UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

FABIO DE STEFANI

GRAMÁTICAS LIVRES DE CONTEXTO: SIMPLIFICAÇÃO E TRANSFORMAÇÃO IMPLEMENTADAS NO GESPAS

CRICIÚMA, DEZEMBRO DE 2007

FABIO DE STEFANI

GRAMÁTICAS LIVRES DE CONTEXTO: SIMPLIFICAÇÃO E TRANSFORMAÇÃO IMPLEMENTADAS NO GESPAS

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientadora: Profª. M.Sc. Christine Vieira Scarpato

CRICIÚMA, DEZEMBRO DE 2007

FÁBIO DE STEFANI

GRAMÁTICAS LIVRES DE CONTEXTO: SIMPLIFICAÇÃO E TRANSFORMAÇÃO IMPLEMENTADAS NO GESPAS

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

_____

Profa. M.Sc. Ana Claudia Garcia Barbosa
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

_____

Profª. M.Sc. Christine Vieira Scarpato (UNESC)
Orientador

_____

Profª. M.Sc. Silvana Campos de Azevedo (UNESC)

_____

Prof. Eduardo Menna da Silva (UNESC)

Dedico esta conquista aos meus pais Deniz e Madalena, que sem medirem esforços me educaram e guiaram pelo caminho certo.

AGRADECIMENTOS

Gostaria de agradecer acima de tudo e por tudo a Deus, que sempre me deu ânimo e coragem para sempre seguir em frente nesta jornada que se está completando.

Agradeço:

A minha orientadora Christine Vieira Scarpato por ter me ajudado nesse projeto e por dedicado seu tempo sempre se dispondo a ajudar. E também a todos os professores que compartilharam os seus conhecimentos com paciência e dedicação.

Aos meus pais Deniz e Madalena que sempre me incentivaram e apoiaram nos momentos em que mais precisei, ajudaram em cada passo que dei e compreenderam em cada momento que estive ausente e não pude-lhes dar atenção e carinho que mereciam. Agradeço por abrirem mão dos seus sonhos em prol dos meus e em favor das minhas vontades para que eu alcançasse os meus objetivos, além deles serem os melhores exemplos em minha vida. Souberam me ensinar e mostraram que com esforços e sacrifícios podemos seguir em frente, sem desistir de qualquer obstáculo que encontramos pelo caminho.Peço-lhes desculpas pelas minhas faltas e pelos momentos que não soube compreendê-los. Quero que sintam orgulho, pois essa conquista é de vocês também. Vocês fazem parte de cada momento de minha vida.

De forma especial à minha esposa Evelini que sempre esteve presente nos momentos mais difíceis encontrados. Às vezes deixava de realizar seus afazeres para me dar apoio e atenção. Da compreensão que teve quando dos muitos momentos em que não pude lhe dar atenção merecida. Não tenho muito que lhe dizer, somente que você é muito importante na minha vida. Enfim, meu muito obrigado.

A meu irmão Paulo, que com seu incentivo, sabedoria e inteligência sempre se mostrou prestativo e procurou me ajudar em tudo que precisei. Tenho certeza que

sem sua ajuda e incentivo eu não conseguiria. Agradeço também a minha cunhada Clair, que, juntamente com o meu irmão deram-me dois sobrinhos maravilhosos, Laura e Felipe, que mesmo tão pequenos me deram ânimo e coragem. Seus sorrisos são meu incentivo, minha alegria. Obrigado!

“A programação é mais bem vista como o processo de criação de trabalhos literários, que se destinam a ser lidos.” **Donald E. Knuth**

RESUMO

Este trabalho apresenta a expansão do software GESPAS adicionando as funcionalidades de Transformação e Simplificação de Gramática Livre de Contexto de modo a atingir a gramática equivalente com uma representação clara e objetiva. O GESPAS inicialmente só trabalhava com a geração das tabelas de análise sintática para analisadores ascendentes e descendentes preditivos tabulares não efetuando nenhuma transformação na gramática. O trabalho apresenta um estudo sobre gramáticas e seus tipos, a Gramática Livre de Contexto e seus algoritmos de Transformação e Simplificação. Os algoritmos desenvolvidos para o software são os de Eliminação de Símbolos Inúteis, Produções Vazias, Produções que Substituem Variáveis (Produções Unitárias), Simplificações Combinadas, Fatoração e Eliminação de Recursão à Esquerda. Além do estudo necessário para o desenvolvimento, o trabalho apresenta a modelagem do software, como foram desenvolvidos os algoritmos com a ferramenta de programação Borland C++ Builder 6 e, por fim, apresenta os testes aplicados no software. Também são descritas as novas funcionalidades do software com figuras exemplificando os seus novos processos.

Palavras–chave: Linguagens Formais, Gramáticas Livres de Contexto, Simplificação e Transformação de Gramática Livre de Contexto.

ABSTRACT

This work presents the expansion of the software called GESPAS, adding the functionalities of Transformation and Simplification of Free Grammar of Context, reaching the equivalent grammar with a clear and objective representation. At first, GESPAS just worked with the generation of the tables of syntactic analysis for ascending and descending predictives tabulate analyzers and not making any transformation in the grammar. The work presents a study about grammars and their types, the Free Grammar of Context and their algorithms of Transformation and Simplification. The algorithms developed for the software are the Useless Symbols Elimination, Empty Productions, Productions that Replace Variables (Unitary Productions), Combined Simplifications, Factoring and Elimination of the Left recursion. Besides the necessary study for the development, the work presents the software modelling, how the algorithms were developed, using the programming tool Borland C++ Builder 6 and finally it presents the applied tests in the software. It is also described the new functionalities of the software with illustrations exemplifying their new processes.

Key-words: Formal languages, Free Grammar of Context, Simplification and Transformation of Free Grammar of Context.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

BNF	Backus Naur Form
FNC	Forma Normal de Greiback
FNG	Forma Normal de Chomsky
GLC	Gramática Livre de Contexto
LLC	Linguagem Livre de Contexto
LSC	Linguagem Sensível ao Contexto
YACC	Yet Another Compiler – Compiler

SUMÁRIO

1 INTRODUÇÃO

Gramática é o conjunto de regras individuais para uma determinada linguagem, não necessariamente o que se entende por seu uso adequado. Uma gramática serve para definir qual o subconjunto de sentenças que faz parte de uma determinada linguagem (VIEIRA, 2006).

As regras são formadas por um par ordenado de símbolos, podendo ser variáveis não terminais ou símbolos terminais. As variáveis não terminais são símbolos que auxiliam na geração da palavra da linguagem e os símbolos terminais são o alfabeto da linguagem definida (VIEIRA, 2006).

Podem-se citar diversos usos das gramáticas, como por exemplo: o dialeto de uma região, a modelagem de circuitos lógicos ou redes lógicas, sistemas biológicos, sistemas de animação, hipertextos, hipermídia, compiladores, linguagens de programação, XML, entre outras aplicações. No curso de Ciência da Computação da UNESC, as gramáticas são utilizadas para desenvolver trabalhos nas disciplinas de Linguagens Formais e Compiladores.

Está sendo utilizado há alguns semestres nas disciplinas da UNESC citadas acima, um software denominado GESPAS. Este software foi desenvolvido por Fabiano Martins, ex-acadêmico de computação da UNESC como Trabalho de Conclusão de Curso, com o objetivo de gerar as estruturas gramaticais para serem utilizadas na análise sintática da gramática. Ele trabalha com dois tipos de análise sintática: Preditivo Tabular LL, que é um método da análise sintática descendente e o método Ascendente LR, que pertence a análise sintática ascendente. Ambos são técnicas de geração da tabela de análise sintática e requerem que a gramática obedeça algumas regras. É

necessário que se simplifique e transforme essas gramáticas livres de contexto para obter as estruturas sintáticas.

As gramáticas livres de contexto podem ser simplificadas e transformadas de modo a alcançar uma equivalência, mas com a representação mais clara e direta. Esta simplificação é necessária quando se quer testar alguma propriedade da gramática, sendo geralmente mais indicado trabalhar com uma gramática menor. Além disso, quando destina-se à implementação (por exemplo, na construção de compiladores, como citado anteriormente) a simplificação e transformação é necessária para que este processo seja mais simples. Na forma mais comum de implementação, tem-se uma quantidade de procedimentos igual a de variáveis da gramática.

A simplificação de gramáticas pode ser feita manualmente, no papel, identificando-se as partes redundantes e eliminando-as. Porém, isto pode levar há erros, principalmente em gramáticas maiores e mais complexas. Para evitar isto é necessário que se implemente os algoritmos específicos para automatizar este processo.

Desta forma, esta pesquisa propõe o desenvolvimento dos algoritmos de simplificação e transformação de gramática livre de contexto na ferramenta GESPAS, proporcionando as transformações da gramática como fatoração, eliminação de recursão à esquerda, como também a inserção de funções de simplificação, tais como: eliminação de símbolos inúteis, eliminação de produções vazias e produções que substituem variáveis. Estas alterações (transformação e simplificação) efetuadas automaticamente, poderão auxiliar acadêmicos e profissionais na manipulação de Gramáticas Livres de Contexto.

1.1 OBJETIVO GERAL

Expandir o software GESPAS adicionando as funções de Simplificação e Transformação de Gramática Livre de Contexto.

1.2 OBJETIVOS ESPECÍFICOS

O objetivo geral apresentado neste trabalho é alcançado destacando-se alguns objetivos específicos relacionados que são apresentados a seguir:

a) compreender as gramáticas e os seus tipos;
b) estudar as formas normais e ambigüidade para gramática livre de contexto;
c) compreender as simplificações e transformações de gramática livre de contexto;
d) implementar os algoritmos de simplificação e transformação de gramática livre de contexto no GESPAS;
e) disponibilizar uma ferramenta que poderá auxiliar no aprendizado de Linguagens Formais e Compiladores para comunidade acadêmica.

1.3 JUSTIFICATIVA

O aperfeiçoamento do protótipo do GESPAS adicionando as funcionalidades de simplificação e transformação de Gramática Livre de Contexto, poderá auxiliar na atividade de desenvolvimento do compilador, além da utilização da gramática transformada em outras aplicações. Anteriormente, o GESPAS somente

gerava as estruturas sintáticas para serem utilizadas pelos compiladores, não efetuando nenhum tratamento nas regras da gramática.

O estudo da gramática se fez necessário pelo fato de formar a base da análise sintática das linguagens de programação, também permite descrever a maioria das linguagens de programação sendo considerada a classe mais geral das linguagens.

As gramáticas são eficazes para manipular muitas linguagens naturais, devido a sua característica em fornecer somente uma possível representação para cada significado de uma sentença com uma única árvore de derivação. Isto torna fácil a visualização de sentenças que têm duas árvores de derivação (ambigüidade).

Escolheu-se aplicar os algoritmos de transformação no GESPAS porque eles podem tornar a gramática mais simples de ser manipulada e prepará-la para posteriores aplicações sem reduzir o poder de geração da mesma. Podem ser citados alguns exemplos de algoritmos de transformação: eliminação de símbolos inúteis, eliminação de produções vazias, eliminação de produções que substituem variáveis, simplificações combinadas, fatoração e eliminação de recursão à esquerda (MENEZES, 2005).

Com estas alterações efetuadas no software, os alunos das disciplinas de Compiladores e Linguagens Formais da UNESC poderão utilizar gramáticas mais complexas resultando em trabalhos maiores, importantes e com mais valia perante a comunidade acadêmica.

1.4 ESTRUTURA DO TRABALHO

Tendo em vista os objetivos deste trabalho de pesquisa, os objetos de estudo durante a elaboração estão descritos em sete capítulos, já incluso introdução, o trabalho e a conclusão. Os objetos de pesquisa são: linguagens formais, gramáticas, gramática

livre de contexto, algoritmos de simplificação e transformação de gramática livre de contexto.

A pesquisa efetuada sobre as linguagens formais, gramáticas e seus tipos, englobando os conceitos e características são apresentadas no capítulo 02. No próximo capítulo, apresenta-se um estudo de Gramáticas Livres de Contexto, especificamente suas características, funcionalidades e também um estudo dos algoritmos de simplificação e transformação de gramáticas.

No capítulo 04 é explanado o funcionamento do software GESPAS, este que adquiriu as novas funcionalidades de simplificação e transformação de gramáticas. O capítulo subseqüente é composto pelos trabalhos correlatos apresentando o que está sendo feito em relação ao tema abordado.

O trabalho realizado nesta pesquisa é descrito no capítulo 06, o qual apresenta a modelagem do software, como foi feito o desenvolvimento e uma explicação de seu funcionamento. Por último é feito um fechamento do trabalho, com conclusões e sugestões para trabalhos futuros.

2 GRAMÁTICAS

Antes de iniciar o estudo sobre gramáticas, é interessante conhecer alguns conceitos sobre linguagens formais. Isto é importante porque uma gramática qualquer gera uma linguagem denotada por L(G). Essa linguagem compõe todas as sentenças de símbolos terminais deriváveis a partir do símbolo inicial.

2.1 LINGUAGENS FORMAIS

Uma linguagem formal ou linguagem é um conjunto de palavras sobre um alfabeto. É considerado um meio de comunicação, formado por palavras e regras. Pode ser citado como exemplos de linguagens, o idioma de uma região, as linguagens de programação, um protocolo de comunicação de redes, entre outros. Quando podem ser representadas através de uma representação matemática, define-se que é uma linguagem formal (DIVERIO; MENEZES, 2004).

Uma definição de Linguagens usada por Hopcroft, Motwani e Ullman (2002) é que as linguagens compõem um conjunto de símbolos formado a partir de algum alfabeto.

As linguagens formais têm uma sintaxe definida permitindo que se possa saber se ela pertence ou não a linguagem e, uma semântica precisa de modo que não contenha sentenças sem significado ou ambíguas. São úteis em diversas áreas que utilizam a matemática como ferramenta. Como exemplo de aplicação em informática, pode ser citado para desenvolvimento de programas, desde o nível mais alto de programação até o nível de instruções (código de baixo nível) (VIEIRA, 2006).

Para representar a linguagem, um dos formalismos utilizados é a gramática. Esta é composta por um conjunto finito de regras, as quais aplicadas sucessivamente geram palavras. O conjunto de todas as palavras da gramática gera a linguagem (MENEZES, 2005).

A linguagem não precisa incluir todos os símbolos de um alfabeto, podendo possuir palavras com apenas alguns símbolos ou nenhum. A representação formal de uma linguagem pode ser definida por $L \subseteq \beta$, onde β é o alfabeto e L a linguagem. Interpretando a definição, diz-se que a linguagem é um super conjunto de β (HOPCROFT; MOTWANI; ULLMAN, 2002).

Vieira (2006) diz que um alfabeto é um conjunto finito, não vazio de elementos chamados de símbolos. Através deste conjunto são geradas as palavras. Estas são constituídas por um número finito de símbolos do alfabeto e possuem um tamanho definido pelo módulo da palavra (|x| >= 0). Este tamanho pode ser zero, quando a palavra não possuir nenhum símbolo ou maior que zero quando constitui uma cadeia de símbolos.

Uma linguagem pode ter um número infinito de palavras, mas todas são geradas a partir de um alfabeto finito e fixo. Por definição, uma linguagem que não gere nenhuma palavra é apresentada por 'ϕ' e uma palavra que não contenha nenhum caractere é representada por $\{\varepsilon\}$. A interpretação de 'ϕ' é diferente de $\{\varepsilon\}$. O primeiro indica que o alfabeto não gera nenhuma palavra e o segundo define a composição de uma palavra vazia. Estes são gerados a partir de qualquer linguagem (HOPCROFT; MOTWANI; ULLMAN, 2002).

2.2 GRAMÁTICA

Conforme já foi dito, as gramáticas são utilizadas para representar as linguagens. A gramática é composta por um conjunto finito de produções que aplicadas sucessivamente resultam em palavras. A união de todas estas palavras gera a linguagem. É comum dizer que a linguagem gerada é representada por L(G) (MENEZES, 2005).

Segundo Menezes (2005) as gramáticas são definidas por uma quádrupla[1], composta de símbolos não terminais, terminais, produções e o símbolo inicial. Pode ser representada da seguinte forma G = (N, T, P, S) onde:

N = conjunto finito de variáveis ou não terminais;

T = conjunto finito de terminais (símbolos do alfabeto);

P = Produções (regras gramaticais);

S = símbolo não terminal que inicia as produções.

A produção é um conjunto finito de pares de símbolos do alfabeto sendo denominada regra de produção ou somente regra. É o elemento fundamental da gramática formado por um par ordenado (α, β) e pode ser representado da seguinte forma $\alpha \rightarrow \beta$ ou $\alpha ::= \beta$, onde α é um símbolo não terminal ou a junção de um não terminal com um terminal e β é um terminal (símbolo do alfabeto) ou a união de um não terminal com um terminal. Os símbolos não terminais são utilizados como auxílio para a geração da palavra (VIEIRA, 2006).

Segundo Vieira (2006) o par $\alpha \rightarrow \beta$ ou $\alpha ::= \beta$ é descrito da seguinte maneira: α é definido por β onde a parte da esquerda de uma produção nunca pode ser vazia. Quando possuir diversas regras iniciando de um mesmo não terminal, por exemplo: $\alpha \rightarrow \beta_1$; $\alpha \rightarrow \beta_2$; ...; $\alpha \rightarrow \beta_n$, pode-se abreviar da seguinte forma:

[1] Sistema formal constituído de quatro elementos (MENEZES, 2005).

$\alpha \rightarrow \beta_1 | \beta_2 | ... | \beta_n$. A aplicação sucessiva de produções é chamada de derivação de uma palavra.

A derivação é uma operação de substituição efetuada de acordo com as produções. Um par $<\alpha, \beta>$ de uma relação de derivação é representada da seguinte forma: $\alpha \rightarrow \beta$. Essa relação de derivação "$\rightarrow$" é definida da seguinte forma: $S \rightarrow \alpha$, onde S é o símbolo não terminal da gramática, então o par $S \rightarrow \alpha$ pertence à relação de derivação. A derivação é a troca sucessiva dos símbolos não terminais por símbolos terminais até chegar ao final e resultar em uma palavra da linguagem (MENEZES, 2005).

A gramática é composta de um conjunto de produções e de uma variável de partida, que é um símbolo não terminal. Toda derivação deve começar com o símbolo inicial. As derivações das produções a partir do símbolo inicial são chamadas de formas sentenciais e estas são compostas de símbolos terminais e não terminais. Uma produção pode ser aplicada diversas vezes a uma forma sentencial até que não possuam mais variáveis. Quando chegar a esta etapa é conhecido como sentença. A sentença é uma seqüência formada por símbolos terminais (VIEIRA, 2006).

Será adotada a seguinte convenção em relação às gramáticas para melhor compreensão e entendimento do trabalho: letras maiúsculas serão reconhecidas como símbolos não terminais, letras minúsculas serão os terminais e o símbolo ε para vazio.

A seguir é apresentado um exemplo de gramática com suas produções G = (N, T, P, S) (Figura 1).

```
N = {X, D}
T = {0,1,2,3,4,5,6,7,8,9}
P = {X → D | DX,
     D → 0|1|2|3|4|5|7|8|9}
S =  X
```

Figura 1. Exemplo de gramática

Com o objetivo de chegar ao numero 696 deve-se aplicar a derivação por meio das produções. Este processo deve iniciar com o símbolo inicial X. Depois de substituído todos os símbolos não terminais por terminais resultaram na palavra 696. O processo de derivação pode ser visto na Figura 2.

```
X → DX  - aplicando a segunda regra da gramática
    DDX - substituindo o terminal X pela 2 regra produção
    DDD - substituindo o terminal X pela 1 regra da produção
    6DD - substituindo o 1º D por uma regra do não terminal D
    69D - substituindo o segundo D por uma regra do não terminal D
    696 - substituindo o último D por uma regra do não terminal D
```

Figura 2. Exemplo do processo de derivação

Segundo a Hierarquia de Chomsky, as gramáticas podem ser divididas em quatro tipos: Tipo 3 ou Regular; Tipo 2 ou Livre de Contexto; Tipo 1 ou Sensível ao Contexto; Tipo 0 ou Irrestrita.

Esta hierarquia foi definida por Noam Chomsky, um lingüista que iniciou seus estudos em gramáticas na década de 50. Noam definiu estas classes como modelo para as linguagens naturais e cada uma possui um desempenho computacional distinto (LOUDEN, 2004). Esta classificação pode ser observada de uma forma mais clara na Figura 3.

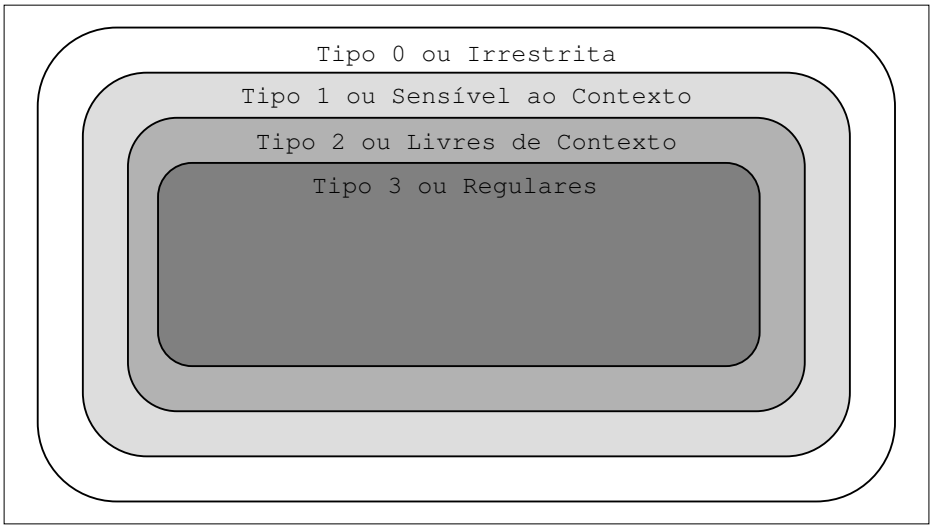


Figura 3. Hierarquia de Chomsky
Fonte: MENEZES, P. (2005).

2.2.1 Gramática Regular

Na gramática regular, o lado esquerdo da produção deve ocorrer somente um símbolo não terminal e único, e do lado direito da produção pode ocorrer símbolos terminais ou a junção de símbolos não terminais com símbolos terminais. A representação formal é apresentada na Figura 4 (VIEIRA, 2006).

$$A \rightarrow aB \quad \text{ou}$$
$$A \rightarrow a \text{ ou seja}$$
$$P = \{ A \rightarrow aX \mid A \in N, a \in T, X \in N \cup \{\varepsilon\} \}$$

Figura 4. Forma Normal da Linguagem Regular
Fonte: VIEIRA, N. J. (2006)

A gramática regular é subdividida em quatro grupos:

a) **gramática linear à direita**: nas regras de produções, os símbolos não terminais sempre tem que estar no lado direito da regra, quando houver. Exemplo: $A \rightarrow aB$ ou $A \rightarrow a$, tal que $a \in T^*$;

b) **gramática linear à esquerda**: os símbolos não terminais têm que ficar sempre do lado esquerdo da gramática. Exemplo: $A \rightarrow Ba$ ou $A \rightarrow a$;

c) **gramática linear unitária à direita**: aplicam-se as mesmas regras da linear à direita com o adicional de ter somente um símbolo terminal, ou seja, $|a| \leq 1$;

d) **gramática linear unitária à esquerda**: aplicam-se as mesmas regras da linear à esquerda, mas o símbolo terminal só pode possuir um símbolo, ou seja, $|a| \leq 1$.

A linguagem regular forma a classe das linguagens mais simples segundo a Hierarquia de Chomsky e fácil de desenvolver algoritmos de reconhecimento, de geração ou conversão entre formalismos de pouca complexidade e de grande eficiência. Em contra partida, possuem fortes limitações de expressividade. É utilizada principalmente em análise léxica e mais recentemente aplicada em sistemas de animações, hipertextos e hipermídia (HOPCROFT; MOTWANI; ULLMAN, 2002).

2.2.2 Gramática Livre de Contexto

A gramática livre de contexto possui suas produções da seguinte forma $A \rightarrow \alpha$, onde A é um símbolo não terminal, e α é uma palavra que pode ser formada pela união de não terminais com terminais. Uma restrição apresentada por este tipo é que do lado esquerdo só pode ter a ocorrência de um não terminal (MENEZES, 2005).

Segundo Menezes (2005) a GLC abrange uma área ampla de linguagens, se comparando com a linguagem regular, incluindo a gramática regular dentro de sua abrangência. Esta gramática gera a Linguagem Livre de Contexto (LLC).

A LLC consegue tratar adequadamente questões de balanceamento de parênteses, blocos estruturados, entre outras características. Uma característica dos algoritmos dessa linguagem é que são simples e possuem uma boa eficiência. Podem ser citadas como exemplos de utilização as linguagens artificiais (linguagens de programação, analisadores sintáticos, tradutores de linguagens, processadores de texto, entre outros) (MENEZES, 2005).

2.2.3 Gramática Sensível ao Contexto

As produções da gramática sensível ao contexto são representadas na seguinte forma $A \rightarrow \beta$, onde A é uma palavra formada por símbolos não terminais podendo ocorrer união com terminais, não podendo ser vazia e β uma palavra que pode ser formada pela união de símbolos terminais com não terminais. Este tipo de gramática apresenta algumas restrições. A primeira é o comprimento do lado esquerdo da produção, deve ser menor ou igual ao comprimento do lado direito da produção e, a segunda é em relação ao símbolo inicial. Se possuir a produção ε, o símbolo inicial não mais poderá aparecer do lado direito de qualquer produção (VIEIRA, 2006).

Vieira (2006) contribui ainda explicando que as formas sentenciais não são decrescentes, ou seja, nunca encolhem. Um exemplo dessa definição pode ser visto na Figura 5. Aplica-se a geração da palavra ‘aaabbbccc’ com a seguinte gramática G = (N, T, P, S) e apresenta-se sua derivação na Figura 6.

```
N = {P, aPBc, cB, Bc, bB}
T = {abc, bb}
P = { P  → aPBc | abc
      cB → Bc
      bB → bb}
S = {P}
```

Figura 5. Exemplo de gramática sensível ao contexto
Fonte: VIEIRA, N. (2006)

```
P → aPBc          - aplicando a primeira regra da gramática
    aaPBcBc       - substituindo o não terminal P pela
                    primeira regra da produção
    aaabcBcBc     - substituindo o não terminal cB pela
                    terceira regra da produção
    aaabBccBc     - substituindo o não terminal bB pela
                    quarta regra da produção
    aaabbccBc     - substituindo o não terminal cB pela
                    terceira regra da produção
    aaabbcBcc     - substituindo o não terminal cB pela
                    terceira regra da produção
    aaabbBccc     - substituindo o não terminal bB pela
                    quarta regra da produção
    aaabbbccc     - derivação
```

Figura 6. Derivação da palavra 'aaabbbccc' de uma gramática sensível ao contexto
Fonte: VIEIRA, N. (2006)

Esta gramática gera a Linguagem Sensível ao Contexto (LSC). As LSC podem ser aceitas por uma Máquina de Turing com Fita Limitada e está contida propriamente na classe das Linguagens Recursivas. As LSC são importantes, pois incluem grande quantidade de linguagens aplicadas (MENEZES, 2005).

2.2.4 Gramática Irrestrita

Na gramática irrestrita, o lado esquerdo da produção pode conter qualquer sentença de símbolos, da mesma forma aplica-se para o lado direito e este pode incluir a

sentença ε. Esta gramática não possui nenhuma restrição nas produções. É representado da seguinte forma:

$$P = \{B \rightarrow a \mid B \in V^{+}, B \in V^{*}\}$$

Segundo Menezes (2005) a gramática irrestrita gera a Linguagem Recursivamente Enumerável. Ela é capaz de representar todas as linguagens que podem ser compiladas mecanicamente, portanto considera-se uma linguagem muita rica e o dispositivo mais geral de computação.

Dentre os tipos de gramáticas apresentados, o trabalho irá conter-se na gramática livre de contexto. Isto ocorre por dois motivos: o software GESPAS somente gera as tabelas de análise sintática e estas são geradas somente para gramáticas livre de contexto; e os algoritmos aplicados no trabalho são desenvolvidos para trabalharem somente com este tipo de gramática, não funcionando para outros tipos.

3 GRAMÁTICAS LIVRES DE CONTEXTO

Uma Gramática Livre de Contexto (GLC) é quádrupla (conforme já foi dito anteriormente) definida formalmente da seguinte maneira: G = (N, T, P, S) com a única restrição que a regra da produção deve ser da forma: $A \rightarrow \alpha$, onde A é uma variável de N e α uma palavra de $(N \cup T)^*$. Em resumo, é uma gramática onde do lado esquerdo de uma produção só pode ter um símbolo não terminal único e do lado direito pode ocorrer uma palavra qualquer constituída por não terminais e/ou terminais (MENEZES, 2005).

Menezes (2005) contribui dizendo que a terminologia "livre de contexto" é resultante da característica de ser a classe mais geral entre as linguagens. Possui esta definição porque a derivação não depende de qualquer análise de símbolos que vêm antes ou depois do não terminal na palavra que está sendo derivada. Outra característica que define este conceito é a sua capacidade de manipular linguagens com duplo balanceamento[2]. A gramática

$$G = (\{E\}, \{+, *, [,], x\}, P, E), \text{ onde } P = \{E \rightarrow E + E \mid E * E \mid [E] \mid x \}$$

permite gerar expressões aritméticas com colchetes balanceados. A derivação que resulta na expressão x + [x * x] é apresentada na Figura 7. Nota-se que foi sempre derivado o terminal mais à esquerda da forma sentencial.

