Desenvolvimento do Compilador da Linguagem Basico

Fábio Alexandrini
fabalex@ifc-riodosul.edu.br
UNIDAVI/IFC-RIODOSUL

Tiago Boechel
tboechel@gmail.com
IFC-RIO DO SUL

Felipe Augusto Schiquetti1
felipe_schiquetti@hotmail.com
IFC-RIO DO SUL

Luis Ricardo Rosa
rosah_rh@hotmail.com
IFC-RIO DO SUL

Rodrigo Becker Probst
rodrigo_beka@hotmail.com
IFC-RIO DO SUL

Resumo: Compiladores é uma disciplina fundamental dos cursos superiores da área de computação, no presente artigo apresenta um breve discurso com seus principais tópicos relacionados a elementos de matemática discreta, conceitos de linguagens, linguagens regulares, linguagens livres de contexto, linguagens sensíveis ao contexto e linguagens recursivas. Está matéria se apresenta com ênfase na formação cientifica do aluno, sendo uma matéria árida e complexa. O objetivo do presente artigo é expressar de forma mais fundamentada os conceitos da disciplina com um teor mais abrangente, para que principalmente o estudante sinta-se mais familiarizado com a disciplina e descubra a sua importância para conseguir aperfeiçoar seus algoritmos e compreender os limites da máquina. Após um estudo focado na matéria corrente organizo-se através do software GALS um analisador léxico e sintático, tendo como base a linguagem de programação BASIC, mas com algumas modificações impostas para o aperfeiçoamento dos analisadores.

Palavras Chave: Linguagem Basico - LinguagemProgramação - Compiladores - Ciência da Computaçã - Tec.Informação

1. INTRODUÇÃO

Um Compilador pode ser denotado como um programa que recebe como entrada um programa fonte e o traduz para um programa equivalente em outra linguagem (LOUDEN, 2004). É um programa que traduz um programa escrito em linguagem de alto nível para um programa equivalente em código de máquina para o processador. O compilador não produz o código de máquina e sim um programa em linguagem simbólica semanticamente equivalente ao programa em linguagem de alto nível, traduzido então em linguagem de máquina.

Para realizar as análises, o compilador deve ter ciência de quais são os tokens apropriados da linguagem, assim como suas palavras chaves e regras para o desenvolvimento de identificadores. As técnicas de aperfeiçoar o código que são usadas em compiladores devem, além de manter a definição do programa original, ser capaz de capturar a maior parte das possibilidades de avanço do código dentro de limites plausíveis de esforços gastos para tal fim.

Esse método tem a vantagem de uma execução de programa muito rápida, assim que o processo de tradução é concluído. A linguagem que um compilador traduz é chamada de linguagem-fonte. O processo de compilação desenvolve-se em diversas etapas que serão melhor compreendidas no decorrer deste artigo.

2. ALFABETOS

Um alfabeto é um conjunto finito não vazio cujos elementos são chamados de símbolos. Dessa maneira, os conceitos de símbolo e alfabeto são introduzidos de forma interdependente: um alfabeto é um conjunto de símbolos, e um símbolo é um elemento qualquer de um alfabeto.

Até certo ponto, pode-se arbitrar os símbolos que nos interessam, e incluir apenas esses símbolos no alfabeto. Para cada aplicação específica, o usuário deve escolher o alfabeto que pretende utilizar.

Uma cadeia de símbolos em um alfabeto pode ser definida como uma função;

V = {a, b, c,.,z }
V = {0, 1}
V = {a, e, i, o, u}

3. ALGORITMOS

Um algoritmo é uma a sequência finita de instruções, e define instruções como uma operação claramente descrita e que possa ser executada em tempo finito.

Um algoritmo nada mais é do que uma receita que mostra passo a passo os procedimentos necessários para a resolução de uma tarefa. Em termos mais técnicos, um algoritmo é uma sequência lógica, finita e definida de instruções que devem ser seguidas para resolver um problema ou executar uma tarefa.

Embora você não perceba, utiliza algoritmos de forma intuitiva e automática diariamente quando executa tarefas comuns. Como estas atividades são simples e dispensam ficar pensando nas instruções necessárias para fazê-las, o algoritmo presente nelas acaba passando despercebido, como exemplo a receita de um bolo (Figura 1);

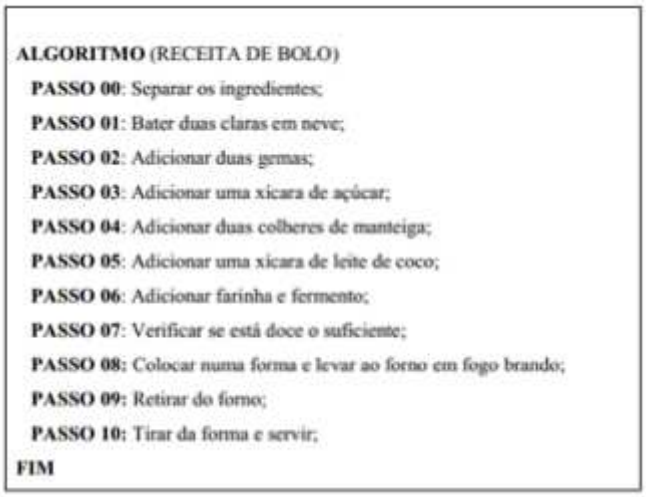

Figura 1: Algoritmo (receita de bolo)

4. GRAMÁTICAS

Chomsky definiu quatro tipos de gramáticas, 0, 1, 2 e 3, que formam uma hierarquia: cada gramática de um tipo é também dos tipos menores.

