UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Elisangela Maschio de Miranda

UMA FERRAMENTA DE APOIO AO PROCESSO DE APRENDIZAGEM DE ALGORITMOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Profª. Anita Maria da Rocha Fernandes, Dra.

Florianópolis, novembro de 2004

UMA FERRAMENTA DE APOIO AO PROCESSO DE APRENDIZAGEM DE ALGORITMOS

Elisangela Maschio de Miranda

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação - Área de Concentração em Sistemas de Conhecimento - e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Raul S. Wazlawick, Dr.
Coordenador do Curso de
Pós-Graduação em Ciência da Computação

Banca Examinadora:

_____

Profª. Anita Maria da Rocha Fernandes, Dra.
Orientadora

_____

Prof. Rogério Cid Bastos, Dr.
Membro da Banca

_____

Profª. Silvia Modesto Nassar, Drª.
Membro da Banca

_____

Prof. Marcelo Thiry, Dr.
Membro da Banca

“**Sonhe alto ...**

Queira o melhor do melhor ...

Se pensamos pequeno ...

Coisas pequenas teremos ...

Mas se desejarmos fortemente o melhor ...

O melhor vai se instalar em nossa vida.

Porque sou do tamanho daquilo que vejo,

E não do tamanho da minha altura”.

(Carlos Drummond de Andrade)

DEDICATÓRIA

Dedico este trabalho a pessoas especiais, que sempre estiveram e estão a meu lado, e com quem posso contar em todos os dias de minha vida. Dedico a meus pais, João e Elaine, que sempre foram um exemplo de vida e tornaram possível esta conquista, e também a minha irmã, incansável, e sempre disposta a não me deixar desanimar. Amo Vocês !!!

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus e a meu anjo da guarda, que com certeza me guardaram e me protegeram durante estes dois anos e meio.

Várias foram as pessoas que estiveram presentes neste tempo. Gostaria de agradecer a todas, pois com certeza sem elas não estaria aqui. Há momentos em que uma palavra, um abraço significam muito.

Aos meus avós, Primo Maschio (*in memorian*) e Cezira Circe Maschio (*in memorian*), que com certeza lá de cima estavam cuidando dos meus passos e zelando por mim. Avós, mais um sonho de vocês conquistado. A Tia Bere e Laline, por todas as novenas que rezaram por mim, todas as velas acesas a meia noite, e por sempre acreditarem no meu potencial, acima de tudo. A toda minha família, tios, primos, por tudo.

A Anita, minha orientadora, pela força e até pelos momentos de briga, que fizeram com que eu levantasse a cabeça e quisesse ir em frente. Anita, conseguimos! A Marlei, que principalmente no final esteve entre eu e a Anita, procurando manter as duas de cabeça fria, e a primeira pessoa que disse no momento em que mais precisava: "Eu sei que você vai conseguir. Força!". Obrigada Marlei! Ao Rudimar, meu father, meu co-orientador, que também sempre esteve torcendo e sempre se manteve ao meu lado.

Duas famílias tiveram papel importantíssimo nesta conquista: a família Gomes e a família Fernandes. Tio Rui, Tia Elfi e Fernando, muito obrigada pela força, pela casa, pelo companheirismo, por acreditarem em mim. Tio Rui e Tia Elfi, por serem meus pais adotivos, me acolherem de braços abertos e cuidarem de mim durante a maratona. Fernando, meu companheiro e amigo, que me acompanhou no início do mestrado, e até o final sempre acreditou que eu conseguiria chegar. Nunca esquecerei suas palavras, me incentivando e me fazendo acreditar em mim mesma. Obrigada ! A Família Fernandes – Anita, Ana, Betinha, Lu, Tio Luis, Tia Dilma, Dona Anita – que me acolheu e me incentivou em muitos momentos. Obrigada pelas conversas, pelas festas, pelas "caipiras veneno", pelas risadas, por tudo.

Existiram amigos especiais, que tiveram papel marcante nestes momentos. Alguns estiveram comigo durante todo o trajeto, outros surgiram no final, mas se tornaram especiais pela forma e momento em que surgiram. Fofete e Cami, não só amigos, vocês são irmãos. Obrigada por tudo, pela força, por permanecerem comigo nos meus piores e melhores momentos, por não desistirem de mim. Com vocês aprendi o verdadeiro significado de amizade. Vocês moram em meu coração. Algumas pessoas surgiram ao final e se tornaram importantes em minha vida: Michel Lemons da Silva, Rubens Burkot Junior, Elir M. Venturini Martins e Roberto Gonçalves Augusto Junior. Obrigada pela companhia, pelas risadas, pelo incentivo, por me aceitarem de braços abertos e poder dizer: encontrei novos amigos. Michel, mais ainda pra ti, obrigada pelo carinho, pela atenção, por cada palavra de incentivo, por se indignar quando algo não estava correto, pelas famosas palavras "vai terminar a dissertação", e por tantas outras coisas que não citarei aqui. Nem sabes o quanto foste importante. Obrigada bp, de coração. Santiago, Anne e Fernanda, sem vocês também não teria chego aqui. Santiago, com sua paciência ao me explicar o CIFluxProg, e sempre me incentivando, a todo momento. Anne, que quando o desespero bateu surgiu como um furacão na minha casa e disse: "eu faço milagres". Nasceu minha irmã. E a Fernanda, que com seu jeitinho rebelde conseguia me fazer rir nos momentos mais tumultuados, e sempre tinha uma palavra amiga, assim como os outros dois.

Gostaria de poder citar diversos amigos, mas os que eu esquecer, me perdoem. Bueno, Biscui, Andréa, Mateus, os meninos do LIA, Julia, Renate, André, Adriana, Filhote, Elisangela, Evandro, Profª. Cris, Oswaldo, Lucinéia, Marcia (Rudimar), Bi, Natali, Leandro, Carlos, e tantos outros, que muitas vezes com uma palavra, um estimulo, um sorriso, conseguiam me animar novamente e fazer acreditar que o sonho era possível.

Aos professores do curso de Ciência da Computação da UNIVALI – Itajaí por todo apoio e força durante todo o processo de dissertação, e ao coordenador do curso, Luca, por ter me incentivado e permitido realizar o trabalho dentro do curso. Obrigada!

A todos aqueles que, direta ou indiretamente, fizeram parte do trajeto e me ajudaram a vencer um obstáculo, meu muito obrigada.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASA	Animação e Simulação de Algoritmos
ATN	*Augmented Transition Network*
BNF	*Backus-Naur Form*
GIA	Grupo de Inteligência Aplicada
PDL	*Program Design Language*
PLN	Processamento em Linguagem Natural
RTN	*Recursive Transition Network*
SE	Sistema Especialista
SENAC	Serviço Nacional de Aprendizagem Comercial

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema para auxiliar os docentes nas avaliações da disciplina de algoritmos, mantendo a coerência e padrão das avaliações. As turmas iniciais nos cursos de ciência da computação e áreas afins possuem muitas vezes mais de 50 alunos, o que pode levar o professor a perder o padrão da correção, correndo o risco de não manter a mesma forma de correção para todos os alunos. Pensando nisto, foi elaborado um sistema que auxilia a manter uma coerência na correção das avaliações, o qual conta com dois módulos, sendo o Módulo do Professor e o Módulo do Aluno. O professor entra com os enunciados dos algoritmos e as informações mais relevantes a respeito dos mesmos. O acadêmico entra com sua resolução do problema, e o algoritmo será avaliado utilizando compiladores, através do uso da análise léxica, sintática e semântica, e o uso da análise pragmática, integrante da técnica de Inteligência Artificial denominada Processamento de Linguagem Natural. Logo após será gerado um arquivo texto com os erros encontrados, que será avaliado por um sistema especialista, e serão geradas as notas e comentários específicos. O sistema foi implementado utilizando a linguagem de programação C++, permitindo ao acadêmico realizar seus exercícios e ter uma avaliação a respeito dos mesmos, bem como ao professor uma visão de como os acadêmicos estão se saindo em sua disciplina.

ABSTRACT

This dissertation presents the development of a system to aid the professors in the evaluations of algorithms discipline, maintaining the coherence and the standard of the evaluations. The initial groups, in the Computer Science Courses and similar areas, have more fifty students many times, and this can lead the professor to lose the standard of correction, running the risk of not to maintain the same correction for all students. Thinking in this, a system that aids to maintain a coherence in the correction of the evaluations was ellaborated, which figures on two modules, being the Professor Module and the Student Module. The professor enters with the enunciations of algorithms and the information more important about them. The student enters with his/her resolution of the problem, and the algorithm will be evaluated using compilers through the use of semantics, syntax and lexicon analysis, and the use of programming analysis, integrated of the Artificial Intelligence technique called Processing of Natural Language. After that, a text file will be generated with the errors found, which will be evaluated by a specialist system, and the grades and specific comments will be generated. The system was implemented using the C++ language , allowing to the student to do his/her exercises and to have an evaluation about them, as well as to the professsor a view about how the students are performing in their discipline.

1. INTRODUÇÃO

1.1 APRESENTAÇÃO

A área da computação é formada por tecnologias novas em fase de grande expansão, contínuas modificações e estágios de maturidade heterogêneos. Os professores são continuamente desafiados DELGADO et. al (2004).

Em geral, as técnicas de ensino dos professores não estão maduras, e sofrem adaptações freqüentes. Um exemplo disto é a construção de algoritmos. O processo de construir algoritmos e transformá-los em programas é um verdadeiro processo de alfabetização. KOLIVER (2004) frisa que uma dificuldade encontrada no ensino de algoritmos é a falta de uma metodologia de ensino. Apesar dos vários debates realizados, não foi encontrada nenhuma abordagem que seja considerada correta.

Tanto quanto o processo de alfabetização, de acordo com DELGADO et. al (2004), a formalização ou expressão de situações e procedimentos, objetivos do estudo da computação cria diversas dificuldades, dentre as quais pode-se citar:

- Manipulação e interpretação de uma nova representação semiótica.
- Interpretação dos elementos e de suas inter-relações, de uma situação problema.
- Adaptação ao pensamento algorítmico.

O presente trabalho apresenta uma ferramenta que visa auxiliar o professor da disciplina de algoritmos na avaliação das soluções geradas pelos acadêmicos. Esta ferramenta efetua essa correção e ainda mantém o padrão e coerência na avaliação, bem como aproxima a forma de avaliação dos diversos professores da disciplina.

1.2 JUSTIFICATIVA

A Inteligência Artificial pode ser definida, conforme LUNGE (2004), como um ramo da ciência da computação que se ocupa da automação do comportamento inteligente. Existem várias técnicas de Inteligência Artificial, tais como Raciocínio Baseado em Casos, que procura resolver problemas baseando-se em casos passados, Sistemas Especialistas, que procuram se basear no especialista humano, e Processamento de Linguagem Natural, que procura estudar uma forma de que os computadores possam compreender a linguagem humana, escrita ou falada. Conforme BARR e FEIGENBAUM (1986), o estudo de processamento de linguagem natural procura fazer com que os computadores possam entender a linguagem natural humana, tornando-se mais fáceis de utilizar. Esta característica pode ser extremamente útil para utilização no ensino, pelo fato de haver uma maior interação entre estudantes e programa. Um exemplo seria a utilização de Processamento de Linguagem Natural na avaliação de algoritmos, através do uso de análise pragmática.

Um professor, ao ministrar a disciplina de algoritmos, pode se utilizar de portugol, como forma de auxílio à disciplina de algoritmos. ESMIN (2000) explica que Portugol é uma pseudo-linguagem algorítmica muito utilizada na descrição de algoritmos, destaca-se por usar comandos em português, facilitando o aprendizado da lógica de programação, e desta forma habituando o aluno com o formalismo da programação.

Os professores, ao corrigirem uma prova ou trabalho de algoritmos, devem avaliar uma série de fatores, tais como a parte léxica, sintática e semântica, pois cada aluno ordena seu pensamento de uma forma, produzindo-se diferentes soluções para um determinado problema. Na grande maioria dos cursos de computação, as turmas de algoritmos são turmas com mais de 50 alunos, por ser uma disciplina dos primeiros períodos, e mesmo pelo grau de dificuldade encontrado pelos alunos para sua compreensão, pela mudança da forma de pensamento. O professor, ao corrigir provas ou trabalhos, pode perder o padrão de correção da avaliação, correndo o risco de não gerar

o conceito correto para todos os alunos. Daí surge a idéia da geração de uma ferramenta que auxilie o professor nesta questão.

O estudo de compiladores, principalmente das análises léxica, sintática e semântica, e da análise pragmática utilizada em processamento de linguagem natural, bem como a implementação de uma ferramenta com o intuito de auxiliar o professor na avaliação dos algoritmos torna-se a principal justificativa para o desenvolvimento deste trabalho. Pode-se também acelerar a correção de avaliações pelo professor e manter o padrão e coerência dentre os alunos.

1.3 OBJETIVOS

1.3.1 Geral

Implementar um aplicativo que tenha por finalidade auxiliar o professor na avaliação de algoritmos desenvolvidos por acadêmicos, fornecendo nota e parecer a respeito do mesmo, utilizando-se de compiladores – análises léxica, sintática e semântica, e das técnicas de Processamento de Linguagem Natural – análise pragmática, e de Sistemas Especialistas.

1.3.2 Específicos

Os objetivos específicos desta dissertação são:

- Estudar e compreender todos os módulos da Ferramenta CIFluxProg, para desenvolvimento do sistema gerado por este trabalho.
- Pesquisar e compreender os conceitos a respeito da técnica de Inteligência Artificial denominada Processamento de Linguagem Natural, principalmente a parte referente à análise pragmática.
- Pesquisar a respeito de compiladores e forma de implementação dos mesmos.

- Desenvolver as regras de produção necessárias para a implementação do Sistema Especialista.
- Determinar os requisitos exigidos pelo sistema.
- Implementar o sistema proposto.

1.4 LIMITAÇÕES DO TRABALHO

Algumas limitações foram impostas no trabalho, pois o mesmo se tornaria inviável para ser encerrado em tempo hábil sem as mesmas.

Uma das limitações foi restringir as estruturas do próprio algoritmo. O trabalho não abrange estruturas como vetores, matrizes, funções, procedimentos e ponteiros. O trabalho se atém a algoritmos sem estrutura alguma, laços de repetição e laços de seleção.

Outra limitação imposta foram as quantidades de entradas e saídas a serem informadas pelo professor. Foram limitadas a duas entradas e quatro saídas no máximo, como uma forma de testar o algoritmo.

Os erros a serem encontrados nas análises morfológica (léxica), sintática, semântica e pragmática foram classificados conforme sua importância, não sendo abrangidos todos os erros. Esta classificação foi elaborada conforme reunião realizada com professores da disciplina de algoritmos.

1.5 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em seis capítulos, que versam sobre Educação na Informática, Algoritmos, Processamento de Linguagem Natural, Sistemas Especialistas, Compiladores, a metodologia empregada no desenvolvimento deste trabalho, a aplicação desenvolvida e finalmente conclusões e recomendações.

O Capítulo 2 apresenta a fundamentação teórica do trabalho, abordando os assuntos pertinentes ao entendimento do trabalho realizado, transcorrendo sobre Educação na Informática, Algoritmos, Processamento de Linguagem Natural, Sistemas Especialistas e Compiladores.

No Capítulo 3 é descrita a metodologia empregada no trabalho.

O Capítulo 4 contém a aplicação prática em si e a forma como foi desenvolvida.

Finalmente, no Capítulo 5 apresentam-se as conclusões e recomendações deste trabalho.

2. FUNDAMENTAÇÃO TEÓRICA

Na Fundamentação Teórica serão apresentados os tópicos referentes ao trabalho desenvolvido. Em Educação em Informática, será dada uma visão geral da educação superior voltada para o ensino de Informática. Logo em seguida é fornecida uma visão sobre algoritmos, a forma e ferramentas de auxílio ao ensino da disciplina de Algoritmos desenvolvidas. Também são abordados conceitos a respeito da técnica de Processamento de Linguagem Natural, bem como Compiladores e Sistema Especialista.

2.1 EDUCAÇÃO NA INFORMÁTICA

2.1.1 Introdução

Nos dias atuais, cada vez mais a informática é necessária e vital, podendo ser encontrada em todos os lugares, de pequenos comércios a grandes corporações. Por este motivo, o mercado exige profissionais mais qualificados, aumentando a demanda por ensino de informática, e por profissionais competentes e treinados para áreas específicas.

Conforme CHAVES (2003), o problema em educação referente ao ensino de informática é de extrema importância e precisa abranger todos os níveis de ensino, tanto no ensino formal quanto no informal. Esta preocupação torna-se realidade pelo fato de ter-se uma diversidade de cursos de informática, em todos os níveis, tais como cursos de graduação e pós-graduação, cursos técnicos em nível de segundo grau, pré-escola utilizando o computador como uma ferramenta lúdica de ensino, cursos específicos voltados para pessoas que desejam utilizar o computador como ferramenta de trabalho, e vários outros.

Dando continuidade a seu pensamento, CHAVES (2003) coloca que a distinção entre esses vários níveis não se dá somente pelo aprofundamento de um

mesmo conjunto de questões, mas também pelo conteúdo, enfoque, metodologia diferenciados de um para outro, dependendo do nível do curso ou dos objetivos educacionais, condicionados pela clientela a quem estão destinados os cursos ou programas de ensino.

Conforme a SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (2000), os cursos de graduação em informática e computação tem por objetivo formar profissionais para o desenvolvimento tecnológico da computação – tanto na área de hardware, software ou ambas, para aplicação das tecnologias e atendimento das necessidades da sociedade atual, bem como a formação de professores para o ensino médio e profissional. Dentre essas necessidades a serem atendidas, pode-se citar armazenamento de grandes volumes de informação e sua recuperação em tempo hábil, sistemas para apoio ao ensino e a aprendizagem, comunicação segura, rápida e confiável, entre outras.

SANTOS et. al (2003) informa que para formar um profissional de Ciência da Computação competente e com atuação transformadora no mercado de trabalho, é necessário destacar-se os seguintes aspectos científicos:

- raciocínio lógico-matemático, que o torna capaz de resolver problemas complexos, modela-los matematicamente, e construir soluções viáveis computacionalmente;
- assimilação e aplicação de novas tecnologias para soluções computacionais;
- capacidade em construir e definir conceitos fundamentais na área da computação, utilizando linguagens adequadas à formulação e à solução computacional dos problemas;
- capacidade de resolver, de forma eficiente, problemas em ambientes computacionais;

capacidade de discutir valores humanísticos, sociais, éticos e ambientais, incentivando o desenvolvimento do espírito crítico, e possuindo capacidade de liderança e trabalho em equipe.

FERNANDES (2002) destaca que existem quatro abordagens pedagógicas diferentes na introdução à computação:

- em largura: é apresentado ao aluno uma grande quantidade de conhecimentos, problemas e soluções fundamentais de computação e suas sub-áreas;
- por programação: o acadêmico aprende a programar em uma linguagem de programação específica, por um dos paradigmas a seguir: imperativo, funcional, ou orientado a objetos;
- por algoritmos: o aluno desenvolve e aprofunda sua habilidade em analisar e sintetizar algoritmos, ficando para uma próxima matéria o aprendizado de uma linguagem de programação;
- por hardware: o aluno tem contato com modelos de construção de artefatos de computação, partindo de elementos de circuitos digitais.