[2] Blocos estruturados ($begin^n$ end^n e similares) e parênteses balanceados $(^n$ $)^n$.

```
E →E + E                (regra E → E + E)
     x + E              (regra E → x)
     x + [E]            (regra E → [E])
     x + [E * E]        (regra E → E * E)
     x + [x * E]        (regra E → x)
     x + [x * x]        (regra E → x)
```

Figura 7. Exemplo de derivação
Fonte: MENEZES, P. (2005)

No desenvolvimento de compiladores, as GLC tiveram uma contribuição importante. Elas transformaram a implementação de analisadores sintáticos ou *parsers* em um trabalho rápido. Recentemente, também estão sendo empregadas para descrever formatos de documentos para tráfego na internet, como por exemplo, XML (HOPCROFT; ULLMAN; MOTWANI, 2002).

Somente é gerada a estrutura de análise sintática para GLC, por isso todos os algoritmos apresentados nesse trabalho são voltados a este tipo de gramática. As principais características da GLC são as seguintes: compreendem um universo amplo de linguagens, efetuando o tratamento de forma adequada os casos de balanceamento de parênteses, estruturas de blocos, são fáceis de serem manipuladas e, os algoritmos que as reconhecem e geram são relativamente simples e possuem uma eficiência razoável.

3.1 BACKUS NAUR FORM

Louden (2004) diz que o *Backus Naur Form* (BNF) é utilizado para representar uma GLC de forma mais clara. Este permite a criação de dispositivos de geração da sentença e cada produção equivale a uma regra de substituição. Utiliza-se para a produção dos dispositivos de geração da sentença. Cada símbolo da metalinguagem é associado a uma ou mais cadeias de símbolos, sendo equivalente às regras da gramática.

O BNF é representado por símbolos da metalinguagem definidos da seguinte forma:

a) as variáveis não terminais são delimitadas pelos símbolos ‘<’ e ‘>’.
   Exemplo: < X >;
b) as palavras não delimitadas são terminais;
c) as produções são representadas da seguinte forma: A ::= α. O não terminal fica a esquerda do símbolo e a regra definida a direita;
d) o ‘|’ (*pipe*) chamado de ‘ou’ é usado para separar as cadeias que ficam do lado direito do símbolo ::=;
e) ε representa a cadeia vazia (LOUDEN, 2004).

Esta convenção não é universal, podem ocorrer variações em relação aos símbolos utilizados de autor para autor. Em um arquivo texto, é complicado representar o símbolo vazio (ε), portanto é comum utilizar a letra ‘i’ com o ‘^’, ficando ‘î’ para representar o vazio em um arquivo texto. Um exemplo de gramática em formato BNF pode ser visto na Figura 8 (LOUDEN, 2004).

```
<identificador> ::= <letra> | <identificador><letra>
<identificador> ::= <identificador> <digito>
<letra>         ::= a | b | c | ... | z | î
<digito>        ::= 0 | 1 | 2 | ... | 9 | î
```

Figura 8. Exemplo de gramática em formato BNF

3.2 AMBIGÜIDADE

Quando uma palavra possuir duas ou mais árvores de derivação, esta é considerada uma gramática ambígua. Em alguns casos, para conseguir trabalhar com uma GLC é necessário que a gramática não seja ambígua (MENEZES, 2005). Para

gerar a expressão x + x *x com a gramática G = ({E}, {+, *, [,], x}, P, E), onde P = {E →E + E | E * E | [E] | x } tem-se diversas formas, tais como as descritas na Figura 9 a seguir:

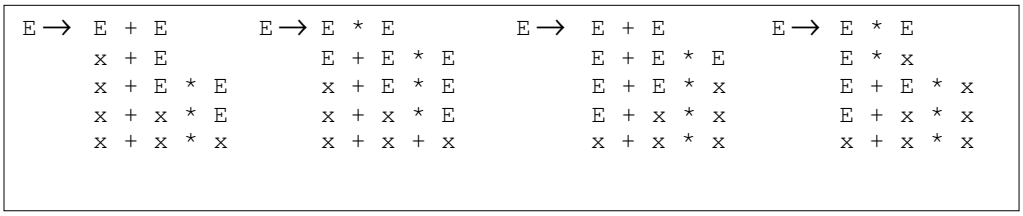


Figura 9. Formas de derivação para a palavra x + x * x

Portanto, esta gramática é considerada ambígua, porque possui diversas árvores de derivação que chegam a um mesmo resultado (Figura 10).

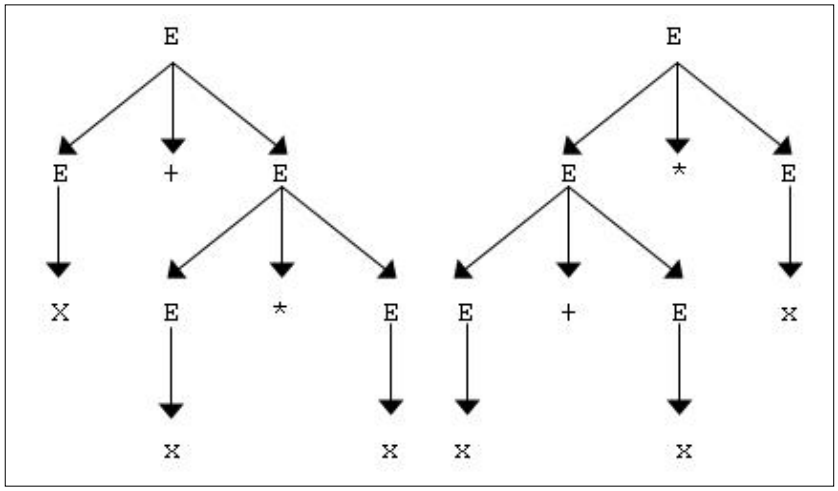


Figura 10. Gramática Ambígua: árvores diferentes para a palavra x + x * x
Fonte: MENEZES, P. (2005)

Segundo Hopcroft, Ullman e Motwani (2002) não existe nenhum algoritmo que defina se uma gramática é ambígua. Algumas são totalmente ambíguas, sendo impossível eliminar a sua ambigüidade. Normalmente é causada por dois motivos:

a) a precedência de operadores não é respeitada. Na Figura 10 verifica-se uma gramática que possui esse problema. Existem duas árvores que resultam na mesma palavra, mas com formas de interpretação diferente;
b) quando se tem uma seqüência de operadores idênticos pode-se efetuar o agrupamento a partir da direita ou da esquerda. Se substituir os símbolos '*' por '+' na árvore de derivação apresentada na Figura 6, resultaria em duas árvores diferentes que correspondam a mesma palavra. Neste caso, tem-se a possibilidade de escolher qual a forma de efetuar a derivação, mas para eliminar a ambigüidade é necessário escolher uma forma.

Podem ser utilizados dois métodos para o tratamento de ambigüidades. Um método é o estabelecimento de uma regra que especifique qual a árvore de derivação é a correta. A árvore correta é aquela em que espelha o real significado da aplicação. Por exemplo, em operações matemáticas, pode-se utilizar a convenção formal da matemática que a multiplicação e divisão são precedentes de adição e subtração. O tratamento de precedência se dá agrupando os operadores em classes de igual precedência e para cada uma é necessário adicionar uma regra. A segunda forma de eliminar a ambigüidade é aplicar a associatividade. Uma expressão aritmética 31 – 2 – 36, não daria de saber qual seria a forma de interpretação correta. Segundo a convenção definida pela matemática, quando a expressão contém uma série de operações deve-se aplicar a associação à esquerda, ficando a (31-2)-36 como a expressão correta (LOUDEN, 2004).

A prática de impor precedência pode ser obtida inserindo não terminais diferentes, cada um representando expressões que compartilhem um nível de força de ligação. Especificando:

a) um fator é uma expressão que não pode ser separado por qualquer operador. Cita-se como exemplo os identificadores e qualquer expressão entre parênteses. Não é possível separar de um identificador as letras que associam a um operador. Os parênteses já são utilizados para impedir que o que está dentro dos parênteses se torne operando de qualquer operador fora dos parênteses;
b) um termo é uma expressão que não pode ser quebrada pelo operador '+'. Por exemplo, para quebrar o termo 'a * b', se for utilizado a associatividade à esquerda e colocar 'a1 *' a sua esquerda é feito o agrupamento de forma '(a1 * a) * b' o que separa 'a * b'. Porém, a inserção de um termo aditivo não pode quebrar 'a * b';
c) após considera-se uma expressão qualquer regra, até mesmo aquelas que podem ser quebradas por um * ou + adjacentes (LOUDEN, 2004).

3.3 SIMPLIFICAÇÃO DE GLC

Existem diferentes métodos de análise, cada um exigindo características específicas, mas se faz necessária a transformação da gramática sem reduzir o seu processamento computacional. Essas simplificações são utilizadas na construção e otimização de algoritmos (MENEZES, 2005). Serão abordadas nesse trabalho as seguintes simplificações:

a) **eliminação de símbolos inúteis**: exclusão de símbolos terminais e não terminais que não são utilizados para a geração da sentença;

b) **eliminação de produções vazias**: tem o propósito de excluir as regras da forma A $\rightarrow \varepsilon$ quando o símbolo não terminal A não for o símbolo inicial da gramática;

c) **eliminação de produções que substituem variáveis (produções unitárias)**: são excluídas as regras gramaticais onde um símbolo não terminal chama outro não terminal A $\rightarrow$ B. Esta forma não contribui para a geração da sentença;

d) **simplificações combinadas**: união das três simplificações citadas anteriormente em uma ordem de execução definida (MENEZES, 2005).

3.3.1 Eliminação de Símbolos Inúteis

O algoritmo de eliminação de símbolos inúteis tem como objetivo a exclusão dos símbolos que não fazem referência à geração da sentença. São considerados símbolos inúteis àqueles que não participam da geração da sentença (estéreis) e que não são alcançáveis (inacessíveis) (VIEIRA, 2006).

Um símbolo estéril não gera nenhuma seqüência de terminais pertencente a uma sentença e o inacessível não aparece em nenhuma forma sentencial da gramática. A gramática S $\rightarrow$AB | a; A $\rightarrow$ b possui todos os seus símbolos geradores. Ao eliminar o símbolo B deve-se eliminar a produção S $\rightarrow$ AB deixando a gramática da seguinte forma S $\rightarrow$ a; A $\rightarrow$ b. Neste ponto foi eliminado um símbolo inútil para a geração da sentença. Analisando a gramática resultante, verifica-se que A $\rightarrow$ b não pode ser acessado, devendo ser eliminada ficando somente a produção S $\rightarrow$ a. A linguagem gerada por essa gramática é {a}, a mesma linguagem gerada pela gramática original (MENEZES, 2005).

Segundo Vieira (2006) deve-se aplicar o algoritmo de eliminação de símbolos inúteis para não ficar com símbolos inúteis e após eliminar os símbolos inacessíveis. O algoritmo para a eliminação de símbolos inúteis é divido em duas etapas e está descrito a seguir:

a) **primeira etapa - qualquer variável gera terminais**: o algoritmo restringe o conjunto de símbolos não terminais, analisando as produções da gramática a partir de terminais gerados. Resulta na gramática $G_1 = (N_1, T, P_1, S)$ na qual $N_1 \subseteq N$ e P_1 possui os mesmos elementos de P, exceto as produções que os símbolos não terminais do cabeçalho da produção não pertençam a N_1. O algoritmo que executa esta etapa está definido na Figura 11;

```
N1 = ϕ
repita N1 = N1 ∪{ A | A → α ∈ P e α ∈ (T ∪ N1)*}
até que o cardinal de N1 não aumente;
```

Figura 11. Exclusão dos simbolos inúteis – Etapa 01
Fonte: MENEZES, P. (2005)

b) **segunda etapa - qualquer símbolo é atingível a partir do símbolo inicial**: após a execução da primeira etapa, o algoritmo analisa as produções da gramática a partir do símbolo inicial verificando se todos os símbolos terminais estão sendo aplicados. Possui como resultado a gramática $G_2 = (N_2, T_2, P_2, S)$ onde $N_2 \subseteq N_1$ e $T_2 \subseteq T$. Se a segunda etapa for executada antes da primeira, o resultado esperado poderá não ser satisfatório. A segunda etapa do algoritmo é apresentada na Figura 12.

$T_1 = \phi$
$N_2 = \{S\}$;
repita
$N_2 = N_2 \cup \{ A \mid X \rightarrow \alpha A \beta \in P_1, X \in N_2 \}$;
$T_2 = T_2 \cup \{ a \mid X \rightarrow \alpha a \beta \in P_1, X \in N_2 \}$;
Até que os cardinais de N_2 e T_2 não aumentem;

Figura 12. Exclusão de simbolos inúteis – Etapa 02
Fonte: MENEZES, P. (2005)

Considerando a seguinte gramática G = ({S, A, B, C}, {a,b,c}, P, S) onde P = {S → aAa | bBb, A → a | S, C → c}, ao aplicar a primeira etapa do algoritmo, obtêm-se como resultado a seguinte gramática G_1 = ({A,C,S}, {a, c}, {S → aAa, A → a | S, C → c}, S). A produção S → bBb foi excluída porque o símbolo não terminal 'B' do lado esquerdo da produção não pertence ao novo conjunto de não terminais. O algoritmo vai sendo executado até que não ocorra mais nenhuma alteração no conjunto (Figura 13) (MENEZES, 2005).

Iteração	Variáveis
Início	ε
1	{A, C}
2	{A, C, S}
3	{A, C, S}

Figura 13. Exclusão de Símbolos Inúteis – Etapa 01
Fonte: MENEZES, P. (2005)

Aplicando a segunda etapa, resultou na gramática G_2 = ({S, A}, {a}, {S → aAa, A → a | S}, S). Foi excluída a produção C → c, pois o símbolo 'c' não entrou no novo conjunto de variáveis terminais. Na Figura 14 são apresentadas as iterações do algoritmo (MENEZES, 2005).

Iteração	Variáveis	Terminais
Início	{S}	ε
1	{S, A}	{a}
2	{S, A}	{a}

Figura 14. Exclusão de Simbolos Inúteis – Etapa 02
Fonte: MENEZES, P. (2005)

3.3.2 Eliminação de Produções Vazias

Menezes (2005) diz que esta simplificação de gramática pode resultar em modificações nas produções da gramática. Tem como objetivo excluir produções da forma $A \rightarrow \varepsilon$ (A não terminal e ε símbolo vazio). A implementação do algoritmo, é dividida em três etapas:

a) **primeira etapa - variáveis que constituem produções vazias**: considera todas as variáveis que geram a palavra vazia ($A \rightarrow \varepsilon$), com exceção se o A for o símbolo inicial da gramática e todas as variáveis que indiretamente geram a palavra vazia ($B \rightarrow A$) (Figura 15);

$V_\varepsilon = \{ A \mid A \rightarrow \varepsilon \}$;
repita
$V_\varepsilon = V_\varepsilon \cup \{ X \mid X \rightarrow X_1 \ldots X_n \in P$ tal que $X_1 \ldots X_n \in V_\varepsilon \}$
Até que o cardinal de V_ε não aumente;

Figura 15. Exclusão das produções que substituem variáveis – Etapa 01
Fonte: MENEZES, P. (2005)

b) **segunda etapa - exclusão de produções vazias**: cada produção que possuir uma variável que gere a palavra vazia do lado direito é adicionada uma produção sem essa variável. Resulta na gramática

G_1 = (N, T, P_1, S). A Figura 16 representa a segunda etapa deste algoritmo;

```
P1 = { A | A → ε }
repita
 para toda A → α  ∈ P1,X ∈ Vε tal que α = α1Xα2, α1α2 ≠ ε
           faça P1 = P1 ∪{ A →  α1α2}
até que o cardinal de P1 não aumente;
```

Figura 16. Exclusão das produções que substituem variáveis – Etapa 02
Fonte: MENEZES, P. (2005)

c) **terceira etapa - geração da palavra vazia, se necessário**: quando a palavra vazia pertencer a linguagem é gerado uma produção com este fim. Se o símbolo inicial pertencer ao conjunto de $V\varepsilon$, adicionar a P_1 as regras S' $\rightarrow$ S e S' $\rightarrow \varepsilon$, incluindo esse novo não terminal S' em N' = N $\cup$ { S' }. Caso contrário, trocar os nomes de S por S' e de N por N'. Adicionado esta produção, resulta na gramática G_2 = (N', T, P_1, S).

Menezes (2005) apresenta um exemplo com a gramática G = ({S, X, Y}, {a, b}, P, S) na qual P = {S $\rightarrow$ aXa | bXb | ε, X $\rightarrow$ a | b | Y, Y $\rightarrow \varepsilon$}. O algoritmo executa suas iterações até que não são mais acrescentadas produções que geram o símbolo ε (Figura 17). A produção S $\rightarrow \varepsilon$ e Y $\rightarrow \varepsilon$ foram excluídas da gramática porque geram a palavra vazia.

Iteração	$V\varepsilon$
Início	{S, Y}
1	{S, Y, X}
2	{S, Y, X}

Figura 17. Exclusão de produções vazias – Etapa 01
Fonte: MENEZES, P. (2005)

A execução da segunda etapa pode ser verificada na Figura 18 a seguir e resulta na gramática G_1 = ({S, X, Y}, {a, b}, {S → aXa | bXb | aa | bb, X → a | b | Y}, S). Nesta etapa foram geradas as seguintes produções: S → aa | bb resultante da segunda etapa do algoritmo, onde são geradas produções para o símbolo não terminal que possua um símbolo gerador de palavra vazia no seu lado direito. As novas produções são formadas com as regras do símbolo gerador de palavra vazia retirando o símbolo gerador (MENEZES, 2005).

Iteração	Produções
Início	{S → aXa | bXb, X → a | b | Y}
1	{S → aXa | bXb | aa | bb, X → a | b | Y}
2	{S → aXa | bXb | aa | bb, X → a | b | Y}

Figura 18. Exclusão de produções vazias – Etapa 02
Fonte: MENEZES, P. (2005)

Como a gramática inicialmente gerava a palavra vazia, deve ser adicionada à produção de geração do símbolo ε. Para fazer a gramática gerar o símbolo inicial, deve ser adicionada a produção S → ε ao símbolo inicial e transferir todas as produções de S para o S', ficando as produções da seguinte forma: S' → S|ε, S → aXa | bXb | aa | bb, X → a | b | Y. A gramática resultante seria G_2 = ({S, X, Y}, {a, b}, { S' → S|ε, S → aXa | bXb | aa | bb, X → a | b | Y }, S). O símbolo não terminal Y não está mais sendo utilizado, resultando em um símbolo inútil. Neste caso, observa-se que não é qualquer combinação de simplificações de gramática que atinge o resultado desejado (MENEZES, 2005).

3.3.3 Produções que Substituem Variáveis (produções unitárias)

Segundo Menezes (2005) uma produção que substitui diretamente uma variável por outra ($A \rightarrow B$) não adiciona informação na geração da sentença, exceto se a variável A possa ser substituída por B e $B \rightarrow \alpha$. Dessa forma, a produção $A \rightarrow B$ por ser substituída por $A \rightarrow \alpha$. O algoritmo para efetuar esse processo é dividido em duas etapas:

a) **etapa 1 - fecho transitivo de cada variável**: o conjunto de variáveis que podem substitui-la transitivamente, por exemplo: $A \rightarrow B$ e $B \rightarrow C$, então B e C pertencem ao fecho transitivo de A. A variável A consegue chegar às outras duas variáveis, conforme ilustra a Figura 19;

```
para toda A ∈ V
faça FECHO-A = { B | A ≠ B e A →+ B usando
      exclusivamente produções de P da forma X → Y };
```

Figura 19. Exclusão das produções que substituem variáveis – Etapa 01
Fonte: MENEZES, P. (2005)

b) **etapa 02 - exclusão das produções que substituem variáveis**: substitui as produções da forma de $A \rightarrow B$ por produções $A \rightarrow \alpha$, onde α é atingível a partir de A por meio de seu fecho. Após esta etapa é resultante na gramática $G_1 = (N, T, P_1, S)$. O algoritmo da etapa 2 está definido na Figura 20.

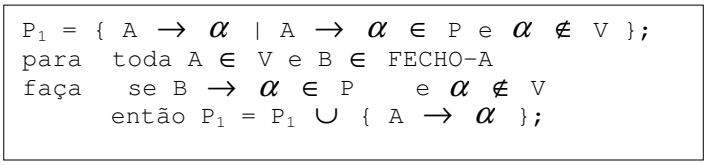


Figura 20. Exclusão das produções que substituem variáveis – Etapa 02
Fonte: MENEZES, P. (2005)

Considerando como exemplo a seguinte gramática G = ({S, X}, {a, b}, {S → aXa | bXb, X → a | b | S | ε }, S), executando a primeira etapa do algoritmo, gerará o seguinte fecho transitivo para cada variável: FECHO-S = Ø e FECHO-X = {S} (MENEZES, 2005).

A iteração da segunda etapa do algoritmo pode ser verificada na Figura 21 e a gramática resultante é a seguinte: G = ({S, X}, {a, b}, {S →aXa | bXb, X→a | b | ε| aXa | bXb}, S). Nesta etapa são adicionadas as produções de S para X, pois o símbolo X referencia o não terminal S. (MENEZES, 2005).

Iteração	Produções
Inicial	{S → aXa | bXb, X → a | b | ε }
S	{S → aXa | bXb , X → a | b | ε }
X	{S → aXa | bXb, X → a | b | ε | aXa |bXb}

Figura 21. Exclusão das produções que substituem variáveis – Etapa 02
Fonte: MENEZES, P. (2005)

3.3.4 Simplificações Combinadas

O algoritmo de Simplificações combinadas é a execução dos algoritmos de simplificação de GLC em uma determinada ordem para se obter o resultado desejado. Deve-se observar que nem sempre após a execução dos algoritmos, se obtém o resultado desejado. Por exemplo: ao aplicar os algoritmos em uma gramática que não possua

símbolos inúteis, mas possui produções que substituem variáveis, poderá após a execução desse segundo algoritmo a gramática começar a ter símbolos inúteis, pois foi alterada (MENEZES, 2005).

A ordem de execução dos algoritmos recomendada é a seguinte: Eliminação de Produções Vazias; Eliminação de Produções que Substituem Variáveis e Eliminação de Símbolos Inúteis (VIEIRA, 2006).

3.4 RECURSÃO

Uma gramática é dita recursiva quando uma variável deriva ela mesma ($A \rightarrow A\varepsilon$). No desenvolvimento de algoritmos reconhecedores, isso não é desejado. A recursão é dividida em dois tipos: Recursão à Esquerda e Recursão à Direita (AHO; SETHI; ULLMAN, 1995).

São consideradas gramáticas com recursão à esquerda quando existir algum $A \in N$ tal que $A + \rightarrow A\alpha$, $\alpha \in (N \cup T)^*$. Uma gramática com recursão à direita é aquela que existe $A \in N$ tal que $A + \rightarrow \alpha A$, $\alpha \in (N \cup T)^*$. Quando a recursão for de um passo, ou seja, $A \rightarrow A\alpha \in P$ ou $A \rightarrow \alpha A \in P$ é considerada recursão direta, sendo possível efetuar a remoção de forma simples. A recursão indireta é mais trabalhosa de ser eliminada e exige que a gramática esteja simplificada (PRICE; TOSCANI, 2005).

A seguir são apresentados os algoritmos para a eliminação de recursão direta e indireta para a recursão à esquerda.

3.4.1 Eliminação de Recursão Direta

A recursão direta é aquele em que o símbolo não terminal referencia a ele mesmo, por exemplo, A $\rightarrow$ A. A eliminação dessa recursão é feita aplicando o algoritmo apresentado na Figura 22 a seguir (REIS, 2006).

Susbstituir cada regra
$A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_n | \beta_1 | \beta_2 | \ldots | \beta_m$,
onde nenhum β_i começa por A, por:
$A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_m A'$
$A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_n A' | \varepsilon$

Figura 22. Algoritmo de eliminação de recursão direta
Fonte: REIS, C. (2006)

3.4.2 Eliminação de Recursão Indireta

Segundo Reis (2006) a eliminação de recursão indireta é feita através do algoritmo apresentado na Figura 23.

a) ordene os não terminais de G em uma ordem qualquer (A_1, A_2,..., A_n)
b) para i de 1 até *n* faça
  para j de 1 até (i – 1) faça
    troque $A \rightarrow A_j\gamma$ por $A_i \rightarrow \delta_1\gamma | \delta_2\gamma | \ldots | \delta_k\gamma$, onde δ_1, δ_2, ..., δ_k são os lados direitos das A_j-produções (ou seja, $A_j \rightarrow \delta_1 | \ldots | \delta_k$);
  fim_para
  elimine as recursões diretas de A_i-produções
 fimpara

Figura 23. Algoritmo de renomeção dos não terminais – Etapa 01
Fonte: REIS, C. (2006)

Utilizando como exemplo a gramática G = ({S, A}, {a}, {S →Aa,A→ Sb|cA|a}, S), a eliminação de recursão para esta gramática ficaria da seguinte maneira: executando a primeira etapa do algoritmo (a) resultaria nas produções: A_1 = S, A_2 = A e n = 2. Na primeira etapa do algoritmo b é feita alteração das produções que causam recursão indireta em recursão direta. A produção será alterada A → Sb, pois causa recursão com a produção S → Aa ficando A → Aab. Neste ponto não existem mais recursões indiretas.

O próximo passo é a eliminação de recursão direta. Das produções resultantes da primeira etapa (S → Aa, A → Aab | cA | a) existe recursão direta com o símbolo não terminal A. Deve ser adicionado um símbolo não terminal para eliminar a recursão das produções A → Aab | cA. No final do algoritmo a gramática resultante é G = ({S, A}, {a,b,c}, {S→Aa, A→cAA' | aA', A'→ abA'|ε}) (PRICE; TOSCANI, 2005). O processo de alteração pode ser visto na Figura 24.

```
1) i = 1 (não faz j pois o laço é de 1 até 0);
      Eliminando as recursões diretas das S-produções:
            Não faz nada (pois não tem);
2) i = 2 e j = 1
      Trocar as produções tipo A → Sγ:
            P'= S → Aa, A → Aab|cA|a;
      Eliminar as recursões diretas das A-produções:
            P'' = S → Aa, A → cAA'|aA', A'→abA'|ε
```

Figura 24. Exemplo de eliminação de recursão
Fonte: PRICE, A.; TOSCANI S. (2005)

3.5 FATORAÇÃO

Um gramática quando possuir duas ou mais produções iniciando com a mesma forma sentencial do lado direito, tem-se uma indecisão de qual produção utilizar.

A fatoração é utilizada para eliminar esta indecisão e quando a gramática está fatorada, diz-se que é determinística (PRICE; TOSCANI, 2005).

A fatoração é uma introdução de um não terminal nas produções, quando se tem uma indecisão de qual produção utilizar. Por exemplo, você tem as produções $A \rightarrow aBx \mid aBy$, nesse caso, você irá ficar com uma indecisão de qual produção utilizar. A eliminação dessa indecisão é realizada efetuando a fatoração da gramática, incluindo um novo não terminal ficando da seguinte maneira: $A \rightarrow aBX$; $X \rightarrow x \mid y$. A fatoração possui muita utilidade para a criação de uma gramática adequada à análise sintática preditiva (PRICE; TOSCANI, 2005).

Divide-se a fatoração em dois tipos: o determinismo direto e o indireto. A solução do determinismo direto é efetuando a lógica aplicada no exemplo acima e, o determinismo indireto é retirado fazendo alterações nas regras de produção às derivações necessárias para torná-la em determinismo direto. Após pode-se resolver com o determinismo direto.

Segundo Louden (2004) o algoritmo geral apresentado para a fatoração é apresentado na Figura 25. Conforme o algoritmo vai sendo executado, o número de escolhas de produções para os não terminais vai diminuindo, garantindo a saída do algoritmo.

```
enquanto houver alterações na gramática faça
      para cada não terminal A faça
            seja α um prefixo de comprimento máximo
            compartilhado por duas ou mais escolhas de
            produções para A
            se α ≠ ε então
                  sejam A → a1 | a2 | ... |an todas as escolhas
                  de produções para A e suponha que a1, ..., ak
                  compartilhem α, de forma que A → α β1 |
                  ... | α βk | α βk+1 | ... | αn, βj não
                  tenham prefixo comum compartilhado e
                  αk+1, ..., αn não compartilhem α
            substituir a regra A → α1 | α2 | ... | αn pelas
            regras A → α A′ | αk+1 | ... | αn
              A′ → β1 | ... | βk
```

Figura 25. Algoritmo para fatoração de uma gramática
Fonte: LOUDEN, K. (2004)

Louden (2004) apresenta um exemplo de fatoração com a gramática G = ({S, A, B}, {a, b}, {S → aA | aB, A → aA | a, B → b}, S). Executando o algoritmo, será alterada as seguintes produções S → aA | aB e A → aA | a, isto porque nessas produções é causada um indeterminismo de qual produção utilizar, pois inicia com o mesmo símbolo. A solução para este problema é adição de um símbolo não terminal auxiliar para a geração da produção. Por exemplo, para o caso do S → aA | aB ficaria da S → aS', S' → A | B. A gramática resultante após a execução do algoritmo é G = ({S, A, B}, {a, b}, {S → aS', S' → A | B, A → aA', A' → a | ε, B → b }).