Nossa definição não é exatamente a mesma usada por Chomsky, mas é praticamente equivalente.

As regras de uma gramática tipo 0 são regras da forma a ® b, com a e b quaisquer. Gramáticas tipo 1 são as gramáticas com regras da forma a ® b, em que se exige | a | £| b |; é entretanto permitida uma regra que viola esta restrição: uma gramática tipo 1 pode ter a regra S ® e, se S não aparece do lado direito de nenhuma regra.

Gramáticas tipo 2 são as gramáticas com regras da forma A ® b, onde A é um símbolo não terminal, e b é uma sequência qualquer de V*, possivelmente vazia.

Gramáticas tipo 3 só podem ter regras dos três tipos descritos a seguir:
A ® aB, onde A e B são não terminais, e a é um terminal;
A ® a onde A é um não terminal, e a é um terminal;
A ® e, onde A é um não terminal.

Se uma linguagem tem uma gramática tipo 0, ela é uma linguagem tipo 0; se tem uma gramática tipo 1, ela é uma linguagem tipo 1, ou uma linguagem sensível ao contexto, se tem uma gramática tipo 2, ela é uma linguagem tipo 2, ou uma linguagem livre de contexto; se tem uma gramática tipo 3, ela é uma linguagem tipo 3, ou uma linguagem regular.

As gramáticas tipo 3 são chamadas regulares pela simplicidade da estrutura de suas linguagens, garantida pelos rígidos formatos de suas regras.

As gramáticas tipo 2 são chamadas de livres de contexto porque uma regra A ® b indica que o não terminal A, independentemente do contexto em que estiver inserido, pode ser substituído por b.

Finalmente, as gramáticas tipo 1 são chamadas de sensíveis ao contexto.

Tabela 1: Hierarquia de Chomsky

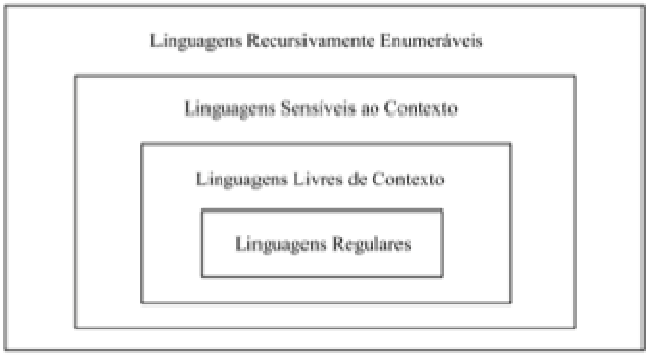

Figura 2: Hierarquia de Chomsky

5. AUTÔMATOS

Autômato finito determinístico;

Um Autômato Finito Determinístico, sobre um alfabeto é um sistema, onde um conjunto de estados finito, não vazio é um alfabeto de entrada finito e a função de transição é o estado inicial e o conjunto de estados finais.

O nome determinístico faz referência ao fato de que é uma função que determina precisamente o próximo estado a ser assumido, em resumo são reconhecedores de linguagens regulares definidos através de quíntuplas de forma a seguir:

É um conjunto finito não vazio de estados do autômato;

É um conjunto de símbolos, denominado alfabeto de entrada do autômato;

É a função de transição de estados do autômato e seu papel é o de indicar as transições possíveis em cada configuração do autômato. Esta função fornece para cada par "estado e símbolo de entrada" um novo estado para onde o autômato deverá mover-se.

É denominado estado inicial do autômato finito. É o estado para o qual o reconhecedor deve ser levado antes de iniciar suas atividades.

É um subconjunto do conjunto Q dos estados do autômato, e contém todos os estados de aceitação ou estados finais do autômato finito. Estes estados são aqueles em que o autômato deve terminar o reconhecimento das cadeias de entrada que pertencem à linguagem que o autômato define. Nenhuma outra cadeia deve ser capaz de levar o autômato a qualquer destes estados.

Por exemplo:

M = ({A, B}, {0, 1}, f, A, {B}) f = (A, 0) Þ A
(A, 1) Þ B
(B, 1) Þ B
(B, 0) Þ A

Para este autômato finito, reconhecem-se os seguintes elementos:
Estados do autômato: A e B
Símbolos do alfabeto de entrada: 0 e 1
Estado final: B
Estado inicial: A

Linguagem reconhecida: cadeias de dígitos binários terminadas obrigatoriamente por um dígito 1.