Dentre a área de formação básica, pode-se destacar o ensino de programação. CASTRO e CASTRO JÚNIOR (2002) coloca que a aprendizagem de conceitos e métodos para se construir programas de computador não é uma atividade trivial, por requerer o uso de habilidades de alto nível e raciocínio abstrato, sendo uma atividade desafiadora. SOCIEDADE BRASILEIRA DE COMPUTAÇÃO (2000) destaca que a programação deixou de ser uma “arte” e passou a ser ciência, contendo um conjunto de princípios, técnicas e formalismos, com o intuito de produzir softwares estruturados e confiáveis. Em grande parte das instituições de ensino superior que oferecem cursos de computação, a aprendizagem de programação se dá, inicialmente, pela disciplina de algoritmos, para posteriormente aprender uma linguagem de programação, ou mesmo aprende a linguagem de programação ao mesmo tempo em que aprende a reestruturar seu pensamento. A seguir, tem-se uma descrição de algoritmos, pois sobre sua avaliação é que trata este trabalho.

2.2 ALGORITMOS

2.2.1 Lógica

De acordo com OLIVEIRA (1998), a lógica em si pode ser atribuída à mente de Aristóteles, que codificou o assunto de tal forma que permanece por mais de dois milênios, e a lógica moderna tem por pai o alemão Gottlob Frege, do século XIX. A lógica desenvolveu-se na Filosofia, percorrendo ainda os caminhos da Lingüística, Matemática e Ciência da Computação.

SOUZA et al. (2000a) afirma que lógica é um termo muito amplo, sendo utilizado da filosofia até as ciências exatas, passando por todas as demais ciências. Uma definição do termo lógica, geralmente, estará vinculada à área que está sendo estudada. Uma definição generalizada poderia ser: lógica é o estudo do pensamento, do raciocínio, da resolução de problemas de forma coerente.

Segundo MANZANO e OLIVEIRA (1996), "a lógica é a ciência que estuda as leis e critérios de validade que regem o pensamento e a demonstração, ou seja, ciência dos princípios formais do raciocínio". De acordo com o mesmo, é necessário ter domínio sobre o pensamento, bem como possuir a "Arte de Pensar".

MANZANO e OLIVEIRA (1996) destaca que a lógica deve ser usada por todos, principalmente por profissionais de informática, pois por meio dela solucionam-se problemas com eficiência e eficácia, com o uso de recursos computacionais. A lógica não tem por intuito ensinar uma pessoa a pensar, mas sim desenvolver e aperfeiçoar esta técnica, sendo que a mesma deve ser praticada constantemente.

Conforme SOUZA et al. (2000a), a lógica liga-se intimamente à linguagem artificial e a linguagem natural. A lógica irá representar a semântica (estudo do significado dos símbolos) e a sintaxe (estudo da relação dos símbolos entre si).

SOUZA et. al (2000b) comenta que o aprimoramento do raciocínio lógico pode ser efetuado através da elaboração de soluções para problemas do mundo real, com o objetivo de enfatizar a naturalidade à lógica de procedimentos, seja para tarefas computacionais ou não computacionais, de tal forma que forneça um sólido alicerce de raciocínio para solucionar problemas, nos mais variados domínios de aplicação.

A lógica de programação, de acordo com ESMIN (2000), consiste em aprender a pensar na mesma seqüência de execução dos programas de computador. Aprende-se, dessa forma, a pensar como serão executadas as ações, partindo do estudo de um problema até chegar a construção de um algoritmo, que seria a solução deste mesmo problema.

Dando continuidade ao raciocínio, ESMIN (2000) comenta que, quanto maior o domínio da lógica de programação, mais fácil será o detalhamento das tarefas envolvidas na solução do problema proposto, e mais eficiente ainda será o algoritmo criado.

MANZANO e OLIVEIRA (1996) apresenta várias formas de um profissional de informática representar seu pensamento lógico, dentre as quais pode-se citar: fluxograma, diagrama de bloco e algoritmo.

O Fluxograma, de acordo com MANZANO e OLIVEIRA (1996), tem por principal função descrever o fluxo, manual ou mecânico, com ênfase nos dados e informações. Compõe-se de alguns desenhos geométricos básicos, que tem por função indicar a entrada, processamento e saída dos dados, acompanhados dos procedimentos de raciocínio lógico, que deverão solucionar o problema em questão. Já o Diagrama de Bloco tem por objetivo descrever o método e a seqüência do processo dos planos em um computador. É representado por símbolos geométricos, que estabelecerão as seqüências de operações a serem efetuadas em um processamento computacional. O algoritmo trata de uma série finita de passos com o intento de chegar a resolução de um determinado problema.

2.2.2 Algoritmo

De acordo com MANZANO e OLIVEIRA (1996), o termo algoritmo data do ano de 830 d.C. devido a um estudioso e matemático originário da Pérsia, de nome Mohammed Ibn Musa Abu Djefar, pelo mesmo ter escrito um importante livro sobre álgebra. Sendo conhecido por Al-Khwarismi, seu nome passou a ser muito usado, o que foi causando mudanças na pronúncia. De Al-Khwarism passou a Al-Karismi, Algarismi, chegando a Algarismo. Algarismo é a representação numérica do sistema de cálculos utilizado nos dias atuais, e deste radical provêm o termo **Algoritmo**, utilizado em computação.

Conforme SOUZA et. al (2000b), a idéia de utilizar algoritmos para controlar o funcionamento de um computador deve-se à Ada Augusta. Sua principal contribuição foi a introdução dos conceitos de sub-rotina, laços e salto condicional.

SALVETTI e BARBOSA (1998) definem algoritmo como uma seqüência finita de instruções ou operações básicas, que, quando executadas, resolvem um problema computacional de qualquer instância, apoiando-se na estratégia de ordenação da seqüência de instruções estabelecida durante a análise do problema. O desenvolvimento do mesmo não pode perder-se nos tipos de dados e sua representação.

Já MANZANO e OLIVEIRA (1996) conceitua-o como "um processo matemático ou de resolução de um grupo de problemas semelhantes, em que se estipulam, com generalidade e sem restrições". Pode-se ainda dizer que são regras formais com o intuito de obter um resultado ou solucionar um problema, através de fórmulas de expressões aritméticas.

Para a matemática, de acordo com SOUZA et al. (2000a), o algoritmo trata de um processo de cálculo, ou da resolução de um grupo de problemas semelhantes, no qual são estipuladas regras formais para se chegar a solução de um determinado problema, com generalidades e sem restrições.

Conforme DAZZI (1998), para a solução de um determinado problema podem existir diversos caminhos. Algoritmo é, exatamente, um dos caminhos para solucionar o mesmo.

Algoritmo é, para OLIVEIRA (1998), uma seqüência finita de passos lógicos escrito numa linguagem natural, em caso particular aqui pseudo-linguagem.

Dando continuidade ao seu pensamento, OLIVEIRA (1998) afirma que o melhor exemplo para a compreensão de um algoritmo é uma receita de bolo. Neste caso são descritos todos os passos para a construção de um bolo numa seqüência lógica, como por exemplo: um bolo não vai para o forno sem estar feito; os ovos não vão com a casca, pois antes são quebrados. Estes detalhes são de conhecimento da pessoa que está fazendo o bolo, e para isso ele segue uma seqüência de passos lógicos, que denomina-se receita (o algoritmo).

MANZANO e OLIVEIRA (1996) destaca que algoritmo, na realidade, é uma "receita" de como fazer. Informa, ainda, que é a transcrição, passo a passo, de um determinado problema, para chegar a solução do mesmo.

Segundo DAZZI (1998), o algoritmo deve possuir duas virtudes: legibilidade e portabilidade. Legibilidade trata da clareza do algoritmo, ou seja, ser compreendido por qualquer pessoa que não o tenha construído. Por sua vez, portabilidade diz questão ao algoritmo não ser orientado a uma determinada linguagem de programação, pois são várias as existentes. O algoritmo deve se preocupar com a lógica em si, e poder ser implementado em qualquer linguagem de programação.

WILT apud SOUZA et. al (2000b) aponta três fatores importantíssimos na elaboração de algoritmos: correção, eficiência e facilidade de implementação. Estes fatores exigem um projeto bem determinado, onde haja uma conjunção dos fatores: robustez (sem perder a eficiência), legibilidade e facilidade de execução das ações. Para mantê-los na solução de problemas complexos, é necessário haver algum tipo de metodologia para diminuir a complexidade do desenvolvimento.

Segundo KNUTH apud SOUZA et. al (2000b), as seguintes características distinguem um algoritmo de um conjunto de ações:

- um algoritmo sempre termina;
- cada ação é descrita sem ambigüidades e precisamente;
- cada ação é muito simples, sendo que pode ser executada em um intervalo de tempo; e
- um algoritmo sempre produz um ou mais resultados, podendo não exigir dados de entrada.

SALVETTI e BARBOSA (1998) destaca que sob o ponto de vista lógico, um algoritmo é constituído por três estruturas: seqüencial, repetitiva e seletiva. A constituição de um algoritmo baseia-se em qualquer combinação dessas três estruturas.

O desenvolvimento de um algoritmo, segundo SALVETTI e BARBOSA (1998), pode ser feito por meio da técnica *top-down*, a qual identifica partes ou etapas na resolução do problema. Inicialmente elabora-se um esboço, detalhando cada etapa, até obter uma seqüência de operações básicas sobre os tipos de dados considerados. A implementação pode ser realizada em qualquer linguagem de programação, podendo ser trivial (simples transcrição de operações básicas) ou trabalhosa (dependendo das características da linguagem escolhida e dos tipos de dados nela definidos).

Pode-se realizar a descrição de um algoritmo de duas formas: através de fluxograma ou de uma linguagem algorítmica, que será descrita a seguir.

2.2.2.1 Portugol

Portugol é uma pseudo-linguagem algorítmica muito utilizada na descrição de algoritmos, destaca-se por usar comandos em português, facilitando o aprendizado da lógica de programação, e desta forma habituando o iniciante com o formalismo da programação (ESMIN, 2000) .

Segundo MANZANO e OLIVEIRA (1996), o portugol pode ser classificado como uma técnica narrativa denominada de pseudocódigo, ou conhecida também por português estruturado. Baseia-se em uma PDL - *Program Design Language* (Linguagem de Projeto de Programação), sendo que sua forma original de escrita é conhecida como inglês estruturado, muito parecida com a notação da linguagem PASCAL.

Ainda segundo MANZANO e OLIVEIRA (1996), o portugol é usado como referência genérica para uma linguagem de projeto de programação, e tem por finalidade mostrar uma notação para elaborações de algoritmos, que será usada na definição, criação e desenvolvimento de uma linguagem computacional e sua documentação.

A representação de algoritmos em portugol, conforme ESMIN (2000), é rica em detalhes, como a definição dos tipos das variáveis usadas no algoritmo, encontrando muita aceitação por se assemelhar em demasia à forma em que são escritos os programas.

Comparando este processo aos fluxogramas destaca-se uma vantagem, visto que segundo MANZANO e OLIVEIRA (1996), é mais fácil escrever que desenhar (na grande maioria dos casos), e a codificação acaba se tornando uma simples transcrição de palavras chave. Corroborando ESMIN (2000) aponta o fato do Portugol ser uma linguagem simples e permitir o detalhamento dos algoritmos. A tradução de um algoritmo em Portugol para um programa computacional através de uma linguagem de programação é muito fácil e clara, o que facilita muito o ensino/aprendizagem da própria linguagem de programação. Outro destaque, de acordo com ASCÊNSIO (2002), é o fato de a passagem para qualquer linguagem de programação ser quase imediata, bastando ao aluno saber as palavras reservadas da respectiva linguagem de programação.

ESMIN (2000) salienta que o inconveniente dos algoritmos é o fato de não poderem ser executados no computador, pois o iniciante precisa imaginar a sua execução, e essa tarefa não é fácil. SOUZA et. al (2000b) apresenta outros problemas,

tais como a influência da linguagem de programação utilizada no momento da definição do algoritmo e o fato de não haver utilização de recursos visuais, o que deixa de estimular um fator importante no aprendizado dos alunos, que é a visão.

2.2.3 O Ensino de Algoritmos

O ensino de Algoritmos, em cursos de Computação, tem por objetivo ordenar o pensamento do aluno, fazendo com que o mesmo aprenda a pensar na mesma seqüência lógica utilizada pelo computador. A importância dessa disciplina será percebida mais tarde, quando o aluno iniciar o aprendizado em linguagens de programação e precisar ordenar os passos para resolução de um problema, repassando-os ao computador, sob forma de comandos.

Conforme BOOKSHEAR et al. (2000), o algoritmo é uma codificação do raciocínio necessário para resolver determinado problema, e essa capacidade torna possível a construção de máquinas com comportamento inteligente – sendo essa inteligência moldada através de software, o que torna a disciplina de algoritmos essencial ao campo de Computação. O próximo passo seria representar o algoritmo desenvolvido de uma forma apropriada ao entendimento da máquina ou do aluno, através de comandos compreensíveis e sem ambigüidade, ou seja, transportá-lo a uma linguagem de programação.

GIRAFFA et. al (2003) coloca que o objetivo da disciplina de algoritmos é o aluno desenvolver habilidades cognitivas que permitam que o mesmo aprenda a resolver problemas, utilizando-se do computador como ferramenta. Os professores, seguindo o raciocínio de GIRAFFA et. al (2003), devem se preocupar com fatores como as diferentes formas de resolução de um mesmo problema, e os diferentes estilos cognitivos de um aluno. De acordo com GARDNER apud DOMINGUES (2003), cada ser humano possui um conjunto inato de competências intelectuais humanas chamadas de Inteligências Múltiplas. Dentre estas inteligências, existe a classificada como Lógico-matemática, chamada também de raciocínio científico ou indutivo, que é a inteligência mais desenvolvida ou potencialmente mais desenvolvida nos alunos dos cursos de

Computação. GIRAFFA et. al (2003) ainda destaca que o tipo de inteligência predominante em um indivíduo não irá padronizar seu raciocínio ou a forma que o mesmo resolve problemas.

O professor da disciplina de algoritmos, de acordo com GIRAFFA et. al (2003), deve conduzir a disciplina de forma que o próprio aluno descubra seu estilo de raciocínio e forma de solução de problemas, auxiliando o aluno a modelar sua solução em forma de algoritmo.

Conforme MENEZES e NOBRE (2002), o professor da disciplina de algoritmos vivencia algumas dificuldades durante o processo de aprendizagem, destacando-se:

- reconhecer as habilidades inatas de seus alunos no processo de ensino;
- apresentar diferenciadas técnicas para solução dos problemas;
- trabalhar a capacidade do aluno de buscar soluções e escolha da estrutura a ser utilizada; e
- promover a cooperação e colaboração entre os alunos.

Outro problema seria a dificuldade dos alunos em abstrair do problema o que se deseja. Normalmente, na disciplina de algoritmos, é dada a teoria, logo após exemplos e exercícios referentes ao tema abordado. Estes exercícios normalmente são constituídos de uma parte textual, situando o aluno no problema. A grande dificuldade encontrada é abstrair deste texto o que se deseja, mais especificamente, as entradas, o processamento e a saída.

CASTRO e CASTRO JÚNIOR (2002) salienta que a abordagem tradicional, que é o curso baseado na programação imperativa, tem trazido sérios problemas aos alunos que nunca programaram. Esta abordagem faz com que o aluno comece a pensar um problema passo a passo, e isto não é o que é feito no dia a dia. KOLIVER et. al (2004) coloca que o despreparo da maior parte dos estudantes em questão de enfrentar uma disciplina que pretende auxilia-los no desenvolvimento de soluções para problemas

genéricos, em passos e de maneira lógica, é um dos fatores para os altos índices de desistência e reprovação.

KOLIVER et. al (2004) informa que outro fator que causa dificuldade aos professores é a heterogeneidade do público. Alguns acadêmicos possuem uma engenhosidade superior nata, o que faz com que possuam uma facilidade natural para a elaboração de soluções algorítmicas para problemas. Isso faz com que os instrutores tenham problemas em questão de definição do programa da disciplina e grau de complexidade dos problemas propostos.

Dando seqüência aos problemas encontrados, normalmente os professores utilizam no ensino de algoritmos a pseudo-linguagem Portugol, complementada pelo fluxograma (até parte da disciplina). Parte dos alunos sentem dificuldades em compreender estas estruturas. Conseguem analisar o problema, retirar dele as entradas, processamento e saídas solicitadas, mas no momento de transportar para a pseudo-linguagem, sentem dificuldade em compreender de que forma isso deve ser feito.

Outro ponto interessante para ser analisado é a forma como os estudantes chegam ao terceiro grau, conforme informa KOLIVER et. al (2004). Os calouros normalmente chegam carregando vícios oriundos de práticas mecanicistas freqüentes nos primeiro e segundo graus, que acostumam o estudante a realizar aplicações de fórmulas matemáticas sem realizar qualquer tipo de análise do problema. Desta forma, o acadêmico entra na universidade sem conhecer um método geral de resolução de problemas. Isto faz com que muitos professores da disciplina possuam uma expectativa equivocada relativa ao conhecimento do aluno, esperando que o mesmo possua habilidades para análise e resolução de problemas. A ausência de disciplinas no ensino secundário que abordem a lógica também é um fator determinante para os problemas encontrados. Mesmo aplicando a lógica no dia a dia, grande parte dos estudantes possuem problemas ao aplica-la de forma satisfatória na construção de soluções algorítmicas sem os princípios básicos que a norteiam.

Por ser uma disciplina inicial, e uma disciplina com um grau de dificuldade grande para os alunos que não conseguem se adaptar à forma de pensamento passo a passo, muitas vezes o professor sente dificuldades, durante as aulas, de avaliar qual é o problema encontrado pelos alunos. Normalmente as turmas de algoritmos são turmas grandes, e a maior parte dos alunos não evidenciam de forma verbal os problemas encontrados. Essas dúvidas somente se tornam claras durante a aplicação de uma prova, ou mesmo de exercícios válidos como nota.

Pode-se também citar, como problema, a questão da correção de provas e exercícios que o professor aplica aos alunos. Como citado anteriormente, normalmente as turmas da disciplina de Algoritmos são turmas grandes, primeiro por ser uma disciplina ministrada no primeiro período - portanto, todos os alunos que ingressam através do vestibular matriculam-se nela -, pelo alto índice de reprovação e também deve-se levar em conta as transferências realizadas. Ao corrigir provas de algoritmos, o professor deve levar em consideração que cada aluno resolve problemas de formas diferentes, portanto, um mesmo problema pode possuir diversos caminhos de resolução, e avaliar todos os problemas um a um, estabelecendo um padrão para a correção. Mas devido a grande quantidade de provas a serem corrigidas o professor, involuntariamente, poderá alterar a forma de avaliação de um aluno para outro.

Apesar dos problemas citados anteriormente, a disciplina de algoritmos, baseada em Portugol e fluxograma, ainda é ministrada na grande maioria dos cursos de Computação, pois através dela o aluno consegue ter embasamento para aprender diversas linguagens de programação. A disciplina de algoritmos em si ensina o aluno a compreender um problema, definir sua resolução e aplicar esta resolução no formato da pseudo-linguagem. Na grande maioria das vezes, juntamente ao ensino de Algoritmos é ensinado aos acadêmicos alguma linguagem de programação, normalmente C ou Pascal. Com o algoritmo elaborado, torna-se mais fácil o desenvolvimento do programa, pois a parte essencial, a parte lógica do problema, já está resolvida. Portanto, o aluno deve somente transferir a resolução para a linguagem de programação.

2.2.4 Ferramentas de Auxílio ao Ensino de Algoritmos

Nas próximas seções, algumas ferramentas desenvolvidas para auxiliar o ensino da disciplina de algoritmos serão apresentadas.

2.2.4.1 ASA

Conforme MAYERHOFER (1995), a ferramenta ASA (Animação e Simulação de Algoritmos) – desenvolvida pelo SENAC, permite ao usuário construir e testar passo a passo, seus algoritmos. No entanto, a mesma utiliza um formato que engloba portugol e fluxograma em uma só solução, apresentando as instruções escritas (portugol) e setas de direção interligando as instruções, o que demonstra a direção do fluxo de execução do algoritmo.