3.6 FORMAS NORMAIS

As formas normais estabelecem restrições rígidas em relação as produções, não alterando o poder de geração da gramática. É dividido em duas formas: Forma Normal de Chomsky (FNC) e Forma Normal de Greiback (FNG). Essa estrutura é usada principalmente no desenvolvimento de algoritmos e na prova de teoremas. Considere que A, B e C são não terminais, o α um terminal e 'a' uma palavra: Na FNC, as

produções são da forma: A $\rightarrow$ BC ou A $\rightarrow$ α, na FNG já as produções são da forma A $\rightarrow$ α a (VIEIRA, 2006).

3.6.1 Forma Normal de Chomsky

Quando a gramática gerar a palavra vazia ou as produções não estiverem na forma A $\rightarrow$ BC ou A $\rightarrow \alpha$, onde A, B, C são variáveis e α é um terminal, não é possível aplicar a FNC. Toda a linguagem livre de contexto ε-livre pode ser gerada por uma GLC na FNC (REIS, 2006).

O algoritmo para a transformação de uma GLC em uma gramática na FNC é dividido em três etapas e estas estão descritas a seguir:

a) **etapa 1 - simplificação da gramática**: deve-se aplicar os algoritmos de simplificação de GLC já vistos anteriormente. (eliminação de produções vazias, eliminação de produções que substituem variáveis e eliminação de símbolos inúteis). Após a execução dessa etapa do algoritmo, é obtido como resultado a seguinte gramática: $G_1 = (V_1, T_1, P_1, S)$;

b) **etapa 2 - transformação do lado direito das produções de comprimento maior ou igual a dois**: o lado direito das produções de comprimento maior ou igual a dois deve ser composto somente de símbolos não terminais. A exclusão de um símbolo terminal ‘a‘ pode ser realizada substituindo-se o terminal por uma variável intermediária C_a e incluindo a produção $C_a \rightarrow$ a. A gramática resultante é $G_2 = (V_2, T_1, P_2, S)$ onde V_2 e P_2 são construídos com a execução do algoritmo apresentado na Figura 26;

```
V2 = V1;
P2 = P1;
para toda A →X1X2...Xn ∈ P2 tal que n ≥ 2     faça
      se para r ∈ { 1, ..., n}, Xr é um símbolo terminal
      então ( suponha Xr = a )
            V2 = V2 ∪{ Ca };
            Substitui a por Ca em A → X1X2...Xn ∈ P2;
            P2 = P2 ∪ { Ca → a };
```

Figura 26. FNC – Etapa 02
Fonte: MENEZES, P. (2005)

c) **etapa 3 - transformação do lado direito das produções de comprimento maior ou igual a três, em produções com exatamente duas variáveis**: as produções que possuem o comprimento do lado direito maior que dois devem ser transformados em produções com exatamente duas variáveis. Depois da execução da etapa anterior, o lado direito das produções da forma $A \rightarrow B_1|B_2|...|B_n$ ($n \geq 2$) é composto por não terminais. A conclusão da transformação é dada quando se garante que o lado direito das produções é composto somente por dois símbolos não terminais. Isto é possível gerando-se $B_1|B_2|...|B_n$ em diversas etapas e usando-se variáveis intermediárias. O algoritmo que apresenta esta etapa está definido na Figura 27 e após a execução resulta na gramática $G_3 = (V_3, T_1, P_3, S)$ (MENEZES, 2005).

```
V3 = V2;
P3 = P2;
para toda A →B1B2...Bn ∈ P3 tal que n ≥ 3     faça
      P3 = P3 – { A → B1B2...Bn};
      V3 = V3 ∪{ D1, ..., Dn - 2 };
      P3 = P3 ∪ { A → B1D1, D1 → B2D2, ..., Dn-3 → Bn-
                  2Dn-2,
            Dn-2 → Bn-1Bn }
```

Figura 27. FNC – Etapa 03
Fonte: MENEZES, P. (2005)

3.6.2 Forma Normal de Greiback

Uma GLC está na FNG quando todas as produções de uma gramática estão na forma $A \rightarrow a\alpha$, onde A é um não terminal, 'a' um terminal e α uma palavra. A palavra vazia não é gerada por esta gramática. Toda LLC ε-livre pode ser gerada por uma GLC na FNG (MENEZES, 2005).

Segundo Menezes (2005) o algoritmo para efetuar esta transformação é dividido em seis etapas, as quais serão descritas a seguir:

a) **etapa 1 - análoga a etapa 01 do algoritmo de transformação de FNC:** após a execução é resultado a seguinte gramática $G_1 = (V_1, T_1, P_1, S)$;

b) **etapa 2 - renomeação das variáveis em uma ordem crescente qualquer**: tem-se que renomear as variáveis em uma ordem crescente, como por exemplo, $A_1, A_2, ..., A_n$ em que n é o cardinal do conjunto de variáveis. Critérios diferentes de renomeações podem resultar em diferentes gramáticas, mas todas são equivalentes. A etapa 2 resulta na seguinte gramática $G_2 = (V_2, T_1, P_2, A_j)$;

c) **etapa 3 - Transformação de produções para a forma $A_r \rightarrow A_s\alpha$, em que $r \leq s$**: a primeira variável do lado direito tem que ser maior ou igual a do lado esquerdo, considerando a ordenação efetuada na etapa 02. As produções $A_r \rightarrow A_s\alpha$ tais que r > s são modificadas, substituindo-se a variável A_s pelas suas correspondentes produções ($A_s \rightarrow \beta_1 \mid ... \mid \beta_m$) resultando em $A_r \rightarrow \beta_1\alpha \mid ... \mid \beta_m\alpha$ e assim sucessivamente. Existe um limite para as produções crescentes, que pode ser a geração de um terminal ($A_r \rightarrow a\alpha$) ou de uma recursão ($A_r \rightarrow A_r\alpha$). O algoritmo da etapa 03 é apresentado na Figura 28. Nesta figura também é apresentado

o algoritmo da etapa 04. Ambos são executados no mesmo processo. A gramática resultando é $G_3 = (V_3, T_1, P_3, A_j)$;

```
P3 = P2
para r variando de 1 até n
faça
      para s variando de 1 até r - 1
      Etapa 03
      faça  para toda Ar → As α ∈ P3
            faça  excluir Ar → As α de P3;
                  para toda As → β ∈ P3
                  faça P3 = P3 ∪ { Ar → β α }

      para toda Ar → Ar α ∈ P3
      Etapa 04
      faça  excluir Ar → Ar α de P3;
            V3 = V3 ∪ { Br };
            P3 = P3 ∪{ Br → α } ∪ { Br → α Br};
      para toda Ar → ϕ ∈ P3 tal que Ø não inicia por Ar e
      alguma Ar → Ar α foi excluída
      faça  P3 = P3 ∪ { Ar → ϕ Br }
```

Figura 28. FNG – Etapas 03 e 04
Fonte: MENEZES, P. (2005)

d) **etapa 4 - exclusão das recursões da forma $A_r \rightarrow A_r\alpha$**: a recursão à esquerda pode ser gerada na etapa anterior ou já existir na gramática. Esta deve ser excluída, introduzindo variáveis e incluindo-se a recursão à direita ($B_r \rightarrow \alpha B_r$). Esta etapa é executada junto com a etapa 03 (Figura 28);

e) **etapa 5 - um terminal no início do lado direito de cada produção**: todas as regras da forma $A_r \rightarrow A_s\alpha$ são tais que r < s. As produções da maior variável A_n só podem iniciar por um terminal no lado direito. Assim, se, $A_{n-1} \rightarrow A_n\alpha$ for substituído o A_n pelas suas correspondentes produções, o lado direito de A_{n-1} também iniciará por um terminal. A repetição do algoritmo para A_{n-2}, ..., A_1 resultará na forma $A_r \rightarrow a\alpha$. A

gramática resultante nesta etapa é $G_4 = (V_3, T_1, P_4, A_j)$ onde P_4 é construído conforme o algoritmo da Figura 29. Tem-se que garantir que as produções relativas às variáveis auxiliares B_r iniciam por um terminal do lado direito. Este processo é apresentado na Figura 30;

```
P4 = P3
para r variando de n-1 até 1 e toda Ar → As α ∈ P4
faça  excluir Ar → As α ∈ P4;
      para toda As → β de P4
      faça  P4 = P4 ∪{ Ar → β α };
```

Figura 29. FNG – Etapa 05 Parte 1
Fonte: MENEZES, P. (2005)

```
para toda Br → As βr
faça  excluir Br → As βr de P4;
      para toda As → a α
      faça  P4 = P4 ∪{ Br → a α βr
```

Figura 30. FNG – Etapa 05 Parte 2
Fonte: MENEZES, P. (2005)

f) **etapa 6 - produções na forma A → aα**: é análoga a etapa 3 correspondente no algoritmo de transformação de FNC. Todas as produções devem possuir somente duas variáveis do lado direito.

4 GESPAS – SOFTWARE GERADOR DE ESTRUTURAS SINTÁTICAS

O GESPAS é um protótipo de software desenvolvido pelo ex-acadêmico Fabiano da Silva Martins como Trabalho de Conclusão do Curso de Ciência da Computação pela UNESC. O protótipo foi desenvolvido na ferramenta Borland C++ Builder 4 com a linguagem C++. Utilizou-se esta ferramenta por ser difundida no meio acadêmico e também porque os programas gerados são leves (MARTINS, 2005). Veja na Figura 31 o software GESPAS com uma gramática carregada.

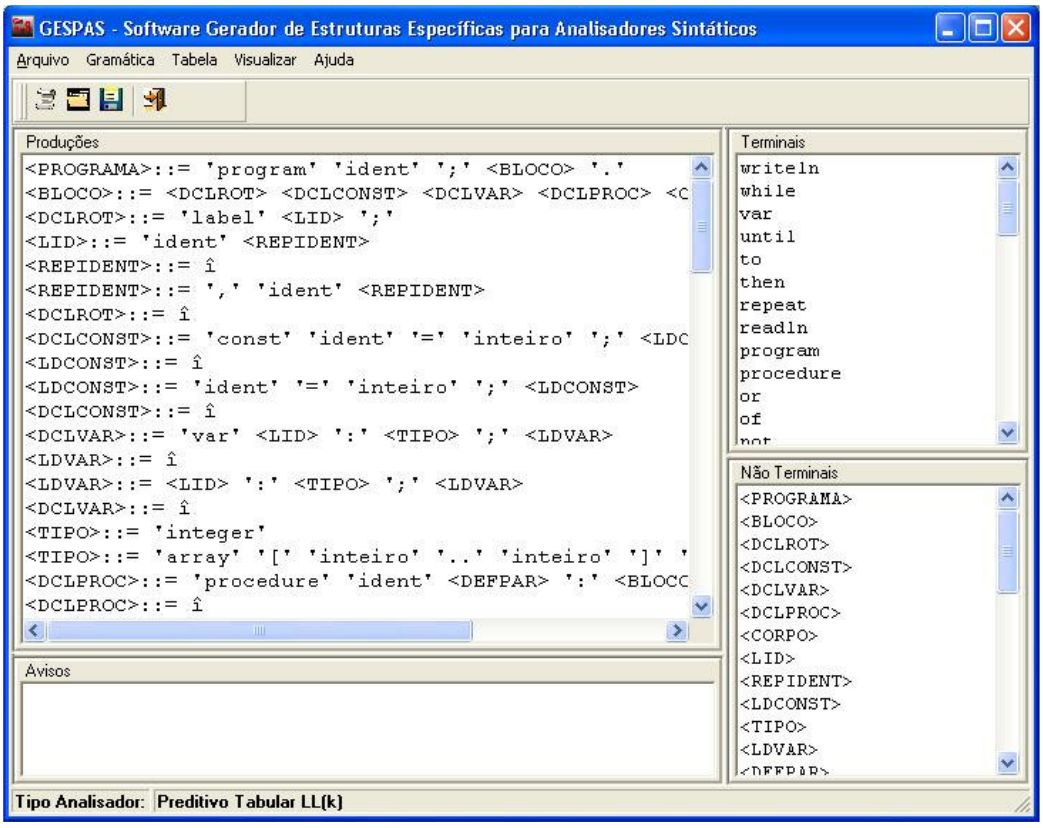


Figura 31. Tela inicial do GESPAS com a gramática carregada
Fonte: MARTINS, F. (2005)

O GESPAS possui como finalidade a geração de tabelas de *parsing* para analisadores sintáticos ascendentes utilizando a técnica SLR e, descendente preditivo tabular. Estas poderão auxiliar acadêmicos e profissionais na implementação de

compiladores. A entrada da gramática pode ser feita com o editor de texto do próprio software ou, por meio de importação de um arquivo em formato bnf. Para o software conseguir aplicar esta análise, a gramática deve estar no formato BNF (MARTINS, 2005). A forma correta de entrada da gramática no software pode ser observada no Anexo A.

Após o software aberto, ao escolher Arquivo → Novo Projeto, o software disponibiliza as opções de criar tabela para analisadores Descendentes Preditivos ou, Ascendentes SLR conforme a Figura 32. Um projeto é a definição de qual análise sintática irá ser aplicada e a gramática a ser trabalhada. O software também disponibiliza a opção para você salvar o projeto para utilização futura (MARTINS, 2005).

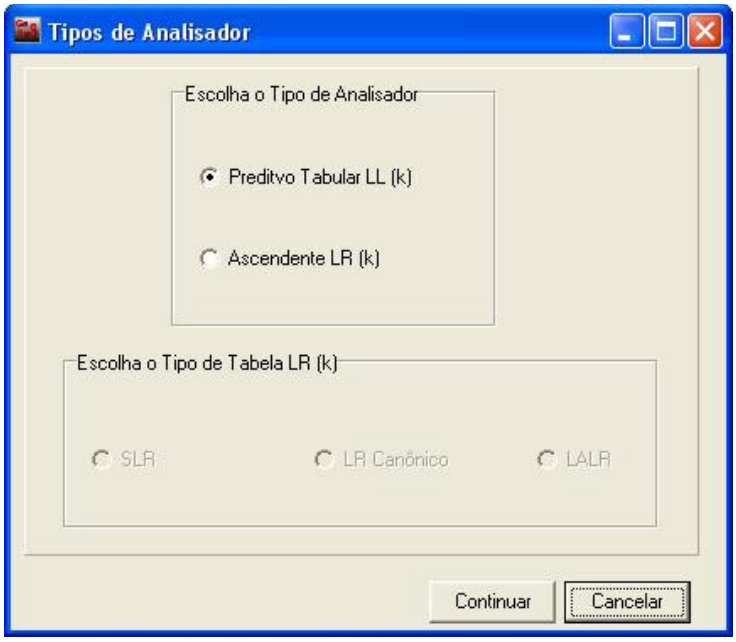


Figura 32. Escolha do analisador sintático
Fonte: MARTINS, F. (2005)

Segundo Martins (2005) depois de feita a escolha da análise sintática que será aplicado, é necessária que se efetue a entrada da gramática no software. Conforme

já foi dito, a gramática deve estar no formato BNF. Os não terminais devem estar entre os símbolos “<” e “>” (maior e menor) e os terminais escritos em letras minúsculas entre aspas. O símbolo “ î ” (letra i com acento circunflexo) significa a palavra vazia. A separação do terminal das cadeias que ele deriva é realizado utilizando o “ ::= ”.

Quando terminado o processo de entrada das regras da gramática, é necessário que se reconheçam os símbolos terminais/não terminais. Para efetuar esse reconhecimento, deve-se ir ao seguinte caminho: Gramática → Encontrar Terminais/NT. O processo de reconhecimento é realizado com a leitura dos caracteres do editor de texto e verificando os símbolos utilizados para separação de símbolos conforme as regras de definição de gramática do GESPAS e separando os terminais e não terminais. Terminando essa análise, o software inclui o símbolo “$” que representa o fim do arquivo (MARTINS, 2005).

Segundo Martins (2005) ao reconhecer um símbolo terminal ou não terminal, o software adiciona-os nos respectivos lugares em sua tela. Os lugares são os espaços para os símbolos terminais, não terminais. É gerada uma lista com os símbolos reconhecidos. Ao finalizar esse processo, o software adiciona uma mensagem informativa no local designado a avisos. Se durante esse processo ocorrer algum erro de análise é inserida uma mensagem nos avisos. Na Figura 33, poderá ser visto a tela do GESPAS com os terminais, não terminais e o espaço reservado para avisos.

Seguindo o procedimento, é necessário efetuar a codificação da gramática. Isto é possível ser efetuado pelo caminho Gramática → Codificar Gramática. A funcionalidade deste processo é ler os terminais e não terminais e definir um código para cada. Após esta definição, ele gera uma estrutura que guarda as produções codificadas e cada elemento é substituído pelo código definido. Ao final, o software

adiciona uma mensagem no local de avisos e a gramática está apta para gerar a tabela de análise (MARTINS, 2005).

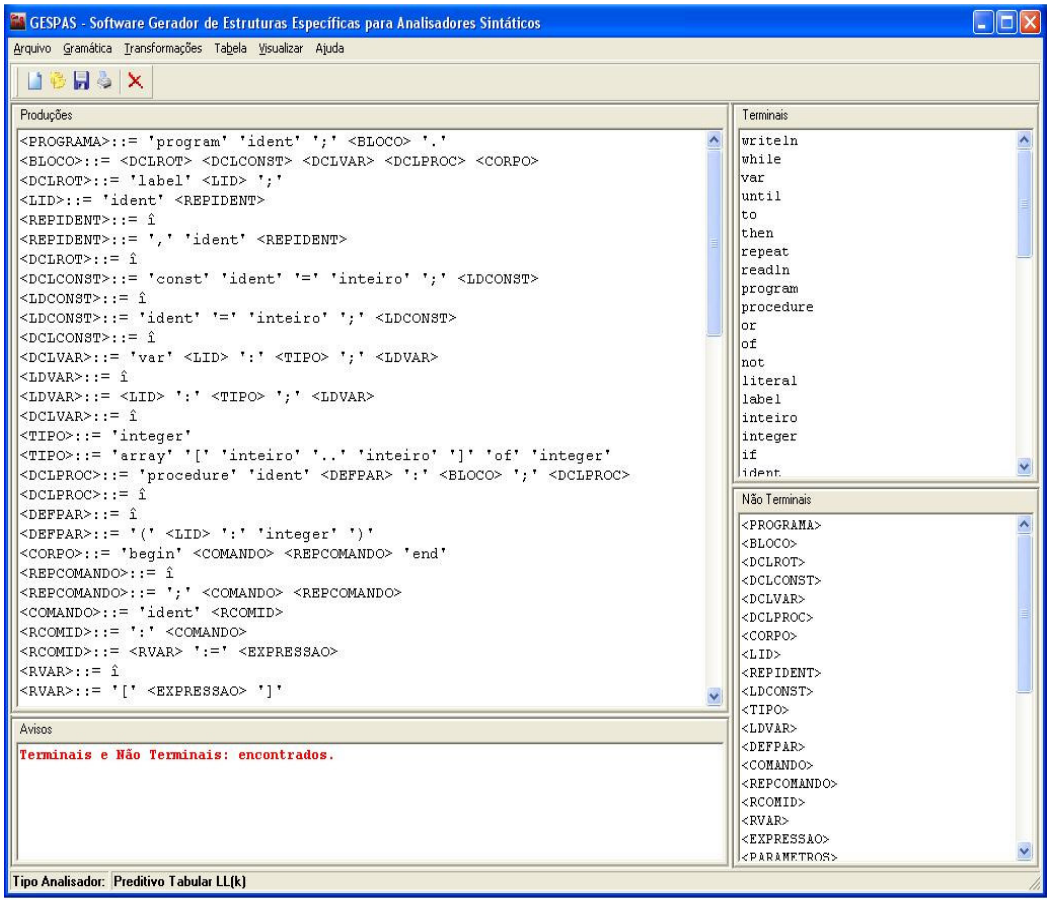


Figura 33. Tela no GESPAS com seus espaços reservados
Fonte: MARTINS, F. (2005)

Terminado o processo de codificação da gramática, pode-se gerar a tabela de análise, escolhendo a opção em Tabela → Gerar. Se for escolhido o tipo de analisador descendente preditivo, o software faz uma busca e verifica se a gramática não possui recursão. Se não possuir, segue calculando os conjuntos *first* e *follow*. Estes conjuntos são utilizados para facilitar a construção da tabela de análise sintática. Quando se escolhe o tipo de análise SLR, o software antes de gerar os conjuntos *first* e *follow*, calcula duas outras funções, a *closure* e *goto*. Essas funções são utilizadas para a construção do Conjunto Canônico dos Itens LR que é utilizado para a geração da tabela

de análise sintática. Uma mensagem é adicionada no campo de avisos ao término do processo (MARTINS, 2005).

Segundo Martins (2005) pode ser visualizada e salvo em formato HTML a tabela de análise sintática e os conjuntos *first* e *follow*. Para isso, deve-se acessar o caminho Visualizar → Tabela. A tabela de análise sintática e os conjuntos para uma gramática com a analise preditiva tabular é apresentada na Figura 34. Na Figura 35 são apresentadas as informações de uma gramática que foi aplicada a análise SLR.

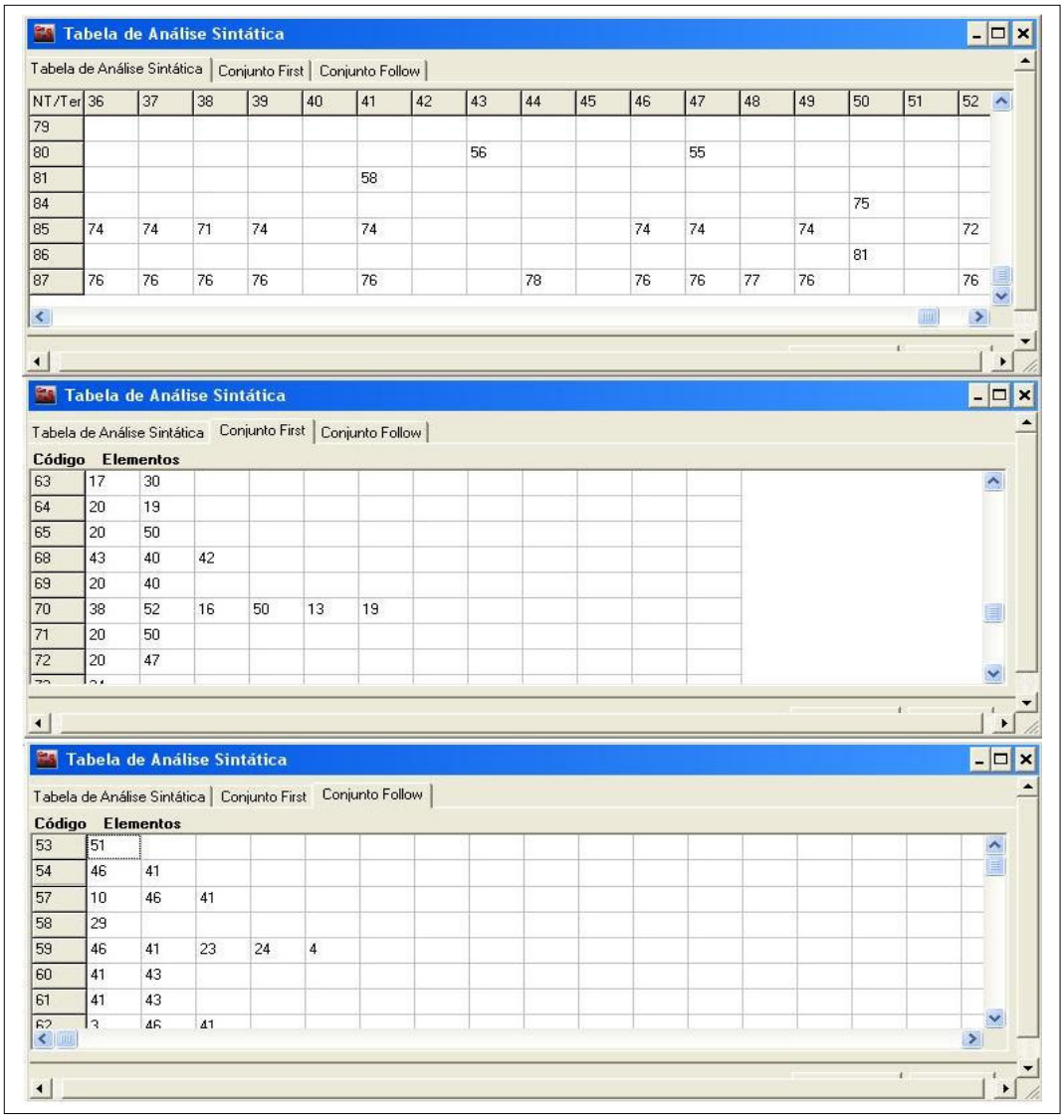


Figura 34. Tabela de Análise Sintática e conjuntos para o tipo preditivo tabular
Fonte: MARTINS, F. (2005)

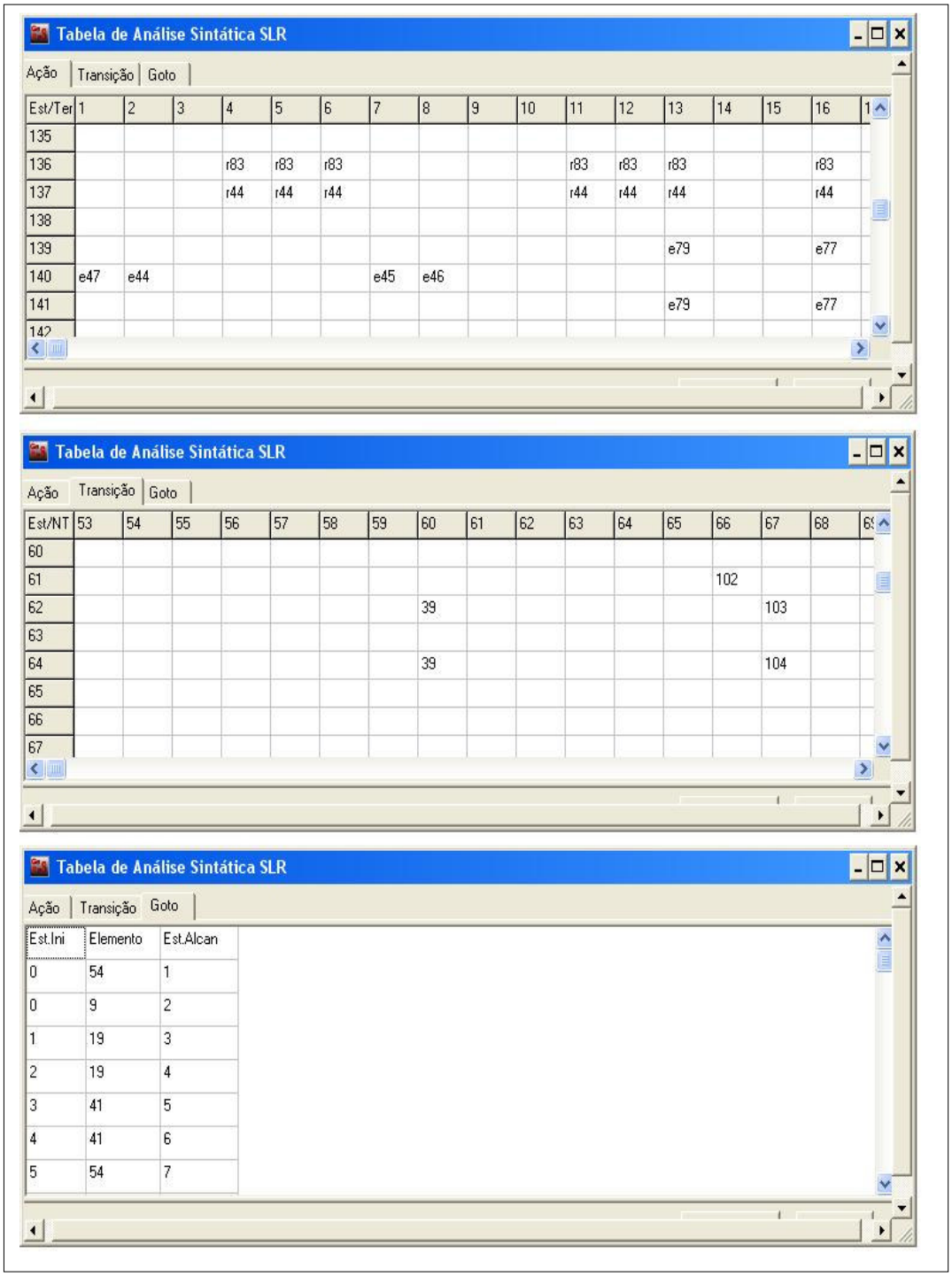


Figura 35. Tabela de Análise Sintática e Conjuntos *Closure* e *Goto* para o tipo SLR
Fonte: MARTINS, F. (2005)

Se todas as etapas anteriores foram executadas sem nenhuma advertência, pode-se efetuar a exportação da gramática codificada e a tabela para as linguagens C++, Delphi, Java, Pascal e HTML ou visualizá-las no software. Para efetuar a exportação, deve-se acessar a opção Tabela → Exportar → Linguagem a ser escolhida. A visualização da gramática codificada pode-ser vista através de Visualizar → Gramática Codificada. A tela de visualização (Figura 36) apresenta as regras da gramática em forma de números, com a codificação da gramática feita anteriormente (MARTINS, 2005).