6. AUTOMATOS FINITOS NÃO DETERMINISTICOS

Passa-se agora ao estudo dos autômatos não determinísticos. Em oposição ao que acontece com os determinísticos, a função de transição de um autômato não determinístico não precisa determinar exatamente qual deve ser o próximo estado. Em vez disso, a função de transição fornece uma lista de estados para os quais a transição poderia ser feita. Essa lista pode ser vazia, ou ter um número qualquer positivo de elementos.

Essa possibilidade de escolha entre vários caminhos a serem seguidos nos leva a modificar a definição de aceitação. Um autômato determinístico, aceita se "o último estado atingido é final".

Mas um não determinístico aceita se "existe uma sequência de escolhas tal que o último estado atingido é final". Pode-se alternativamente imaginar que o autômato não determinístico "escolhe", "adivinha", o caminho certo para a aceitação, uma vez que a existência de escolhas erradas, que não levam a um estado final, é irrelevante.

7. ANÁLISES

Análise léxica é o processo de analisar a entrada de linhas de caracteres e produzir uma sequencia de símbolos chamados "símbolos léxicos" (tokens), que podem ser manipulados mais facilmente por um leitor de saída.

Análise sintática, assim como as outras referentes à língua, é um exercício muito próximo da matemática, pois envolve um raciocínio lógico do tipo: “se você encontrar tal elemento, então admita que esse elemento seja um objeto”. Promover esse tipo de raciocínio no estudo das sentenças é desenvolver uma análise formal, porque as categorias sintáticas são formas que não dependem do conteúdo que expressam. Resumidamente este é o processo de analisar uma sequencia de entrada para determinar sua estrutura gramatical.

Análise semântica é a terceira fase da compilação onde se verifica os erros semânticos, (por exemplo, uma multiplicação entre tipos de dados diferentes) no programa-fonte e coleta as informações necessárias para a próxima fase da compilação.

8. ÁRVORES

São estruturas de dados extremamente úteis em muitas aplicações. Uma árvore é formada por um conjunto finito *T* de elementos denominados vértices ou nós de tal modo que se T = 0 a árvore é vazia, caso contrário tem-se um nó especial chamado raiz da árvore (*r*), e cujos elementos restantes são particionados em *m*>=1 conjuntos distintos não vazios, as subárvores de *r*, sendo cada um destes conjuntos por sua vez uma árvore. A seguir pode-se verificar uma arvore de nove nós, sendo (A) a raiz;

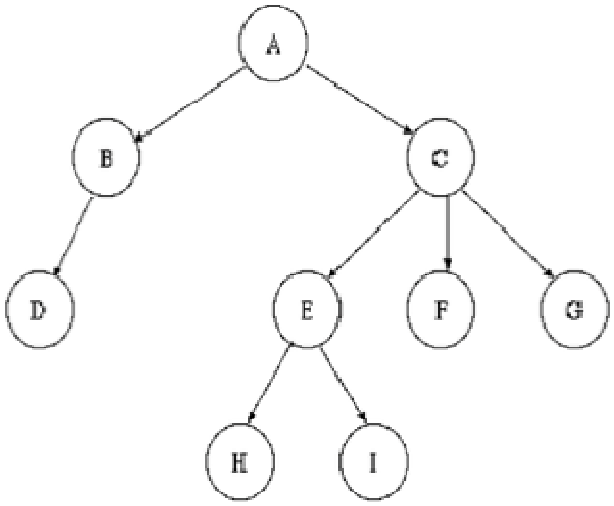

Figura 3: Arvore de nove nós sendo (A) á raiz

Uma árvore estritamente binária é uma árvore binária em que cada nó tem 0 ou 2 filhos. Uma árvore binária cheia é uma árvore em que se um nó tem alguma sub-árvore vazia então ele está no último nível. Uma árvore completa é aquela em se *n* é um nó com algumas de subárvores vazias, então *n* se localiza no penúltimo ou no último nível. Portanto, toda árvore cheia é completa e estritamente binária;