O ASA aborda Sistemas Numéricos, Memória, Variáveis, Operações Lógicas, Operações Seqüenciais, Laços de Seleção, Laços de Repetição (Enquanto ... Faça e Para ... Faça), Representações (Fluxogramas, Diagramas de Nassi-Shneiderman e Pseudocódigo), Vetores, Ordenação e Matrizes. Ele é estruturado em três módulos, sendo: Lições, Construtor e Analisador (MAYERHOFER, 1995).

O módulo das lições é onde o aluno irá ter contato com a nova matéria a ser apresentada. A apresentação é como as lições dadas no método normal, seja através de um livro texto, aula expositiva ou fita de vídeo. As animações neste módulo são divididas em duas categorias (MAYERHOFER, 1995):

- Conceito: representações concretas de algo abstrato a ser ensinado, como a representação da memória do computador.
- Simulação: correspondem à representação mais plausível possível do mundo real.

A aplicação do que foi ensinado no módulo de lições é realizado no módulo construtor. O aluno cria e executa a simulação, que pode ter mais que uma

representação. Já o módulo Analisador trata da correção e análise do algoritmo desenvolvido.

A utilização da divisão Lições, Construtor e Analisador permite aos desenvolvedores de software educacional lançar mão do recurso de visualização, operar com um grande volume de exercícios e empregar meios para rastrear o processo de aprendizagem e construção do conhecimento.

2.2.4.2 Portugol/Plus

De acordo com ESMIN (2000), Portugol/Plus é uma ferramenta de apoio à lógica de programação baseado em Portugol, possuindo um editor de algoritmos e um compilador. Basicamente, o acadêmico digitaria seu código em Portugol, compilaria o mesmo e seria gerado um programa objeto na linguagem de programação Pascal, e através do compilador Pascal é executado. O acadêmico poderia observar, tal como uma linguagem de programação, seu programa em funcionamento.

2.2.4.3 AMBAP

Conforme ALMEIDA et al. (2002), o ambiente AMBAP tem por intuito auxiliar o aluno iniciante no aprendizado de programação. Ele permite que o aluno construa seu programa em uma linguagem algorítmica, executá-lo e compreender conceitos como variáveis, comandos, recursão, através da simulação. A arquitetura do sistema conta com:

- Módulo Simulador: representa graficamente os resultados das ações do módulo interpretador e gera uma interface de entrada e saída de dados com o usuário. Este módulo é composto pelo Gerenciador de Entrada e Saída (interface amigável de entrada e saída durante a execução de uma solução do aluno), Analisador de Expressões (mostra com detalhes a realização da avaliação das expressões executadas no momento pelo interpretador) e Simulador Gráfico de Memória (mostra, passo a passo, o resultado das ações

do interpretador em uma forma gráfica, simulando o estado abstrato da memória do computador).

- Interpretador: executa as ações definidas pelo usuário, podendo ser em quatro linguagens diferentes – fluxograma, portugol, um conjunto de assembly ou um subconjunto de linguagem de microprogramação.
- Editor: oferece um ambiente adequado para descrever a solução do programa proposto, independente de qual das quatro linguagens.
- Tradutor: responsável pela conversão automática entre as quatro linguagens definidas anteriormente.
- Tutor: auxilia e supervisiona o aluno durante o processo de aprendizado.

2.2.4.4 CIFluxProg

Conforme SANTIAGO e DAZZI (2004), o CIFluxProg permite a construção e execução de fluxogramas e algoritmos desenvolvidos em Português Estruturado (Portugol), o que torna o aprendizado dos conceitos de Algoritmos mais claros aos acadêmicos de primeiro período dos cursos de Informática. Ao elaborarem o algoritmo e testarem no ambiente, os acadêmicos podem visualizar se o algoritmo em Portugol ou Fluxograma foi elaborado corretamente ou não, e se existem erros sintáticos e/ou semânticos.

Dando continuidade, SANTIAGO e DAZZI (2004) informa que a ferramenta é composta por dois ambientes distintos: um ambiente para desenvolvimento de Fluxogramas, e outro ambiente para elaboração de algoritmos em Português Estruturado. Ambos contam um ambiente gráfico, arquivo de ajuda, e seu código pode ser executado através do interpretador de código. O usuário pode, também, executar funções padrão como abrir arquivos de código existentes e salvar algoritmos apresentados. Todas as soluções implementadas em um módulo podem ser abertas em outro, ou seja, se o acadêmico formulou um algoritmo em portugol e deseja ver sua execução em fluxograma, é só abrí-lo no módulo fluxograma.

SANTIAGO e DAZZI (2004) descreve o módulo de portugol (Figura 1) da seguinte forma: o usuário deverá digitar o algoritmo na caixa de texto disponibilizada pelo ambiente. Tão logo o usuário entre, é solicitado o nome do novo algoritmo, e o ambiente monta a estrutura primária, que seria:

```
programa <nome fornecido pelo usuário>
declarações
inicio
fim
```

O usuário irá digitar o algoritmo entre a estrutura primária, e poderá contar com a ajuda de uma barra de estruturas, onde poderá encontrar os comandos primários leia e escreva, a estrutura de seleção SE ... SENÃO, e os laços de repetição PARA .. FAÇA, FAÇA ... ENQUANTO e ENQUANTO ... FAÇA. Após digitar o algoritmo, ao clicar em executar, o interpretador do algoritmo entra em ação, e realiza a análise morfológica e sintática do algoritmo, e, ao encontrar erro, para a execução do programa e exibe o erro encontrado. Conforme o algoritmo vai sendo executado, em uma janela a direita é exibido o teste de mesa do mesmo. A Figura 1 mostra a interface do ambiente no módulo de Portugol.

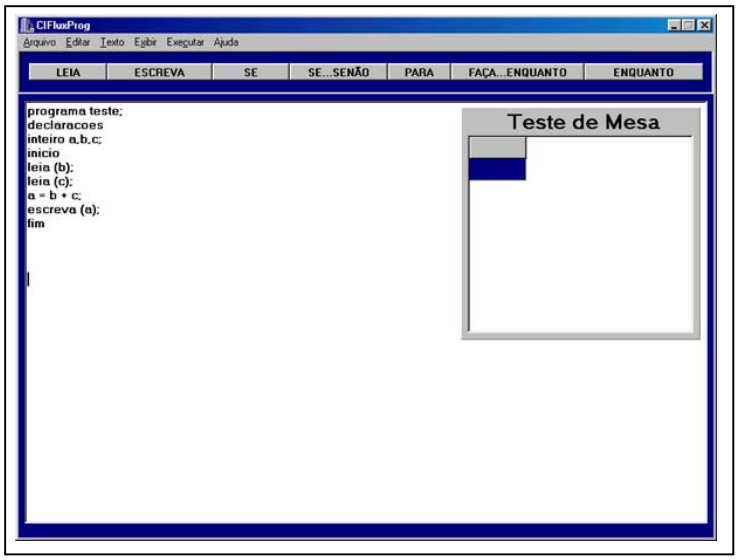

Figura 1: Interface do ambiente CIFluxProg no módulo de Portugol

O módulo de Fluxograma (Figura 2) possui uma caixa de edição, uma barra de ferramentas com os principais símbolos representativos dos comandos principais e uma barra de menus. Todos os símbolos da barra de ferramentas possuem campos editáveis, onde serão inseridos nomes e valores de variáveis, condições lógicas, etc. Um detalhe importante deste módulo é o suporte ao aninhamento de símbolos (SANTIAGO, 2004).

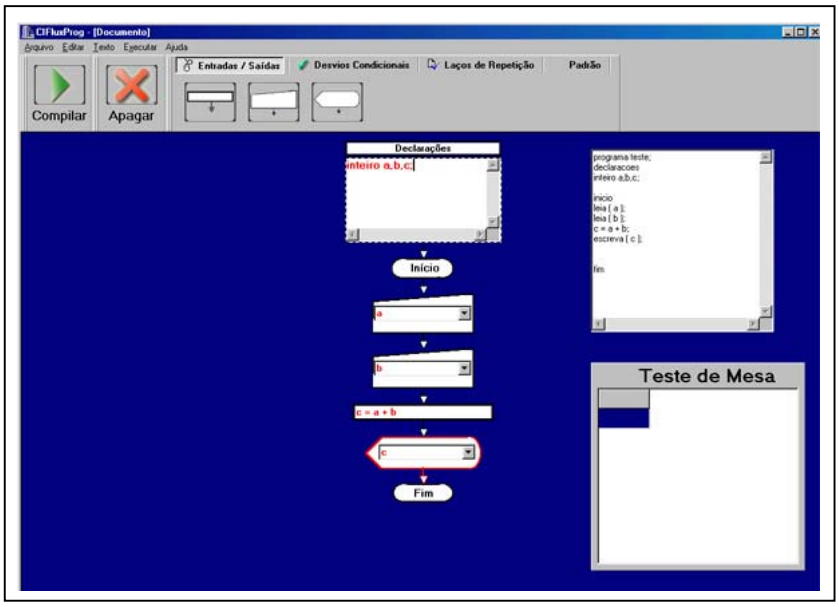

Figura 2: Ambiente de desenvolvimento de Fluxogramas

Aninhamento é quando se tem dentro de um símbolo o mesmo símbolo ou outro do mesmo tipo, por exemplo, de um laço de repetição um outro símbolo do mesmo tipo. Este tipo de encadeamento deve ser suportado por um sistema que trabalhe com algoritmos, pois é bastante usado na resolução de problemas (SANTIAGO, 2004).

Da mesma forma que o módulo de algoritmos, tão logo se inicie um novo programa são inseridas as estruturas básicas, e o usuário trabalha a partir daí. O usuário pode contar também com o teste de mesa, conforme o algoritmo gerado pelo mesmo for sendo executado, e com uma versão em portugol do algoritmo gerado em fluxograma. Conforme o acadêmico for inserindo os respectivos símbolos e dados, vai sendo exibido na caixa de texto na lateral direita a versão do mesmo algoritmo em Português Estruturado.

SANTIAGO e DAZZI (2004) informa que a ferramenta foi levada à sala de aula para realização de testes. Foram passados aos acadêmicos dois enunciados para serem desenvolvidos no CIFluxProg. Logo após, foi repassado aos acadêmicos um questionário, para que os mesmos avaliassem a utilização da ferramenta, e o quanto a mesma facilitou na resolução de problemas, e possíveis pontos a serem modificados. A ferramenta mostrou-se funcional, e um excelente auxílio ao professor da disciplina de

Algoritmos, pelo fato de os acadêmicos poderem visualizar suas resoluções aos problemas apresentados, aumentando assim sua compreensão da matéria.

2.2.4.5 RAFF

Conforme SOUSA et al. (2004), o RAFF é um compilador com gramática parecida com a linguagem de Programação C que utiliza termos da Língua Portuguesa em seu vocabulário, sendo de simples compreensão, principalmente para os iniciantes em programação. O compilador foi desenvolvido em Borland C++ Builder.

Dando continuidade, SOUSA et al. (2004) informa que o compilador desenvolvido possui quatro módulos:

- Analisador Léxico: produz uma seqüência de tokens utilizados na análise sintática e elimina comentários e caracteres neutros.
- Analisador Sintático: relata erros de sintaxe encontrados no código fonte e realiza a verificação de tipos.
- Analisador Semântico: verifica erros semânticos e captura as informações de tipo para a fase de geração de código.
- Gerador do Código.

Além disto, fornece os seguintes relatórios:

- Relatório dos erros ocorridos durante alguma das fases da compilação;
- Relatório com o número de linhas compiladas, tamanho do arquivo executável e tempo de compilação.
- Código fonte do programa.

2.2.4.5 SistLog

Conforme MIRANDA (2000), o SistLog foi desenvolvido em Borland Delphi 5.0, utilizando a técnica de Inteligência Artificial denominada Raciocínio Baseado em

Casos, e tendo por objetivo auxiliar e padronizar a forma de avaliação de provas e exercícios da disciplina de Algoritmos e Programação.

No sistema desenvolvido, MIRANDA (2000) informa que o usuário responde a nove questões:

- A lógica do programa está correta?
- As estruturas de controle foram utilizadas adequadamente?
- As saídas são as solicitadas no problema?
- Os dados que o usuário deve informar foram lidos?
- As variáveis foram declaradas?
- Todas as variáveis foram declaradas corretamente quanto ao tipo?
- As variáveis que precisam ser inicializadas foram inicializadas corretamente?
- Todas as variáveis declaradas foram utilizadas?
- Quanto a sintaxe, o algoritmo está correto?

Após responder as perguntas, o usuário fornecerá os pesos de cada item, de cada questão e o peso de cada item para a recuperação dos casos similares. O sistema pesquisará a base de raciocínio baseado em casos para encontrar a nota do aluno, com 98% de similaridade. Após encontrar a nota, será realizada uma nova pesquisa em outra base de raciocínio baseado em casos, que fornecerá ao professor observações pertinentes ao aluno, baseadas na média por cada item (MIRANDA, 2000).

2.2.4.6 Ambiente SICAS

O ambiente SICAS, de acordo com MENDES e GOMES (2004), é um ambiente que tem por finalidade facilitar o aprendizado do aluno em programação, pois o ambiente permite que os acadêmicos inventem estratégias e mecanismos para a construção de algoritmos que resolvam problemas de programação, sendo deixado de lado aspectos como a análise sintática. O ambiente foi construído desta forma pelo fato de a dificuldade na construção de um algoritmo residir na concepção do mesmo, e não

na codificação da solução construída. O ambiente é dividido em dois módulos: modo professor e modo aluno.

O Modo do Professor permite que o professor oriente e verifique as soluções elaboradas pelos alunos. O professor pode indicar enunciados de problemas e fornecer soluções a serem consultadas, testadas e editadas pelos acadêmicos. Estas soluções podem estar corretas ou não, dependendo da estratégia pedagógica a ser aplicada pelo professor. O professor também pode acessar as soluções elaboradas pelos alunos para verificação dos problemas enfrentados pelos mesmos, e também fornecer, para cada enunciado, um conjunto de dados de entrada e os respectivos resultados esperados, de forma que os acadêmicos possam testar seus algoritmos. MENDES e GOMES (2004).

O acadêmico, em seu módulo, possui um ambiente para resolução de problemas propostos. O acadêmico poderá visualizar uma simulação animada de sua resolução, e se o professor cadastrou o conjunto de dados de entrada e respectivas saídas, poder testar se o algoritmo elaborado se comporta bem em qualquer caso. Na Figura 3 pode ser visualizada a área de edição do algoritmo. MENDES e GOMES (2004).

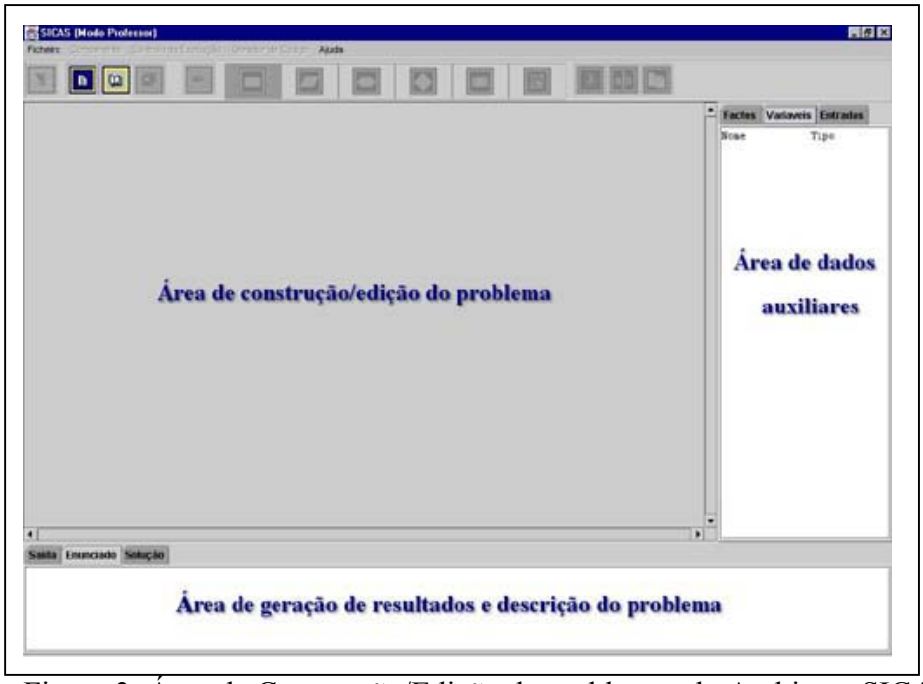

Figura 3: Área de Construção/Edição de problemas do Ambiente SICAS

Na área de construção/edição do problema, o acadêmico não precisa, necessariamente, iniciar por uma nova resolução. Ele pode editar/alterar uma resolução sua ou disponibilizada pelo professor. Caso o acadêmico opte por iniciar uma resolução desde o princípio, é direcionado para um ambiente de criação onde, em forma de fluxograma, poderá construir a seqüência de operações que executadas produzirão a solução do problema. MENDES e GOMES (2004).

A área de geração de resultados e descrição do problema, conforme MENDES e GOMES (2004), é dividida nas subseções Saída, Enunciado e Solução. Em Saída pode-se encontrar a saída gerada pela resolução do acadêmico e também o respectivo código em Linguagem C ou Linguagem JAVA, caso alguma destas soluções tenha sido selecionada. Na subseção enunciado é apresentado o enunciado a ser resolvido. Em solução é indicado, pelo professor, um conjunto de caracteres representativos da saída esperada para o problema definido, assumindo o conjunto de valores de entrada estabelecidos.

Por fim, a área de dados auxiliares possui as subseções Fatos, Variáveis e Entradas. Os fatos são úteis durante a simulação da resolução construída pois permitem conhecer, durante a execução, a correspondência entre cada variável e o valor associado a ela. A subseção variáveis é uma espécie de ambiente de consulta rápida que fornece informações sobre as variáveis declaradas e seus respectivos tipos. Já a subseção Entradas, mesmo que não seja visível no modo aluno, tem por objetivo permitir que o acadêmico, em conjunto com a subseção Solução, teste sua resolução.

Ao encerrar o processo de construção do problema, a solução pode ser simulada pelo sistema, através do modo de execução. Ao acadêmico são permitidos os seguintes controles durante o progresso de uma execução:

- Direção da execução: o acadêmico pode selecionar se deseja que a execução se processe pelas vias normais, ou seja, da primeira a última instrução, ou que possa recuar na execução.

- Movimento Realizado: o acadêmico pode optar pela execução passo-a-passo (avançando de uma instrução para outra só com autorização do acadêmico), passo lento (execução contínua mas lenta) ou passo ininterrupto mas acelerado (rápido).

Além destes controles, o acadêmico pode parar a resolução a qualquer momento e retornar mais tarde, cada componente a ser executado é destacado para que o acadêmico possa se situar no que está ocorrendo, e pode acompanhar as seções fatos e saídas. A Figura 4 mostra a execução de uma resolução.