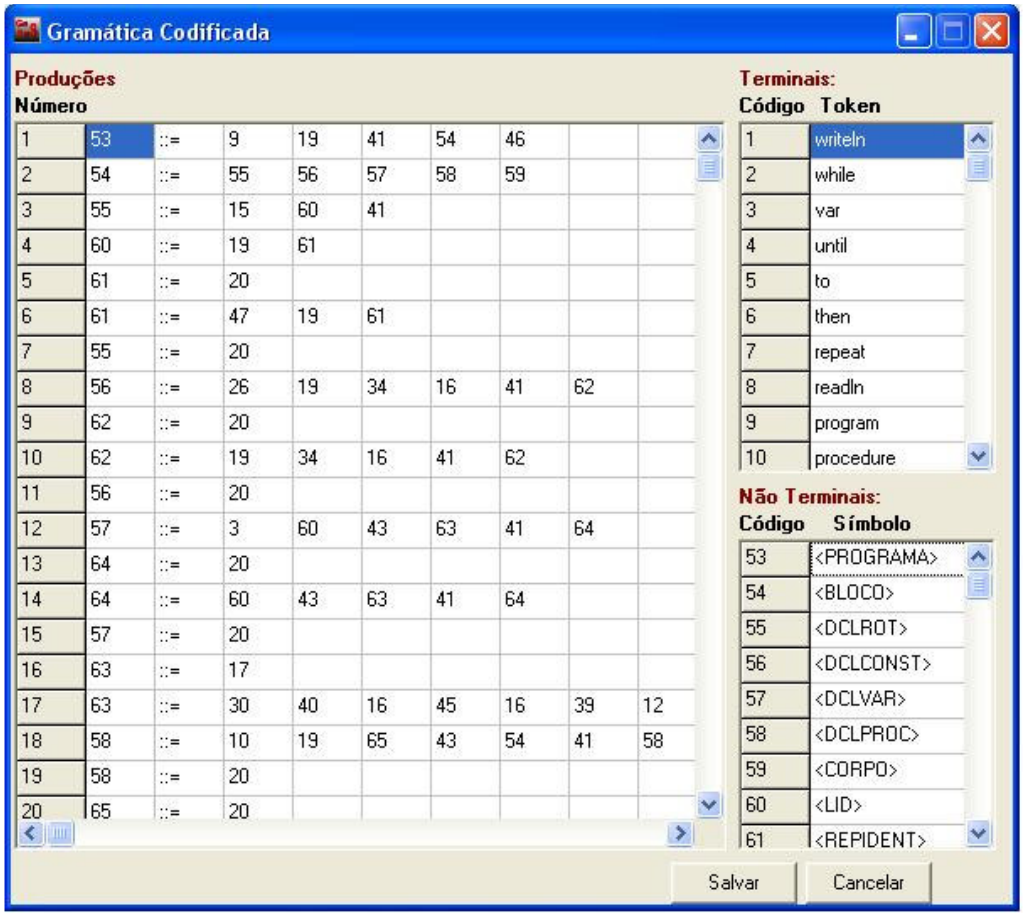


Figura 36. Visualização da Gramática Codificada
Fonte: MARTINS, F. (2005)

5 TRABALHOS CORRELATOS

Neste capítulo serão apresentados alguns trabalhos relacionados ao assunto de linguagens formais e gramáticas, podendo ser citados como exemplo o GALS, YACC e o GAS descritos a seguir. Em relação ao tema abordado no trabalho, não foi localizado nenhum software que apresente as funcionalidades implementadas no software.

5.1 GALS

O GALS é uma ferramenta desenvolvida pelo acadêmico Carlos Eduardo Gesser como Trabalho de Conclusão de curso do Curso de Bacharelado em Ciência da Computação na Universidade Federal de Santa Catarina em 2002. Esta ferramenta propicia a geração de analisadores léxicos e sintáticos. Permite escolher se vai ser gerado somente o analisador léxico, solicitando que o usuário defina os aspectos léxicos, se vai gerar somente o analisar sintático, onde não gera nenhuma informação sobre o analisador léxico e semântico, ficando a cargo do usuário de implementá-los. Uma terceira opção disponível seria a geração do Analisador Léxico e Sintático, onde gera o conjunto dos mesmos e o analisador semântico não é gerado, ficando o usuário responsável de defini-lo (GESSER, 2003).

A ferramenta trabalha com a linguagem regular, sendo feita a entrada dos dados de duas formas: por meio de um arquivo texto ou digitado diretamente em sua interface. Poderá ser utilizado os analisadores sintáticos Descendente Recursivo, LL(1), SLR(1), LALR(1), LR(1) canônico. Ainda tem a opção de gerar a análise nas linguagens Java, C++ e Delphi (GESSER, 2003).

5.2 YET ANOTHER COMPILER – COMPILER

O *Yet Another Compiler – Compiler* (YACC) é um gerador de *parser* desenvolvido para a plataforma UNIX no ano de 1975 por Steve Johnson. Trabalha de forma integrada com o LEX[3], onde recebe como entrada o arquivo gerado com o LEX em código C, com as regras e ações para serem analisadas. Atualmente utiliza como gerador de *parser* os algoritmos de análise sintática LALR(1) e a gramática LR (HUBERT, 2004).

Sua tradução significa "mais um compilador de compiladores". Está tradução foi comum para todos os compiladores, mas atualmente são mais conhecidos como geradores de análise sintática. Um detalhe a ser ressaltado é que este software é desenvolvido para a plataforma UNIX e, em conjunto com o LEX (HUBERT, 2004).

O YACC possui uma versão gráfica desenvolvida posteriormente chamada Bison.

5.3 GAS

O GAS foi desenvolvido por Edson Linhares como Trabalho de Conclusão de Curso de Bacharelado em Ciência da Computação na Universidade Federal de Santa Catarina em 1991. É um software gerador de análise sintática que executa em ambiente MS-DOS, ou seja, não possui uma interface gráfica. O GAS tem a finalidade de gerar análise sintática para as classes LL (1), SLR (1), LALR (1) e LR (1). A sua análise se dá

[3] O Lex é um software gerador da estrutura de análise léxica desenvolvida para a plataforma UNIX. Tem como entrada uma expressão regular e é gerado como saída uma rotina de análise léxica em C. A entrada é dividida em tokens e os software determina a classificação de cada um como literal, número, palavra reservada entre outros descritos pela linguagem. Além da expressão regular, o software necessita de uma instrução auxiliar para manipular os tokens. As entradas devem ser feitas através de comandos, sendo preciso o usuário estar familiarizado com isto (HUBERT, 2004).

por meio da importação de um arquivo de texto padrão ASCII e a gramática contida neste arquivo esteja de forma correta. Neste arquivo, também deverá vir especificado os terminais, os não terminais e as produções para o software poder efetuar a análise. Apresenta ainda a função de imprimir ou visualizar a tabela gerada para a linguagem de programação C, Pascal ou Modula 2.

O GAS tem seu uso restrito por não possuir uma interface gráfica e também por ter poucas possibilidades de manipulação da tabela, como por exemplo, a exportação em um arquivo de alguma linguagem.

6 GRAMÁTICAS LIVRES DE CONTEXTO: SIMPLIFICAÇÃO E TRANSFORMAÇÃO IMPLEMENTADAS NO GESPAS

O GESPAS, conforme já apresentado anteriormente, gera a tabela de análise sintática para o Descendente Preditivo Tabular quanto para o Ascendente. No ascendente é utilizada a técnica SLR. Porém, não efetua nenhuma transformação ou simplificação na gramática. Estas transformações ou simplificações quando necessárias eram feiras manualmente. Com a extensão do GESPAS elas já podem ser aplicadas na gramática no próprio software, auxiliando desta forma acadêmicos e profissionais.

O trabalho apresentado consiste na continuidade da implementação do software GESPAS, adicionando as funcionalidades de simplificação e transformação de GLC. Os algoritmos incluídos no software foram os de Fatoração, Eliminação de Recursão à Esquerda, Eliminação de Símbolos Inúteis, Eliminação de Produções Vazias, Eliminação de Produções que Substituem Variáveis (produções unitárias) e Simplificações Combinadas. Estas simplificações e transformações alteram a gramática de modo a alcançar uma gramática equivalente.

A ferramenta de desenvolvimento inicial utilizada no software GESPAS foi a Borland C++ Builder 4, dando-se continuidade nesta etapa com a ferramenta Borland C++ Builder 6. Uma versão mais atual da utilizada no inicio do software. Segundo Martins (2005) escolheu-se essa ferramenta para desenvolver o software por ser muito utilizada no meio acadêmico e, também por gerar programas que não necessitem de muito recurso computacional.

A alteração do protótipo do software disponibilizou para os usuários os processos de simplificação e transformação de gramáticas, a possibilidade de desfazer a

execução do último algoritmo e a comparação entre a gramática alterada com a gramática inicial que se está trabalhando.

Realizada a entrada da gramática no software seguindo as regras necessárias, (Anexo A) deve-se executar o processo de separação dos terminais/não terminais da gramática. Isto é possível de ser executado pelo caminho **Gramática>Encontrar terminais/NT.** Nesta etapa são disponibilizadas as opções de execução dos algoritmos por meio do menu Transformações (Figura 37), onde se tem as opções de Eliminação de Símbolos Inúteis, Eliminação de Produções Vazias, Eliminação de Produções que Substituem Variáveis, Simplificações Combinadas, Fatoração e Eliminação de Recursão à Esquerda. Também é disponibilizada neste caminho a opção de desfazer a execução do último algoritmo e a comparação da gramática original com a gramática alterada. Na hora de gerar a tabela de Análise Sintática, o software verifica se há Recursão a Esquerda, se a gramática está fatorada, se existe símbolos inúteis e inacessíveis e também se existem produções duplicadas na gramática. Quando existir alguma destas ocorrências, o sistema sugere a opção de executar o algoritmo referente a cada uma das situações.

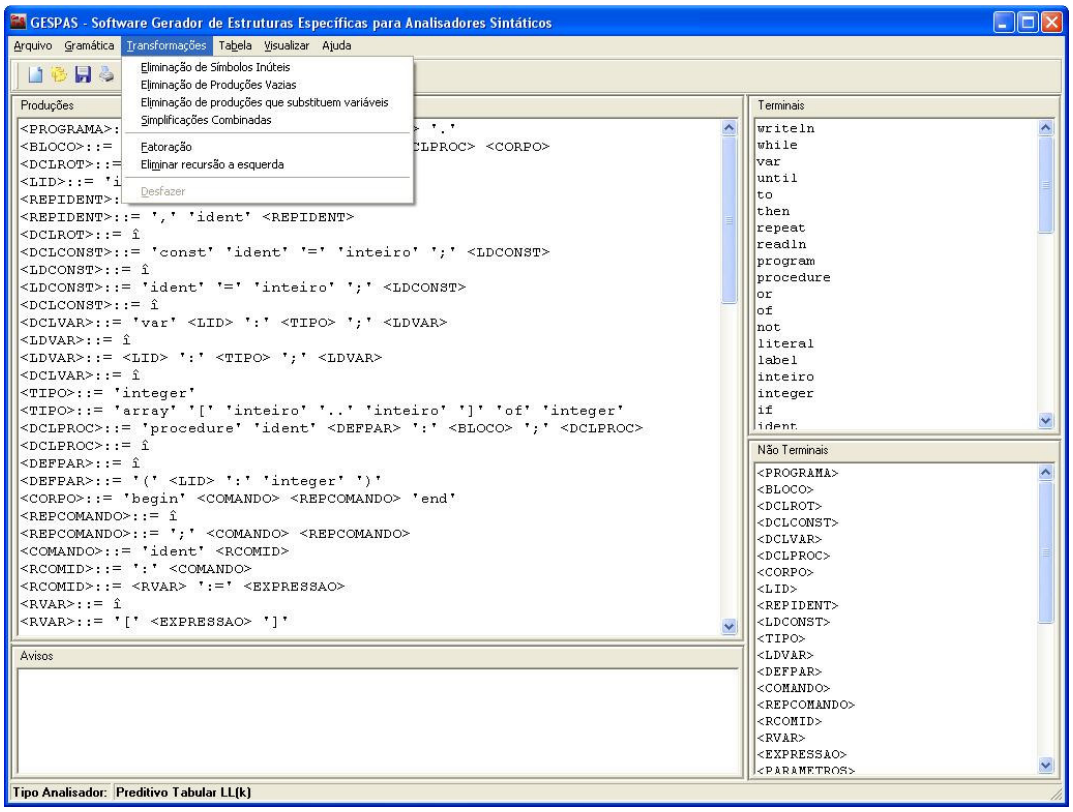


Figura 37. Menu de opções dos algoritmos

Escolhido qual algoritmo deseja-se executar, é iniciado o processo de alteração da gramática (Figura 38).

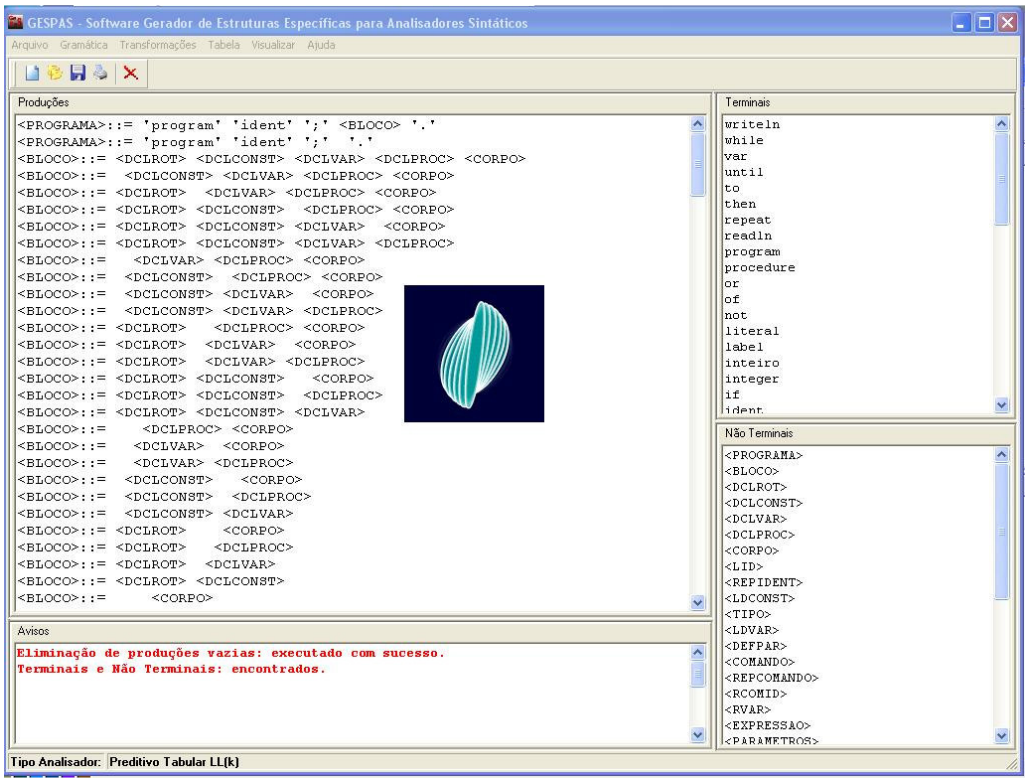


Figura 38. GESPAS executando algoritmo

Quando terminar o processo de execução do algoritmo, a nova gramática é carregada para o GESPAS e inicia-se o processo de separação dos terminais/não terminais de forma automática. Também é apresentada uma mensagem para o usuário, questionando se ele deseja comparar a gramática alterada com a gramática anterior. Caso escolha a opção SIM, será aberta uma janela (Figura 39) onde do lado esquerdo constará a gramática original e do lado direito a gramática alterada. No título desta janela é apresentado qual foi o último algoritmo executado pelo software.

Dispõe-se também nesta tela a opção de impressão das gramáticas. O relatório (Apêndice A) é composto por um cabeçalho com o nome do projeto, o último algoritmo que foi executado e a data de impressão. Após essas informações, a impressão é constituída da gramática original e a gramática alterada.

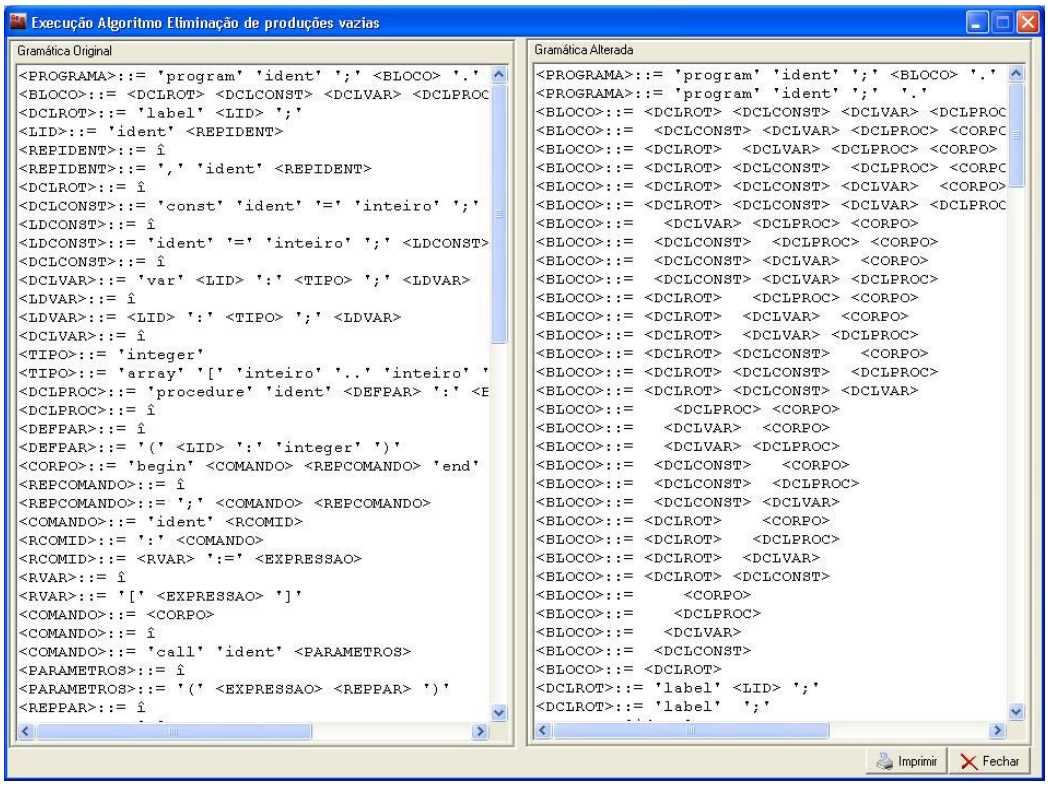


Figura 39. Tela de comparação entre as gramáticas

Na interface principal do software foi feita uma pequena alteração. Foram trocados os ícones dos botões, criados teclas de atalhos para as principais funções do software, como a de criar um novo projeto (CTRL + N), abrir um projeto (CTRL + A), salvar um projeto (CTRL + S), carregar um arquivo BNF (CTRL + F). Adicionou-se também um botão com a possibilidade de impressão da gramática carregada no GESPAS. Esta impressão pode ser vista no Apêndice B. A impressão é constituída de um cabeçalho com o nome do projeto e a data de impressão, a gramática com suas produções, os símbolos não terminais e após os símbolos terminais.

Adotou-se uma metodologia de estudo durante o período de desenvolvimento do trabalho. A primeira etapa a ser concluída foi o levantamento bibliográfico. Neste ponto, levantaram-se as principais literaturas sobre gramáticas. Após esta etapa, iniciou-se um processo de estudo de gramáticas especialmente a

adotada no trabalho, a GLC. Nesta etapa, englobou-se também o estudo de cada algoritmo a ser aplicado no GESPAS, trabalhando-se com exemplos para melhor entendimento de seu funcionamento.

A próxima foi a modelagem do GESPAS utilizando a UML. Modelou-se o software com a UML por ser o esquema de representação gráfica amplamente utilizada para modelagem de sistemas e porque unificou vários esquemas de notações populares (BOOCH; RUMBAUGH; JACOBSON, 2000). Depois de modelar, iniciou-se o desenvolvimento do software. Utilizou-se para o desenvolvimento a ferramenta Borland C++ Builder 6. E por fim realizaram-se os testes aplicando gramáticas de exemplos no GESPAS, verificando seus resultados e efetuando as correções de erros que surgiram.

6.1 ALGORITMOS UTILIZADOS NA IMPLEMENTAÇÃO

Os algoritmos de simplificações e transformações na gramática referenciados no capítulo 3 não foram utilizados para o desenvolvimento do software. Foram utilizados somente para auxilio no entendimento da lógica do algoritmo. O código utilizado no software foi desenvolvido a partir do entendimento de exemplos com gramáticas. Este novo código foi elaborado para que se tenha um algoritmo apresentado em pseudocódigo e não em forma matemática como os descritos no capítulo 3.

O desenvolvimento do código das classes dos algoritmos foi utilizado basicamente a classe *TStringList*[4], manipulação de *Strings*[5], matrizes e vetores. Alguns padrões de programação foram adotados para melhor entendimento do código. As variáveis que são declaradas globalmente na classe são iniciadas com a letra ‘i’ de

[4] Classe composta por uma lista de *Strings* (CANTÙ, 2003).
[5] Tipo de dado utilizado para armazenar caracteres (HUBBARD, 2003).

instância e após a inicial do tipo de dado. Por exemplo, uma variável *String* referente a produção declarada de forma global ficaria da seguinte forma ‘isProducao’. Para variáveis locais de uma função, o que mudaria é que as variáveis iniciariam com a letra ‘l’ de local. Aplicando o padrão no exemplo dado anteriormente ficaria da seguinte forma ‘lsProducao’.

Quando é passado algum argumento para os métodos e funções é utilizado como prefixo a letra ‘a’ de argumento e a inicial do tipo de dado. Outro padrão adotado foi a ordenação das variáveis por seu tamanho em *bits* e por sua quantidade de caracteres no nome. A seqüência de declaração de variáveis segue a seguinte ordem: objetos, classes, variáveis em ordem decrescente de tamanho. Procurou-se também manter o código legível com comentários no código especificando o que cada função tem por objetivo. Foi criada uma classe para cada algoritmo e uma classe geral, contendo funções que são utilizadas por todos os algoritmos de transformação e simplificação de gramática. As classes dos algoritmos são compostas de uma função principal, que é chamada de qualquer lugar do software ou de funções auxiliares.

6.1.1 Eliminação de Símbolos Inúteis

A classe de eliminação de símbolos inúteis é composta de uma função principal que é carregada a partir da tela principal do GESPAS. Esta função utiliza como argumentos a gramática a ser alterada, o conjunto de terminais e o de não terminais retornando a gramática alterada. Esta classe também possui algumas funções auxiliares que são utilizadas para a eliminação de símbolos inúteis. O algoritmo foi dividido em duas etapas. A primeira é a verificação se a produção é geradora de terminais e a segunda verifica se os símbolos não terminais do cabeçalho da produção

são atingíveis a partir do símbolo inicial. O pseudocódigo do algoritmo pode ser visto na Figura 40.

Na primeira etapa é feito um laço de repetição até que não ocorra mais alteração na gramática. Dentro desse laço é feito um outro nas produções da gramática verificando se os símbolos não terminais de cada produção geram terminais de forma direta. Depois de feita a verificação da geração de forma direta, é verificada a geração de terminais de forma indireta. Para ser feita essa verificação foi criado um conjunto[6] de símbolos que inicialmente recebe os terminais da gramática. Caso a produção satisfazer essa regra, é adicionado ao conjunto de símbolos o não terminal do cabeçalho da produção e marcado essa produção para ser excluída. Essa marcação é feita adicionando o código da gramática em um vetor. Ao término do laço de repetição das produções são apagadas as produções marcadas e, verificado se necessita fazer o laço novamente. Permanece nesse laço até que não ocorra mais alteração na gramática. Terminado esses laços, é gerada uma gramática somente com as produções que os símbolos pertençam ao conjunto de símbolos gerados no laço anteriormente. Nesta etapa, já se exclui as produções que não são geradoras.

Quando terminar a primeira etapa, o algoritmo passa para a segunda parte, onde é verificado se os símbolos não terminais do cabeçalho da produção são atingíveis a partir do símbolo inicial. É feito um laço de repetição até que não se tenha mais alteração na gramática e dentro desse laço é feito um outro nas produções da gramática. É verificado para cada produção se o símbolo não terminal do cabeçalho da produção é atingível a partir do símbolo inicial. Quando a produção satisfizer esse requisito, é adicionado o símbolo não terminal em um conjunto. Terminado esse laço é gerada uma

[6] Lista de *Strings*

gramática, contendo somente as produções que possuam os símbolos não terminais separados no processo anterior e retorna essa gramática para ser carregada no GESPAS.

```
lslConjunto = terminais da gramática;
lslConjuntoNT = ϕ
faça
   para  i = 0; i < qtde produções gramática; incrementa i
      lsproducao = produção da gramática na posição i;
      se todos os símbolos da produção são úteis então
         lslconjunto = símbolo NT do cabeçalho da produção
         lslconjuntoNT = símbolo NT do cabeçalho da produção
         adiciona código da produção (i) no vetor de exclusão
      fim se
   fim para
   para i = 0; i < qtde itens vetor; incrementa i
      apaga produção da gramática na posição i;
   fim para
enquanto houver alterações na gramática

para i = 0; i < qtde produções gramática; incrementa i
   lsproducao = produção da gramática na posição i;
   se os símbolos da produção estão todos contidos no conjunto
                 lslConjuntoNT então
      nova_gramatica = adiciona lsproducao
   fim se
fim para

lslgramatica = nova_gramatica;
limpar produções da nova gramática;
faca
   para i = 0; i <  qtde produções gramática; incrementa i
      lsproducao = produção da gramática na posição i;
      se a produção é atingível a partir do símbolo inicial então
         lslConjuntoNT = NT do cabeçalho da produção
         adiciona código da produção (i) no vetor de exclusão
      fim se
   fim para
   para i = 0; i < qtde itens vetor; incrementa i
      apaga produção da gramática na posição i;
   fim para
enquanto houver alterações na gramática

lslConjuntoNT = símbolo NT do cabeçalho da 1° produção da gramática
para i = 0; i < qtde produções da gramática; incrementa i
   lsproducao = produção da gramática na posição i;
   se o símbolo NT do cabeçalho da produções pertencer ao conjunto
                 lslConjuntoNT
      nova_gramatica = adiciona lsproducao;
   fim se
fim para
retornar nova_gramatica;
```

Figura 40. Pseudocódigo algoritmo de Eliminação de Símbolos Inúteis

Um exemplo do algoritmo é apresentado com a gramática G = ({S, A, B, C}, {a, b, c}, P, S) onde P = {S → aAa | bBb; A → a | S; C → c}. Inicialmente o algoritmo inicia o conjunto dos símbolos com os símbolos terminais da gramática e inicia o conjunto de símbolos não terminais com vazio. O algoritmo inicia os laços verificando para cada produção da gramática se os símbolos do lado direito da produção pertencem ao conjunto dos símbolos. As iterações do algoritmo podem ser vistas na Figura 41.

Laço externo	Iteração	Produção	Conjunto	Excluir
1	1	S → aAa	a b c	N
	2	A → bBb	a b c	N
	3	A → a	a b c A	S
	4	A → S	a b c A	N
	5	C → c	a b c A C	S
	Excluir produções marcadas			
2	1	S → aAa	a b c A C S	S
	2	S → bBb	a b c A C S	N
	3	A → S	a b c A C S	S
	Excluir produções marcadas			
3	1	S → bBb	a b c A C S	N
	Excluir produções marcadas			
4	1	A → bBb	a b c A C S	N
	Excluir produções marcadas			

Figura 41. Iterações da primeira etapa do algoritmo

Verifica-se que na quarta iteração do laço externo é que o algoritmo termina a execução da primeira etapa, pois não ocorre mais nenhuma alteração na gramática. Ainda executando a primeira etapa do algoritmo são adicionadas em uma nova gramática somente as produções onde todos os seus símbolos pertençam ao conjunto dos símbolos. As produções resultantes desta etapa são as seguintes: P = {A → aAa; A → a | S; C → c}. A produção S → bBb foi excluída pois o símbolo ‘b’ não é gerado de forma direta por nenhuma produção.

Na segunda etapa do algoritmo é verificado para cada produção se o símbolo não terminal do cabeçalho da produção é atingível a partir do símbolo inicial. O

conjunto de símbolos não terminais é inicializado com o símbolo não terminal do cabeçalho da primeira produção da gramática. Na Figura 42 são apresentadas as iterações feitas nesta segunda parte do algoritmo.

Laço externo	Iteração	Produção	Conjunto NT	Excluir
1	1	S → aAa	S	S
	2	A → a	S A	S
	3	A → S	S A	S
	4	C → c	S A	N
	Excluir produções marcadas			
2	1	C → c	S A	N
	Excluir produções marcadas			
3	1	C → c	S A	N
	Excluir produções marcadas			

Figura 42. Iterações da segunda etapa do algoritmo

Ao término desta etapa, é gerada uma nova gramática somente com as produções que o símbolo não terminal do cabeçalho estiver contido no conjunto gerado anteriormente. Nesta etapa as produções resultantes são P = {S → aAa; A → a | S}. A produção C → c foi excluída, pois não era atingível a partir do símbolo inicial.