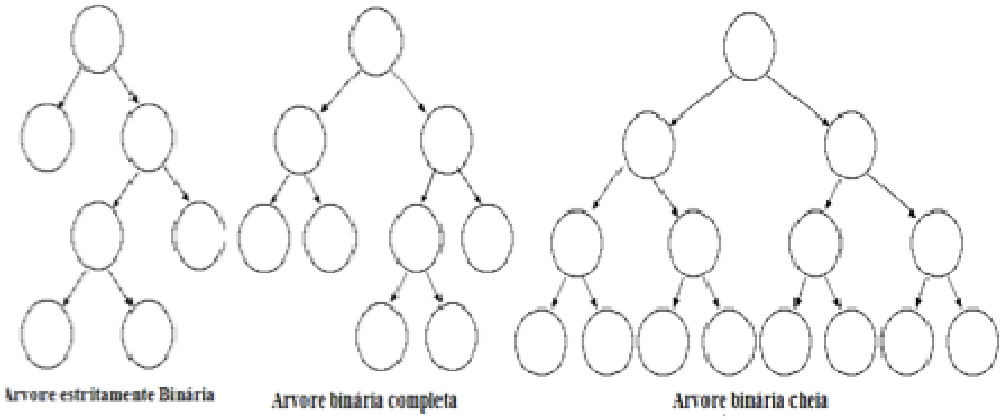

Figura 4: Árvore estritamente binária, binária completa e binária cheia

As árvores binárias são estruturas importantes toda vez que uma decisão binária deve ser tomada em algum ponto de um algoritmo. Ao Supor que precisa-se descobrir números duplicados em uma lista não ordenada de números. Uma maneira é comparar cada novo número com todos os números já lidos. Isto aumenta em muito a complexidade do algoritmo. Outra possibilidade é manter uma lista ordenada dos números e a cada número lido fazer uma

busca na lista. Outra solução é usar uma árvore binária para manter os números. O primeiro número lido é colocado na raiz da árvore. Cada novo número lido é comparado com o elemento raiz, caso seja igual é uma duplicata e volta-se a ler outro número, pode-se verificar melhor no exemplo:

Supondo-se que os números 7-8-2-5-8-3-5-10-4 foram os dados de entrada fornecidos pelo usuário, sua árvore binária seria construída da seguinte forma;

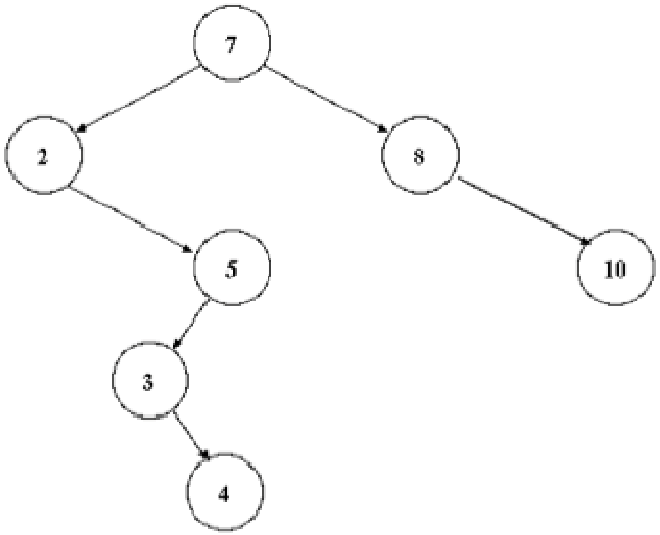

Figura 5: Arvore binária com dados fornecidos pelo usuário.

10. GALS

O GALS é uma ferramenta freeware de geração de ***analisadores léxicos e sintáticos***, dai o seu nome (GALS), seu uso é feito através de sua interface gráfica, com base em definições regulares e uma gramática.

Gera os analisadores para três linguagens ; Java, C++ ou Delphi. Tem a opção de gerar o analisador léxico, o analisador sintático ou ambos (GESSER, 2003, p.39).

O download do software GALS pode ser feito no site http://gals.sourceforge.net/ onde também se encontra tutoriais e fóruns de como utiliza-lo.

Foi criado por Carlos Eduardo Gesser em seu Trabalho de conclusão de curso (TCC) de ciência da computação na universidade federal de santa catarina, sob orientação do professor Dr. Olinto José Varela Furtado.

11. DESENVOLVIMENTO DO RECONHECEDOR LÉXICO/SINTÁTICO

Teve-se algum tempo para decidir como de criar a própria linguagem, e decidiu-se que, como aprendizes de cientistas da computação, deve-se criar algo básico mas que tivesse uma possível utilidade prática no futuro, portanto, optao-se por criar uma linguagem que ajuda nos primeiros passos para crianças aprenderem a lógica da programação.

Volta-se para os primórdios, em aulas de Algoritmos, e decidiu-se que precisaria ter criação de variáveis, o uso do se (IF) e do enquanto (WHILE) que são, em nossa opinião o básico. Daí o nome da nossa linguagem.. BASICO.

Após o estudo realizado construí-se um ANALISADOR LÉXICO E SINTÁTICO BASICO - IF/WHILE através da ferramenta de desenvolvimento GALS (Figura 6), o analisador é Léxico e Sintático das palavras reservadas IF e WHILE, tendo como base a linguagem BASIC, mas com algumas modificações que aperfeiçoam e simplificam o entendimento da verificação dos analisadores.