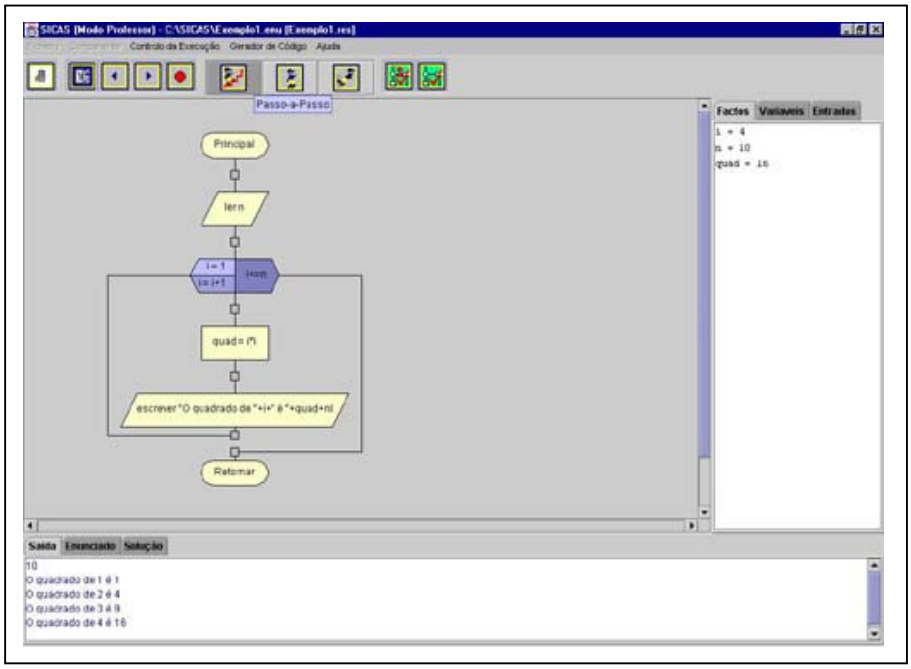

Figura 4: Execução de uma solução no Ambiente SICAS

O ambiente está sendo avaliado por professores com experiência em ensino de programação, e todas as sugestões/correções estão sendo registradas, analisadas e repensadas para a elaboração de uma nova versão da ferramenta.

2.2.4.7 C-Tutor

O C-Tutor, de acordo com SONG et al (1997), é um Sistema Tutor Inteligente que tem por objetivo ensinar a linguagem de programação C para iniciantes. Ele é

composto por dois subsistemas: um programa analisador e um ambiente de aprendizagem.

O programa analisador é composto por um sistema de engenharia reversa e um sistema didático. O programa Ex-Bug (*Execution-guided* de Bugger) realiza a análise dos programas do estudante, e GOES (*Goal Extraction System*) é um sistema de engenharia reversa que gera uma descrição do problema tomando por base um programa fonte. O programa fonte é gerado pelo professor, e considerado como o código correto. Ele também transforma as entradas do programa em forma canônica com o objetivo de absorver variações sintáticas. Já o ambiente de ensino do C-Tutor é o Curriculum Network, que constrói o conhecimento de metas e planos como gráficos genéticos, e ensina conceitos e habilidades em programação. A figura 5 mostra uma visão geral do C –Tutor. SONG et al (1997).

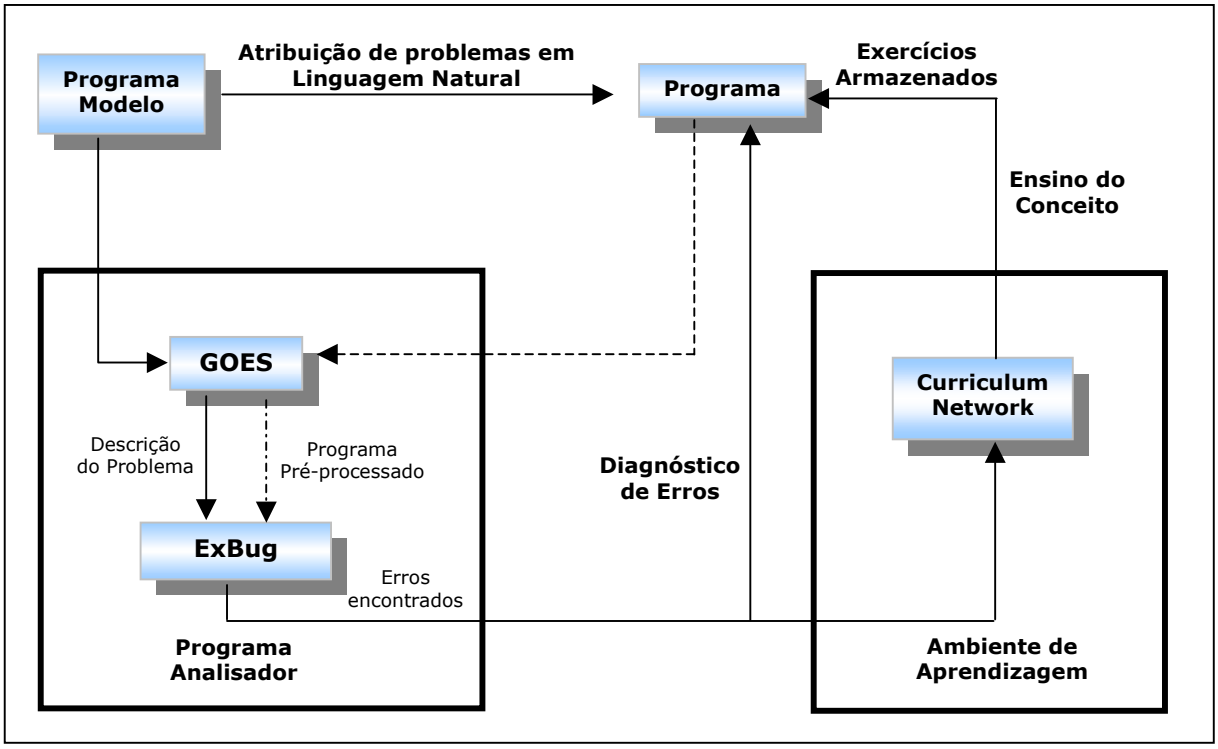

Figura 5: Visão geral do C-Tutor

Fonte: adaptado de SONG et al (1997).

A base de conhecimento do C-Tutor é representada por uma estrutura de frame. As metas e planejamentos são os principais componentes da base de conhecimento.

Planos descrevem seqüências de ação estereotipadas nos programas, e metas representam conceitos comuns de programação tais como a troca de dois valores ou mesmo encontrar o maior valor entre informações dadas. SONG et al (1997).

Existem dois módulos no C-Tutor: o sistema do estudante e o sistema do estudante-professor. No módulo que se refere ao modelo do estudante no Curriculum Network, o sistema ensina um conceito de programação e apresenta exercícios. O programa gerado é enviado ao GOES para eliminar as variações sintáticas. O ExBug analisa o programa processado, e fornece um diagnóstico baseado em intenção. Baseado neste diagnóstico, o sistema decide se deve apresentar ao aluno um novo exercício, ou um novo conceito. No módulo estudante-professor existe um agente externo, que é o professor, e o ExBug é uma espécie de assistente do professor em um curso real de C. Tomando por base o problema descrito pelo professor, o estudante escreve um programa e GOES o transforma em uma forma canônica, para então o ExBug analisa-lo. SONG et al (1997).

A análise de um programa gerado por um acadêmico novato é difícil, pois geralmente contém muitos erros. Ao tratar-se do C-Tutor, o GOES gera automaticamente uma descrição do problema elaborado pelo professor. Esta descrição é gerada da seguinte forma: em primeiro lugar são extraídos planos do programa gerado pelo professor. Depois, são extraídas as metas dos planos em uma hierarquia metas/planos. Por fim são adicionadas informações sobre os objetos a serem usados pelo programa. SONG et al (1997).

Já o ExBug, por tratar-se de um analisador de programas baseado em conhecimento, possui duas formas de análise: análise dinâmica e análise estática. A análise dinâmica encontra os erros através de sintomas dos erros gerados durante a execução do programa, e a análise estática detectam os erros através da análise do comportamento interno ou verificação do programa. A análise estática pode detectar erros difíceis de encontrar durante a análise dinâmica, mas para haver uma análise completa é necessário utilizar as duas.

O ambiente de aprendizagem é construído como um Curriculum Network. A aprendizagem é composta de três ciclos: a fase do estudo dos conceitos, a fase de programação e a fase de revisão. Na fase de estudo dos conceitos, o sistema seleciona o conceito mais fácil não estudado na base de conhecimento e ensina o conceito. Então o sistema seleciona um exercício apropriado relacionado com o plano e todas as metas são estudadas. O ExBug analisa o programa do estudante e fornece um diagnóstico baseado em intenção para o estudante. Se o estudante falhou na programação de algum plano, o sistema vai para a fase de revisão, onde o sistema mostra os conceitos do plano com a mensagem de retorno do ExBug. Depois da fase de revisão, o sistema apresenta um novo problema para o estudante. A fase de programação e revisão serão repetidas até o problema estar correto.

2.2.4.8 PL-Detective

A ferramenta PL-Detective, conforme DIWAN et. al (2004), constrói associações e demonstrações de um curso para fundamentos de linguagem de programação. Ele suporta uma linguagem denominada MYSTERY, e exporta sete interfaces, em que cada uma controla a semântica de um simples aspecto do MYSTERY.

O conteúdo é estruturado de duas formas. Em um primeiro momento, que seria a análise da linguagem, é estruturado como um quebra-cabeças. Os estudantes procuram descobrir a semântica de uma determinada implementação MYSTERY, através da execução de programas e observação dos resultados. Na segunda parte, os estudantes selecionam implementações para a interface semântica de forma que um dado programa MYSTERY produza resultados específicos. Em ambas as associações, o instrutor limita a quantidade de tentativas de cada grupo, para prevenir que os alunos tentem por tentativa e erro. (DIWAN et. al, 2004).

Levando-se em conta a perspectiva do estudante, o objetivo da análise é determinar a semântica de uma implementação MYSTERY interrogando o PL-Detective, através da submissão de programas em uma interface web. A resposta do

sistema é alguma saída produzida durante a execução do programa submetido. Recebendo a resposta o grupo deve decidir se a semântica está correta ou pela submissão de um novo programa. Então o grupo produz um relatório que define as descobertas semânticas e descreve como o grupo chegou a suas conclusões. (DIWAN et. al, 2004).

Quanto a perspectiva do professor, o objetivo do mesmo é levar o estudante a entender os conceitos fazendo uma reflexão sobre as diferentes interpretações destes conceitos. Em primeiro lugar o professor deve decidir qual a interface semântica mais relevante para o material a ser exibido. O professor seleciona a implementação de todas as interfaces que não são relevantes para a tarefa em questão e informa aos estudantes o que as implementações são. Já as interfaces relevantes podem seguir por dois caminhos: (i) o professor pode escolher suas implementações; (ii) o professor pode usar um procedimento de randomização para gerar uma configuração diferente para cada grupo.

Quando um estudante testa o sistema, o teste pode ou não compilar com sucesso. Compilando com sucesso, ele pode ou não executar com sucesso. No caso de não se obter sucesso na compilação ou execução, deve-se providenciar saídas que sejam úteis para o estudante, mas que não detalhem tanto como se resolve o problema.

A seguir, é descrita a técnica de Processamento de Linguagem Natural, bem como suas etapas, que serão utilizadas no desenvolvimento das análises referentes ao sistema desenvolvido.

2.3 PROCESSAMENTO DE LINGUAGEM NATURAL

2.3.1 Linguagem Natural

A comunicação, de acordo com LACERDA (1996) é uma atividade diária complexa, envolvendo sentidos como visão, tato, audição e uma capacidade motora

apurada, que permite o uso perfeito da língua e das mãos. O responsável pelo funcionamento das funções sensoriais e motoras, mais o armazenamento de informações e regras lingüísticas utilizadas é o cérebro. Um só problema com o cérebro pode comprometer totalmente a capacidade de comunicação de um indivíduo, tornando-o parcial ou completamente isolado do mundo.

RUSSEL E NORVING (2003) colocam que a comunicação é a troca de informações através da produção e percepção de sinais retirados de um sistema de sinais convencionais. O que distingue os seres humanos dos animais é o sistema complexo de mensagens estruturadas denominado linguagem que permite com que o ser humano comunique a maior parte do que sabe sobre o mundo. LUGER (2002) informa que a comunicação através da linguagem natural – por texto ou por fala -, depende muito do conhecimento e expectativas referentes ao domínio do discurso. A compreensão da linguagem não trata somente de transmissão das palavras: existem inferências sobre o objetivo, conhecimento e suposições do locutor, como também o contexto da interação.

LACERDA (1996) destaca que quando uma pessoa fala algo a outra, a mensagem é comunicada de um emissor para um receptor, sendo que o receptor, para compreender, deve:

- captar a mensagem, seja por som, gestos ou por escrita;
- identificar as palavras e suas conexões, e verificar se são aceitáveis;
- verificar se a inter-relação entre as palavras está de acordo com um padrão aceitável pela linguagem usada;
- verificar se o significado das palavras combinam entre si;
- procurar identificar qual o significado da mensagem.

Ao identificar o significado da mensagem, o receptor tem condições de compreender o que lhe foi dito, reagindo com outra resposta ou de qualquer outra forma.

RUSSEL E NORVING (2003) salientam que o principal objetivo da linguagem natural é a comunicação entre as pessoas, e isso só é atingido quando elas se entendem. As pessoas comunicam-se em línguas diversificadas, utilizando uma forma comum de falar (sentenças com mesma forma, sotaques, expressões típicas, etc). Para que se elaborem frases sintaticamente corretas, usa-se por base as gramáticas existentes, que objetivam estabelecer um padrão comum de comunicação.

Um computador para interpretar uma comunicação em processamento em linguagem natural, conforme GEVARTER (1984), é necessário os seguintes conhecimentos:

- estrutura das sentenças;
- significado das palavras;
- morfologia das palavras;
- modelo de crenças;
- regras de conversação;
- um conjunto de informações gerais sobre a palavra.

OLIVEIRA (1997) informa que uma língua natural é formada por um conjunto de orações (conjunto de frases), e as frases podem ser divididas em palavras. As palavras podem ser subdivididas em unidades mínimas de sons (fonemas) e de significados (morfemas). FARACO (2000) coloca que o morfema irá estudar as palavras do ponto de vista morfológico, ou seja, quanto a sua forma e alguns aspectos de sua flexão.

Dando continuidade, FARACO (2000) informa que o morfema pode se dividir em dois: Morfema Lexical e Morfema Gramatical. O morfema lexical é à parte do significado real da palavra, que se relaciona ao mundo extralingüístico. Pode-se citar como exemplo a palavra **esquerdo**. O morfema **esquerd** significa posição no espaço. Este é um elemento fixo, ao qual se anexam outros morfemas, sendo chamado de radical, raiz, ou morfema lexical da palavra. ANDRÉ (1982) coloca que a raiz é o elemento fundamental da palavra, não podendo ser decomposto, e possuindo o sentido

básico, sendo comum às palavras da mesma família. Já o morfema gramatical pode ser citado como a porção da palavra que a situa no sistema lingüístico a qual pertence. Os morfemas gramaticais não ocorrem livremente, devendo se unir a radicais para permitir a flexão das palavras, ou a criação de novos termos lingüísticos. Para se indicar o feminino da palavra médico, por exemplo, emprega-se o morfema gramatical **a**, gerando a palavra médica.

Conforme FARACO (2000), ao se querer qualificar uma pessoa ou situação relacionada secundariamente com a medicina, forma-se uma nova palavra ao unir-se a palavra original o morfema gramatical **para**, formando-se **paramédico**. Pode-se considerar morfemas gramaticais a vogal temática, desinências e afixos.

ANDRÉ (1982) informa que vogal temática é a vogal acrescida ao radical de certas palavras, que prepara o radical para receber as desinências, permitindo noções gramaticais como plural, tempo, etc. Já as desinências são os elementos que encerram as palavras, indicando as flexões gramaticais. Podem ser nominais, verbais e verbo-nominais. Afixos são elementos que se agregam à raiz (ou radical) para formação de uma nova palavra, derivada da primeira. Podem ser divididos em prefixos (se antepõem à raiz. Exemplo: **RE**luzir) e sufixos (se pospõem a raiz. Exemplo: sapat**EIRO**)

As palavras podem ser classificadas em classes ou categorias, conforme OLIVEIRA (1997), de acordo com sua denotação. Uma palavra pode pertencer a uma ou mais categorias sintáticas, dependendo da posição em uma oração. FARACO (2000) informa que conforme a NGB (Nomenclatura Gramatical Brasileira), as palavras da língua portuguesa são divididas em dez classes, denominadas classes ou categorias gramaticais. São elas substantivo, adjetivo, verbo, pronome, numeral, artigo, advérbio, preposição, conjunção e interjeição. Estas classes podem ser variáveis – palavras que apresentam mudança na forma, quando contextualizadas ou invariáveis – não apresentam mudanças na forma, mesmo quando contextualizadas.

OLIVEIRA (1997) coloca que frase é um conjunto de palavras com um significado completo, podendo descrever ações, expressar emoções, etc. ANDRÉ

(1982) coloca que uma frase é uma unidade de discurso suficiente para prover comunicação, entendendo-se por discurso a língua apreciada na fala ou na escrita. A frase pode ser formada por uma palavra, uma expressão, uma oração ou um período, e pode possuir verbo ou não. O que importa é a manifestação de um propósito de comunicação.

A análise sintática, de acordo com FARACO (2000), tem por objetivo estudar os períodos e orações que o compõe. Para isso, divide-se o período (frase formada por uma ou mais orações) em orações (frase ou parte de frase que se organiza em torno de um verbo ou de uma locução verbal), e depois determina-se a função de cada termo da oração.

Como sintagma, de acordo com OLIVEIRA (1997), pode-se entender os elementos da estrutura da oração, ou então como o grupo de palavras classificado conforme a categoria sintática do seu elemento núcleo. Os sintagmas nominais possuem como elemento núcleo um substantivo, os sintagmas verbais possuem como núcleo um verbo, e assim por diante. Pode-se citar como exemplo:

Sintagma Nominal: Ana, O rapaz, A uva verde.
Sintagma Verbal: chegou cedo, leu uma carta.

2.3.2 Histórico de Processamento de Linguagem Natural

Conforme LACERDA (1996), a pesquisa envolvendo lingüística computacional iniciou-se nos anos 40. A habilidade do computador em manipular símbolos foi aproveitada para compilar listas com ocorrência de palavras e concordâncias, mas logo se tornou claro que o computador poderia executar funções lingüísticas muito mais poderosas.

Dando continuidade, LACERDA (1996) destaca que em 1949 Warren Weaver propôs que os computadores poderiam ser úteis na função de tradutores. Desta pesquisa

resultou a chamada Tradução por Máquina, que era realizada através da procura de palavras em um dicionário bilíngüe, que mapeava na língua de saída às palavras expressas na língua de entrada. Esta abordagem encontrou problemas relativos à seleção das palavras apropriadas e o trabalho de organizá-las em uma sentença na língua de saída. Posteriormente, este conceito de tradução foi abandonado.

OLIVEIRA (2002) afirma que nos anos 60 os computadores já possuíam a habilidade de aceitar e responder questões em inglês relacionadas a vários assuntos – álgebra, medicina, relações de parentesco, entre outras, e podiam conduzir uma pesquisa psiquiátrica, a nível rudimentar, em inglês, galês ou alemão. LACERDA (1996) destaca que esse novo grupo de programas marcaram o início do trabalho, em Inteligência Artificial, sobre o entendimento da linguagem natural. O foco principal da pesquisa em Processamento de Linguagem Natural tornou-se a compreensão pelo computador de uma sentença humana.