6.1.2 Eliminação de Produções Vazias

O segundo algoritmo a ser desenvolvido foi o de Eliminação de Produções Vazias. A classe deste algoritmo é composta por um método que é acessado através da tela principal do software. Este método recebe como parâmetros a gramática, o conjunto de não terminais e o de terminais retornando a gramática alterada. É composta também por métodos e funções auxiliares no processo de eliminação. O correto funcionamento do algoritmo dá-se dividindo o seu processo em duas etapas. Na primeira etapa, é verificado se as produções da gramática geram as palavras vazias de forma direta ou de forma indireta, e na segunda são geradas as novas produções paras as que produzem a palavra vazia. Na Figura 43 é apresentado o pseudocódigo do algoritmo.

O funcionamento da primeira etapa ocorre através de um laço de repetição, até que não ocorra mais alteração na gramática e dentro desse laço é feito um outro percorrendo as produções da gramática. Para cada produção é verificado se a mesma gera a palavra vazia de forma direta ($A \rightarrow \varepsilon$) ou de forma indireta ($A \rightarrow B;\ B \rightarrow \varepsilon$). Quando a produção satisfizer essa característica, é adicionado o símbolo não terminal do cabeçalho da produção em um conjunto e é marcada essa produção para ser excluída. Quando terminar o laço das produções, o sistema exclui as produções que foram marcadas e verifica se precisa iniciar novamente o processo.

A segunda etapa do algoritmo também constitui de um laço nas produções, onde são geradas as novas produções para aquelas que geram a palavra vazia. As produções da gramática original, excetuando as que geram a palavra vazia e as novas produções são adicionadas em uma outra gramática que é retornado para a interface principal do software.

```
lslConjuntoVazio = ϕ;
lbGeraVazio = false;
faca
   para i = 0; i < qtde produções da gramática; incrementa i
      lsProducao = produção da gramática na posição i;
      se a produção gerar a palavra vazia de forma direta então
         lslConjuntoVazio = símbolo NT do cabeçalho da produção;
         adiciona código da produção (i) no vetor de exclusão
      fim se
      se a produção gerar a palavra vazia de forma indireta então
         lslConjuntoVazio = símbolo NT do cabeçalho da produção;
      fim se
   fim para
   para i = 0; i < qtde itens vetor; incrementa i
      apaga produção da gramática na posição i;
   fim para
enquanto não houver alterações no conjunto vazio
se a gramática gerar a palavra vazia então
   lbGeraVazio = true;
fim se
para i = 0; i < qtde produções da gramática; incrementa i
   lsProducao = produção da gramática na posição i;
   nova_gramatica = adiciona lsproducao;
   gramatica_temp = substitui as produções vazias para a produção
   para j = 0; j< qtde produções da gramatica_temp; incrementa j
       nova_gramatica = adiciona produção na posicao j da
            gramatica_temp;
   fim para
fim para
se lbGeraVazio então
   adiciona produção vazia para o símbolo inicial da gramática;
fim se
```

Figura 43. Pseudocódigo do algoritmo de Eliminação de Produções Vazias

Com a gramática G = ({S, X, Y}, {a, b}, P, S) onde P ={S → aXa | bXb | ε, X → a | b | Y; Y → ε } é possível apresentar um exemplo do algoritmo. Na primeira etapa do algoritmo é feita uma verificação para cada produção se é gerada a produção vazia. As produções que são geradoras são marcadas para serem apagadas. A Figura 44 apresenta as iterações do algoritmo.

Laço Externo	Iteração	Produção	Conjunto	Excluir
1	1	S → aXa		N
	2	S → bXb		N
	3	S → ε	S	S
	4	X → a	S	N
	5	X → b	S	N
	6	X → Y	S	N
	7	Y → ε	S Y	S
	Apagar as produções marcadas			
2	1	S → aXa	S Y	N
	2	S → bXb	S Y	N
	3	X → a	S Y	N
	4	X → b	S Y	N
	5	X → Y	S Y X	N
	Apagar as produções marcadas			
3	1	S → aXa	S Y X	N
	2	S → bXb	S Y X	N
	3	X → a	S Y X	N
	4	X → b	S Y X	N
	5	X → b	S Y X	N
	6	S → Y	S Y X	N
	Apagar as produções marcadas			

Figura 44. Iteração da primeira parte do algoritmo

Na terceira iteração do laço externo, o algoritmo termina a sua primeira etapa, pois não ocorreram mais modificações no conjunto de símbolos vazios. Após é feito a verificação se a gramática gera a palavra vazia. Para esta gramática a palavra vazia é gerada através do símbolo inicial com a produção S → ε. Na segunda etapa do algoritmo são geradas as novas produções excluindo os símbolos geradores de palavra vazia. As iterações do algoritmo podem ser vistas na Figura 45. A gramática resultante desta etapa é P = {S → aXa | aa | bXb | bb; X → a | b | Y}. A gramática inicialmente gerava a palavra vazia, portanto deverá ser adicionada a produção geradora da palavra vazia para o símbolo inicial da gramática. A gramática resultante deste algoritmo é G = ({S, X, Y}, {a, b}, P, S) aonde P = {S → aXa | aa | bXb | bb | ε; X → a | b | Y }.

Iteração	Produção	Novas produções
1	S → aXa	S → aa
2	S → bXb	S → bb
3	X → a	
4	X → b	
5	X → Y	

Figura 45. Iteração da segunda etapa do algoritmo

6.1.3 Eliminação de Produções que Substituem Variáveis

O algoritmo de Eliminação de Produções que Substituem Variáveis foi o terceiro a ser implementado no GESPAS. Como todas as outras classes dos algoritmos, esta é composta de uma função principal e de funções auxiliares. A função principal recebe como argumentos a gramática, o conjunto de terminais e o de não terminais, retornando a gramática com as alterações. Esta função foi dividida em dois processos.

A primeira faz um laço de repetição nas produções da gramática, verificando se ela gera produção unitária. Se gerar, é calculada a seqüência de símbolos não terminais utilizados para se chegar à produção em questão e armazenado em uma matriz. A produção que é geradora de produção unitária é marcada para ser excluída. Terminado esse laço, o algoritmo exclui as produções marcadas ficando somente as produções que não geram produções unitárias na gramática.

O segundo processo é feito também através de um laço nas produções resultantes e adicionando-as em uma nova gramática. Neste ponto também é gerada as novas produções relacionadas a produção unitária. Esta geração constitui-se da adição de todas as produções dos símbolos não terminais calculados na primeira etapa que está na matriz para o símbolo não terminal do cabeçalho da produção que se está adicionando. Terminado esta etapa, é retornada a gramática que é carregada no

GESPAS para se dar continuidade aos trabalhos com o software. Na Figura 46 é apresentado o pseudocódigo do algoritmo.

```
Cria matriz [qtde nt da gramatica][qtde producoes gramatica];
Inicializa a matriz;
lsNTold = ' ';
para i = 0; i< qtde produções gramática; incrementa i
   lsproducao = produção da gramática na posição I;
   lsNT = separa NT do cabeçalho da produção;
   se lsNT <> lsNTold entao
      matriz[contador][0] = lsNT;
      incrementa contador;
      lsNTold = lsNT;
   fim se
   se produção gera produção unitária então
      lsNTProducaoUnitaria = separa NT unitario da produção;
      lslConjuntoFecho = calcula o conjunto de símbolos que são
            atingidos a partir do símbolo inicial até o simbolo NT
      adiciona os simbolos do conjunto lslConjuntoFecho na matriz
      adiciona código da produção (i) no vetor de exclusão
   fim se
fim para
para i = 0; i < qtde itens vetor; incrementa i
   apaga produção da gramática na posição i;
fim para
para i = 0; i < qtde produções da gramática; incrementa i
   lsSimbolo = matriz[i][0];
   se lsSimbolo <> ' ' então
      para j = 0; j < qtde produções da gramática; incrementa j
         lsProducao = produção da gramática na posição j;
         lsSimboloNT = símbolo NT do cabeçalho da produção
         se lsSimbolo == lsSimboloNt então
            nova_gramatica = produção da gramática na posição j
         fim se
      fim para
      para j = 0; j < qtde produções gramática; incrementa j
         lsSimbolo_Gerar = matriz[i][j];
         se lsSimbolo_Gerar <> ' ' então
            nova_gramatica_temp = gerar novas produções para o
                  símbolo_gerar
            adiciona as produções da nova_gramatica_temp na
                  nova_gramatica
         fim se
      fim para
   fim se
fim para
retorna nova_gramatica
```

Figura 46. Pseudocódigo do algoritmo de eliminação de produções unitárias

Utilizando a gramática G = ({S, X}, {a, b}, P, S) aonde P = {S → aXa | bXb; X → a | b | S | ε }. Executando a primeira etapa do algoritmo é gerada a matriz de símbolos utilizados para a geração das novas produções (Figura 47).

Iteração	Produção	Fecho	Matriz	Excluir
1	S → aXa		Matriz[1][1] = S	N
2	S → bXb			N
3	X → a		Matriz[2][1] = X	N
4	X → b			N
5	X → S	S	Matriz[2][2] = S	S
6	S → ε			N

Figura 47. Primera etapa do algoritmo

Ao término desta etapa, são excluídas as produções marcadas. Após é gerada uma nova gramática com as produções da gramática e também são inseridas as novas produções geradas através do fecho transitivo calculado. No final da execução do algoritmo, a gramática possui como resultado as seguintes produções P = (S → aXa | bXb; X → a | b | ε | aXa | bXb). As podições X → aXa | bXb foram inseridas para o símbolo X, pois o X possuí uma produção unitária e o seu fecho transitivo é S.

6.1.4 Simplificações Combinadas

O desenvolvimento do algoritmo de Simplificações Combinadas é realizado efetuando-se a chamada dos três métodos descritos anteriormente na seguinte ordem: primeiro é executado o algoritmo de Eliminação de Produções Vazias, após o de Eliminação de Produções que Substituem Variáveis e por último a Eliminação de Símbolos Inúteis. Como nas classes dos outros algoritmos, esta possui um método principal que é acessado através da interface principal do software, passando como argumento a gramática e os conjuntos de símbolos terminais e não terminais. Ao

término da execução desses algoritmos é retornada a gramática alterada que é carregada na tela principal do GESPAS.

6.1.5 Fatoração

Dando continuidade ao desenvolvimento do software, trabalhou-se com o algoritmo de Fatoração. Esse algoritmo foi desenvolvido em uma classe composta de uma função principal e funções auxiliares a principal. A lógica aplicada no algoritmo é feita da seguinte forma: inicia-se um laço de repetição parando quando não ocorrer mais alterações na gramática. Dentro desse laço é feito um segundo nas produções da gramática. É analisado para cada produção se a mesma é geradora de não determinismo na gramática. As produções que são geradoras são marcadas em uma matriz composta do símbolo não terminal do cabeçalho da produção e do código das produções da gramática.

Terminado o primeiro laço é feito outro na matriz gerada separando o símbolo não terminal e o código das produções que são geradoras de indeterminismo adicionando os códigos em um vetor. Nesta etapa, o sistema gera as novas produções retirando o indeterminismo. As produções geradas são adicionadas em uma gramática nova e, no final desse laço é comparada a quantidade de produções com a gramática anterior. Se for diferente, repete todo o processo novamente. Quando terminar definitivamente esse processo, a gramática é retornada da função que é carregada no GESPAS. A Figura 48 apresenta o código do algoritmo.

```
Cria matriz[qtde NT][qtde produções];
lsntold = simbolo NT do cabeçalho da primeira produção;
licountlinhamat = 0;
licountcolummat = 0;
Faca
   Para I = 0; I < qtde produções da gramática; incrementa i
      lsProducao = produção da gramática na posição I;
      lsnt = simbolo NT do cabeçalho da produção;
      se lsnt <> lsntold entao
         incrementa licountcolummat;
         incrementa licountlinhamat;
         matriz[licountlinhamat][licountcolummat] = lsnt;
         lsntold = lsnt;
      fim se
      se producao gera indeterminismo entao
         matriz[licountlinhamat][licountcolummat] = I;
         incrementa licountcolummat;
      fim se
   Fim para

   Para I  = 0; I < qtde produções da gramática; incrementa i
      Gerar vetor de produções adicionando os códigos das produções
            adicionados na matriz para cada simbolo NT;
      Gerar as novas produções para simbolo NT da matriz adicionado
            em uma nova gramatica
   Fim para
Enquanto houver alterações na gramática

Retornar a nova gramatica
```

Figura 48. Código do algoritmo de Fatoracao

Com a gramática G = ({S, A, B}, {a, b}, P, S) aonde P = (S → aA | aB; A → aA | a; B → b) é possível ser visualizado um exemplo do algoritmo. O algoritmo em sua execução vai verificando se as produções estão fatoradas e já vai gerando as novas produções eliminando a fatoração. As iterações do algoritmo são apresentadas na Figura 49. A gramática resultante após a execução é G = ({S, S`, A, A`, B}, {a, b}, P, S) aonde P = {S → aS`; S`→ A | B; A → aA`; A` → A | ε; B → b}.

Laço	Iteração	Produção	matriz	NT	Nova produção
1	1	S → aA	Matriz[1][1] = S Matriz[1][2] = 0		
	2	S → aB	Matriz[1][3] = 1		
	3	A → aA	Matriz[2][1] = A Matriz[2][2] = 2		
	4	A → a	Matriz[2][3] = 3		
	5	B → b	Matriz[3][1] = B		
	1			S	S → aS` S` → A S` → B
	2			A	A → aA` A` → A A` → î
	3			B	B → b
2	1	S → aS`	Matriz[1][1] = S		
	2	S` → A	Matriz[1][1] = S`		
	3	S` → B			
	4	A → aA`	Matriz[1][1] = A		
	5	A` → A	Matriz[1][1] = A`		
	6	A` → î			
	7	B → b	Matriz[1][1] = B		
	1			S	S → aS`
	2			S`	S` → A S` → B
	3			A	A → aA`
	4			A`	A` → A A` → î
	5			B	B → b

Figura 49. Sequencia execução algoritmo fatoração

6.1.6 Eliminação de Recursão à Esquerda

O último algoritmo a ser desenvolvido foi o de eliminação de Recursão à Esquerda. A primeira parte do algoritmo é a separação dos símbolos não terminais definidos no cabeçalho da produção e adicionando-os em um conjunto. Após é feito um laço nas produções da gramática verificando se ela gera recursão indireta. Se a produção gerar recursão, são geradas as novas produções e adicionado em uma nova gramática. As produções já analisadas também são adicionadas em um conjunto.

Quando terminar esse processo, é feito um outro laço nas produções da nova gramática, verificando se é gerada a recursão direta. Caso exista recursão direta, esta

produção é armazenada em um conjunto de produções que só geram recursão. As produções que não geram recursão são armazenadas em um outro conjunto. A próxima etapa é composta de um laço nos símbolos não terminais da gramática geradas no primeiro processo do algoritmo e são geradas as novas produções sem recursão e gera uma nova gramática. É adicionado inicialmente na nova gramática as produções que não geram recursão e após as produções novas geradas pela eliminação de recursão. Terminado todos esses processos, é retornada a gramática para ser carregado na tela do GESPAS e dado continuidade aos processos normais do sistema. O algoritmo pode ser visto na figura 50.

```
Adicionar os NT da gramatica em uma lista de strings (ordem_NT);
Para I = 0; I < qtde produções da gramatica; incrementa I;
   lsproducao = produção da gramatica na posicao I;
   se a produção gerar recursão indireta entao
      lslnalisadas = lsprodução;
       gerar a produção sem recursão para a produção
       adicionar as novas produções na nova gramatica;
   else
       se a produção não pertencer ao conjunto lslanalisadas entao
          adiciona a produção na nova gramatica
       fim se
   fim se
Fim para

Para I = 0 ; I < qtde produções da gramática; incrementa I;
   lsprodução = produção da gramática na posição I;
   se a produção gerar recursão direta então
      lslProdRecursão recebe lsProdução
   else
      lsProdSemRecursão recebe lsProdução
   fim se
Fim para

Para I  = 0; I <  qtde itens ordem NT; incrementa i
   lsNT = NT do conjunto ordem_nt na posição I;
   gerar as novas produções para cada símbolo lsNT;
   adicionar as novas produções na nova gramática
Fim para

Retorno a nova_gramática
```

Figura 50. Algoritmo de Eliminação de Recursão

Pode ser visto com a gramática G = ({S, A}, {a, b, c}, P, S) aonde P = {S→ Aa; A → Sb | cA | a} um exemplo da execução do algoritmo. A primeira etapa

do algoritmo é a criação do conjunto de símbolos não terminais. Esse conjunto fica com os seguintes elementos: S, A. A primeira etapa é a eliminação de recursões indiretas transformando em recursões diretas. O processo do algoritmo é apresentado na Figura 51.

Iteração	Produção	Novas produções	Produções Analisadas
1	S → Aa	A → Aab	S → Aa
2	A → Sb		A → Sb
3	A → cA		A → cA
4	A → a		A → a

Figura 51. Transformação de recursão indireta em direta

A gramática resultante desta etapa é P = {S → Aa; A → Aab | cA | a}. O segundo processo é a eliminação de recursão direta. A execução do algoritmo é apresentado na Figura 52.

Iteração	Produção	Produções com Recursão	Produções sem Recursão
1	S → Aa		S → Aa
2	A → Aab	A → Aab	
3	A → cA		A → cA
4	A → a		A → a

Iteração	Símbolo	Novas Produções
1	S	S → Aa
2	A	A → cAA` A → aA` A` → abA` A` → î

Figura 52. Eliminação de Recursão direta

Ao término da execução do algoritmo, o algoritmo resulta na gramática G = ({S, A, A`}, {a, b, c}, P, S) aonde P = {S → Aa; A → cAA` | aA`; A` → abA` | ε }.

6.2 MODELAGEM

A modelagem do software foi feita com UML, conforme descrito. A modelagem permite visualizar as principais funcionalidades de software, seqüência em que as operações devem ser executadas, entre outros detalhes (LARMAN, 2000). Foram definidos três diagramas de modelo: o diagrama de casos de uso, de seqüência e de atividades.

O diagrama de Casos de Uso (*Use Case*) permite descrever as funcionalidades possíveis de serem executadas no software (FOWLER; SCOTT, 2000). Foi complementado o diagrama de Casos de Uso inicial do GESPAS, adicionando as novas funcionalidades oferecidas pelo software. Na Figura 53 é apresentado este diagrama.

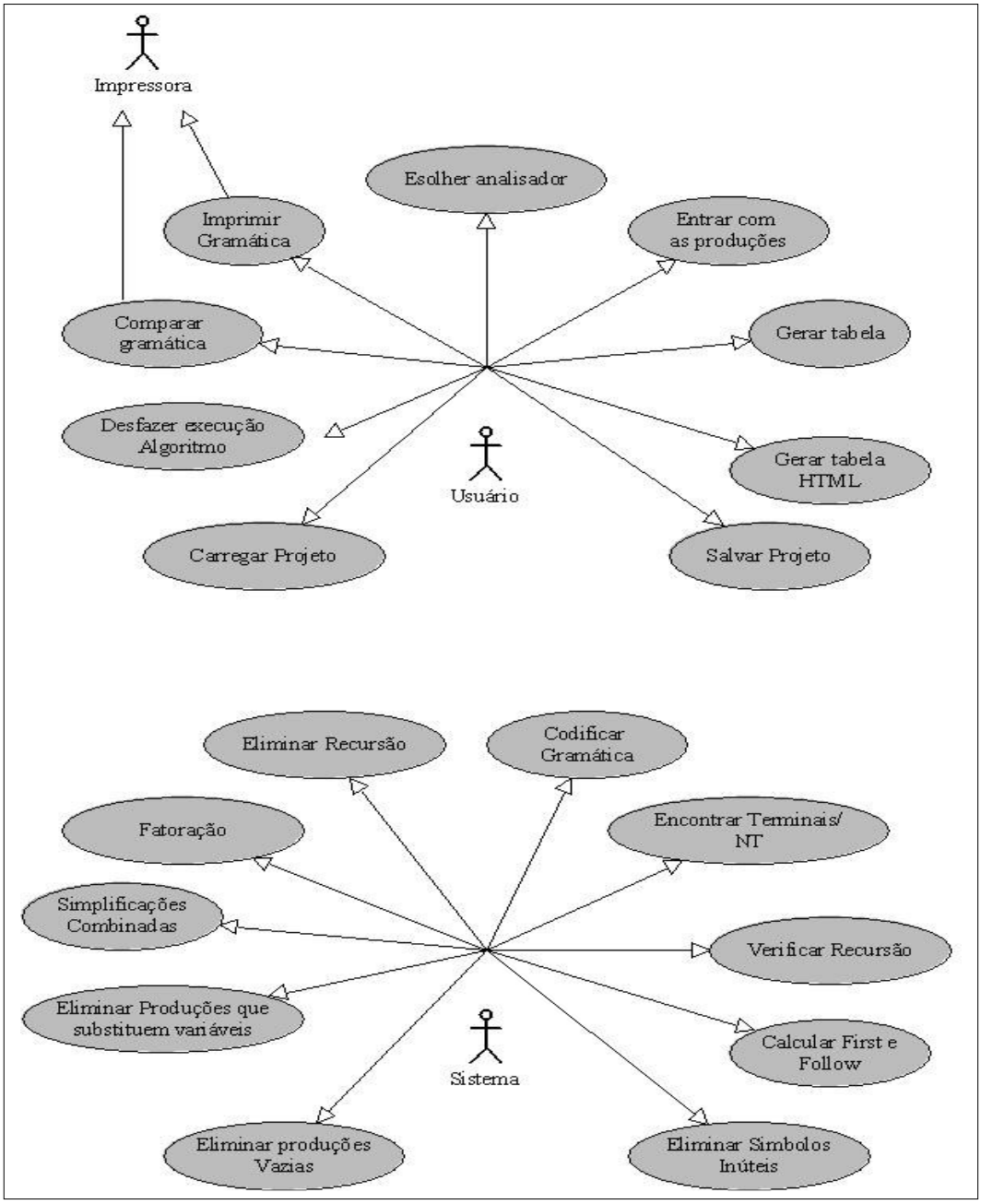


Figura 53. Diagrama de Caso de Uso

Na Figura 54 é apresentado o Diagrama de Seqüência. Este permite uma boa definição do projeto como os relacionamentos necessários, métodos e comportamentos entre as classes. Eles descrevem ao longo de uma linha de tempo a seqüência de comunicação entre os objetos (FOWLER; SCOTT, 2000).

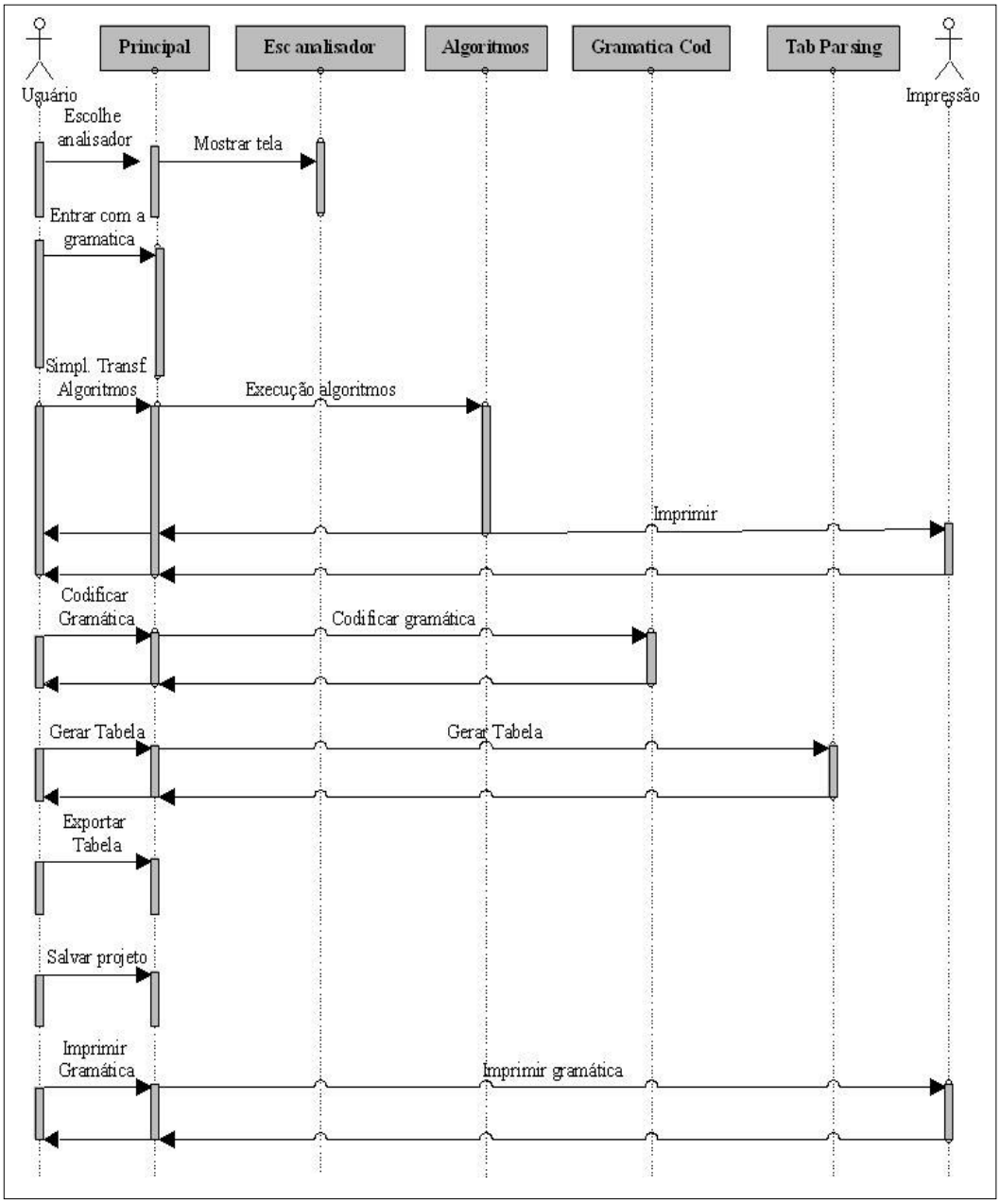


Figura 54. Diagrama de sequência

Foi desenvolvido mais um diagrama especificando o comportamento do sistema, o diagrama de atividades. É responsável por mostrar o complemento das classes em todos os seus casos de uso. Fornece uma descrição global dos processos disponíveis no sistema (FOWLER; SCOTT, 2000). A Figura 55 apresenta o diagrama de atividades.