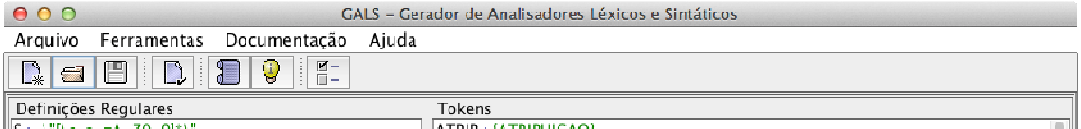

Figura 6: Código escrito na IDE Gals

Os primeiros passos para a criação de uma estrutura léxica estudam os tokens que serão reconhecidos pelo analisador.

Antes de especificar de fato os tokens, é preciso verificar que a especificação léxica é dividida em duas partes: Definições Regulares e Definição dos tokens, eles são definidos na segunda parte.

Nas definições regulares são definidas expressões auxiliares, para serem utilizadas em sua definição. No programa, uso-se vários tipos de definições regulares, como operadores relacionais, operadores lógicos, identificadores e números, transformando todos em tokens incluindo as palavras reservadas que em especiais são o IF e WHILE, e as palavras necessárias para um bom entendimento do funcionamento desses algoritmos. OBS: no código, não considerou-se a nomeação das linhas como acontece com o BASIC original, portanto, no exemplo aparecerá como sendo um número.

Com os tokens prontos, verifico-se as exceções exigidas pela IDE de programação, podendo verificar o funcionamento da Analise Léxica do código escrevendo o código de teste (Figura 7):

A = 2
IF (A < 3) GOTO 30 ELSE GOTO 50
A = 5
WHILE (A<3) GOTO 30

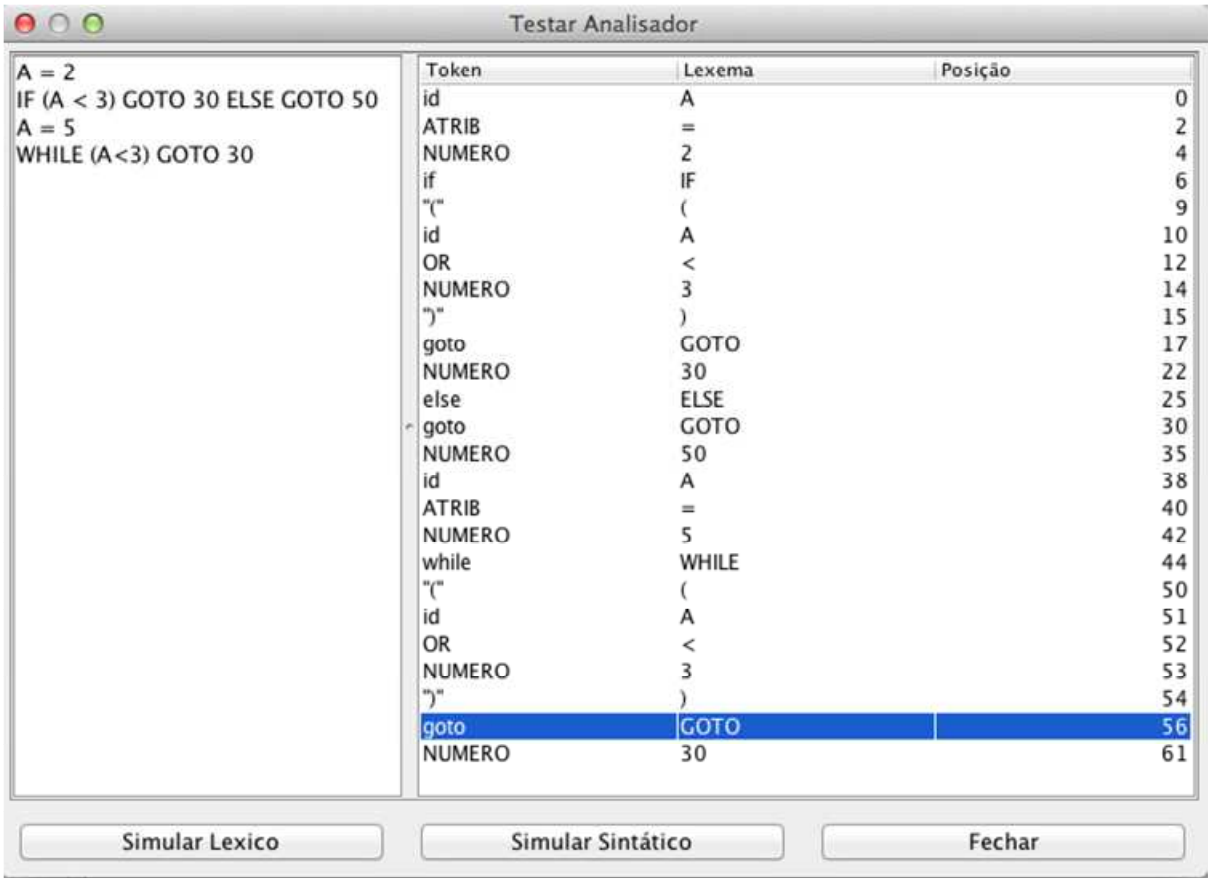

Figura 7: Programa rodando a análise Léxica

A especificação sintática é feita de produções para uma gramática de expressão IF da seguinte forma:

```
<SE> ::= if "("<OP_REL>")"  <INIC>
       | if "("<OP_REL>")"  <GOTO>
       | if "("<OP_REL>")"  <INIC> <SENAO>
       | if "("<OP_REL>")"  <INIC> <GOTO>
       | if "("<OP_REL> <OP_LOG>")"  <INIC>
       | if "("<OP_REL> <OP_LOG>")"  <INIC> <SENAO>
       | if "("<OP_REL> <OP_LOG>")"  <INIC> <GOTO>;
```