Pode-se identificar quatro categorias históricas de programas em linguagem natural, conforme OLIVEIRA (2002):

- Programas que objetivavam a geração de um número reduzido de resultados em domínios específicos, como o BASEBALL, SAD-SAM, STUDENT e o ELIZA. A simplicidade do processo fazia com que muitos dos problemas gerados na utilização de linguagem natural pudessem ser ignorados.
- Programas como o PROTO-SYNTHEX1, em que se armazenava uma representação do texto, e recorria a engenhos de indexação para auxiliar a recuperação de palavras ou frases. Os sistemas não eram restritos a um domínio únicos, mas eram semanticamente fracos e não possuíam poderes dedutivos.
- Sistemas de lógica limitada, que objetivavam a tradução de frases de entrada para uma notação formal utilizada na base de dados. A intenção era realizar deduções partindo das informações mantidas na base de dados, mesmo que somente alguns processos de conversação diários pudessem ser explorados.

Como exemplo desse tipo de sistemas cita-se o SIR, TLC, DEACON e o CONVERSE.

- Sistemas com base de conhecimento, que utilizavam informações a respeito de um assunto específico para compreensão das frases de entrada. Alguns desses programas utilizavam a técnica de Inteligência Artificial (IA) denominada Sistema Especialista, e exibiam vários poderes dedutivos. Dentre estes sistemas cita-se o LUN AR e o SHRDLU.

Dentre todos os exemplos, OLIVEIRA (2002) cita que se destaca o programa ELIZA, desenvolvido por Joseph Weizenbaum, em 1966. Este programa caracterizava-se por assumir o papel de um psiquiatra que conversava sobre os problemas de um paciente humano. O ELIZA, no entanto, trabalhava somente com truques semânticos, não havendo compreensão a respeito do tema da conversa.

LUGER (2002) destaca o SHRDLU, desenvolvido por Terry Winograd no ano de 1972. Este programa conversava sobre um mundo de blocos, sendo estes blocos constituídos de diferentes formas e cores, tendo uma garra para movimentá-los. Por não envolver problemas mais complexos de raciocínio de senso comum, a técnica de representação do conhecimento era relativamente fácil. Mas mesmo assim o SHRDLU estabeleceu um modelo de integração entre sintaxe e semântica, e também mostrou que um programa com conhecimento suficiente sobre o domínio de um discurso pode se comunicar significativamente utilizando linguagem natural.

INSITE (2002) coloca que talvez futuramente os computadores possam se igualar a capacidade humana de compreender e compor textos, mas atualmente essa capacidade ainda é muito limitada.

Conforme afirmação de BARR e FEIGENBAUM (1986), os pesquisadores de processamento de linguagem natural esperam que sua pesquisa ajude a compreender melhor a linguagem e a natureza da inteligência, bem como consigam construir sistemas de linguagem natural úteis e práticos. Os computadores, como a mente humana, possuem a habilidade de manipular símbolos, em processos como tomada de decisões.

Dessa forma, desenvolvendo e testando modelos computacionais de processamento de linguagem que se aproximem da performance humana, os pesquisadores acreditam que possam compreender melhor a linguagem humana.

2.3.3 Processamento de Linguagem Natural

BARR e FEIGENBAUM (1986) coloca que o caminho mais comum para que as pessoas se comuniquem é falando ou escrevendo através de linguagem natural, seja em inglês, francês, português ou chinês. As linguagens de computadores possuem um formato mais rígido, para que possam ser convertidas em uma seqüência de instruções de computador. O estudo de processamento de linguagem natural procura fazer com que os computadores possam entender a linguagem natural humana, tornando-se mais fáceis de utilizar.

De acordo com OLIVEIRA (2002) o fato de se processar uma linguagem natural permite aos seres humanos comunicarem-se com computadores de uma forma mais próxima a deles, ou seja, uma linguagem a qual já estão acostumados. Desta forma, eliminar-se-ia o aprendizado de uma linguagem artificial ou formas inusitadas de interação.

OLIVEIRA (2003) afirma que a principal preocupação da pesquisa sobre processamento de linguagem natural é o uso da linguagem natural, ou seja, a utilização de agentes computacionais que se utilizam da linguagem natural para obter informações a respeito de outros agentes, sejam eles humanos ou máquinas, possibilitando ou causando mudanças em outros agentes, e, dessa forma, mudando o mundo.

O computador possui muita dificuldade em entender os escritos em linguagem humana, de acordo com INSITE (2002), porque as linguagens de computador são extremamente precisas, com estruturas lógicas bem definidas e regras fixas, o que permite que o mesmo saiba como proceder a cada comando. Ao avaliar-se a linguagem humana, constata-se que uma frase pode conter ambigüidades, interpretações que fogem do contexto do conhecimento do mundo, de regras gramaticais, culturais e abstratas.

Dando continuidade, Insite (2003) afirma que o estudo do Processamento de Linguagem Natural possui por objetivo dotar os computadores da capacidade de compreender e compor textos.

OLIVEIRA (1997) coloca que o Processamento de Linguagem Natural (PLN) está subdividida em duas sub-áreas:

- A interpretação de linguagem natural, que possui mecanismos que tentam compreender alguma língua natural para traduzi-la para alguma representação compreensível e utilizada pelo computador;
- na geração de linguagem natural, o computador traduz alguma representação interna em um texto compreensível pelo indivíduo. O texto de saída deve ser o mais próximo possível de um texto de um falante daquela língua.

Conforme BARR e FEIGENBAUM (1986), toda pesquisa em linguagem natural possui objetivos como:

- Desenvolvimento de sistemas práticos e úteis, que possam compreender a linguagem natural e atender a tradução por máquina, compreensão de textos, uma interface natural com banco de dados e obter um diálogo natural com a máquina.
- Uma melhor compreensão da linguagem humana e da natureza da inteligência humana.

MARTIN (2002) coloca que para capturar o conhecimento em Processamento em Linguagem Natural, são usados modelos formais incluindo máquinas de estado, sistemas de regras formais e lógicas, e teoria da probabilidade.

Os problemas relacionados a PLN, de acordo com OLIVEIRA (2003), são os seguintes:

- Normalmente, as frases escritas em uma determinada língua são descrições incompletas das informações a serem transmitidas. Exemplo:

  Há alguns gatos lá fora
  - Há alguns gatos no quintal.
  - Há dois gatos no quintal.
  - Negrinho e Miudinha estão no quintal.

  Uma mesma expressão pode significar diferentes coisas em contextos diferentes. Exemplo

  Onde está a água
  - Em um laboratório químico ela precisa ser pura.
  - Se o indivíduo está sedento, ela precisa ser potável.
  - Ao lidar com goteiras, ela pode estar imunda.

- Como novas palavras, expressões e significados podem ser gerados livremente, nenhum programa em Linguagem Natural pode ser completo. Exemplo: Eu "xeroco" uma cópia para você.
- Existem diversas formas de expressar a mesma idéia. Exemplo:
  - Minha mãe nasceu no dia 27 de outubro.
  - O aniversário de minha mãe é no dia 27 de outubro.

Conforme RICH (1994), três fatores contribuem para dificultar a compreensão da linguagem natural por computadores:

- a complexidade da representação alvo, que determinará o grau de compreensão do sistema. Quanto maior o nível de compreensão desejada, maior deve ser a complexidade da representação alvo;
- o tipo de mapeamento gerado;
- o nível de interação entre os componentes da linguagem, que é bastante alto nas linguagens humanas. Uma simples mudança em uma palavra ou símbolo pode mudar completamente a estrutura e o sentido de uma declaração.

LACERDA (1996) informa que existem vários tipos de mapeamentos entre uma realidade e uma estrutura que descreve esta realidade, o que gera entendimentos diferentes. São quatro os tipos de mapeamento: mapeamento um-para-um, muitos-para-um, um-para-muitos e muitos-para-muitos.

Dando continuidade, LACERDA (1996) informa que o mapeamento um-para-um possui como exemplo típico às expressões aritméticas, pois a prioridade entre operadores foi previamente convencionada. Este tipo de mapeamento é o que normalmente é encontrado nas linguagens de programação, por estas não admitirem ambigüidades.

LACERDA (1996) coloca que o mapeamento muitos-para-um é o mais comum, e ocorre ao se mapear de uma linguagem natural para uma representação alvo mais simples. Este tipo de mapeamento requer sempre grande conhecimento não lingüístico. Como exemplo, pode-se utilizar a mesma idéia central: não saber dirigir. Pode-se originar frases como:

- Ele não sabe dirigir.
- Ele é um motorista barbeiro.
- Na direção ele não é de nada.

No mapeamento um-para-muitos, continua LACERDA (1996), se dá ao se originar diversas interpretações de uma mesma expressão. Estes são casos de ambigüidade. Como exemplo, pode-se utilizar a frase: Eles estão no carro. Desta sentença pode-se originar uma série de interpretações, tais como:

- Eles estão andando pela estrada com o carro.
- Eles estão parados, porém dentro do carro.
- Eles estão abastecendo o carro.
- Eles estão limpando o carro.

De acordo com LACERDA (1996), o mapeamento muitos-para-muitos é aquele em que ocorre simultaneamente os mapeamentos muitos-para-um e um-para-muitos, acontecendo praticamente em todas as línguas.

Um programa de PLN, de acordo com OLIVEIRA (1997), deve ter conhecimento sobre a estrutura da língua, como conhecimento sobre as palavras, como combinar as palavras em sentenças, o significado de cada palavra, a combinação destas mesmas palavras em sentenças, e assim por diante. Por outro lado, para simular um

comportamento lingüístico, um indivíduo deve conhecer a estrutura da língua utilizada, ter conhecimento do mundo em geral e do emprego da conversa mantida. Dessa forma, um sistema de processamento de linguagem natural precisaria de métodos de codificação para poder utilizar este conhecimento e utilizá-lo de forma a produzir um comportamento apropriado. Além disso, o conhecimento a respeito do mundo atual possui um papel crucial na forma de interpretação de uma sentença particular.

Para que haja interpretação de uma sentença em linguagem natural, conforme OLIVEIRA (2002), existe a necessidade de manter informações morfológicas, sintáticas e semânticas armazenadas em um dicionário, juntamente com palavras que o sistema compreenda. LACERDA (1996) destaca que para obter-se sucesso na compreensão de frases individuais, pode-se dividir em três componentes: análise sintática, análise semântica e análise pragmática. OLIVEIRA (2002) informa que existe um quarto componente: a análise morfológica. Estes componentes serão descritos a seguir.

2.3.3.1 Análise Morfológica

OLIVEIRA (1997) informa que o conhecimento morfológico diz respeito à construção das palavras relacionadas ao seu morfema. LUGER (2002) diz que a análise morfológica é importante na determinação do papel de uma palavra numa sentença, incluindo tempo, número e sua parte no discurso.

O analisador morfológico, conforme informa OLIVEIRA (2002), irá identificar expressões ou palavras isoladas em uma sentença, sendo auxiliado por delimitadores como pontuação e espaços em branco, e as palavras sendo classificadas conforme sua categoria gramatical. Neste contexto, a morfologia trabalha as palavras conforme sua estrutura, forma, flexão e classificação, no que se refere a cada um dos tipos de palavras.

OLIVEIRA (2002) identifica que o autômato finito é um reconhecedor utilizado para a análise morfológica, tendo sido proposto uma forma de compreensão de vocabulários extensos por autômatos determinísticos acíclicos minimizados.

2.3.3.2 Análise Sintática

LACERDA (1996) esclarece que análise sintática é transformar a seqüência de palavras em estruturas de palavras que irão mostrar o relacionamento entre as palavras. A rejeição irá ocorrer ao haver violação das regras de combinação das palavras na linguagem pelas palavras. OLIVEIRA (1997) informa que essa forma de conhecimento identifica a forma de relacionamento de uma palavra com outra. RUSSELL e NORVING (2003) colocam que análise sintática é o processo de construção de uma árvore de análise para uma cadeia de entrada. LUNGE (2004) informa que este foi o componente da análise lingüística mais bem formalizado e de automatização mais bem sucedida.

Com as informações do analisador morfológico, e a gramática da linguagem analisada, o analisador sintático constrói árvores de derivação para cada sentença que mostra a relação entre as palavras. OLIVEIRA (1997) confirma que se a sentença for ambígua poderá ser construída mais de uma árvore.

OLIVEIRA (2002) coloca que a análise sintática de uma oração em português deve considerar sintagmas como termos essenciais (sujeito e predicado), termos integrantes (complementos verbal e nominal) e termos acessórios (adjunto adverbial, adjunto adnominal e aposto). Já a análise do período deve considerar o tipo de período (simples ou composto), sua composição (por subordinação, por coordenação) e a classificação das orações (absoluta, principal, coordenada ou subordinada).

O maior problema nos sistemas de processamento de linguagem natural, de acordo com OLIVEIRA (2002), é a transformação de uma frase ambígua em outra não ambígua, sendo essa transformação conhecida como *parsing*. OLIVEIRA (1997) informa que o analisador mapeia um conjunto de palavras em um conjunto de padrões sintaticamente significativos. LACERDA (1996) coloca que o processo de *parsing* permite a geração de uma estrutura gráfica conhecida como árvore sintática, que representará a estrutura sintática da sentença analisada. Como exemplo, tem-se a sentença **O menino chutou a bola**, que gera a árvore sintática da Figura 6.

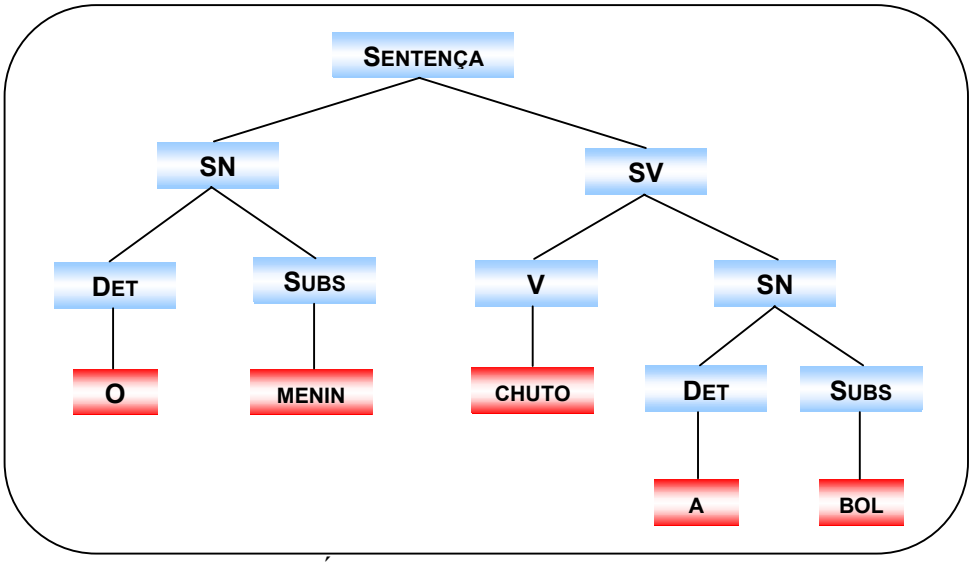

Figura 6: Exemplo de Árvore Sintática.
Fonte: OLIVEIRA (1997)

OLIVEIRA (1997) coloca que, quando o analisador sintático necessita decidir entre diversas estruturas possíveis de uma determinada frase, encontra-se um problema. As informações semânticas, neste caso, servirão para decidir qual a estrutura sintática correta. As abordagens desenvolvidas para solucionar este problema podem ser classificadas em:

- Gramáticas Semânticas.
- Gramáticas de Caso.
- Filtragem semântica das estruturas sintáticas geradas.
- Desenfatizar a análise sintática e acionar o processo de conhecimento semântico e não sintático (conhecido como analisadores livres de sintaxe).

Gramáticas

Conforme OLIVEIRA (1997), uma gramática de uma determinada língua trata de um esquema para especificar as sentenças permitidas por aquela língua, indicando a regra sintática para se combinar palavras em frases e cláusulas. Em programas que utilizam Processamento de Linguagem Natural, a função da gramática é a análise das

sentenças e ajuda na determinação de seus significados, obtendo-se uma resposta apropriada.

LACERDA (1996) coloca que, formalmente, uma gramática G é formada pela quádrupla G = (VN,VT,S,P), sendo V = VN $\cup$ VT, onde:

- VN - vocabulário não terminal.
- VT – vocabulário terminal.
- S – símbolo inicial.
- P – um conjunto finito de produções.

Cada produção P possui o formato X $\rightarrow$ Y, onde X e Y pertencem a V e X não é nulo.

Um estudante ou mesmo professor de língua portuguesa sentirá dificuldade em compreender uma representação gramatical formulada para uma linguagem de computador pela mesma ser específica para a área de computação. Fazendo-se uma analogia com a gramática da língua portuguesa, as representações acima possuiriam os seguintes significados:

- VN – categorias sintáticas (frase verbal, nome, sentença, etc).
- VT – palavras e conectivos (terra, gira, etc).
- P – regras que definem a relação entre VN e VT.
- S – conjunto de símbolos não terminais que, a partir deles, consiga-se gerar exatamente os símbolos terminais imaginados na linguagem.

A seguir tem-se algumas definições de gramáticas.

Gramática Regular (Tipo 3)

LACERDA (1996) informa que este tipo de gramática caracteriza-se por possuir no lado direito somente um símbolo terminal, ou então um símbolo terminar seguido de um único símbolo não terminal. Suas produções podem ser da forma X$\rightarrow$

αY ou $X \rightarrow \alpha$, onde X e Y são símbolos não terminais isolados e α é um símbolo terminal único.

Esta gramática, considerando-se o processamento sintático da linguagem natural, de acordo com OLIVEIRA (2002), é bastante simples e facilmente reconhecida, apresentando um poder de expressão limitado, e possuindo pouca utilidade no processamento de linguagem natural.

Gramática Livre de Contexto (Tipo 2)

Conforme OLIVEIRA (2002), este tipo de gramática é muito útil na descrição de gramáticas em linguagem natural. Geralmente, são mais poderosas que as gramáticas regulares, permitindo a representação da linguagem com um certo grau de complexidade. No entanto, um dos maiores problemas encontrados neste tipo de gramática, no tratamento de linguagem natural, é a dificuldade em expressar dependências simples.

LACERDA (1996) coloca que as produções desse tipo de gramática são $X \rightarrow Y$, onde X tem que ser um único símbolo não terminal. A maioria das linguagens humanas enquadra-se neste tipo de gramática, possuindo uma complexidade linear.

Em gramáticas livres de contexto, conforme RUSSELL e NORVING (2003), o lado esquerdo consiste em um único símbolo de não-terminal. Então, desta forma, cada regra define a reescrita do não-terminal como o lado direito em qualquer contexto. As gramáticas livres de contexto são populares em linguagens naturais e linguagem de programação.

As Gramáticas livres de contexto, de acordo com RABUSKE (1995), utilizam a notação *Backus-Naur Form* (BNF). Como exemplo, a definição BNF da sentença "O vaqueiro tange a boiada", como mostra a Tabela 1.

Tabela 1: BNF da sentença “O vaqueiro tange a boiada”

Definição	Notação BNF
<sentença>	::= <frase_nominal><frase_verbal>
<frase_nominal>	::= <artigo><nome>
<frase_verbal>	::= <verbo><frase_nominal>
<artigo>	::= <o>
<artigo>	::= <a>
<verbo>	::= <tange>
<nome>	::= <vaqueiro>
<nome>	::= <boiada>

Fonte: RABUSKE (1995)

Dando continuidade, RABUSKE (1995) coloca que a notação BNF acima permite gerar a árvore de análise da Figura 7.