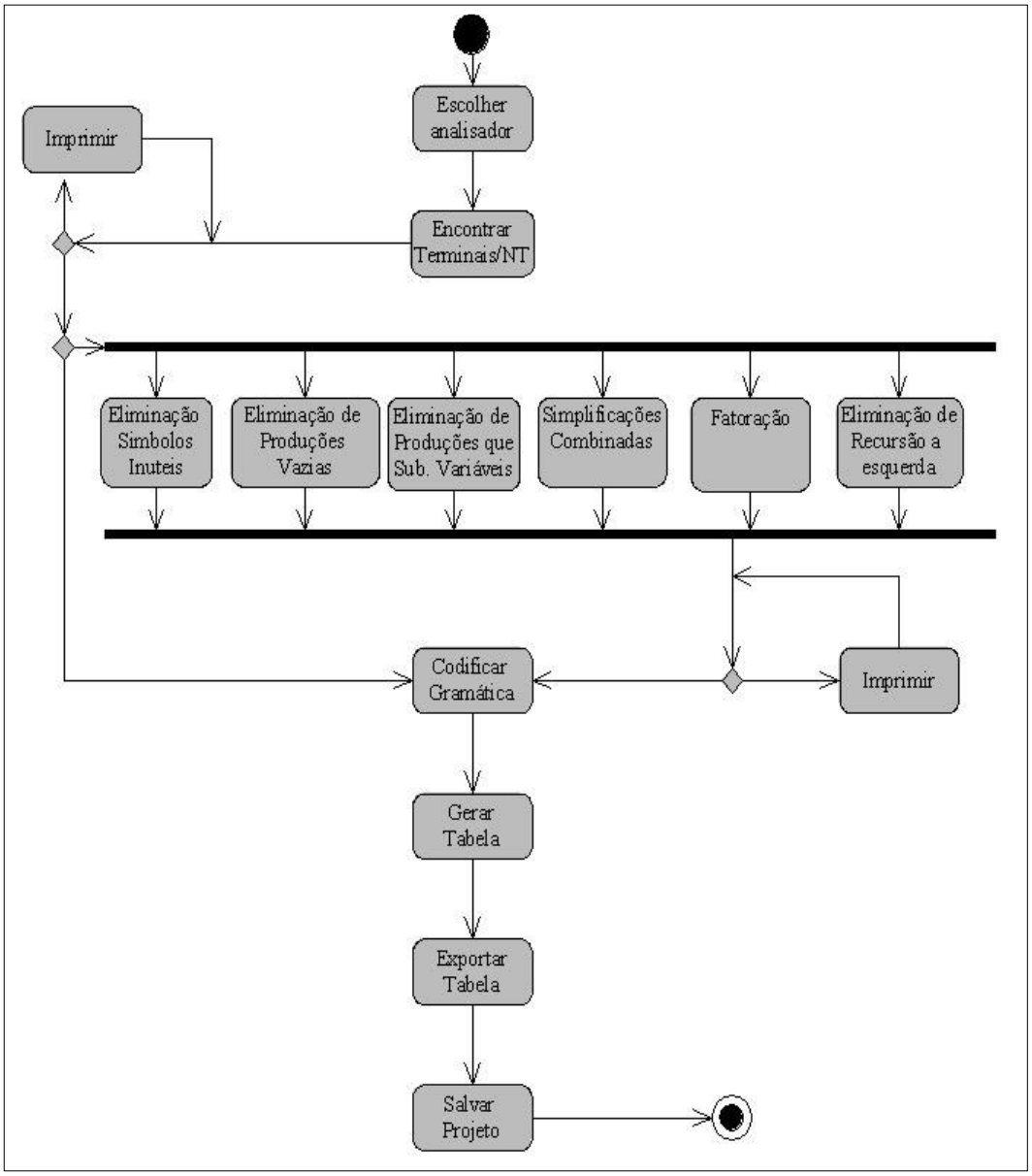


Figura 55. Diagrama de atividades

Após o término da modelagem do software, iniciou-se o processo de implementação, efetuando uma divisão dos algoritmos em classes. Foi criada uma classe para cada algoritmo, contendo os métodos e funções necessárias para o funcionamento. A chamada dos algoritmos é feita na tela principal do GESPAS (Figura 43). Quando a função do algoritmo é carregada, instancia-se uma variável referente a classe do algoritmo, sendo chamado o método da classe. O que difere os métodos da

tela principal o GESPAS é a criação da classe e o nome do método que se está chamando. Na Figura 56 é apresentada a chamada do algoritmo de Eliminação de Símbolos Inúteis.

```
//Criação dos Objetos
TStringList *lslgramatica_alterada   = new TStringList();
TStringList *lslnaoterminais         = new TStringList();
TStringList *lslgramatica            = new TStringList();
TStringList *lslterminais            = new TStringList();
TCursor      Save_Cursor             = Screen->Cursor;
int          i;

fmSplash->ativaSplash();
lslgramatica->AddStrings(redProducoes->Lines );
lslterminais->AddStrings(redTerminais->Lines);
lslnaoterminais->AddStrings(redNaoTerminais->Lines);
redBackup->Lines = redProducoes->Lines;

// criação do objeto algoritmos
EliminacaoSimbolosInuteis *la = new EliminacaoSimbolosInuteis();

Screen->Cursor = crHourGlass;
try
{
   //chamada do algoritmo
   lslgramatica_alterada->AddStrings(la->
      eliminar_simbolos_inuteis(
      lslgramatica,lslterminais, lslnaoterminais));
}
__finally
{
        Screen->Cursor = Save_Cursor;
        fmSplash->desativaSplash();
}
this->redAvisos->Lines->Append("Eliminação de símbolos inúteis:
                               executado com sucesso.");
this->carregargramaticaalterada(redBackup,
      lslgramatica_alterada, "Eliminação de Símbolos inúteis");
this->encontrarElementrosAlgoritmos();
this->mm_desfazer->Enabled = true;
```

Figura 56. Exemplo de chamada dos algoritmos na tela principal

6.3 TESTES

Os testes foram aplicados para cada algoritmo desenvolvido, foram aplicados testes com gramáticas simples disponíveis em livros e também geradas para testes (Apêndice C). Foram utilizadas em média quatro a cinco gramáticas diferentes

para cada algoritmo validando o seu funcionamento. Neste momento, também já foi executado o processo de correção de erros para os algoritmos que apresentaram inconsistências na execução. Os principais erros encontrados foram de lógica aplicada no algoritmo e manipulação de caracteres.

Validando os testes com as gramáticas simples, iniciaram-se os testes com gramáticas mais complexas. Foram coletadas essas gramáticas de livros, exemplos do próprio software e algumas desenvolvidas para testes. O único algoritmo que apresentou erro na execução em algumas gramáticas foi o de Fatoração. O erro encontrado foi de manipulação de caracteres e foi corrigido alterando-se a lógica aplicada no algoritmo. Nos demais algoritmos não ocorreram problemas. As gramáticas utilizadas para testes podem ser vistas no Apêndice D.

6.4 RESULTADOS OBTIDOS

Os resultados obtidos consideram-se satisfatórios, mas para efetivar a sua eficiência devem-se efetuar testes com gramáticas maiores ou disponibilizar para usuários diversificados efetuarem testes com diferentes gramáticas.

Fazendo um resumo geral do software, verifica-se que o software ficou com seus processos de forma simplificada. Possui uma interface simples de ser trabalhada com os principais processos disponíveis na tela e os demais são acessados através de um menu de opções. Enquanto está executando algum algoritmo é apresentado um *feedback* para o usuário através de um circulo girando deixando a real situação do algoritmo. No término da execução do algoritmo é possível comparar ou até imprimir a gramática alterada pelo algoritmo com a original, possibilitando um melhor entendimento do algoritmo por acadêmicos.

Uma funcionalidade adicionada no software que poderá auxiliar os acadêmicos é a impressão da gramática com a separação das produções, os símbolos não terminais e os terminais, permitindo que se identifiquem melhor as palavras geradas pela gramática.

Durante a fase de testes foram encontrados alguns erros na execução dos algoritmos sendo corrigidos nesta etapa. Após as correções foram efetuados os testes novamente e não ocorreram mais erros.

O algoritmo de remoção de ambigüidade não foi implementado no GESPAS, pois segundo Hopcroft, Ullman e Motwani (2002) não existe nenhum algoritmo que defina se uma gramática é ambígua. Algumas gramáticas são totalmente ambíguas impossíveis de se eliminar esta ambigüidade. Para a remoção de ambigüidade é necessário intervenção do usuário definindo a regra de remoção da ambigüidade. Em uma expressão matemática 31 – 2 – 36 não se tem como saber qual operação deve ser executada primeiro se é 31 – 2 ou 2 – 36. E para isso que o usuário deve definir a regra de remoção de ambigüidade.

CONCLUSÃO

O objetivo principal do trabalho somente pode ser alcançado através da total compreensão dos objetivos específicos. Para tanto, foi necessário o estudo de gramáticas, seus tipos, os algoritmos a serem aplicados no software, o entendimento da ferramenta de programação utilizada e conhecimento de programação visual. Durante o desenvolvimento do trabalho, foram divididas algumas etapas para melhor organização, cada uma com sua finalidade. Podem ser citadas como etapas: levantamento bibliográfico, estudo das gramáticas e os algoritmos a serem aplicados, a modelagem do software, seu desenvolvimento, testes entre outras etapas.

Durante a etapa de implementação foi feito um estudo sobre a ferramenta Borland C++ Builder 6, a qual foi utilizada. Pesquisou-se também sobre a linguagem C++ para verificar quais as melhores funções de manipulação de *strings* que poderiam ser aplicadas no software. Por meio desta pesquisa, conseguiu-se chegar a melhor forma de desenvolvimento dos algoritmos utilizando os recursos disponíveis na linguagem.

Somente com todos os objetivos específicos alcançados é que se conseguiu disponibilizar as alterações do protótipo do GESPAS descritas no capítulo 06. As alterações disponibilizadas permitem que usuários possam utilizar gramáticas complexas, não necessitando efetuar transformações de forma manual, quando necessário, suscetíveis a erros. As funcionalidades apresentadas neste trabalho não foram encontradas em nenhum outro software similar, fazendo com que se inicie uma nova linha de pesquisa.

Em resumo, as alterações feitas no software ficaram agrupadas no menu Transformações, com a possibilidade de comparar a gramática alterada e desfazer a execução. Além das alterações propostas foram feitas algumas melhorias no software

como a alteração dos ícones da interface principal, adição de teclas de atalho para as principais funcionalidades e a possibilidade de impressão da gramática.

Durante a produção deste trabalho, muitas dificuldades surgiram. Dificuldade na compreensão dos algoritmos, no entendimento do conteúdo para transformá-lo neste trabalho, a falta de material publicado na área e também a falta de tempo para as pesquisas necessárias. Dentre as dificuldades citadas, a principal foi a de compreensão dos algoritmos. Todas as bibliografias utilizadas representam os algoritmos de forma complexa, através de notação matemática, dificultando o entendimento para a implementação em código C++.

A solução para esta dificuldade foi o entendimento do funcionamento do algoritmo através da verificação de seu funcionamento com gramáticas. Para cada algoritmo foram selecionados exemplos e estudado as suas ações, gerando um script de sua lógica. Após transformou-se para o código do programa. As demais dificuldades foram superadas através de dedicação e pesquisa.

Como trabalho futuro se sugere a expansão do software adicionando a possibilidade da manipulação de outros tipos de gramáticas, o desenvolvimento do software para arquitetura multiplataforma. Hoje o GESPAS está restrito a plataforma Windows, sabendo que outras plataformas estão se destacando no mercado. Pode ser adicionado também os algoritmos de transformação das Formas Normais de *Chomsky* e *Greibach* e a manipulação de Autômatos de Pilha.

REFERÊNCIAS

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores:** princípios, técnicas e ferramentas. Tradução Daniel de Ariosto Pinto. Rio de Janeiro: Livros Técnicos e Científicos, 1995.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML**: guia do usuário. Tradução Fábio Freitas da Silva. Rio de Janeiro: Campus, 2000.

CANTÙ, Marco. **Dominando o delphi 7:** a bíblia. Tradução Kátia Aparecida Roque. São Paulo: Pearson Makron Books, 2005.

DIVERIO, Tiarajú A.; MENEZES, Paulo B. **Teoria da computação**: máquinas universais e computabilidade. 2. ed. Porto Alegre: Sagra Luzzatto, 2004.

FOWLER, Martin; SCOTT, Kendal. **UML essencial**: um breve guia para a linguagem padrão de modelagem de objetos. Tradução Vera Pezerico e Christian Thomas Price. Porto Alegre: Bookman, 2000.

GESSER, Carlos Eduardo. **GALS:** gerador de analisadores léxicos e sintáticos. Florianópolis, 2003. Trabalho de Conclusão do Curso em Ciência da Computação. Universidade Federal de Santa Catarina, 2003. Disponível em < http://gals.sourceforge.net/help.html> Acesso em: 07 mar 2007.

HOPCROFT, John E.; MOTWANI, Rajeev; ULLMAN, Jefrey D. **Introdução à teoria de autômatos, linguagens e computação.** Tradução Vandenberg D. de Souza. Rio de Janeiro: Campus/Elsevier, 2002.

HUBBARD, John R. **Programação em C++.** Tradução de Edson Furmankiewicz. 2. ed. Porto Alegre: Bookman, 2003.

HUBERT, Bert. Lex e YACC primer/HOWTO, Tradução de Tiago Bortoletto Vaz. Disponível em: < http://codigolivre.org.br/projects/lex-yacc/ > Acesso em: 12 nov. 2006.

LARMAN, Craig. **Utilizando UML e padrões:** uma introdução à análise e ao projeto orientados a objetos. Tradução Luiz Augusto Meirelles Salgado. Porto Alegre: Bookman, 2000.

LOUDEN, Kenneth C. **Compiladores:** princípios e práticas. Tradução Flávio Soares Correa da Silva. São Paulo: Pioneira Thomson Learning, 2004.

MARTINS, Fabiano. **GESPAS:** software gerador de estruturas específicas para analisadores sintáticos. Criciúma, 2005. Trabalho de Conclusão do Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, 2005.

MENEZES, Paulo Blauth. **Linguagens formais e autômatos.** 5. ed. Porto Alegre: Sagra Luzzato, 2005.

PRICE, Ana Maria de Alencar; TOSCANI, Simão Sirineo. **Implementação de linguagens de programação:** compiladores. 3. ed. Porto Alegre: Sagra Luzzatto, 2005.

REIS, Carla Alessandra Lima. **Linguagens formais.** Material de aula da disciplina de linguagens formais. Universidade Federal do Pará. Disponível em: <http://www.cultura.ufpa.br/clima/> Acesso em: 14 ago. 2006.

VIEIRA, Newton José. **Introdução aos fundamentos da computação:** linguagens e máquinas. São Paulo: Pioneira Thomson Learning, 2006.

BIBLIOGRAFICA RECOMENDADA

CAMPANI, Carlos A. P. **Disciplina de Linguagens Formais.** Material de aula da disciplina de Linguagens Formais. Universidade Federal de Pelotas. Disponível em: < http://minerva.ufpel.edu.br/~campani/lingformal.pdf> Acesso em: 25 out 2006.

DEITEL, Harvey M.; DEITEL, Paul J. **C++**: como programar. Tradução Carlos Arthur Lang Lisboa e Maria Lúcia Lang Lisboa. Porto Alegre: Bookman, 2001.

HOLLINGWORTH, Jarrod et al. **C++ Builder developer's guide.** Indianápolis: SAMS, 2000.

JAMSA, Kris. **Aprendendo C++.** Tradução Jeremias René D. Pereira dos Santos. São Paulo: Makron Books, 1999.

KERNIGHAN, Brian W.; PIKE Rob. **A prática da programação.** Tradução Kátia Roque. Rio de Janeiro: Campus, 2000.

LIBERTY, Jessé. **Aprenda em 24 horas C++.** Tradução Ana Maria Netto Guz. 2. ed. Rio de Janeiro: Campus, 1998.

MATEUS, César Augusto. **C++ Builder 5:** guia prático. São Paulo: Érica, 2000.

OLINTO, José Varela Furtado. **Linguagens formais e compiladores.** Material de aula da disciplina de Linguagens Formais e Compiladores. Universidade Federal de Santa Catarina. Disponível em: < http://www.inf.ufsc.br/~olinto/apostila-lfc.doc> Acesso em: 25 out. 2006.

PRESSMAN, Roger S. **Engenharia de software.** Tradução José Carlos Barbosa dos Santos. São Paulo: Pearson Makron Books, 1995.

SCHILDT, Herbert. **Borland C++ Builder:** referência completa. Rio de Janeiro: Campus, 2001.

APÊNDICE A – IMPRESSÃO DA COMPARAÇÃO DAS GRAMÁTICAS

Nome projeto: C:\Gespas_novo\Projetos\gramatica com recursao indireta.ges
Algoritmo: Eliminação de recursão a esquerda
Impresso em: 10/11/2007

Gramática Original

<PROGRAMA> ::= 'nome' 'ident' ';' <BLOCO> 'fimprog'
<BLOCO> ::= <DCLCONST> <DCLVAR> <DCLPROC> <CORPO>
<DCLCONST> ::= 'const' 'ident' ':=' <TIPO> ';' <LDCONST>
<DCLCONST> ::= î
<LDCONST> ::= î
<LDCONST> ::= 'const' 'ident' ':=' <TIPO> ';' <LDCONST>
<DCLVAR> ::= 'var' <LID> ':' <TIPO> ';' <LDVAR>
<DCLVAR> ::= î
<LID> ::= 'ident' <REPIDENT>
<REPIDENT> ::= î
<REPIDENT> ::= ',' 'ident' <REPIDENT>
<LDVAR> ::= î
<LDVAR> ::= <LID> ':' <TIPO> ';' <LDVAR>
<TIPO> ::= 'integer'
<TIPO> ::= 'string'
<TIPO> ::= 'real'
<DCLPROC> ::= 'procedure' 'ident' <DEFPAR> ';' '{' <BLOCO> '}' <DCLPROC>
<DCLPROC> ::= î
<DEFPAR> ::= î
<DEFPAR> ::= '(' <LID> ':' 'integer' ')'
<CORPO> ::= 'inicio' <COMANDO> <REPCOMANDO> 'fim'
<COMANDO> ::= 'ident' ':=' <EXPRESSAO>
<COMANDO> ::= 'writeln' '(' <ITEMSAIDA> <REPITEM> ')'
<COMANDO> ::= <CORPO>
<COMANDO> ::= î
<COMANDO> ::= 'if' '(' <EXPRESSAO> ')' 'then' <CORPO> <ELSEPARTE>
<COMANDO> ::= 'while' '(' <EXPRESSAO> ')' 'do' <CORPO>
<COMANDO> ::= 'repeat' <CORPO> 'until' '(' <EXPRESSAO> ')'
<COMANDO> ::= 'readln' '(' 'ident' ')'
<COMANDO> ::= 'for' '(' <EXPRESSAO> ')' 'to' '(' <EXPRESSAO> ')' 'do' <CORPO>
<REPCOMANDO> ::= î
<REPCOMANDO> ::= ';' <COMANDO> <REPCOMANDO>
<ELSEPARTE> ::= î
<ELSEPARTE> ::= 'else' <CORPO>
<ITEMSAIDA> ::= 'literal'
<ITEMSAIDA> ::= <EXPRESSAO>
<REPITEM> ::= î
<REPITEM> ::= ',' <ITEMSAIDA> <REPITEM>
<EXPRESSAO> ::= <EXPSIMP> <REPEXPSIMP>
<REPEXPSIMP> ::= î
<REPEXPSIMP> ::= '=' <EXPSIMP>
<REPEXPSIMP> ::= '<' <EXPSIMP>
<REPEXPSIMP> ::= '>' <EXPSIMP>
<REPEXPSIMP> ::= '>=' <EXPSIMP>
<REPEXPSIMP> ::= '<=' <EXPSIMP>
<REPEXPSIMP> ::= '<>' <EXPSIMP>
<EXPSIMP> ::= '+' <TERMO> <REPEXP>
<EXPSIMP> ::= '-' <TERMO> <REPEXP>
<EXPSIMP> ::= <TERMO> <REPEXP>
<REPEXP> ::= '+' <TERMO> <REPEXP>
<REPEXP> ::= '-' <TERMO> <REPEXP>
<REPEXP> ::= î

<TERMO> ::= <FATOR> <REPTERMO>
<REPTERMO> ::= î
<REPTERMO> ::= '*' <FATOR> <REPTERMO>
<REPTERMO> ::= '/' <FATOR> <REPTERMO>
<FATOR> ::= <TERMO> 'numinteger'
<FATOR> ::= '(' <EXPRESSAO> ')'
<FATOR> ::= 'ident'
<FATOR> ::= 'string'
<FATOR> ::= 'literal'
<FATOR> ::= 'numreal'

Gramática Alterada
<PROGRAMA> ::= 'nome' 'ident' ';' <BLOCO> 'fimprog'
<BLOCO> ::= <DCLCONST> <DCLVAR> <DCLPROC> <CORPO>
<DCLCONST> ::= 'const' 'ident' ':=' <TIPO> ';' <LDCONST>
<DCLCONST> ::= î
<LDCONST> ::= î
<LDCONST> ::= 'const' 'ident' ':=' <TIPO> ';' <LDCONST>
<DCLVAR> ::= 'var' <LID> ':' <TIPO> ';' <LDVAR>
<DCLVAR> ::= î
<LID> ::= 'ident' <REPIDENT>
<REPIDENT> ::= î
<REPIDENT> ::= ',' 'ident' <REPIDENT>
<LDVAR> ::= î
<LDVAR> ::= <LID> ':' <TIPO> ';' <LDVAR>
<TIPO> ::= 'integer'
<TIPO> ::= 'string'
<TIPO> ::= 'real'
<DCLPROC> ::= 'procedure' 'ident' <DEFPAR> ';' '{' <BLOCO> '}' <DCLPROC>
<DCLPROC> ::= î
<DEFPAR> ::= î
<DEFPAR> ::= '(' <LID> ':' 'integer' ')'
<CORPO> ::= 'inicio' <COMANDO> <REPCOMANDO> 'fim'
<COMANDO> ::= 'ident' ':=' <EXPRESSAO>
<COMANDO> ::= 'writeln' '(' <ITEMSAIDA> <REPITEM> ')'
<COMANDO> ::= <CORPO>
<COMANDO> ::= î
<COMANDO> ::= 'if' '(' <EXPRESSAO> ')' 'then' <CORPO> <ELSEPARTE>
<COMANDO> ::= 'while' '(' <EXPRESSAO> ')' 'do' <CORPO>
<COMANDO> ::= 'repeat' <CORPO> 'until' '(' <EXPRESSAO> ')'
<COMANDO> ::= 'readln' '(' 'ident' ')'
<COMANDO> ::= 'for' '(' <EXPRESSAO> ')' 'to' '(' <EXPRESSAO> ')' 'do' <CORPO>
<REPCOMANDO> ::= î
<REPCOMANDO> ::= ';' <COMANDO> <REPCOMANDO>
<ELSEPARTE> ::= î
<ELSEPARTE> ::= 'else' <CORPO>
<ITEMSAIDA> ::= 'literal'
<ITEMSAIDA> ::= <EXPRESSAO>
<REPITEM> ::= î
<REPITEM> ::= ',' <ITEMSAIDA> <REPITEM>
<EXPRESSAO> ::= <EXPSIMP> <REPEXPSIMP>
<REPEXPSIMP> ::= î
<REPEXPSIMP> ::= '=' <EXPSIMP>
<REPEXPSIMP> ::= '<' <EXPSIMP>
<REPEXPSIMP> ::= '>' <EXPSIMP>
<REPEXPSIMP> ::= '>=' <EXPSIMP>
<REPEXPSIMP> ::= '<=' <EXPSIMP>
<REPEXPSIMP> ::= '<>' <EXPSIMP>
<EXPSIMP> ::= '+' <TERMO> <REPEXP>
<EXPSIMP> ::= '-' <TERMO> <REPEXP>

<EXPSIMP> ::= <TERMO> <REPEXP>
<REPEXP> ::= '+' <TERMO> <REPEXP>
<REPEXP> ::= '-' <TERMO> <REPEXP>
<REPEXP> ::= î
<TERMO> ::= <FATOR> <REPTERMO>
<REPTERMO> ::= î
<REPTERMO> ::= '*' <FATOR> <REPTERMO>
<REPTERMO> ::= '/' <FATOR> <REPTERMO>
<FATOR> ::= '(' <EXPRESSAO> ')' <FATOR">
<FATOR> ::= 'ident' <FATOR">
<FATOR> ::= 'string' <FATOR">
<FATOR> ::= 'literal' <FATOR">
<FATOR> ::= 'numreal' <FATOR">
<FATOR"> ::= <REPTERMO> 'numinteger' <FATOR">
<FATOR"> ::= î

APÊNDICE B – IMPRESSÃO DE GRAMÁTICA

Nome projeto: C:\Gespas\Projetos\Gramatica_correta.ges
Impresso em: 16/09/2007

Gramática

<PROGRAMA>::= 'programa' <DECLARACAOVARIAVEIS> 'inicio' <COMANDOS> 'fim'
<DECLARACAOVARIAVEIS> ::= 'var' <LISTAVARIAVEL> ';'
<DECLARACAOVARIAVEIS> ::= î
<COMANDOS> ::= <COMANDOSIMPLES> ';' <COMANDOS>
<COMANDOS> ::= <COMANDOESTRUTURADO> ';' <COMANDOS>
<COMANDOS> ::= î
<COMANDOSIMPLES> ::= <ATRIBUICAO>
<COMANDOSIMPLES> ::= <LEITURA>
<COMANDOSIMPLES> ::= <ESCRITA>
<COMANDOESTRUTURADO> ::= <SELECAO>
<COMANDOESTRUTURADO> ::= <REPETICAO>
<ATRIBUICAO> ::= <VARIAVEL> ':=' <EXPRESSAO>
<LEITURA> ::= 'leia' '(' <LISTAVARIAVEL> ')'
<ESCRITA> ::= 'mostre' '(' <LISTAVARIAVEL> ')'
<SELECAO> ::= 'se' <EXPRESSAO> 'entao' <COMANDOS> <SENAO> 'fimse'
<SENAO> ::= 'senao' <COMANDOS>
<SENAO> ::= î
<REPETICAO> ::= 'enquanto' <EXPRESSAO> 'faca' <COMANDOS> 'fimenquanto'
<LISTAVARIAVEL> ::= <VARIAVEL> <OUTRAVARIAVEL>
<OUTRAVARIAVEL> ::= ',' <VARIAVEL> <OUTRAVARIAVEL>
<OUTRAVARIAVEL> ::= î
<VARIAVEL> ::= 'identificador' <VARIAVELX>
<VARIAVELX> ::= 'inteiro'
<VARIAVELX> ::= 'real'
<EXPRESSAO> ::= <EXPRESSAOARITMETICALOGICA> <EXPRESSAORELACIONAL>
<EXPRESSAORELACIONAL> ::= '=' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '<>' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '<' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '<=' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '>' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '>=' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= î
<EXPRESSAOARITMETICALOGICA> ::= <ELEMENTO> <TERMO>
<ELEMENTO> ::= 'numinteiro'
<ELEMENTO> ::= 'numreal'
<ELEMENTO> ::= <VARIAVEL>
<ELEMENTO> ::= '(' <EXPRESSAO> ')'
<TERMO> ::= '+' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= '-' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= '*' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= '/' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= 'e' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= 'ou' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= î

Não Terminais

<PROGRAMA>
<DECLARACAOVARIAVEIS>
<COMANDOS>
<LISTAVARIAVEL>
<COMANDOSIMPLES>
<COMANDOESTRUTURADO>
<ATRIBUICAO>
<LEITURA>

<ESCRITA>
<SELECAO>
<REPETICAO>
<VARIAVEL>
<EXPRESSAO>
<SENAO>
<OUTRAVARIAVEL>
<VARIAVELX>
<EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL>
<ELEMENTO>
<TERMO>

Terminais
var
senao
se
real
programa
ou
numreal
numinteiro
mostre
leia
inteiro
inicio
identificador
î
fimse
fimenquanto
fim
faca
entao
enquanto
e
>=
>
=
<>
<=
<
+
;
:=
/
,
*
)
(
$
-

APÊNDICE C – GRAMÁTICAS SIMPLES UTILIZADAS PARA TESTES

```
//------------------------------
```
Eliminação de Sìmbolos Inúteis

```
<A> ::= <A> <B> <C>
<A> ::= <A> <E> <F>
<A> ::= <B> <D>
<B> ::= <B> '0'
<B> ::= '0'
<C> ::= '0' <C>
<C> ::= <E> <B>
<D> ::= '1' <D>
<D> ::= '1'
<E> ::= <B> <E>
<F> ::= '1' <F> '1'
<F> ::= '1'

Resultado Gespas -> OK

<A> ::= <B> <D>
<B> ::= <B> '0'
<B> ::= '0'
<D> ::= '1' <D>
<D> ::= '1'

//--
Gramatica 2

<S> ::= 'a' <A> 'a'
<S> ::= 'b' <B> 'b'
<A> ::= 'a'
<A> ::= <S>
<C> ::= 'c'

Resultado Gespas -> OK

<S> ::= 'a' <A> 'a'
<A> ::= 'a'
<A> ::= <S>

//--
Gramatica 3

<S> ::= 'a' <A>
<A> ::= 'a'
<A> ::= 'b' <B>
<B> ::= 'b'
<B> ::= 'd' <D>
<C> ::= 'c' <C>
<C> ::= 'c'
<D: ::= 'd' <D>

Resultado Gespas -> OK

<S> ::= 'a' <A>
<A> ::= 'a'
<A> ::= 'b' <B>
<B> ::= 'b'
```

```
//------------------------------
```
Eliminação de e-produções

```
<S> ::= 'a' <X> 'a'
<S> ::= 'b' <X> 'b'
<S> ::= î
<X> ::= 'a'
<X> ::= 'b'
<X> ::= <Y>
<Y> ::= î

Resultado Gespas -> OK

<S> ::= 'a' <X> 'a'
<S> ::= 'a'  'a'
<S> ::= 'b' <X> 'b'
<S> ::= 'b'  'b'
<S> ::= î
<X> ::= 'a'
<X> ::= 'b'
<X> ::= <Y>

//--
Gramatica 2

<P> ::= <A> <P> <B>
<P> ::= <C>
<A> ::= <A> 'a' 'a' <A>
<A> ::= î
<B> ::= <B> <B> 'b'
<B> ::= <C>
<C> ::= 'c' <C>
<C> ::= î

Resultado Gespas -> OK

<P> ::= <A> <P> <B>
<P> ::=  <P> <B>
<P> ::= <A>  <B>
<P> ::= <A> <P>
<P> ::=   <B>
<P> ::=  <P>
<P> ::= <A>
<P> ::= <C>
<P> ::= î
<A> ::= <A> 'a' 'a' <A>
<A> ::=  'a' 'a' <A>
<A> ::= <A> 'a' 'a'
<A> ::=  'a' 'a'
<B> ::= <B> <B> 'b'
<B> ::=  <B> 'b'
<B> ::=   'b'
<B> ::= <C>
<C> ::= 'c' <C>
<C> ::= 'c'
```

```
Gramatica 3

<S> ::= 'b' <D> <C> 'e'
<D> ::= 'd' <D>
<D> ::= î
<C> ::= 'c' <C>
<C> ::= î

Resultado Gespas -> OK

<S> ::= 'b' <D> <C> 'e'
<S> ::= 'b'  <C> 'e'
<S> ::= 'b' <D>  'e'
<S> ::= 'b'   'e'
<D> ::= 'd' <D>
<D> ::= 'd'
<C> ::= 'c' <C>
<C> ::= 'c'

//--
Gramatica 4

<S> ::= 'a' <B>
<B> ::= 'b' <B>
<B> ::= î

Resultado Gespas -> OK

<S> ::= 'a' <B>
<S> ::= 'a'
<B> ::= 'b' <B>
<B> ::= 'b'

//--
Gramatica 5

<S> ::= 'b' <D> <C> 'e'
<D> ::= 'd' <D>
<D> ::= î
<C> ::= 'c' <C>
<C> ::= î

Resultado Gespas -> OK

<S> ::= 'b' <D> <C> 'e'
<S> ::= 'b'  <C> 'e'
<S> ::= 'b' <D>  'e'
<S> ::= 'b'   'e'
<D> ::= 'd' <D>
<D> ::= 'd'
<C> ::= 'c' <C>
<C> ::= 'c'
```

```
Gramatica 6

<S> ::= 'a' <S>
<S> ::= î

Resultado Gespas -> OK

<S> ::= 'a' <S>
<S> ::= 'a'
<S> ::= î


//----------------------
Eliminaçao de produções
unitárias


<S> ::= 'a' <X> 'a'
<S> ::= 'b' <X> 'b'
<X> ::= 'a'
<X> ::= 'b'
<X> ::= <S>
<X> ::= î

Resultado Gespas -> OK

<S> ::= 'a' <X> 'a'
<S> ::= 'b' <X> 'b'
<X> ::= 'a'
<X> ::= 'b'
<X> ::= î
<X> ::= 'a' <X> 'a'
<X> ::= 'b' <X> 'b'

//--
Gramatica 2

<E> ::= <E> '+' <T>
<E> ::= <T>
<T> ::= <T> '*' <F>
<T> ::= <F>
<F> ::= '(' <E> ')'
<F> ::= 't'

Resultado Gespas -> OK

<E> ::= <E> '+' <T>
<E> ::= <T> '*' <F>
<E> ::= '(' <E> ')'
<E> ::= 't'
<T> ::= <T> '*' <F>
<T> ::= '(' <E> ')'
<T> ::= 't'
<F> ::= '(' <E> ')'
<F> ::= 't'
```