Pode ser utilizado na gramática qualquer token já declarado como símbolo terminal. Os símbolos não terminais precisam ser previamente declarados em sua área específica, como o ELSE. Veja o exemplo a seguir(Figura 8):

```
A = 2
IF (A < 3) GOTO 30 ELSE GOTO 50
A = 5
WHILE (A<3) GOTO 30
```

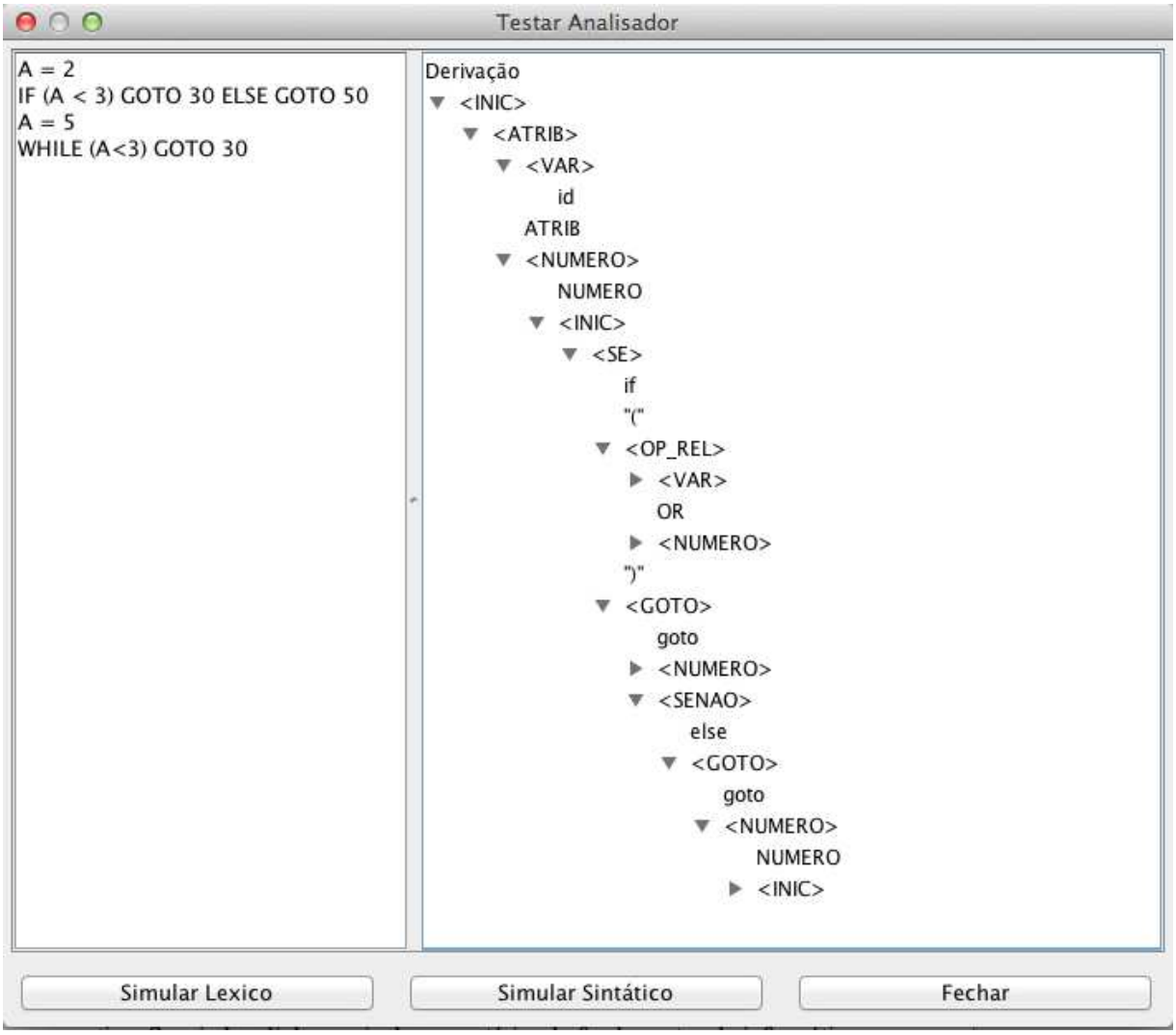

Figura 8: Programa rodando a análise Sintática

12. BASICO

Tendo como base os analisadores retirados do GALS, foi possível a criação de um compilador utilizando a linguagem JAVA, que consegue interpretar a linguagem que está sendo escrita e analisa lexica, sintática e semanticamente. Resultando em um código compilado com sucesso (Figuras 9,10, 11) ou um código com erro. Além disso, este compilador permite que o código escrito seja salvo para usá-lo posteriormente.

