBOCK, Hanspeter; WOB, Albrecht; LOBERBAUER, Markus. **The compiler generator coco/r**. Disponível em: <<http://www.ssw.unilinz.ac.at/Coco/Doc/User Manual.pdf>>. Acesso em: 13 set. 2006.

REIS, Rogério; MOREIRA, Nelma. **Yappy: yet another lr (1) parser generator for python**. Disponível em: <<http://www.ncc.up.pt/fado/Yappy/>>. Acesso em: 20 out. 2006.

SCHROER, Friedrich Wilhelm. **Accent: a compiler compiler for the entire class of contextfree grammars**. Disponível em: <<http://accent.compilertools.net/Accent.html>>. Acesso em: out. 2006.