Gramatica 3

```
<E> ::= <T>
<E> ::= <E> '+' <T>
<T> ::= <F>
<T> ::= <T> '*' <F>
<F> ::= <I>
<F> ::= '(' <E> ')'
<I> ::= 'a'
<I> ::= 'b'
<I> ::= <I> 'a'
<I> ::= <I> 'b'
<I> ::= <I> '0'
<I> ::= <I> '1'
```

Resultado Gespas -> OK

```
<E> ::= <E> '+' <T>
<E> ::= <T> '*' <F>
<E> ::= '(' <E> ')'
<E> ::= 'a'
<E> ::= 'b'
<E> ::= <I> 'a'
<E> ::= <I> 'b'
<E> ::= <I> '0'
<E> ::= <I> '1'
<T> ::= <T> '*' <F>
<T> ::= '(' <E> ')'
<T> ::= 'a'
<T> ::= 'b'
<T> ::= <I> 'a'
<T> ::= <I> 'b'
<T> ::= <I> '0'
<T> ::= <I> '1'
<F> ::= '(' <E> ')'
<F> ::= 'a'
<F> ::= 'b'
<F> ::= <I> 'a'
<F> ::= <I> 'b'
<F> ::= <I> '0'
<F> ::= <I> '1'
<I> ::= 'a'
<I> ::= 'b'
<I> ::= <I> 'a'
<I> ::= <I> 'b'
<I> ::= <I> '0'
<I> ::= <I> '1'
```

Gramatica 4

```
<S> ::= 'b' <S>
<S> ::= <A>
<A> ::= 'a' <A>
<A> ::= 'a'
```

Resultado Gespas - OK

```
<S> ::= 'b' <S>
<S> ::= 'a' <A>
<S> ::= 'a'
<A> ::= 'a' <A>
<A> ::= 'a'
```

```
//--
```

Gramatica 5

```
<S> ::= 'a' <S> 'b'
<S> ::= <A>
<A> ::= 'a' <A>
<A> ::= <B>
<B> ::= 'b' <B> 'c'
<B> ::= 'b' 'c'
```

Resultado Gespas - OK

```
<S> ::= 'a' <S> 'b'
<S> ::= 'a' <A>
<S> ::= 'b' <B> 'c'
<S> ::= 'b' 'c'
<A> ::= 'a' <A>
<A> ::= 'b' <B> 'c'
<A> ::= 'b' 'c'
<B> ::= 'b' <B> 'c'
<B> ::= 'b' 'c'
```

```
//------------------------------
```

Simplificações Combinadas

```
<S> ::= 'a' <X> 'a'
<S> ::= 'b' <X> 'b'
<S> ::= î
<X> ::= 'a'
<X> ::= 'b'
<X> ::= <Y>
<Y> ::= î
```

Resultado GESPAS

```
<S> ::= 'a' <X> 'a'
<S> ::= 'a'  'a'
<S> ::= 'b' <X> 'b'
<S> ::= 'b'  'b'
<X> ::= 'a'
<X> ::= 'b'
```

```
//------------------------------
Fatoracao

<S> ::= 'a' <A>
<S> ::= 'a' <B>
<A> ::= 'a' <A>
<A> ::= 'a'
<B> ::= 'b'

resultado Gespas

<S> ::= 'a' <S">
<S"> ::=  <A>
<S"> ::=  <B>
<A> ::= 'a' <A">
<A"> ::=  <A>
<A"> ::= î
<B> ::= 'b'

//--
Gramatica 2

<S> ::= <A> 'b'
<S> ::= 'a' 'b'
<S> ::= 'b' 'a' <A>
<A> ::= 'a' 'a' 'b'
<A> ::= 'b'

Resultado

<S> ::= 'b' <S"">
<S> ::= 'a' <S">
<S""> ::=  'b'
<S""> ::=  'a' <A>
<S"> ::=  'a' 'b' 'b'
<S"> ::=  'b'
<A> ::= 'a' 'a' 'b'
<A> ::= 'b'

//--
Gramatica 3

<S> ::= 'a' <B>
<S> ::= 'a' <C>

resultado

<S> ::= 'a' <S">
<S"> ::=  <B>
<S"> ::=  <C>
```

```
Gramatica 4

<S> ::= 'a' 'a' 'b' 'b'
<S> ::= 'a' 'a' 'c'

resultado

<S> ::= 'a' <S">
<S"> ::= 'a' <S"">
<S""> ::=  'b' 'b'
<S""> ::=  'c'

//--
Gramatica 5

<S> ::= 'b' <A>
<S> ::= 'a' <J>
<S> ::= 'a' <C>
<j> ::= <B> 'b'
<C> ::= 'c'

resultado

<S> ::= 'a' <S">
<S> ::= 'b' <A>
<S"> ::=  <J>
<S"> ::=  <C>
<j> ::= <B> 'b'
<C> ::= 'c'

//--
Gramatica 6

<S> ::= 'a' <J>
<S> ::= 'a' <C>
<S> ::= 'b' <A>
<J> ::= <B> 'b'
<C> ::= 'c'

resultado

<S> ::= 'a' <S">
<S> ::= 'b' <A>
<S"> ::=  <J>
<S"> ::=  <C>
<J> ::= <B> 'b'
<C> ::= 'c'
```

Gramatica 7

<S> ::= 'a' <J>
<S> ::= <C> 'b'
<C> ::= 'a' 'd'
<C> ::= 'b'
<J> ::= <M>
<M> ::= î

resultado

<S> ::= 'a' <S">
<S> ::= 'b' 'b'
<S"> ::= 'd' 'b'
<S"> ::= <J>
<C> ::= 'a' 'd'
<C> ::= 'b'
<J> ::= <M>
<M> ::= î

//------------------------------
Eliminação de recursão
<S> ::= <A> 'a'
<A> ::= <S> 'b'
<A> ::= 'c' <A>
<A> ::= 'a'

Resultado

<S> ::= <A> 'a'
<A> ::= 'c' <A> <A">
<A> ::= 'a' <A">
<A"> ::= 'a' 'b' <A">
<A"> ::= î

Gramatica 1

<S> ::= <A> <A>
<S> ::= 'a'
<A> ::= <S> <S>
<A> ::= 'b'

resultado

<S> ::= <A> <A>
<S> ::= 'a'
<A> ::= 'a' <S> <A">
<A> ::= 'b' <A">
<A"> ::= <A> <S> <A">
<A"> ::= î

Gramática 2
<EXP> ::= <EXP> '+' <TERMO>
<EXP> ::= <EXP> '-' <TERMO>
<EXP> ::= <TERMO>

resultado

<EXP"> ::= '+' <TERMO> <EXP">
<EXP"> ::= '-' <TERMO> <EXP">
<EXP"> ::= î<E"> ::= î

APÊNDICE D – GRAMÁTICAS UTILIZADAS PARA TESTES

Gramática 1

```
<PROGRAMA>::= 'program' 'ident' ';' <BLOCO> '.'
<BLOCO>::= <DCLROT> <DCLCONST> <DCLVAR> <DCLPROC> <CORPO>
<DCLROT>::= 'label' <LID> ';'
<LID>::= 'ident' <REPIDENT>
<REPIDENT>::= î
<REPIDENT>::= ',' 'ident' <REPIDENT>
<DCLROT>::= î
<DCLCONST>::= 'const' 'ident' '=' 'inteiro' ';' <LDCONST>
<LDCONST>::= î
<LDCONST>::= 'ident' '=' 'inteiro' ';' <LDCONST>
<DCLCONST>::= î
<DCLVAR>::= 'var' <LID> ':' <TIPO> ';' <LDVAR>
<LDVAR>::= î
<LDVAR>::= <LID> ':' <TIPO> ';' <LDVAR>
<DCLVAR>::= î
<TIPO>::= 'integer'
<TIPO>::= 'array' '[' 'inteiro' '..' 'inteiro' ']' 'of' 'integer'
<DCLPROC>::= 'procedure' 'ident' <DEFPAR> ':' <BLOCO> ';' <DCLPROC>
<DCLPROC>::= î
<DEFPAR>::= î
<DEFPAR>::= '(' <LID> ':' 'integer' ')'
<CORPO>::= 'begin' <COMANDO> <REPCOMANDO> 'end'
<REPCOMANDO>::= î
<REPCOMANDO>::= ';' <COMANDO> <REPCOMANDO>
<COMANDO>::= 'ident' <RCOMID>
<RCOMID>::= ':' <COMANDO>
<RCOMID>::= <RVAR> ':=' <EXPRESSAO>
<RVAR>::= î
<RVAR>::= '[' <EXPRESSAO> ']'
<COMANDO>::= <CORPO>
<COMANDO>::= î
<COMANDO>::= 'call' 'ident' <PARAMETROS>
<PARAMETROS>::= î
<PARAMETROS>::= '(' <EXPRESSAO> <REPPAR> ')'
<REPPAR>::= î
<REPPAR>::= ',' <EXPRESSAO> <REPPAR>
<COMANDO>::= 'goto' 'ident'
<COMANDO>::= 'if' <EXPRESSAO> 'then' <COMANDO> <ELSEPARTE>
<ELSEPARTE>::= 'else' <COMANDO>
<COMANDO>::= 'while' <EXPRESSAO> 'do' <COMANDO>
<COMANDO>::= 'repeat' <COMANDO> 'until' <EXPRESSAO>
<COMANDO>::= 'readln' '(' <VARIAVEL> <REPVARIAVEL> ')'
<VARIAVEL>::= 'ident' <VARIAVEL1>
<VARIAVEL1>::= î
<VARIAVEL1>::= '[' <EXPRESSAO> ']'
<REPVARIAVEL>::= î
<REPVARIAVEL>::= ',' <VARIAVEL> <REPVARIAVEL>
<COMANDO>::= 'writeln' '(' <ITEMSAIDA> <REPITEM> ')'
<ITEMSAIDA>::= 'literal'
<ITEMSAIDA>::= <EXPRESSAO>
<REPITEM>::= î
<REPITEM>::= ',' <ITEMSAIDA> <REPITEM>
<COMANDO>::= 'case' <EXPRESSAO> 'of' <CONDCASE> 'end'
<CONDCASE>::= 'inteiro' <RPINTEIRO> ':' <COMANDO> <CONTCASE>
<RPINTEIRO>::= ',' 'inteiro' <RPINTEIRO>
<RPINTEIRO>::= î
<CONTCASE>::= î
```

```
<CONTCASE>::= ';' <CONDCASE>
<COMANDO>::= 'for' 'ident' ':=' <EXPRESSAO> 'to' <EXPRESSAO> 'do'
<COMANDO>
<EXPRESSAO>::= <EXPSIMP> <REPEXPSIMP>
<REPEXPSIMP>::= î
<REPEXPSIMP>::= '=' <EXPSIMP>
<REPEXPSIMP>::= '<' <EXPSIMP>
<REPEXPSIMP>::= '>' <EXPSIMP>
<REPEXPSIMP>::= '>=' <EXPSIMP>
<REPEXPSIMP>::= '<=' <EXPSIMP>
<REPEXPSIMP>::= '<>' <EXPSIMP>
<EXPSIMP>::= '+' <TERMO> <REPEXP>
<EXPSIMP>::= '-' <TERMO> <REPEXP>
<EXPSIMP>::= <TERMO> <REPEXP>
<REPEXP>::= '+' <TERMO> <REPEXP>
<REPEXP>::= '-' <TERMO> <REPEXP>
<REPEXP>::= 'or' <TERMO> <REPEXP>
<REPEXP>::= î
<TERMO>::= <FATOR> <REPTERMO>
<REPTERMO>::= î
<REPTERMO>::= '*' <FATOR> <REPTERMO>
<REPTERMO>::= '/' <FATOR> <REPTERMO>
<REPTERMO>::= 'and' <FATOR> <REPTERMO>
<FATOR>::= 'inteiro'
<FATOR>::= '(' <EXPRESSAO> ')'
<FATOR>::= 'not' <FATOR>
<FATOR>::= <VARIAVEL>
```

Gramática 2

```
<PROGRAMA>::= 'program' 'ident' ';' <BLOCO> '.'
<BLOCO>::= <DCLROT> <DCLCONST> <DCLVAR> <DCLPROC> <CORPO>
<DCLROT>::= 'label' <LID> ';'
<LID>::= 'ident' <REPIDENT>
<REPIDENT>::= î
<REPIDENT>::= ',' 'ident' <REPIDENT>
<DCLROT>::= î
<DCLCONST>::= 'const' 'ident' '=' 'inteiro' ';' <LDCONST>
<LDCONST>::= î
<LDCONST>::= 'ident' '=' 'inteiro' ';' <LDCONST>
<DCLCONST>::= î
<DCLVAR>::= 'var' <LID> ':' <TIPO> ';' <LDVAR>
<LDVAR>::= î
<LDVAR>::= <LID> ':' <TIPO> ';' <LDVAR>
<DCLVAR>::= î
<TIPO>::= 'integer'
<TIPO>::= 'array' '[' 'inteiro' '..' 'inteiro' ']' 'of' 'integer'
<DCLPROC>::= 'procedure' 'ident' <DEFPAR> ':' <BLOCO> ';' <DCLPROC>
<DCLPROC>::= î
<DEFPAR>::= î
<DEFPAR>::= '(' <LID> ':' 'integer' ')'
<CORPO>::= 'begin' <COMANDO> <REPCOMANDO> 'end'
<REPCOMANDO>::= î
<REPCOMANDO>::= ';' <COMANDO> <REPCOMANDO>
<COMANDO>::= 'ident' <RCOMID>
<RCOMID>::= ':' <COMANDO>
<RCOMID>::= <RVAR> ':=' <EXPRESSAO>
<RVAR>::= î
<RVAR>::= '[' <EXPRESSAO> ']'
<COMANDO>::= <CORPO>
<COMANDO>::= î
```

```
<COMANDO>::= 'call' 'ident' <PARAMETROS>
<PARAMETROS>::= î
<PARAMETROS>::= '(' <EXPRESSAO> <REPPAR> ')'
<REPPAR>::= î
<REPPAR>::= ',' <EXPRESSAO> <REPPAR>
<COMANDO>::= 'goto' 'ident'
<COMANDO>::= 'if' <EXPRESSAO> 'then' <COMANDO> <ELSEPARTE>
<ELSEPARTE>::= 'else' <COMANDO>
<COMANDO>::= 'while' <EXPRESSAO> 'do' <COMANDO>
<COMANDO>::= 'repeat' <COMANDO> 'until' <EXPRESSAO>
<COMANDO>::= 'readln' '(' <VARIAVEL> <REPVARIAVEL> ')'
<VARIAVEL>::= 'ident' <VARIAVEL1>
<VARIAVEL1>::= î
<VARIAVEL1>::= '[' <EXPRESSAO> ']'
<REPVARIAVEL>::= î
<REPVARIAVEL>::= ',' <VARIAVEL> <REPVARIAVEL>
<COMANDO>::= 'writeln' '(' <ITEMSAIDA> <REPITEM> ')'
<ITEMSAIDA>::= 'literal'
<ITEMSAIDA>::= <EXPRESSAO>
<REPITEM>::= î
<REPITEM>::= ',' <ITEMSAIDA> <REPITEM>
<COMANDO>::= 'case' <EXPRESSAO> 'of' <CONDCASE> 'end'
<CONDCASE>::= 'inteiro' <RPINTEIRO> ':' <COMANDO> <CONTCASE>
<RPINTEIRO>::= ',' 'inteiro' <RPINTEIRO>
<RPINTEIRO>::= î
<CONTCASE>::= î
<CONTCASE>::= ';' <CONDCASE>
<COMANDO>::= 'for' 'ident' ':=' <EXPRESSAO> 'to' <EXPRESSAO> 'do'
<COMANDO>
<EXPRESSAO>::= <EXPSIMP> <REPEXPSIMP>
<REPEXPSIMP>::= î
<REPEXPSIMP>::= '=' <EXPSIMP>
<REPEXPSIMP>::= '<' <EXPSIMP>
<REPEXPSIMP>::= '>' <EXPSIMP>
<REPEXPSIMP>::= '>=' <EXPSIMP>
<REPEXPSIMP>::= '<=' <EXPSIMP>
<REPEXPSIMP>::= '<>' <EXPSIMP>
<EXPSIMP>::= '+' <TERMO> <REPEXP>
<EXPSIMP>::= '-' <TERMO> <REPEXP>
<EXPSIMP>::= <TERMO> <REPEXP>
<REPEXP>::= '+' <TERMO> <REPEXP>
<REPEXP>::= '-' <TERMO> <REPEXP>
<REPEXP>::= 'or' <TERMO> <REPEXP>
<REPEXP>::= î
<TERMO>::= <FATOR> <REPTERMO>
<REPTERMO>::= î
<REPTERMO>::= '*' <FATOR> <REPTERMO>
<REPTERMO>::= '/' <FATOR> <REPTERMO>
<REPTERMO>::= 'and' <FATOR> <REPTERMO>
<FATOR>::= 'inteiro'
<FATOR>::= '(' <EXPRESSAO> ')'
<FATOR>::= 'not' <FATOR>
<FATOR>::= <VARIAVEL>
```

Gramática 3

```
<PROGRAMA>::= 'programa' <DECLARACAOVARIAVEIS> 'inicio' <COMANDOS>
'fim'
<DECLARACAOVARIAVEIS> ::= 'var' <LISTAVARIAVEL> ';'
<DECLARACAOVARIAVEIS> ::= î
<COMANDOS> ::= <COMANDOSIMPLES> ';' <COMANDOS>
```

```
<COMANDOS> ::= <COMANDOESTRUTURADO> ';' <COMANDOS>
<COMANDOS> ::= î
<COMANDOSIMPLES> ::= <ATRIBUICAO>
<COMANDOSIMPLES> ::= <LEITURA>
<COMANDOSIMPLES> ::= <ESCRITA>
<COMANDOESTRUTURADO> ::= <SELECAO>
<COMANDOESTRUTURADO> ::= <REPETICAO>
<ATRIBUICAO> ::= <VARIAVEL> ':=' <EXPRESSAO>
<LEITURA> ::= 'leia' '(' <LISTAVARIAVEL> ')'
<ESCRITA> ::= 'mostre' '(' <LISTAVARIAVEL> ')'
<SELECAO> ::= 'se' <EXPRESSAO> 'entao' <COMANDOS> <SENAO> 'fimse'
<SENAO> ::= 'senao' <COMANDOS>
<SENAO> ::= î
<REPETICAO> ::= 'enquanto' <EXPRESSAO> 'faca' <COMANDOS> 'fimenquanto'
<LISTAVARIAVEL> ::= <VARIAVEL> <OUTRAVARIAVEL>
<OUTRAVARIAVEL> ::= ',' <VARIAVEL> <OUTRAVARIAVEL>
<OUTRAVARIAVEL> ::= î
<VARIAVEL> ::= 'identificador' <VARIAVELX>
<VARIAVELX> ::= 'inteiro'
<VARIAVELX> ::= 'real'
<EXPRESSAO> ::= <EXPRESSAOARITMETICALOGICA> <EXPRESSAORELACIONAL>
<EXPRESSAORELACIONAL> ::= '=' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '<>' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '<' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '<=' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '>' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= '>=' <EXPRESSAOARITMETICALOGICA>
<EXPRESSAORELACIONAL> ::= î
<EXPRESSAOARITMETICALOGICA> ::= <ELEMENTO> <TERMO>
<ELEMENTO> ::= 'numinteiro'
<ELEMENTO> ::= 'numreal'
<ELEMENTO> ::= <VARIAVEL>
<ELEMENTO> ::= '(' <EXPRESSAO> ')'
<TERMO> ::= '+' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= '-' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= '*' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= '/' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= 'e' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= 'ou' <EXPRESSAOARITMETICALOGICA>
<TERMO> ::= î
```

APÊNDICE E – ARTIGO

Gramáticas Livres de Contexto: Simplificação e Transformação implementas no GESPAS

Fábio de Stefani [1], Christine Vieira Scarpato [1]

[1] Universidade do Extremo Sul Catarinense (UNESC)
Caixa Postal 3167 – CEP. 88806-000 – Criciúma, SC, Brasil

fabiostefani@gmail.com, cvi@unesc.net

Abstract: This paper describes a developed research that it gives continuity to a software called GESPAS, where the functionalities of Simplification and Transformations of Free Grammar of Context were added. The goal of the software modification is to help academics and professionals in this area, in the process of compilers development and in the study of the Formal Languages too.

Resumo: Este artigo descreve a pesquisa desenvolvida que dá continuidade ao software denominado GESPAS, em que foram adicionado as funcionalidades de Simplificação e Transformações de Gramática Livre de Contexto. As alterações efetuadas no software tem por objetivo auxiliar acadêmicos e profissionais da área no processo de desenvolvimento de compiladores e também no estudo das Linguagens Formais.

1. Introdução

O GESPAS foi desenvolvido por Fabiano Martins, como trabalho de Conclusão de Curso do Ciência da Computação na Universidade do Extremo Sul Catarinense (UNESC). O software tem como objetivo a geração da tabela de análise sintática para os analisadores ascendentes utilizando a técnica SLR e para o descendente preditivo tabular. É possível gerar a tabela e a gramática para as Linguagens de programação C++, Java, Delphi e Pascal. Anteriormente o software não efetuava nenhuma verificação, simplificação ou transformação na gramática deixando o processo mais complexo se fosse necessário alguma alteração pois esta deveria ser feita manualmente.

Para solucionar este problema foi adicionado ao GESPAS os algoritmos de Eliminação de Símbolos Inúteis, Eliminação de Produções Vazias, Eliminação de Produções que Substituem Variáveis (Produções Unitárias), Simplificações Combinadas, Fatoração e Eliminação de Recursão à Esquerda.

Este artigo apresenta as alterações efetuadas no software, essas alterações visam a automação do processo de simplificação e transformação das Gramáticas Livres de Contexto.

2. Gramáticas

As gramáticas são utilizadas para representar as linguagens. Uma linguagem é constituída de um conjunto de palavras sobre um alfabeto. A linguagem também é

considerada um meio de comunicação formado por palavras e regras (DIVERIO; MENEZES, 2004). Pode ser citado como exemplo de linguagens: o idioma de uma região, linguagens de programação, protocolo de comunicação de redes, entre outros.

Segundo Menezes (2005) as gramáticas são constituídas por um conjunto finito de produções e sua representação formal é da seguinte forma G = (N, T, P, S) onde:

N = conjunto finito de variáveis ou símbolos não terminais;
T = conjunto finito de terminais (símbolos do alfabeto);
P = produções ou regras gramaticais;
S = símbolo não terminal que inicia as produções.

A produção é um conjunto finito de pares de símbolos do alfabeto denominada regra da produção. Sua representação é da seguinte forma A ::= B ou $A \rightarrow B$ onde é constituído de símbolos não terminais ou símbolos terminais ou a junção de ambos. A produção é lida da seguinte forma: A é definido por B. Uma outra característica das produções é que são consideradas o elemento fundamental de uma gramática (VIEIRA, 2006).

Quando aplica-se sucessivamente as produções em uma palavra é chamada de derivação. A derivação é uma operação de substistuição efetuada de acordo com as produções. Toda derivação deve iniciar a partir do símbolo inicial da gramática (MENEZES, 2005). Na Figura 1 é apresentado um exemplo de gramática com suas produções.

```
N = {X, D}
T = {0,1,2,3,4,5,6,7,8,9}
P = {X → D | DX,
     D → 0|1|2|3|4|5|7|8|9}
S =  X
```

Figura 01. Exemplo de Gramática

Segundo a Hierarquia de Chomsky as gramáticas podem ser divididas em quatros tipos: Tipo 3 ou Regular; Tipo 2 ou Livre de Contexto; Tipo 1 ou Sensível ao Contexto; Tipo 0 ou Irrestrita. Cada tipo de gramática possui restrições especificas na forma de suas produções (MENEZES, 2005). Nesta pesquisa foi trabalhado com a gramática Tipo 2 ou Gramática Livre de Contexto.

A terminologia livre de contexto é resultado de ser a classe mais geral entre as linguagens. Possui esta definição porque a derivação não depende de qualquer análise de símbolos que vêm antes ou depois do não terminal da palavra que se está derivando (MENEZES, 2005).

As gramáticas livres de contexto tem uma contribuição importante no desenvolvimento de compiladores, elas deixaram esse processo mais rápido. As gramáticas livres de contexto estão sendo aplicadas em documentos de arquivo para tráfego na internet (HOPCROFT; ULLMAN; MOTWANI, 2002).

3. GESPAS

O GESPAS, conforme já apresentado, gera a tabela de análise sintática para o Descendente Preditivo Tabular e para o Ascendente utilizando a ténica SLR. O GESPAS anteriormente não efetuava nenhuma alteração ou verificação nas gramáticas

trabalhadas por ele. Com as alterações efetuadas no software descritas neste trabalho, ele já efetua algumas alterações e verificações antes de gerar a tabela de análise sintática.

Inicialmente o software foi desenvolvido com a ferramenta Borland C++ Builder 4 dando-se continuidade nesta etapa com uma versão mais atual da ferramenta, a Borland C++ Builder 6.

A alteração no software consiste da adição dos algoritmos de Simplificação e Transformação de Gramáticas Livres de Contexto e algumas verificações na gramática no momento que se está executando os processos de encontrar os terminais e não terminais. Os algoritmos adicionados no software foram os de Eliminação de Símbolos Inúteis, Eliminação de Produções Vazias, Eliminação de Produções que Substituem Variáveis (Produções Unitárias), Simplificações Combinadas, Fatoração e Eliminação de Recursão à Esquerda.

O desenvolvimento do trabalho deu-se dividindo o processo em algumas etapas: primeiramente foram desenvolvidas as classes dos algoritmos no software, após foi feita a modelagem do software verificando qual a melhor forma de serem executados os algoritmos e por final a realização dos testes finais.

4. Algoritmos utilizados na implementação

A lógica de cada algoritmo foi desenvolvida analisando exemplos de gramática e também de estudo dos algoritmos publicados em livros e apostilas de aula. Para o desenvolvimento dos algoritmos trabalhou-se praticamente com a classe *TStringList*[7], manipulação de *Strings*[8], matrizes e vetores e estruturas de repetição.

Também adotou-se alguns padrões de programação onde as variáveis locais são declaradas com a letra ‘l’ na frente de seu nome e também é colocado a inicial do tipo de dado após a letra do local de declaração, por exemplo uma variável do tipo inteiro local ficaria ‘llvar’. Para variáveis globais, é colocado a letra ‘i’ de instância, as variáveis passadas como argumentos é colocado a letra ‘a’ de argumento e assim para todos os locais que são utilizados para declarar as variáveis.

4.1 Eliminação de Símbolos Inúteis

A classe de eliminação de símbolos inúteis é composta de uma função principal que é carregada a partir da tela principal do GESPAS. Esta função utiliza como argumentos a gramática a ser alterada, o conjunto de terminais e o de não terminais retornando a gramática alterada. Esta classe também possui algumas funções auxiliares que são utilizadas para a eliminação de símbolos inúteis. O algoritmo foi dividido em duas etapas. A primeira é a verificação se a produção é geradora de terminais e a segunda verifica se os símbolos não terminais do cabeçalho da produção são atingíveis a partir do símbolo inicial.