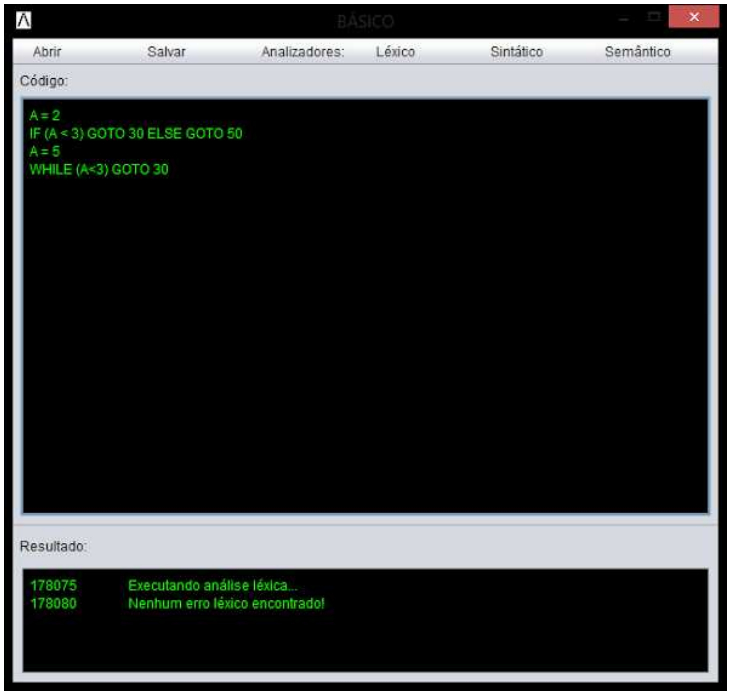

Figura 9: Compilador BASICO executando análise léxica com sucesso

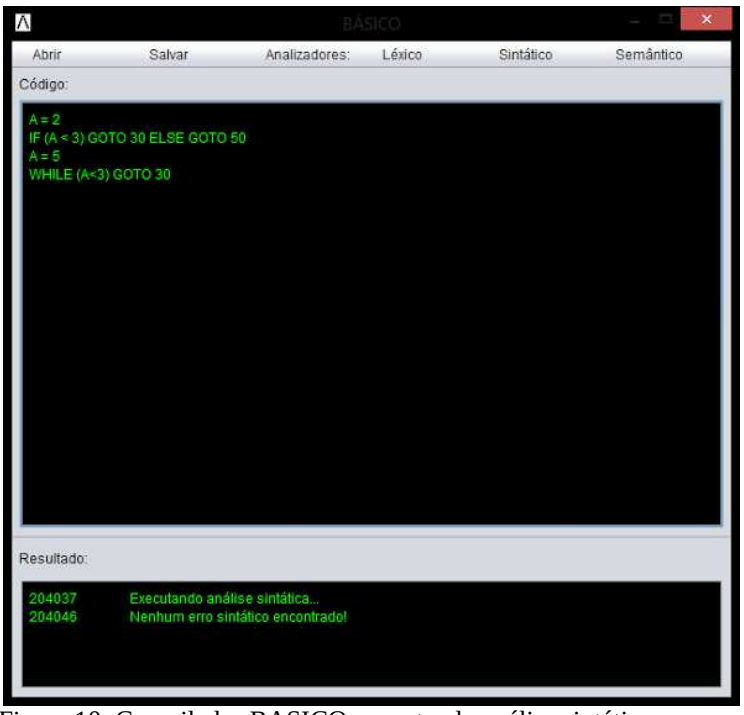

Figura 10: Compilador BASICO executando análise sintática com sucesso

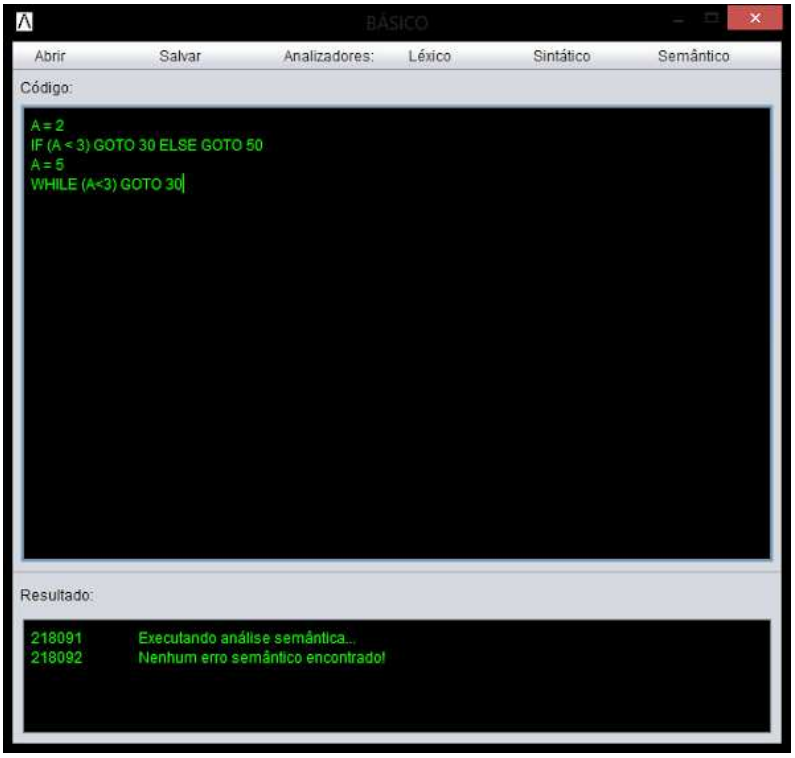

Figura 11: Compilador BASICO executando análise semântica com sucesso

13. CONSIDERAÇÕES FINAIS

Com uma base tão técnica e cientifica a matéria de compiladores tem de a se tornar debilitante, mas que se estudada mais a fundo com um tom mais utilizável e menos teórico ela se torna útil na construção de algoritmos mais sólidos e estáveis.

No decorrer do trabalho verifico-se que com a teoria relativamente colocada em prática fica muito mais fácil absorver os assuntos tratados, prontamente com os códigos fontes aperfeiçoados.

Mas sem deixar de brindar a matérias anteriores que nos fizeram ter o conhecimento para escrever o artigo. Seguindo a linha curricular as matérias de fundamentos de informática e matemática, arquitetura de computadores, teoria da computação e linguagem formais e autômatos tiveram um grande peso para este artigo ser consolidado.

A estrutura geral do compilador está diretamente ligada com seus analisadores, tabelas e resultados, com determinação e pesquisa o código sempre pode ser mais bem apreendido e consequentemente melhorado

14. REFERÊNCIAS

APRENDIZAGEM em matemática: registros de representação semiótica . 7. ed. São Paulo: Papirus, 2010. 160p. (Papirus educação) ISBN 8530807313.

CORTES, Pedro Luiz. Administração de sistemas de informação. São Paulo, SP: Saraiva, 2008. 503 p. ISBN 9788502064508.