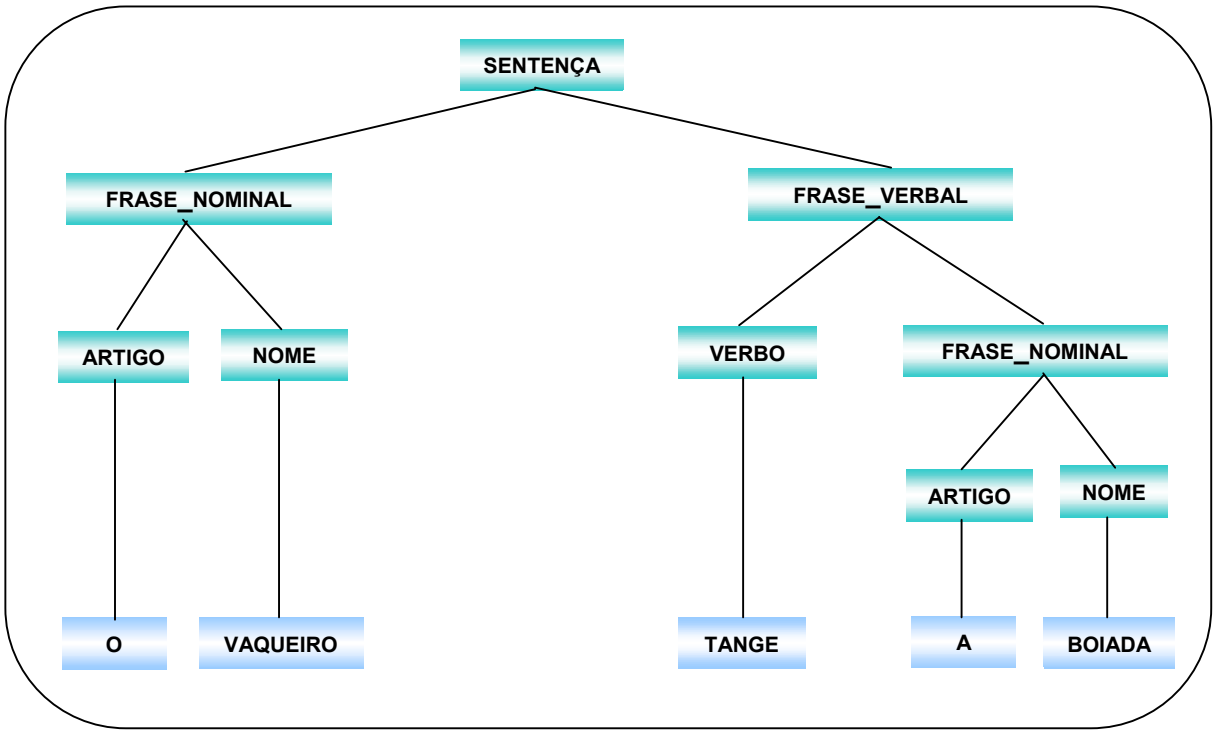

Figura 7: Árvore de análise da estrutura “O vaqueiro tange a boiada”.
Fonte: RABUSKE (1995)

Gramática Sensível ao Contexto (Tipo 1)

De acordo com OLIVEIRA (2002), os problemas de dependência vistos na gramática livre de contexto são resolvidos nesta classe de gramática. Porém, ainda estas gramáticas não abordam adequadamente o tratamento das restrições gramaticais. O problema ocorre quando a decisão de identificar se uma sentença pertence ou não a uma gramática é uma função exponencial sobre o tamanho da sentença, dificultando a implementação do procedimento de verificação, tornando-se uma questão complexa do ponto de vista computacional.

LACERDA (1996) coloca que uma gramática pode ser considerada uma gramática do tipo 1 se a forma de suas produções é restrita, de forma que para cada produção X→ Y da gramática, Y possui pelo menos o mesmo número de símbolos de X.

2.3.3.3 Análise Semântica

De acordo com LACERDA (1996), o termo "semântica" significa significado, ou pode também ser considerado o estudo do significado, que em linguagem natural é a entidade ou ação que ela denota. A análise morfológica identifica as palavras individualmente, a análise sintática serve para determinar a estrutura de uma sentença, e a análise semântica serve para determinar o significado desta mesma sentença. OLIVEIRA (1997) coloca que quando o analisador sintático precisa decidir entre diversas estruturas possíveis para uma determinada frase, encontra-se um problema, decidido pelas informações semântica que auxiliam na escolha de qual a estrutura sintática é a correta. Já LUNGE (2004) coloca que a semântica trata do significado de palavras, frases e sentenças, e os modos pelos quais o significado é transmitido em expressões em linguagem natural.

A análise sintática e semântica são igualmente importantes, e se complementam. LACERDA (1996) coloca que o produto da análise sintática é importante para o analisador semântico determinar o significado da estrutura, e a análise

semântica fornece elementos que facilitam a desambiguação léxica de uma sentença. Existem controvérsias entre qual das duas abordagens deva ser executada em primeiro lugar, e alguns defendem a tese de que ambas devem ser executadas ao mesmo tempo.

OLIVEIRA (2002) coloca que a semântica desdobra-se em duas variações: léxica e gramatical. A semântica léxica procura uma representação conceitual para descrever o sentido, podendo ser realizada uma decomposição semântica das unidades léxicas, ou serem utilizadas redes semânticas. A última forma de representação originou-se da psicologia e baseia-se na forma como os seres humanos categorizam e memorizam conceitos.

Dando continuidade, OLIVEIRA (2002) salienta que a compreensão das palavras é importante, mas a compreensão da relação entre as palavras é tão importante quanto.O enfoque formal na semântica gramatical tenta descrever o sentido de uma frase, pela tradução de sua estrutura sintática para uma fórmula lógica-semântica. Como não existe uma correspondência imediata e biunívoca entre sintaxe e semântica, uma estrutura sintática pode possuir diversas representações semânticas. Como exemplo, a estrutura "Uma professora de capoeira pernambucana" pode se referir a uma pessoa nascida no estado de Pernambuco que ensina capoeira, ou uma pessoa que ensina capoeira no estilo em que é praticada em Pernambuco.

Linguagens formais ou linguagens naturais, conforme RUSSELL e NORVING (2003), associam um significado ou uma semântica a uma cadeia válida. Por exemplo, quando fala-se em aritmética, se "X" e "Y" são expressões, então "X + Y" também é uma expressão, e a semântica desta expressão é a soma de X e Y.

LACERDA (1996) coloca que as abordagens desenvolvidas para solucionar o problema da análise semântica podem ser classificadas em:

- Gramáticas Semânticas.
- Gramáticas de Caso.

- Filtragem semântica as estruturas sintáticas geradas, através dos seguintes métodos:
  - fazer a filtragem à medida em que as ambigüidades forem sendo encontradas. Exemplo: ATN;
  - aguardar até que o processo de análise sintática esteja completado e avaliar as estruturas resultantes quanto à aceitabilidade semântica.
- Desenfatizar a análise sintática e acionar o processo de compreensão por conhecimento semântico e não sintático, podendo usar dependência conceitual e *demon-parsers*.

Gramáticas Semânticas

De acordo com RICH (1994), são gramáticas livres de contexto, onde a escolha das regras de produção e símbolos não terminais é dirigida por funções semânticas e sintáticas. Abaixo, um fragmento simplificado de uma gramática semântica usada no sistema LADDER, que fornece acesso, em linguagem natural, a uma grande base de dados utilizada por oficiais da Marinha:

S → qual é PROPRIEDADE-NAVIO de NAVIO?
PROPRIEDADE-NAVIO → a PROP-NAVIO | o PROP-NAVIO | PROP-NAVIO
PROP-NAVIO → velocidade | comprimento | contingente | largura | tipo
NAVIO → NOME-NAVIO | o mais rápido NAVIO2 | o maior NAVIO2 | NAVIO2
NOME-NAVIO → Kennedy | Kitty Hawk | Constellation | ...
NAVIO2 → TIPO-NAVIO LOC | TIPO-NAVIO
TIPO-NAVIO → PAÍSES TIPO 2 | TIPO2
TIPO2 → porta-aviões | submarino | barco a remo | ...
PAÍSES → americano | francês | ingles | russo | …
LOC → no Mediterrâneo | no Pacífico | ...

Analisando-se, esta gramática possui categorias semânticas não terminais, como NAVIO, LOC e PAÍSES, no lugar das tradicionais categorias sintáticas NP e VP. Desta forma, a estrutura “Qual é a velocidade do [de+o] mais rápido porta-aviões

americano no pacífico?" pode ser considerada sintática e semanticamente correta. No entanto, se considerarmos a estrutura "Qual a altura do mais rápido porta-aviões americano no Pacífico?", a mesma seria rejeitada, pois altura não possui correspondência semântica ao universo da linguagem definida pelas regras de produção, apesar de estar sintaticamente correta.

Gramáticas de Caso

LACERDA (1996) coloca que a noção de casos tem sua utilização para referenciar diversos conceitos relacionados, ou significa a classificação de palavras conforme seu papel sintático em uma sentença, sinalizado por várias formas de flexão. Diferenciando-se das gramáticas sintáticas, de acordo com OLIVEIRA (1997), a gramática de casos atribui papéis temáticos, ou casos, aos componentes de uma determinada estrutura ou frase, sendo que estes papéis devem ser independentes da distribuição desses componentes na estrutura sintática da frase.

Conforme RICH (1994), em línguas como o latim, os casos possuem uma importância básica, sendo que se não forem conhecidos, não poderá ser realizada a tradução correta do latim para o português.

Dando continuidade, RICH (1994) coloca que não se chegou a um consenso sobre a quantidade exata de casos que uma determinada língua pode reunir, classificando a seguinte relação como tendo um grande alcance:

- Agente: tipicamente animado, aquele que instiga a ação.
- Instrumento: inaminado, causador do evento ou ainda é o objeto utilizado para causar o evento.
- Dativo: entidade afetada pela ação, sendo animado.
- Factivo: local onde ocorreu o fato.
- Origem: local de onde algo se desloca.
- Meta: local para onde algo se desloca.
- Beneficiário: ser que o evento ocorre em seu benefício, sendo animado.

- Tempo: tempo de ocorrência do evento.
- Objeto: entidade modificada ou que se modifica.

De acordo com OLIVEIRA (1997), esta análise tem por principal objetivo a representação do significado de uma frase. Frases que possuem significação diferenciada deverão possuir representações diferentes, e frases com igual significação deverão possuir a mesma representação, como em:

Elaine fechou a janela.
A janela foi fechada por Elaine.

As duas frases possuem o mesmo agente = Elaine e paciente = janela, ainda que ambas tenham estruturas sintáticas diferentes.

BRUSER & MOSER apud LACERDA (1996) classificam os tipos de casos em superficiais e profundos. Casos superficiais ou casos de níveis sintáticos são os que as palavras são classificadas conforme seus sufixos, flexões ou são considerados marcadores sintáticos, permitindo a atribuição de um papel especial a qualquer termo da sentença. Já os casos profundos ou casos semânticos são os que classificam os sintagmas nominais de acordo com os papéis conceituais desempenhados por eles na sentença, independente do predicado ou verbo.

Para implementar a gramática de casos, de acordo com TEMPAKU (2000), pode-se recorrer aos diversos mecanismos que dão suporte à implementação de gramáticas, como a *Augmented Transition Network* (ATN).

Augmented Transition Network (ATN)

Ao se trabalhar com redes de transição estendidas, expande-se as redes de transição permitindo que procedimentos sejam associados aos arcos da rede, de acordo com LUGER (2002). Ao atravessar os arcos, o analisador ATN executa estes procedimentos associados. Os procedimentos podem atribuir valores a características gramaticais e realizar testes, sendo que se certas condições não forem satisfeitas pode

causar a falha de uma transição. Estes procedimentos também constroem uma árvore sintática, para ser usada na geração de uma representação semântica interna do significado da sentença.

De acordo com OLIVEIRA (2003), o ATN é derivado de um autômato finito, formado por uma coleção de estados e arcos com nomes (*labels*), um estado inicial distinto e um conjunto distinto de estados finais. Os estados são conectados uns aos outros pelos arcos – originando um grafo direcionado ou rede, e o nome do arco indica um símbolo terminal (palavra), ou tipos de palavras, que devem ocorrer na sentença de entrada para permitir a transição para o próximo estado. Uma sentença é aceita por uma ATN ao existir uma seqüência de arcos que conectam o estado inicial com o estado final, podendo ser seguido pela sentença, como indica a Figura 8.

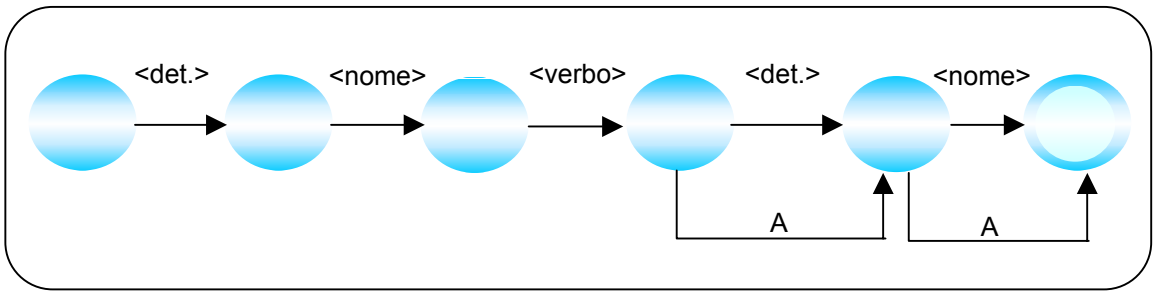

Figura 8: Um Autômato para um Fragmento de Português.
Fonte: OLIVEIRA (2003)

Dando continuidade, OLIVEIRA (2003) informa que um autômato finito básico é enriquecido com vários recursos que podem aumentar seu poder de computação. Uma transição através de um arco como o demonstrado acima é uma aplicação recursiva de uma rede, tendo início no estado apontado. Uma rede com este tipo de recurso possui seu poder ampliado para tratar de linguagens livres de contexto, sendo chamadas *Recursive Transition Network* (RTN).

Cada arco pode ser equipado com um teste e uma seqüência de ações, o que amplia ainda mais o poder destas redes, sendo o teste uma condição arbitrária imposta para que a transição seja efetivada. As ações serão executadas durante a transição pelo arco, e a rede será composta por um conjunto de registros que servem para armazenar segmentos da estrutura final produzida pela gramática. Os testes possuem por função

examinar o conteúdo dos registros e as ações podem atribuir valores arbitrários a eles. Uma rede neste estilo é chamada ATN, e possui poder equivalente a máquina de Turing.

2.3.3.4 Análise Pragmática

LACERDA (1996) destaca que pragmática é o estudo da comunicação e onde ela se situa no conjunto de necessidades de comunicação, emissores, receptores, tempos, lugares, ambiente, convenções lingüísticas e práticas culturais. RUSSELL e NORVING (2003) coloca que a análise pragmática leva em conta que palavras iguais podem possuir diferentes significados, dependendo do contexto em que está inserida. O significado não encontra-se nas palavras em si, mas na interpretação das palavras. LUGER (2002) coloca que pragmática é o estudo das formas de uso da linguagem e de seus efeitos sobre o interlocutor.

Na análise pragmática, conforme OLIVEIRA (1997), faz-se a reinterpretação da estrutura que representa o que foi dito para se determinar o que realmente se queria dizer. Como exemplo, pode-se citar a frase "Você tem horas?". Se esta frase for interpretada somente pelo lado sintático / semântico, será obtida uma resposta do estilo Sim, eu tenho, ou Não, eu não tenho. Observando-a sob o enfoque pragmático, esta frase pode ser interpretada como uma solicitação para informar-se as horas.

Sob a ótica pragmática, conforme LACERDA (1996), a pesquisa em Processamento de Linguagem Natural possui três implicações importantes:

- o significado de uma mensagem expressa de forma oral é representado somente parcialmente pelo conteúdo da mensagem. O receptor deve ser perceptivo quanto as intenções do emissor ao produzir a mensagem;
- a atribuição das intenções do emissor é componente integral no processo de compreensão do receptor;
- a teoria da compreensão da linguagem deveria determinar o âmbito para o qual as estratégias utilizadas pelas pessoas para chegar a explicações

razoáveis sobre o comportamento físico das outras pudessem ser empregados na compreensão dos atos da fala.

2.4 SISTEMAS ESPECIALISTAS

Um Sistema Especialista, de acordo com LUGER (2002), usa o conhecimento específico de um domínio de problema com o intento de obter um desempenho com "qualidade de perícia" naquela área de aplicação. Os projetistas de sistemas especialistas adquirem o conhecimento necessário através de peritos humanos, e o sistema simula a metodologia e atuação do perito humano.

Sistemas Especialistas, conforme BARRETO (1997), conhecidos também por Sistemas Baseados em Conhecimento, são sistemas computacionais que procuram simular o comportamento de um especialista em um determinado domínio. Eles procuram imitar o processo humano de raciocínio, ao imitar o especialista humano.

FERNANDES (2002) coloca que os Sistemas Especialistas podem aprender, analisar, controlar, interpretar, aconselhar, consultar, monitorar, comunicar, instruir, classificar, diagnosticar, predizer, projetar, testar, possuindo também a capacidade de melhorar seu desempenho, aprimorar o seu raciocínio e aperfeiçoar suas decisões. Atendem uma aplicação restrita ou a um domínio limitado do conhecimento, pois quanto menor o domínio e maior conhecimento tiver o programa, mais eficiente o mesmo será.

Os sistemas especialistas são programas particularmente relevantes, de acordo com RAMOS (1995), para a realização de inferências e deduções baseados em problemas que envolvem aspectos não estruturados. Como os especialistas humanos, estes programas usam lógica simbólica e heurística para solucionar um problema, podendo cometer erros. Eles possuem capacidade de lidar com problemas complexos, resolver questões que necessitam um alto nível de juízo humano e perícia, além de

comunicar-se com seu usuário de forma eficaz. Os sistemas especialistas fazem perguntas, dão pareceres e os justificam.

Os Sistemas Especialistas, de acordo com LUGER (2002), são construídos para resolver diversos problemas em domínios, abrangendo áreas como medicina, matemática, engenharia, química, geologia, ciência da computação, economia, direito, defesa e educação. A seguir, um conjunto resumido de problemas para Sistemas Especialistas:

- Interpretação: gerar conclusões de alto nível em cima de dados brutos.
- Predição: projetar conseqüências prováveis de situações disponíveis.
- Diagnóstico: determinar a causa de mau funcionamento em situações complexas baseando-se em sintomas observáveis.
- Projeto: achar uma configuração de componentes do sistema que alcance os objetivos de desempenho e, simultaneamente, siga o conjunto de restrições do projeto.
- Planejamento: tendo-se as condições iniciais e restrições de tempo de execução, estabelecer uma seqüência de ações para alcançar os objetivos propostos.
- Monitoramento: comparar o comportamento de um sistema com o comportamento que se espera que o mesmo tenha.
- Instrução: assistir o processo de educação de um determinado domínio técnico.

Para a construção de um Sistema Especialista, BARRETO (1997) informa que são necessários:

- o especialista, que é a fonte do conhecimento;
- transformar em dados e armazenar no computador o conhecimento a ser adquirido do especialista;
- gerar as regras de raciocínio, que são regras que mostram o raciocínio do especialista a respeito da resolução do problema;

- em grande parte dos casos existe a necessidade de um mecanismo que gere explicações de como o especialista chegou a determinada conclusão.

A Figura 9 mostra os módulos componentes de um sistema especialista típico LUGER (2002).