Na primeira etapa é feito um laço de repetição até que não ocorra mais alteração na gramática. Dentro desse laço é feito um outro nas produções da gramática verificando se os símbolos não terminais de cada produção geram terminais de forma direta. Depois de feita a verificação da geração de forma direta, é verificada a geração

[7] Classe composta por uma lista de *Strings* (CANTÙ, 2003).
[8] Tipo de dado utilizado para armazenar caracteres (HUBBARD, 2003).

de terminais de forma indireta. Para ser feita essa verificação foi criado um conjunto[9] de símbolos que inicialmente recebe os terminais da gramática. Caso a produção satisfazer essa regra, é adicionado ao conjunto de símbolos o não terminal do cabeçalho da produção e marcado essa produção para ser excluída. Essa marcação é feita adicionando o código da gramática em um vetor. Ao término do laço de repetição das produções são apagadas as produções marcadas e, verificado se necessita fazer o laço novamente. Permanece nesse laço até que não ocorra mais alteração na gramática. Terminado esses laços, é gerada uma gramática somente com as produções que os símbolos pertençam ao conjunto de símbolos gerados no laço anteriormente. Nesta etapa, já se exclui as produções que não são geradoras.

Quando terminar a primeira etapa, o algoritmo passa para a segunda parte, onde é verificado se os símbolos não terminais do cabeçalho da produção são atingíveis a partir do símbolo inicial. É feito um laço de repetição até que não se tenha mais alteração na gramática e dentro desse laço é feito um outro nas produções da gramática. É verificado para cada produção se o símbolo não terminal do cabeçalho da produção é atingível a partir do símbolo inicial. Quando a produção satisfizer esse requisito, é adicionado o símbolo não terminal em um conjunto. Terminado esse laço é gerada uma gramática, contendo somente as produções que possuam os símbolos não terminais separados no processo anterior e retorna essa gramática para ser carregada no GESPAS.

Um exemplo do algoritmo é apresentado com a gramática G = ({S, A, B, C}, {a, b, c}, P, S) onde P = {S → aAa | bBb; A → a | S; C → c}. Inicialmente o algoritmo inicia o conjunto dos símbolos com os símbolos terminais da gramática e inicia o conjunto de símbolos não terminais com vazio. O algoritmo inicia os laços verificando para cada produção da gramática se os símbolos do lado direito da produção pertencem ao conjunto dos símbolos. Ainda executando a primeira etapa do algoritmo são adicionadas em uma nova gramática somente as produções onde todos os seus símbolos pertençam ao conjunto dos símbolos. As produções resultantes desta etapa são as seguintes: P = {A → aAa; A → a | S; C → c}. A produção S → bBb foi excluída pois o símbolo ‘b’ não é gerado de forma direta por nenhuma produção.

Na segunda etapa do algoritmo é verificado para cada produção se o símbolo não terminal do cabeçalho da produção é atingível a partir do símbolo inicial. O conjunto de símbolos não terminais é inicializado com o símbolo não terminal do cabeçalho da primeira produção da gramática. Ao término desta etapa, é gerada uma nova gramática somente com as produções que o símbolo não terminal do cabeçalho estiver contido no conjunto gerado anteriormente. Nesta etapa as produções resultantes são P = {S → aAa; A → a | S}. A produção C → c foi excluída, pois não era atingível a partir do símbolo inicial.

4.2. Eliminação de Produções Vazias

O segundo algoritmo a ser desenvolvido foi o de Eliminação de Produções Vazias. A classe deste algoritmo é composta por um método que é acessado através da tela principal do software. Este método recebe como parâmetros a gramática, o conjunto de não terminais e o de terminais retornando a gramática alterada. É composta também por métodos e funções auxiliares no processo de eliminação. O correto funcionamento do algoritmo dá-se dividindo o seu processo em duas etapas. Na primeira etapa, é

[9] Lista de *Strings*

verificado se as produções da gramática geram a palavra vazia de forma direta ou de forma indireta, e na segunda são geradas as novas produções paras as que produzem a palavra vazia.

O funcionamento da primeira etapa ocorre através de um laço de repetição, até que não ocorra mais alteração na gramática e dentro desse laço é feito um outro percorrendo as produções da gramática. Para cada produção é verificado se a mesma gera a palavra vazia de forma direta ($A \rightarrow \varepsilon$) ou de forma indireta ($A \rightarrow B$; $B \rightarrow \varepsilon$). Quando a produção satisfizer essa característica, é adicionado o símbolo não terminal do cabeçalho da produção em um conjunto e é marcada essa produção para ser excluída. Quando terminar o laço das produções, o sistema exclui as produções que foram marcadas e verifica se precisa iniciar novamente o processo.

A segunda etapa do algoritmo também constitui de um laço nas produções, onde são geradas as novas produções para aquelas que geram a palavra vazia. As produções da gramática original, excetuando as que geram a palavra vazia e as novas produções são adicionadas em uma outra gramática que é retornada para a interface principal do software.

Com a gramática G = ({S, X, Y}, {a, b}, P, S) onde P = {$S \rightarrow$ aXa | bXb | ε, $X \rightarrow$ a | b | Y; $Y \rightarrow \varepsilon$ } é possível apresentar um exemplo do algoritmo. Na primeira etapa do algoritmo é feita uma verificação para cada produção se é gerada a produção vazia. As produções que são geradoras são marcadas para serem apagadas.

Na terceira iteração do laço externo, o algoritmo termina a sua primeira etapa, pois não ocorreram mais modificações no conjunto de símbolos vazios. Após é feito a verificação se a gramática gera a palavra vazia. Para esta gramática a palavra vazia é gerada através do símbolo inicial com a produção $S \rightarrow \varepsilon$. Na segunda etapa do algoritmo são geradas as novas produções excluindo os símbolos geradores de palavra vazia. A gramática resultante desta etapa é P = {$S \rightarrow$ aXa | aa | bXb | bb; $X \rightarrow$ a | b | Y}. A gramática inicialmente gerava a palavra vazia, portanto deverá ser adicionada a produção geradora da palavra vazia para o símbolo inicial da gramática. A gramática resultante deste algoritmo é G = ({S, X, Y}, {a, b}, P, S) aonde P = {$S \rightarrow$ aXa | aa | bXb | bb | ε; $X \rightarrow$ a | b | Y }.

4.3. Eliminação de Produções que Substituem Variáveis

O algoritmo de Eliminação de Produções que Substituem Variáveis foi o terceiro a ser implementado no GESPAS. Como todas as outras classes dos algoritmos, esta é composta de uma função principal e de funções auxiliares. A função principal recebe como argumentos a gramática, o conjunto de terminais e o de não terminais, retornando a gramática com as alterações. Esta função foi dividida em dois processos.

A primeira faz um laço de repetição nas produções da gramática, verificando se ela gera produção unitária. Se gerar, é calculada a seqüência de símbolos não terminais utilizados para se chegar à produção em questão e armazenado em uma matriz. A produção que é geradora de produção unitária é marcada para ser excluída. Terminado esse laço, o algoritmo exclui as produções marcadas ficando somente as produções que não geram produções unitárias na gramática.

O segundo processo é feito também através de um laço nas produções resultantes e adicionando-as em uma nova gramática. Neste ponto também é gerada as

novas produções relacionadas a produção unitária. Esta geração constitui-se da adição de todas as produções dos símbolos não terminais calculados na primeira etapa que está na matriz para o símbolo não terminal do cabeçalho da produção que se está adicionando. Terminado esta etapa, é retornada a gramática que é carregada no GESPAS para se dar continuidade aos trabalhos com o software.

Utilizando a gramática G = ({S, X}, {a, b}, P, S) aonde P = {S → aXa | bXb; X → a | b | S | ε } pode-se aplicar um exemplo do algoritmo. Executando a primeira etapa do algoritmo é gerada a matriz de símbolos utilizados para a geração das novas produções.

Ao término desta etapa, são excluídas as produções marcadas. Após é gerada uma nova gramática com as produções da gramática e também são inseridas as novas produções geradas através do fecho transitivo calculado. No final da execução do algoritmo, a gramática possui como resultado as seguintes produções P = (S → aXa | bXb; X → a | b | ε | aXa | bXb). As poduções X → aXa | bXb foram inseridas para o símbolo X, pois o X possuí uma produção unitária e o seu fecho transitivo é S.

4.4. Simplificações Combinadas

O desenvolvimento do algoritmo de Simplificações Combinadas é realizado efetuando-se a chamada dos três métodos descritos anteriormente na seguinte ordem: primeiro é executado o algoritmo de Eliminação de Produções Vazias, após o de Eliminação de Produções que Substituem Variáveis e por último a Eliminação de Símbolos Inúteis. Como nas classes dos outros algoritmos, esta possui um método principal que é acessado através da interface principal do software, passando como argumento a gramática e os conjuntos de símbolos terminais e não terminais. Ao término da execução desses algoritmos é retornada a gramática alterada que é carregada na tela principal do GESPAS.

4.5. Fatoração

Dando continuidade ao desenvolvimento do software, trabalhou-se com o algoritmo de Fatoração. Esse algoritmo foi desenvolvido em uma classe composta de uma função principal e funções auxiliares a principal. A lógica aplicada no algoritmo é feita da seguinte forma: inicia-se um laço de repetição parando quando não ocorrer mais alterações na gramática. Dentro desse laço é feito um segundo nas produções da gramática. É analisado para cada produção se a mesma é geradora de não determinismo na gramática. As produções que são geradoras são marcadas em uma matriz composta do símbolo não terminal do cabeçalho da produção e do código das produções da gramática.

Terminado o primeiro laço é feito outro na matriz gerada separando o símbolo não terminal e o código das produções que são geradoras de indeterminismo adicionando os códigos em um vetor. Nesta etapa, o sistema gera as novas produções retirando o indeterminismo. As produções geradas são adicionadas em uma gramática nova e, no final desse laço é comparada a quantidade de produções com a gramática anterior. Se for diferente, repete todo o processo novamente. Quando terminar definitivamente esse processo, a gramática é retornada da função que é carregada no GESPAS.

Com a gramática G = ({S, A, B}, {a, b}, P, S) aonde P = (S → aA | aB; A → aA | a; B → b) pode-se apresentar um exemplo do algoritmo. O algoritmo em sua execução vai verificando se as produções estão fatoradas e já vai gerando as novas produções eliminando a fatoração. A gramática resultante após a execução é G = ({S, S`, A, A`, B}, {a, b}, P, S) aonde P = {S → aS`; S`→ A | B; A → aA`; A` → A | ε; B → b}.

4.6. Eliminação de Recursão à Esquerda

O último algoritmo a ser desenvolvido foi o de eliminação de Recursão à Esquerda. A primeira parte do algoritmo é a separação dos símbolos não terminais definidos no cabeçalho da produção e adicionando-os em um conjunto. Após é feito um laço nas produções da gramática verificando se ela gera recursão indireta. Se a produção gerar recursão, são geradas as novas produções e adicionado em uma nova gramática. As produções já analisadas também são adicionadas em um conjunto.

Quando terminar esse processo, é feito um outro laço nas produções da nova gramática, verificando se é gerada a recursão direta. Caso exista recursão direta, esta produção é armazenada em um conjunto de produções que só geram recursão. As produções que não geram recursão são armazenadas em um outro conjunto. A próxima etapa é composta de um laço nos símbolos não terminais da gramática geradas no primeiro processo do algoritmo e são geradas as novas produções sem recursão e gera uma nova gramática. É adicionado inicialmente na nova gramática as produções que não geram recursão e após as produções novas geradas pela eliminação de recursão. Terminado todos esses processos, é retornada a gramática para ser carregado na tela do GESPAS e dado continuidade aos processos normais do sistema.

Pode ser visto com a gramática G = ({S, A}, {a, b, c}, P, S) aonde P = {S→ Aa; A → Sb | cA | a} um exemplo da execução do algoritmo. A primeira etapa do algoritmo é a criação do conjunto de símbolos não terminais. Esse conjunto fica com os seguintes elementos: S, A. A primeira etapa é a eliminação de recursões indiretas transformando em recursões diretas.

A gramática resultante desta etapa é P = {S → Aa; A → Aab | cA | a}. O segundo processo é a eliminação de recursão direta. Ao término da execução do algoritmo, resulta a gramática G = ({S, A, A`}, {a, b, c}, P, S) aonde P = {S → Aa; A → cAA` | aA`; A` → abA` | ε }.

5. Modelagem

A modelagem do software foi feita com UML. Esta permite visualizar as principais funcionalidades do software, as seqüências de execução dos processos, entre outros detalhes. A modelagem no software foi utilizada para definir qual a melhor forma de chamada dos algoritmos, se é feita de forma automática ou disponibilizado o processo para o usuário selecionar. Na Figura 2 é apresentado a interface do software.

Deve ser efetuada a entrada da gramática no software da seguinte forma: os símbolos não terminais devem estar entre sinal de maior e menor (‘<’ e ‘>’), exemplo <LISTA> e os símbolos terminais entre aspas (‘ ‘ ’), exemplo ‘simbolos’. A separação do lado direito de uma produção e do lado esquerdo deve ser feita através do conjunto de símbolos ‘::=’, um exemplo de produção carregada no software por ser ‘<A> ::= <B> |‘a’’. Na figura 2 também pode ser visualizada uma gramática carregada

no software. A interface do software possui 4 (quatro) partes, cada uma com uma finalidade específica. Estas partes estão destinadas a inserção da gramática no software, a avisos a serem apresentados para o usuário, para a relação de símbolos terminais e uma para os símbolos não terminais.

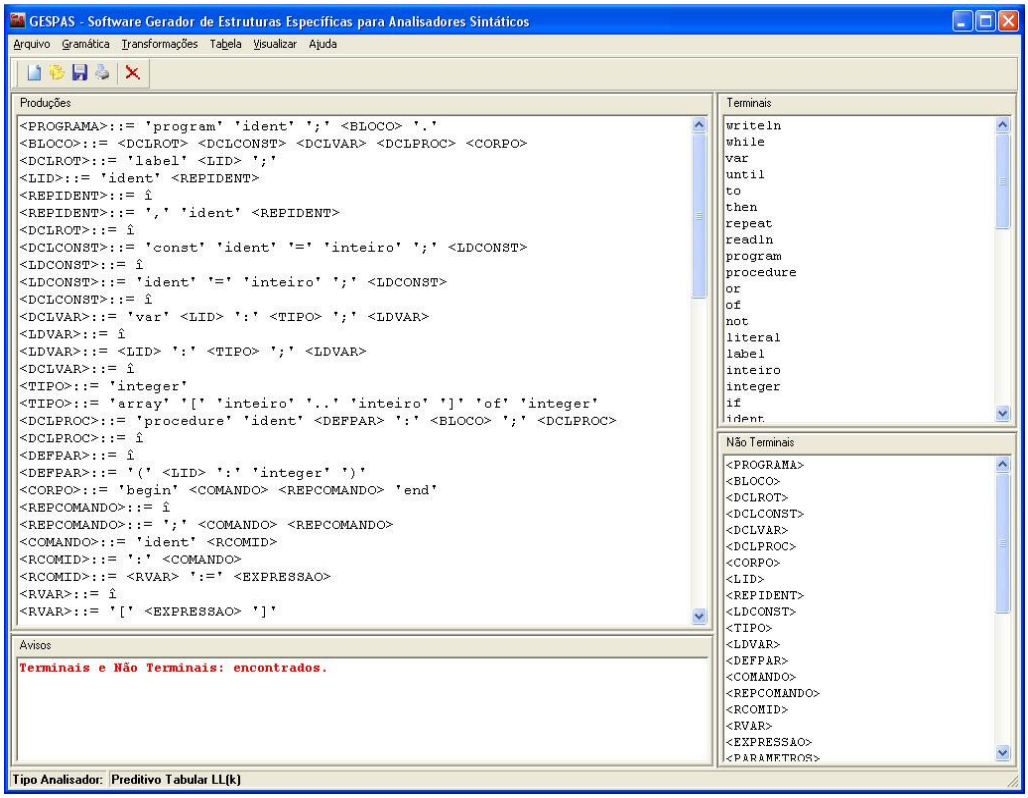


Figura 2. Interface do software Gespas

Após feita a entrada da gramática no software, deve-se executar o processo de reconhecimento dos símbolos terminais e não terminais. Este processo está disponível em Gramática→Encontrar Terminais/NT. Neste momento o software irá fazer algumas verificações na gramática se ela não possui produções duplicadas, linhas em branco no editor e se não existem produções incompletas. Terminado esse processo pode-se codificar a gramática através do processo Gramática→Codificar a Gramática.

Terminado estas duas etapas, o usuário pode executar algum algoritmo de simplificação e transformação ou executar a geração da tabela de análise. Os algoritmos podem ser executados através do menu Transformações. Neste menu está contido a chamada de todos os algoritmos, a opção de desfazer a execução do último algoritmo executado e comparar a gramática original com a gramática alterada. Se o usuário escolher gerar a tabela de análise (Tabela → Gerar), o software verifica se a gramática não possui recursão à esquerda, se a gramática não está fatorada e se possui símbolos inúteis. Caso existir alguns desses problemas na gramática, é apresentado uma mensagem para o usuário se ele deseja executar o algoritmo referente a correção. Após a execução de qualquer algoritmo de alteração na gramática é apresentado uma

mensagem para o usuário se ele deseja comparar a gramática original com a gramática alterada. Esta tela pode ser vista na Figura 3.

Após gerado a tabela de análise sintática, o usuário pode exportar para alguma linguagem de programação disponível no software. As linguagens dispníveis são: Delphi, Java, C++, Pascal. O caminho para efetuar esta exportação é em Tabela → Exportar → Linguagem a ser escolhida Também pode ser visualizado a tabela e a gramática codificada através dos respectivos caminhos: Visualizar → Tabela e Visualizar → Gramática Codificada. No menu arquivo estão disponíveis os processos de salvar, abrir, etc a gramática.

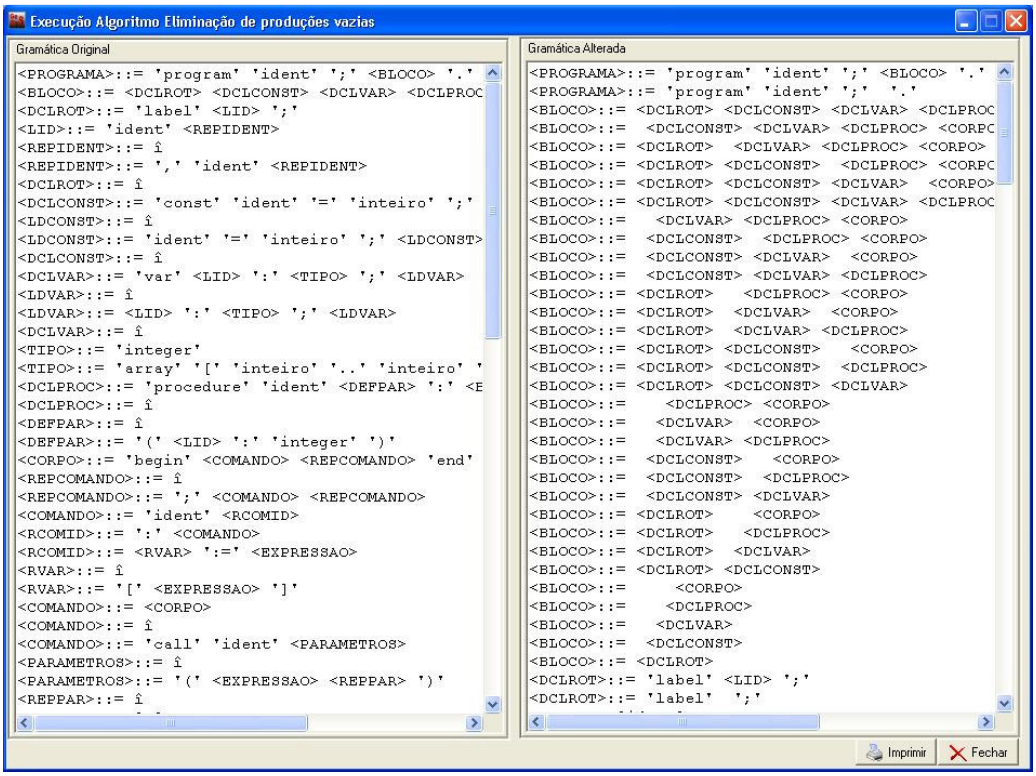


Figura 3. Tela de comparação da gramática original com a alterada

6. Testes

Para cada algoritmo desenvolvido no software foram feitos testes com várias gramáticas retiradas de livros e até mesmo desenvolvidas para validar o funcionamento. Conforme foram apresentando erros foram feitas as devidas correções no software. Após os testes com estas gramáticas, foram efetuados testes com gramáticas mais complexas utilizando todos os processos do software desde o processo de encontrar terminais/NT até a visualização da gramática codificada. Os resultados obtidos com o software foram considerados satisfatórios.

7. Conclusão

A pesquisa desenvolvida, a qual deu-se continuidade ao software GESPAS, explorou as características e fundamentações de Gramáticas e Gramáticas Livres de Contexto, bem

como, seus algoritmos de Simplificação e Transformação de Gramática Livre de Contexto. Com as alterações no software GESPAS automatizou-se o processo de desenvolvimento de compiladores. Isto porque na geração da tabela de análise sintática, o software faz algumas verificações na gramática. Anteriormente se tivesse que ser feito alguma alteração na gramática esta tinha que ser feita de forma manual sucessetível a erros.

O software ficou com seus processos de forma simplificada tornando-se fácil de ser manipulado pelo usuário.

Referências

CANTÙ, Marco. **Dominando o delphi 7**: a bíblia. Tradução Kátia Aparecida Roque. São Paulo: Pearson Makron Books, 2005.

DIVERIO, Tiarajú A.; MENEZES, Paulo B. **Teoria da computação**: máquinas universais e computabilidade. 2. ed. Porto Alegre: Sagra Luzzatto, 2004.

HOPCROFT, John E.; MOTWANI, Rajeev; ULLMAN, Jefrey D. **Introdução à teoria de autômatos, linguagens e computação**. Tradução Vandenberg D. de Souza. Rio de Janeiro: Campus/Elsevier, 2002.

HUBBARD, John R. **Programação em C++**. Tradução de Edson Furmankiewicz. 2. ed. Porto Alegre: Bookman, 2003.

MENEZES, Paulo Blauth**. Linguagens formais e autômatos**. 5. ed. Porto Alegre: Sagra Luzzato, 2005.

VIEIRA, Newton José. **Introdução aos fundamentos da computação**: linguagens e máquinas. São Paulo: Pioneira Thomson Learning, 2006.

ANEXO A – MANUAL DO USUÁRIO

GESPAS - Software Gerador de Estruturas Específicas para Analisadores Sintáticos

MANUAL DO USUÁRIO

(Desenvolvido por Fabiano Martins, 2005)
(Disponível no menu Ajuda do GESPAS)

SUMÁRIO

1.Introdução

O GESPAS é um software que se destina a geração de tabelas de parsing para analisadores sintáticos ascendentes e descendentes preditivos tabulares. Para os ascendentes ele gera a tabela utilizando a técnica SLR. Pretende auxiliar acadêmicos e profissionais, tanto na área de ciência da computação, como na engenharia que desejem implementar um analisador sintático.
Implementado na linguagem C++, muito utilizada em meio acadêmico, dessa forma o código fonte do software pode ser estudado e melhorado conforme a necessidade dos alunos.

2. Menu

2.1 arquivo

2.1.1 Novo Projeto

Inicia um novo projeto onde o usuário tem a opção de criar tabelas para analisadores Descendentes Preditivos (LL) ou Ascendentes(SLR).

2.1.2 Abrir Projeto

Abre um projeto previamente salvo.

2.1.3 Salvar
Salva as alterações feitas no projeto.

2.1.4 Salvar Como
Para salvar o projeto com outro nome ou em outro local.

2.2 Gramática

2.2.1 Encontrar Terminais/NT
Função que idêntica os elementos terminais e não terminais a partir das produções fornecidas pelo usuário.

2.2.2 Codificar Gramática
Função para codificar as produções e os elementos da gramática, este procedimento é necessário para a geração da tabela.

2.3 Tabela

2.3.1 Gerar
Gera a tabela de análise sintática.

2.3.2 Exportar
Exporta a tabela para que possa ser usada na implementação e um analisador sintático nas seguintes linguagens: C++, Java, Delphi., Pascal.

2.4 Visualizar

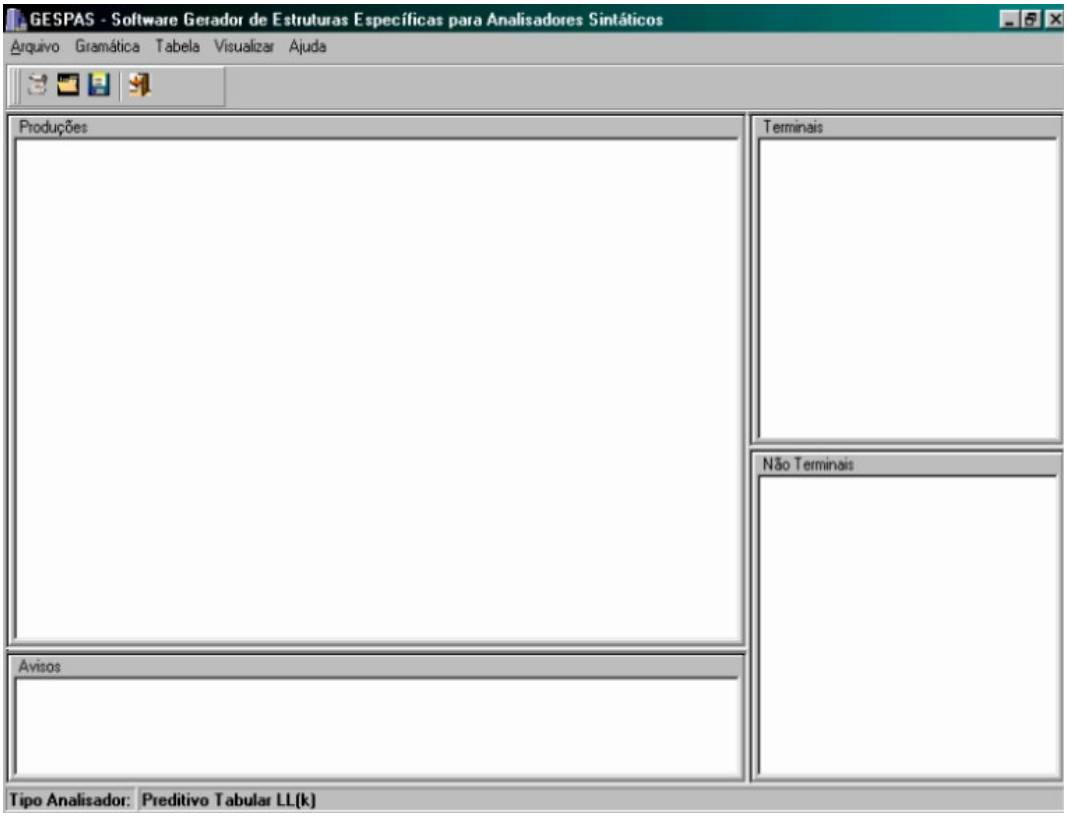
2.4.1 Tabela
Permite a visualização da tabela de análise.

2.4.2 Gramática Codificada
Permite a visualização da gramática codificada.

2.5 Ajuda
Exibe este arquivo de ajuda.

3. Iniciando um novo projeto
A partir do menu **Arquivo>Novo Projeto** o usuário tem a opção de escolher entre criar uma tabela para analisadores Descendentes Preditivos ou Ascendentes SLR.

4. Produções da gramática
Feita a escolha de qual tipo de analisador se pretende que seja gerada a tabela o usuário devera entrar com as produções da gramática no local apropriado.



Exemplo de gramática:

```
<DECLARACAO> ::= <IF-DECL>
<DECLARACAO> ::= 'outra'
<IF-DECL> ::= 'if' '(' <EXP> ')' <DECLARACAO> <ELSE-PARTE>
<ELSE-PARTE> ::= 'else' <DECLARACAO>
<ELSE-PARTE> ::= î
<EXP> ::= '0'
<EXP> ::= '1'
```

Como pôde ser observado os símbolos não terminais deverão ser representados entre os símbolos "<" e ">" (Maior e Menor). Já os terminais entre aspas simples. O símbolo î (a letra " i " com acento circunflexo), representa a palavra vazia. A primeira produção da gramática será considerada o símbolo inicial.
Ao terminar de entrar com as produções o usuário deve por meio do menu **Gramática>Encontrar Terminais/NT** encontrar os elementos da gramática (terminais e não terminais). Após conferir se não precisa adicionar mais nenhuma produção ele pode continuar e codificar a gramática **Gramática>Codificar Gramática**.

A partir deste momento já pode ser gerada a tabela de análise através do menu **Tabela>Gerar**.

5. Carregando um projeto salvo
O usuário pode a qualquer momento salvar um projeto a carrega-lo quando precisar. Quando carregar um projeto o usuário deve executar as operações de codificar a gramática e gerar a tabela, mesmo que já tivessem sido feitas antes do salvamento, para poder visualizar ou exportar a tabela de análise.

6. Exportando a tabela
A tabela de análise sintática pode ser exportada para a implementação de um analisador sintático nas seguintes linguagens: C++, Java, Delphi e Pascal.

7. Exportando para o formato html
Além da exportação para implementação de analisadores sintáticos a tabela pode ser salva em um arquivo html, onde pode ser impressa e usada na documentação. Além da tabela outros elementos podem ser salvos nesse formato como o conjunto first, o conjunto follow e a gramática codificada.