DIVERIO, Tiarajú Asmuz; MENEZES, Paulo Blauth. Teoria da computação: máquinas universais e computabilidade . 2. ed. , 2 reimp. Porto Alegre: Instituto de Informatica da UFRGS: Sagra Luzzatto, 2004. 205 p. (Livros didáticos (; n. 5) ISBN 8524105933.

GALS-GERADOR DE ANALISADORES LÉXICOS E SINTÁTICOS Disponível em:https://projetos.inf.ufsc.br/arquivos_projetos/projeto_353/Gals.pdf
Acesso : em 10 abril.2014.

LIPSCHUTZ, Seymour; LIPSON, Marc. Matemática Discreta. Porto Alegre, RS: Bookmann, 2013. xi, 471 p. (Coleção Schaum) ISBN 9788565837736.

LOUDEN, Kenneth C. Compiladores: princípios e prática . 2. ed. São Paulo: Cengage Learning, 2004. xiv, 569 p. ISBN 9788522104222.

MARTINS, Paulo Roberto (Org.). Algoritmos e estrutura de dados. São Paulo: Pearson Education, 2009. 184p. (Análise e Desenvolvimento de Sistemas ; 3) ISBN 9788576052449.

PATTERSON, David A; HENNESSY, John L. Organização e projeto de computadores. 3. ed. Rio de Janeiro: Elsevier, Campus, 2005. 484 p. ISBN 8535215212.

ROSA, João Luís Garcia. Linguagens formais e autômatos. Rio de Janeiro, RJ: LTC, 2010. 146 p. ISBN 9788521617518.

BEACH, R.; MUHLEMANN, A. P.; PRICE, D. H. R.; PATERSON, A. & SHARP, J. A. A review of manufacturing Flexibility. European Journal of Operational Research, v. 122, 2000, pp. 41-57.

OLIVEIRA, U. R. Gerenciamento de riscos operacionais na indústria por meio da seleção de diferentes tipos de flexibilidade de manufatura. 2009. 246 f. Tese (Doutorado em Engenharia Mecânica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2009.

PADOVEZE, C. L. & BERTOLUCCI, R. G. Proposta de um Modelo para o Gerenciamento do Risco Corporativo. In: Anais XXV Encontro Nacional de Engenharia de Produção, Porto Alegre, 2005

TRIOLA, M. F. Introdução à Estatística. 9ª Edição. Rio de Janeiro: LTC, 2005