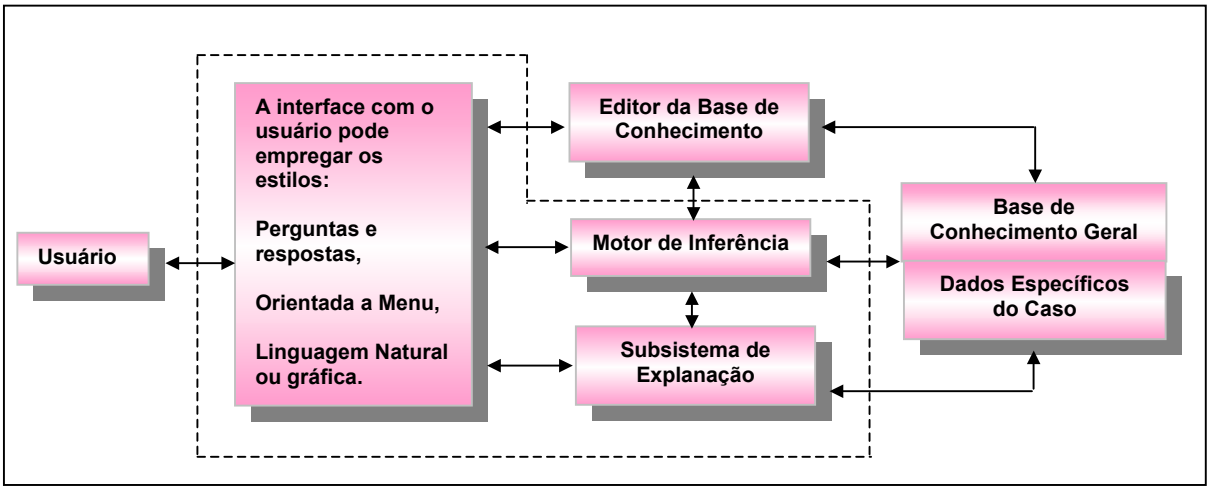

Figura 9: Arquitetura de um Sistema Especialista para domínio particular.
Fonte: LUGER (2002)

Conforme LUGER (2002), o usuário interage com o sistema através da interface, que torna mais simples a comunicação e oculta a complexidade interna. As interfaces podem ser de vários estilos, como orientada a perguntas e respostas, orientada a menu ou gráficas. A resposta final referente a qual utilizar é dada de acordo com a necessidade do usuário, requisitos da base de conhecimento e sistema de inferência.

Dando continuidade, LUGER (2002) informa que a base de conhecimento é o coração do sistema especialista, pois possui o conhecimento do domínio do problema em questão. Ela possui tanto o conhecimento geral como informação específica a respeito do caso. Já o motor de inferência aplica o conhecimento à solução do problema, sendo basicamente um interpretador para a base de conhecimento. O subsistema de explanação faz com que o programa explique seu raciocínio ao usuário, através de, por exemplo, justificativas a respeito das conclusões do sistema e o porque da necessidade de algum dado em particular..

Quando trata-se de Sistema Especialista, pode-se classificar as regras de produção de duas formas:

- *Forward Chaining* (encadeamento para frente): partindo-se de um ponto inicial, chega-se a uma conclusão.
- *Backward Chaining* (encadeamento para trás): inicia com uma hipótese e procura valores para confirmar a mesma.

O desenvolvimento de um Sistema Especialista envolve um investimento considerável de dinheiro e esforço humano. Um Sistema Especialista mal planejado pode levar fracassos embaraçadores ou custosos, de acordo com LUGER (2002). Para evitar este problema foram elaboradas algumas diretrizes para determinar se um problema pode ser resolvido por sistema especialista ou não, que são:

- a necessidade de uma solução justifica o custo e esforço para desenvolvimento de um Sistema Especialista;
- não poder contar com o especialista humano em todas as situações necessárias;
- o problema pode ser resolvido utilizando raciocínio simbólico;
- o domínio do problema é bem estruturado e não requer raciocínio de senso comum;
- o problema não possui resolução através de métodos de computação tradicionais;
- os especialistas estão dispostos a compartilhar seu conhecimento;
- o problema possui tamanho e escopo adequados.

FERNANDES (2002) coloca as seguintes vantagens dos Sistemas Especialistas:

- auxílio na redução de falhas humanas e aceleração de tarefas;
- flexibilidade, estabilidade e maior rapidez na resolução de problemas;
- aumento na qualidade e desempenho na resolução de problemas;

- combinação e preservação do conhecimento dos especialistas;
- integra várias ferramentas;
- não é afetado por questões psicológicas ou fatores externos;
- apresenta maior eficiência e otimização de resultados.

Como desvantagens dos Sistemas Especialistas, RAMOS (1995) salienta as seguintes:

- trabalho rotineiro, sem inspiração ou criatividade;
- não adquirem novos conhecimentos;
- dificuldade em lidar com situações inesperadas;
- entrada, informação e representação simbólica e não sensória;
- sem conhecimento do senso comum.

2.5 COMPILADORES

Conforme AHO et. al (1995), um compilador é um programa que realiza a leitura de um outro programa escrito em uma linguagem fonte e o converte para um programa equivalente escrito em uma linguagem alvo, conforme demonstrado na Figura 10.

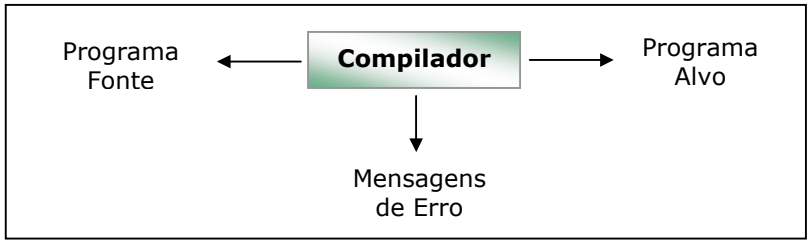

Figura 10: Um compilador.
Fonte: AHO et. al (1995)

Os primeiros compiladores surgiram no início dos anos 50, e eram considerados difíceis de serem escritos. O primeiro compilador FORTRAN, por exemplo, foi construído por 18 homens-ano. AHO et. al (1995)

Existem duas partes na compilação: a análise e a síntese. Na análise o programa fonte é dividido nas partes constituintes e cria uma representação intermediária do mesmo. Já a síntese constrói o programa alvo desejado, partindo da representação intermediária, o que requer técnicas mais especializadas. AHO et. al (1995)

A análise consiste de 3 fases, conforme AHO et. al (1995):

- Análise Linear: os caracteres são lidos da esquerda para a direita e agrupados em *tokens* (seqüência de caracteres com significado coletivo).
- Análise Hierárquica: *tokens* são agrupados hierarquicamente em coleções aninhadas com significado coletivo.
- Análise Semântica: são realizadas verificações para assegurar que os componentes de um programa se combinam de forma significativa.

Um sistema de processamento de linguagem é exibido na Figura 11.

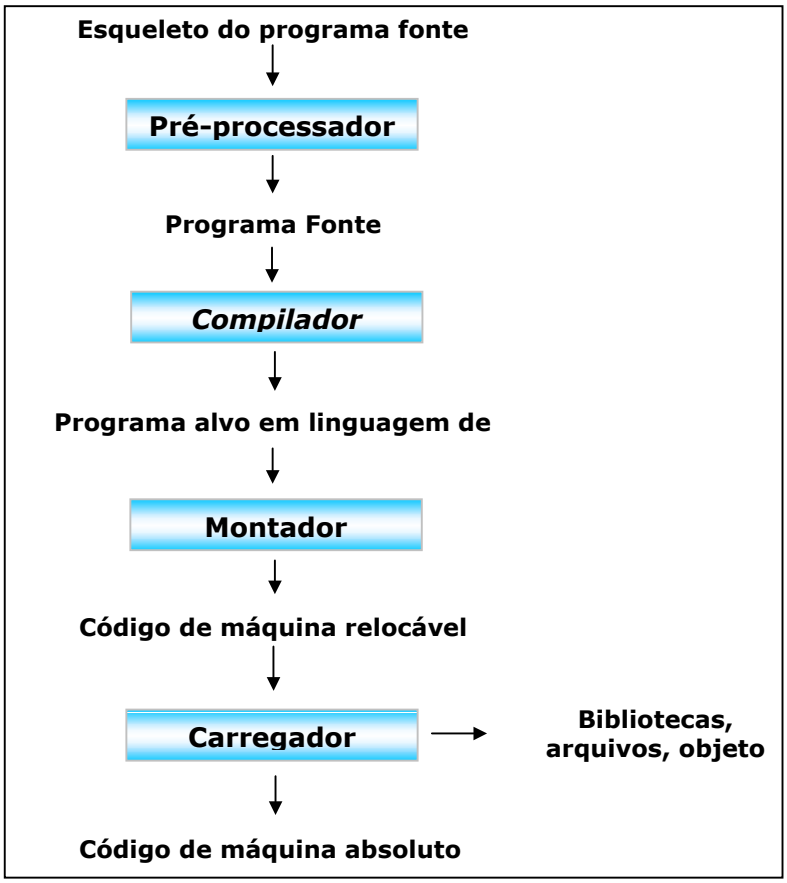

Figura 11: Um sistema de processamento de linguagem.
Fonte: AHO et. al (1995)

Na prática, algumas fases podem ser agrupadas e a representação intermediária entre as mesmas não precisam ser explicitamente construídas. As fases de um compilador são: analisador léxico, analisador sintático, analisador semântico, gerador de código intermediário, otimizador de código e gerador de código. O gerenciamento da tabela de símbolos e a manipulação de erros interagem entre as duas fases. AHO et. al (1995)

As fases de um compilador pode ser visualizadas na Figura 12.

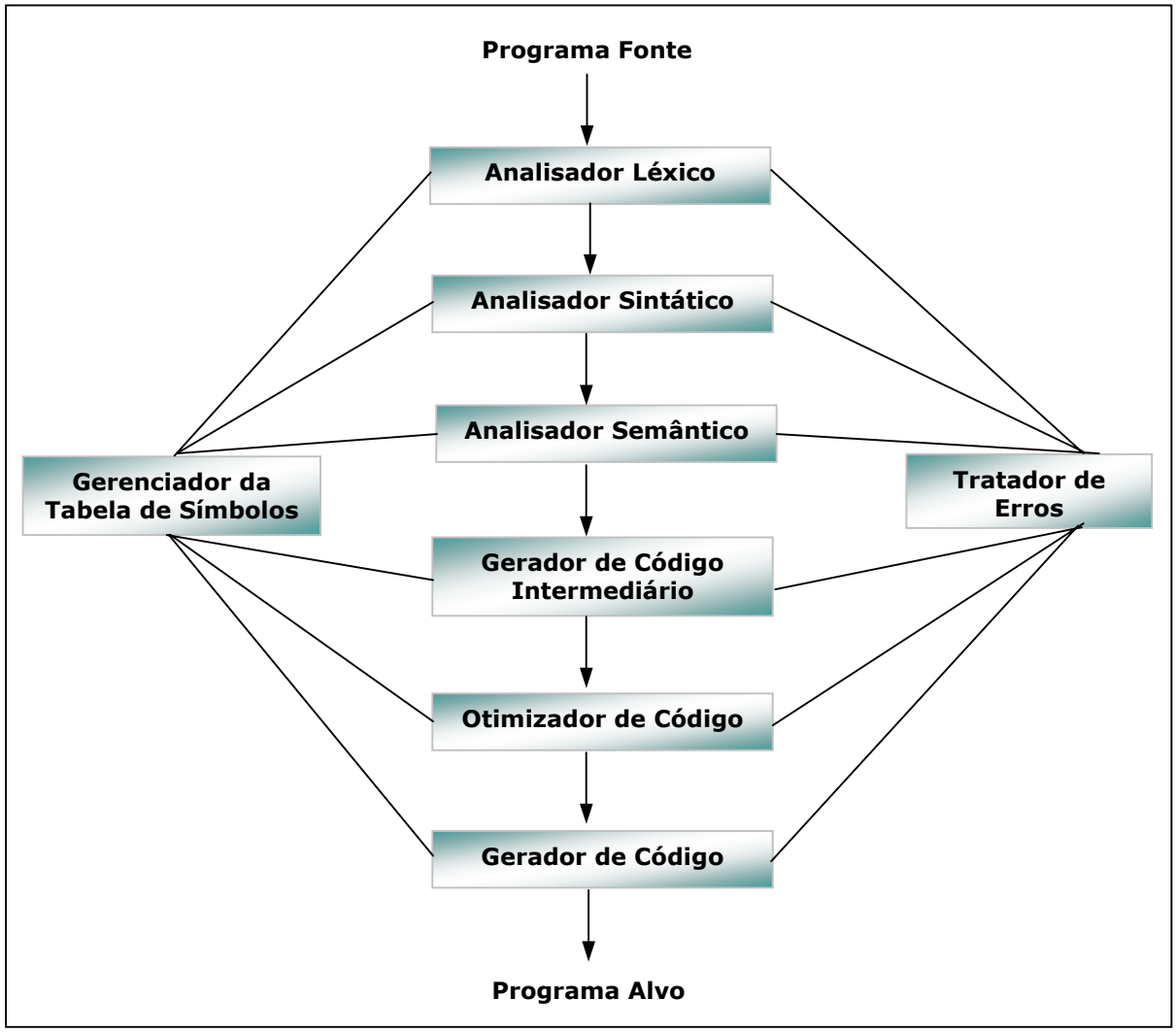

Figura 12: Fases de um compilador.
Fonte: AHO et. al (1995)

Ao se tratar de compiladores, a análise linear é conhecida como análise léxica ou esquadrinhamento (*scanning*). PRICE E TOSCANI (2001) colocam que a função do

analisador léxico é realizar a leitura do programa fonte, um caracter por vez, e realizar sua tradução ar uma seqüência de símbolos léxicos, também chamados tokens. AHO et. al (1995) coloca que o compilador irá ler todos os caracteres de entrada e produzir uma seqüência de *tokens* que serão utilizados pelo *parser* na análise sintática. Quando o parser repassa ao analisador léxico um comando solicitando a obtenção do próximo *token*, o analisador léxico vai lendo os caracteres até conseguir identificar o próximo *token*. O próprio analisador léxico vai removendo do programa fonte comentários, espaços em branco, tabulações, e caracteres de avanço de linha. Na Figura 13 pode-se visualizar a interação entre o analisador léxico e o *parser*.

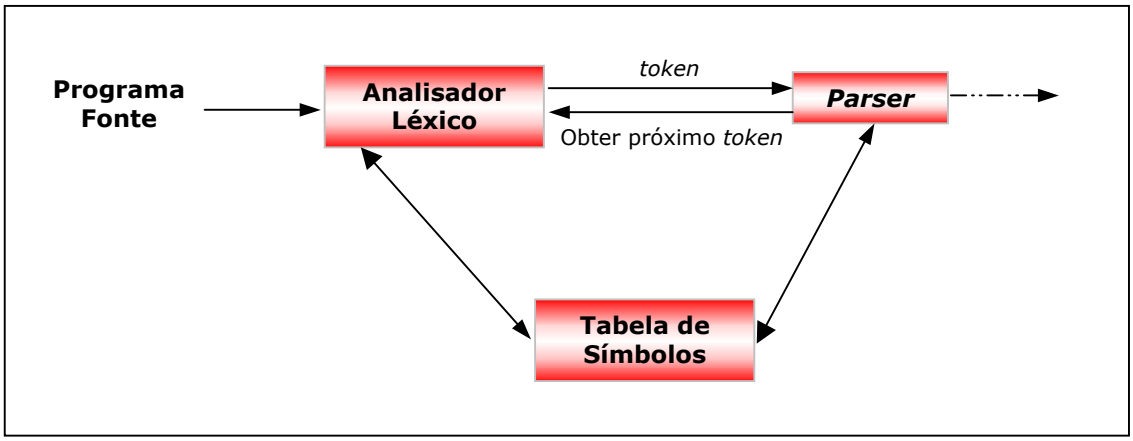

Figura 13: Interação do analisador léxico com o *parser*
Fonte: AHO et. al (1995)

Na especificação dos *tokens* se é trabalhado com expressões regulares, que são uma notação importante para especificar padrões. Cada padrão é um conjunto de cadeias e as expressões servirão como nomes de conjuntos de cadeias (AHO et. al, 1995). PRICE E TOSCANI (2001) colocam que para o analisador léxico o programa fonte é uma seqüência de palavras de uma linguagem regular. Já para o analisador sintático, a seqüência de *tokens* forma uma sentença de uma linguagem livre de contexto.

Uma expressão regular é constituída de expressões regulares mais simples usando-se um conjunto de regras de definição. Cada expressão regular *r* denota uma linguagem *L(r)*. As regras de definição especificam como *L(r)* é formada através da combinação, em várias formas, das linguagens denotadas pelas subexpressões de *r* (AHO et. al, 1995).

Ao se trabalhar com expressões regulares associadas a cada regra existe uma especificação da linguagem denotada pela expressão regular sendo definida. Abaixo as regras para definição de expressões regulares sobre um alfabeto ∑:

- Є é uma expressão regular que mostra o conjunto que contém a cadeia vazia ({Є }).
- Se *a* é um símbolo em ∑, então *a* é uma expressão regular que denota {a}, isto é, o conjunto contendo a cadeia *a*. Apesar de se usar a mesma notação para todos os três, tecnicamente a expressão regular *a* é diferente da cadeia *a* e do símbolo *a*.
- Supondo-se que *r* e *s* sejam expressões regulares denotando as linguagens *L(r)* e *L(s)*. Dessa forma:
  a) *(r)|(s)* é uma expressão regular denotando $L(r) \cup L(s)$.
  b) *(r)(s)* é uma expressão regular denotando *L(r) L(s)*.
  c) *(r)** é uma expressão regular denotando *(L(r))**.
  d) *(r)* é uma expressão regular denotando $L(r)^2$.

Um analisador sintático obtém a cadeia de *tokens* gerados na análise léxica e verifica se a mesma pode ser gerada pela gramática da linguagem-fonte. O analisador sintático deve relatar qualquer erro de sintaxe de forma inteligível e se recuperar de erros que ocorram mais freqüentemente, para assim processar o resto de sua entrada. Existem dois métodos usados mais freqüentemente na construção de compiladores: os analisadores sintáticos *bottom-up* e os *top-down*. Os analisadores *top-down* constroem árvores do topo (raiz) para o fundo (folhas), e os analisadores *bottom-up* começam pelas folhas e trabalham árvore acima até a raiz. Nos dois métodos a entrada é varrida da esquerda para a direita, um símbolo por vez (AHO et. al, 1995). PRICE E TOSCANI (2001) colocam que no método *top-down*, em cada passo, um lado esquerdo de produção é substituído por um lado direito (expansão); já na *bottom-up*, em cada passo, um lado direito de produção é trocado por um símbolo não-terminal (redução).

A análise sintática é responsável pela verificação da boa formação dos comandos da linguagem, seguindo as regras especificadas pela gramática da linguagem.

Quando as sentenças são mal formadas, geralmente o processo de compilação é interrompido e são apresentadas mensagens de erros indicando o problema. No fim da análise sintática, é gerada a representação do programa original de forma hierárquica, onde o programa é representado por uma árvore sintática. Essa árvore sintática nada mais é que a representação compactada de uma árvore PARSE, na qual os operadores aparecem como nós interiores e os operandos de um operador são os filhos do nó daquele operador (AHO et. al, 1995).

O analisador léxico, conforme PRICE E TOSCANI (2001), reconhece o texto fonte original, realizando a leitura dos caracteres e obtendo os *tokens* do programa. Ele lê o texto fonte como uma seqüência de palavras de uma linguagem regular e o reconhece através de um autômato finito. Já o analisador sintático vê o texto como uma sentença que deve satisfazer às regras gramaticais de uma gramática livre de contexto.

AHO et. al (1995) afirma que, ao se tratar de construções léxicas, as mesmas não requerem recursão, ao contrário das sintáticas. As gramáticas livres de contexto nada mais são que uma formalização de regras recursivas que podem ser usadas para guiar a análise sintática. PRICE E TOSCANI (2001) colocam que as gramáticas livres de contexto, ou notação BNF (Backus Naur Form) permitem a descrição da maioria das linguagens de programação usadas atualmente. Da mesma forma, uma linguagem livre de contexto pode ser reconhecida por um autômato a pilha e para ela é possível, tendo por ponto de início a descrição realizada em uma gramática livre de contexto, construir automaticamente um analisador sintático (o YACC se utiliza deste método).

Ao tratar-se de análise semântica, a mais comum trata da verificação da consistência de tipos dos operandos envolvidos em operações aritméticas ou dos parâmetros passados a procedimentos. O código intermediário deve ser fácil de produzir e fácil de traduzir no programa objeto (AHO et. al, 1995).

Tradução dirigida por sintaxe trata de uma técnica com o intuito de realizar tradução (geração de código) juntamente com a análise sintática. As ações semânticas são associadas às regras de produção da gramática, e deste modo quando uma

determinada produção é processada, as ações são executadas. Essa execução das ações podem gerar ou interpretar código, armazenar informações na tabela de símbolos, emitir mensagens de erro, etc (PRICE E TOSCANI, 2001). Na Figura 14 pode-se visualizar a posição do gerador de código intermediário.

Figura 14: Posição do gerador de código intermediário
Fonte: AHO et. al (1995)

Após a realização das análises léxica, sintática e semântica, e estando as três análises corretas, é acionado o gerador de código intermediário. PRICE E TOSCANI (2001) colocam que no processo de geração de código intermediário a árvore de derivação é transformada em um segmento de código. A geração do código pode ser para o código final, mas em grande parte das vezes é gerado primeiramente um código intermediário, pois traduzir o código fonte para código objeto apresenta como vantagens:

- o código intermediário pode ser otimizado, sendo que o código objeto final fica mais eficiente;
- simplifica a implementação do compilador, pois resolve aos poucos as dificuldades relacionadas a passagem do código fonte ao código objeto;
- possibilita a tradução do código intermediário para diversas máquinas.

Como desvantagem na geração de código intermediário pode-se comentar que o compilador requer um passo a mais, pois a passagem de código fonte para código objeto produz uma compilação mais rápida (PRICE E TOSCANI, 2001).

Conforme AHO et. al (1995), nesta parte da análise inicia-se o processo de síntese. A árvore sintática é percorrida em profundidade (*depth first*) para que o código seja gerado das folhas para os nós, e é utilizado a quádrupla – quatro campos que

deixam o código numa linguagem bem acessível para a posterior tradução para código de máquina . No tratamento de erros, os erros sintáticos são os mais freqüentes de ocorrerem. Algumas estratégias de recuperação de erros são:

- Panic Mode - o compilador descarta símbolos de entrada até encontrar um token de sincronização(";" ou "end").
- Phase Level - o compilador tenta corrigir o erro encontrado, sem alterar o código fonte. (colocar um ";" onde não tem ou trocar ":" por ";").
- Produção de Erro - pode ser usado quando sabe-se, de antemão, quais são os erros mais prováveis de acontecer; acrescenta-se à gramática, algumas produções que usam as construções erradas.
- Correção Global - tenta calcular a seqüência mínima de mudanças para a correção de um erro. Essa estratégia tem apenas interesse teórico.

Ao realizar as análises sintática e semântica, o compilador mexe com uma tabela de símbolos. A tabela de símbolos é uma estrutura de dados que contém um registro para cada identificador, com os campos contendo os atributos do identificador, como memória alocada, tipo, escopo, etc (AHO et. al, 1995).

Uma próxima etapa é a geração do código objeto. Esta tarefa trata de transformar uma especificação de código intermediário vinda da etapa de geração de código intermediário para uma especificação de código assembly, e assim ser executado em um computador. Por fim, é realizada a geração do módulo executável, onde é gerado o executável a ser utilizado pelo usuário (AHO et. al, 1995).

3. METODOLOGIA EMPREGADA

O objetivo deste trabalho foi construir um aplicativo que viesse a auxiliar os professores da disciplina de Algoritmos, no curso de Ciência da Computação, na correção e avaliação de algoritmos gerados em Portugol. Utilizando-se da teoria de compiladores foram desenvolvidas as análises léxica, sintática e semântica, e através da técnica de Processamento de Linguagem Natural foi realizada a compreensão do algoritmo, através do uso da análise pragmática. Os erros encontrados durante estas análises foram revertidos em códigos e gravados em um arquivo texto separado. Estes códigos dos erros são lidos em um Sistema Especialista que classifica os tipos de erros, calcula a pontuação do acadêmico no algoritmo e realiza ponderações a respeito dos erros encontrados.

Foi realizada uma pesquisa de natureza aplicada, pois foi utilizado o CIFluxProg, sistema já desenvolvido, e após estudo e verificação de seus módulos, foi desenvolvido o sistema proposto neste trabalho.

As etapas metodológicas seguidas para a elaboração do trabalho foram:

- Estudo da Ferramenta CIFluxProg.
- Levantamento dos quesitos a serem considerados quanto a correção e avaliação de algoritmos junto aos professores da disciplina de Algoritmos e Programação.
- Desenvolvimento das regras de produção referentes ao Sistema Especialista.
- Análise da melhor forma de realizar a compreensão do algoritmo, ou seja, a análise pragmática.
- Implementação do protótipo.

3.1 ESTUDO DA FERRAMENTA CIFLUXPROG

Para realização deste trabalho foi realizado um estudo mais aprofundado a respeito da ferramenta CIFluxProg, pois a estrutura da mesma foi utilizada para o desenvolvimento do CIFluxProgII.

Primeiramente, foram realizadas reuniões com o acadêmico responsável pelo desenvolvimento do CIFluxProg e o professor orientador, bem como o estudo das listagens das linhas de código do sistema. Nestas reuniões o acadêmico explicou o funcionamento e de que forma foram realizadas cada uma das etapas que foram implementadas no CIFluxProg.

Logo após, foram realizados testes com a ferramenta, para análise das possíveis melhorias e implementações necessárias para a realização do projeto. Chegou-se a conclusão que para a realização das etapas propostas seria necessário reiniciar o projeto desde o início, aproveitando-se a gramática da estrutura léxica e sintática, e ir modificando e incorporando as novas funcionalidades na própria gramática.

Em respeito da ferramenta CIFluxProg, foram utilizadas as gramáticas da análise léxica e sintática, e algumas telas que sofreram modificações para poderem suprir as necessidades do CIFluxProgII.

3.2 LEVANTAMENTO DOS QUESITOS

O levantamento dos requisitos a serem considerados pelo sistema foi realizado através de entrevistas com os professores da disciplina de Algoritmos e Programação. Esta etapa foi de suma importância para a realização da análise pragmática e o desenvolvimento do Sistema Especialista.

Primeiramente, foi solicitado a turma de Algoritmos e Programação II da Universidade do Vale do Itajaí – UNIVALI que realizasse exercícios em portugol,

sendo que esses exercícios abrangiam laços de repetição e laços de seleção. Estes exercícios foram repassados aos acadêmicos de segundo período pois, por ser início de semestre, os acadêmicos de primeiro período ainda não haviam visto a matéria. Logo após, foi realizada a correção dos exercícios efetuados pelos alunos, e detectados os problemas encontrados no desenvolvimento dos mesmos. Foi realizada nova reunião com os professores da disciplina, e nesta reunião, avaliados os problemas encontrados e o que deveria ser avaliado pelo sistema. Foi montada uma tabela com os principais erros a serem detectados pela análise morfológica (léxica), sintática e semântica, e apresentados aos professores, para análise e aprovação. Quanto aos erros pragmáticos, através dos exercícios corrigidos foram detectados os principais problemas quanto a interpretação do problema e incorporados como erros.

3.3 DESENVOLVIMENTO DAS REGRAS DE PRODUÇÃO

Para o desenvolvimento das regras de produção, foi montado um primeiro esqueleto e mostrado para os professores da disciplina. Nesta reunião foram discutidos os erros e a pontuação a ser perdida para cada caso de erro. Essas considerações foram de importância vital para o desenvolvimento das regras de produção, e para analisar quanto cada acadêmico deveria perder por erro cometido, ou por conjunto de erros cometidos. Foram desenvolvidas 744 regras de produção, com encadeamento do tipo *forward chaining*. Logo após, foi aplicado um filtro para verificação das regras, e as mesmas chegaram ao número de 273 regras. As regras foram estruturadas de forma a controlar a quantidade de erros por tipo, para assim poder gerar uma nota mais concisa.

3.4 ANÁLISE PARA REALIZAÇÃO DA ANÁLISE PRAGMÁTICA

Para realização da análise pragmática, foram levados em consideração os itens levantados pela correção dos exercícios dos acadêmicos e reunião com os professores. Através destes itens, verificou-se os problemas encontrados, e que grande parte dos erros eram voltados a estruturas como laços de repetição, laços de seleção, montagem de fórmula ou esquecimento de algum item. Através da reunião com os professores da disciplina chegou-se a conclusão que uma das maneiras de avaliação do algoritmo do

acadêmico é a verificação de entradas e saídas, tendo sido este item incorporado à análise pragmática. Para realizar a verificação de entradas e saídas o professor fornece algumas possíveis entradas, e gera as saídas corretas. O algoritmo do acadêmico deve gerar as mesmas saídas. A quantidade de três entradas fictícias foi escolhida como forma de garantia de que realmente as saídas estão corretas, pois com uma entrada somente o acadêmico poderá gerar uma saída correta ao acaso. Se as saídas tiverem problemas, foi incorporada uma procura por pontos chaves, a serem descritos pelos professores.

3.5 IMPLEMENTAÇÃO DO PROTÓTIPO

O protótipo foi implementado em C++, utilizando as gramáticas do CIFluxProg e incorporando maiores funcionalidades as mesmas. Foram realizadas mudanças nas telas do CIFluxProg, como um campo para o acadêmico visualizar o enunciado, erros, e a nota e comentário a respeito do algoritmo desenvolvido. Também foi incorporada uma nova tela para que o professor possa entrar com o enunciado, um algoritmo modelo e pontos chaves do algoritmo em si, sendo que estes itens serão utilizados para a verificação do algoritmo do acadêmico. A descrição da implementação do protótipo pode ser vista no Capítulo 4.

4. O SISTEMA DESENVOLVIDO

O sistema desenvolvido para este trabalho foi denominado CIFluxProgII e utiliza como base o Editor/Compilador do CIFLuxProg. Ao iniciar este projeto, a ferramenta CIFluxProg contava com um ambiente gráfico onde o usuário digitava seu algoritmo em Português Estruturado, e um interpretador, responsável pelas análises morfológica (léxica) e sintática do algoritmo desenvolvido pelo acadêmico. Para desenvolver o CIFluxProgII, foram tomadas por base as funcionalidades existentes no CIFluxProg, e acrescentadas novas. O modelo desenvolvido para o CIFluxProgII pode ser visualizado na Figura 15.

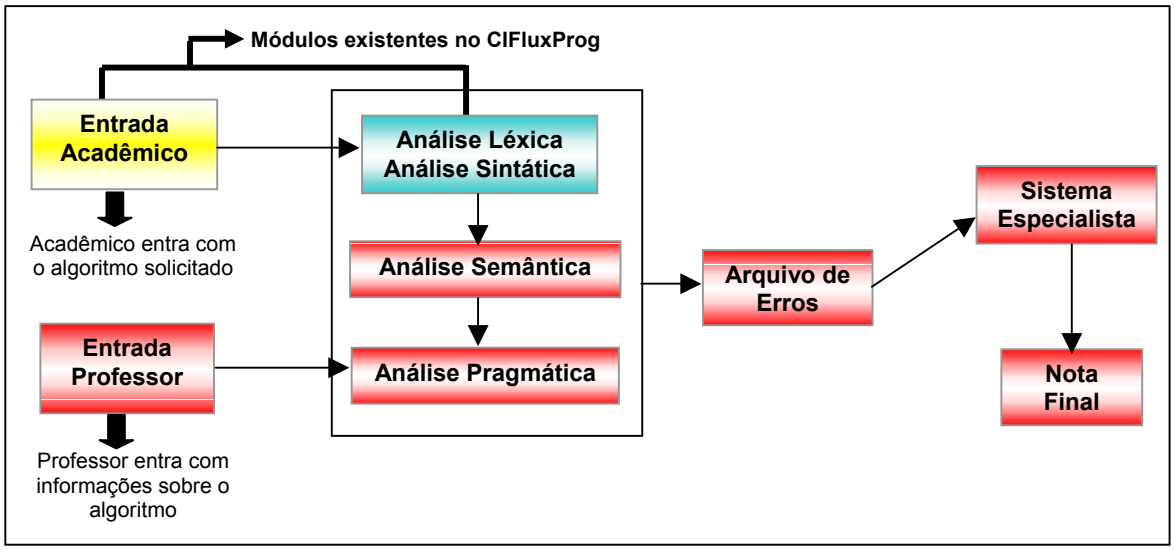

Figura 15: Módulos Principais do Projeto

Não foram utilizadas todas as estruturas encontradas no Português Estruturado, tendo sido desconsideradas estruturas como vetores, matrizes, procedimentos, funções, registros, arquivos e ponteiros.

O sistema desenvolvido é composto por dois módulos: o Módulo do Professor e o Módulo do Aluno. A seguir tem-se uma descrição mais detalhada a respeito dos dois módulos, além de uma descrição da implementação realizada.

4.1 MÓDULO DO ALUNO

Para o desenvolvimento do módulo do aluno utilizou-se a Ferramenta CIFluxProg. A ferramenta CIFLuxProg original conta com um ambiente gráfico em que o aluno entra com o algoritmo desenvolvido por ele, e a ferramenta realiza a análise morfológica, sintática e executa o programa.

Foram realizadas alterações na ferramenta CIFluxProg original. Dentre elas pode-se citar a incorporação do módulo do Professor, o que ocasionou em mudanças na estrutura do módulo do aluno, e a alteração do código fonte existente, para que gerasse um arquivo texto com os códigos dos erros encontrados, para ser utilizado pelo Sistema Especialista (SE). Ao encontrar um erro durante as análises morfológica, sintática e semântica, o sistema informa o erro ao usuário para que o mesmo o conserte, antes de dar continuidade a análise do algoritmo. Os erros, além de serem mostrados ao usuário, são gravados em um arquivo texto separado, para serem utilizados pelo sistema especialista posteriormente. O arquivo texto contém as seguintes informações: código do erro e linha aonde foi encontrado.As análises morfológicas e sintática existiam na primeira versão do CIFluxProg, mas foram revisadas, e foram incorporadas alterações Quanto a análise pragmática, ela será realizada somente após as outras análises, e também, ao encontrar erro, o sistema grava-os no arquivo texto. Após todas as análises terem sido realizadas, o Sistema Especialista realiza a leitura dos erros encontrados e informa ao acadêmico a respectiva nota a respeito do algoritmo desenvolvido e realiza observações a respeito dos erros encontrados.

4.2 MÓDULO DO PROFESSOR

O módulo do Professor é onde o professor entra com o enunciado do algoritmo, e os itens que devem ser considerados pela análise pragmática. Este é um módulo em que o professor irá descrever o algoritmo em si, e que será considerado quando for realizada a análise pragmática do algoritmo, ou seja, se o acadêmico realizou o que o enunciado solicitou. Neste módulo o professor também poderá ter um *feedback* a

respeito de como os alunos estão se saindo no desenvolvimento de cada algoritmo, e os erros encontrados. Estes detalhes serão fornecidos pelo Sistema.

4.3 A IMPLEMENTAÇÃO REALIZADA

A ferramenta CIFluxProgII foi implementada com o objetivo de aceitar algoritmos desenvolvidos em Português Estruturado (Portugol) com uma notação semelhante a da linguagem C, pois esta é a linguagem de programação ensinada aos acadêmicos do primeiro período do curso de Ciência da Computação da UNIVALI. Pretende-se que, com o uso da ferramenta os acadêmicos possam compreender melhor a disciplina de algoritmos, pois através dela os mesmos podem testar o algoritmo desenvolvido, e podem obter uma resposta quanto ao desenvolvimento correto ou não do mesmo, através da nota e comentários gerados.

Para a implementação do protótipo do CIFluxProgII também foi utilizada a linguagem de Programação C++, por ser uma ferramenta portável, versátil e concisa.

Para realização do protótipo, foram incorporadas as seguintes funcionalidades no CIFluxProg:

- Análise Semântica: utilizando-se da árvore sintática, nesta etapa será analisado o sentido do algoritmo. Foram realizadas averiguações quanto, por exemplo, a coerência no uso dos identificadores, utilizando-se uma gramática de atributos.
- Análise Pragmática: na análise pragmática é realizada a compreensão do algoritmo. É realizada uma entrada de dados, em que o professor informa dados referentes ao algoritmo a ser desenvolvido, além das entradas e saídas desejadas. Então, o sistema estuda a relação entre o algoritmo apresentado e o que o professor desejava.
- Sistema Especialista: as informações a respeito dos erros, que se encontram em um arquivo texto gerado após as análises é lido por um Sistema

Especialista, que verifica quais os tipos de erros encontrados e gera como saída a nota final do aluno e comentários a respeito dos erros encontrados.

Além da incorporação destes novos módulos, foi realizada uma revisão na gramática referente à análise morfológica (léxica) e análise sintática, incorporação de novas características, e nova geração das mesmas pelas ferramentas Lex e Yacc. No CIFluxProg, ao encontrar erros no algoritmo, o sistema simplesmente apontava a linha, não identificando o tipo de erro encontrado, tanto na análise morfológica quanto na sintática. Na ferramenta CIFluxProgII desenvolvida, o sistema detecta o tipo de erro, e informa a linha onde foi encontrado, para que o usuário possa consertá-lo, de modo a poder realizar a análise pragmática. Ao encontrar o erro, é informado ao usuário o tipo do erro (léxico, sintático, semântico ou pragmático), qual o erro encontrado e a respectiva linha, e jogado no arquivo texto o código do erro e linha encontrado.

Abaixo segue uma descrição sobre a realização da análise morfológica, sintática, semântica, pragmática e do Sistema Especialista desenvolvidos.

4.3.1 Análise Morfológica (Léxica) e Sintática

O CIFluxProgII foi baseado nos processos de análise léxica e sintática das ferramentas Lex e Yacc. Desta forma foi gerado o modelo sintático, semântico e a gramática do Portugol (Tabela 2), o que criou funcionalidades que auxiliaram a interpretação do código. A especificação BNF gerada pela ferramenta pode ser visualizada no Anexo I.

Lex e Yacc são ferramentas de auxílio à geração de compiladores. A ferramenta Lex gera um analisador léxico em C a partir de uma definição regular da linguagem, e através de padrões da ferramenta analisa os caracteres de uma entrada e converte estes caracteres em *tokens*. Após esta conversão, ele encontra identificadores nesta entrada e gera uma tabela de símbolos. Já a Ferramenta Yacc gera um analisador sintático em linguagem C do tipo *bottom-up*, a partir de gramáticas livres de contexto.

Ela usa regras estabelecidas e analisa os *tokens* gerados pela ferramenta Lex, criando uma árvore sintática, que procura impor uma hierarquia entre os *tokens*.

Tabela 2: Gramática de alguns elementos da linguagem portugol

PORTUGOL	DESCRIÇÃO
Inicio	Identifica o início do algoritmo em portugol
Fim	Identifica o fim do algoritmo em portugol
{	Inicio de bloco
}	Fim de bloco
Se( )	Desvio Condicional
senao	Negação do "se ( )"
enquanto( )	Laço condicional
para __ ate __ passo __	Laço condicional com repetição incremental
inteiro	Tipo de dado numérico inteiro
real	Tipo de dado numérico real
logico	Tipo lógico de dados
cadeia	Tipo de dado de cadeia de caracteres
verdadeiro	Valor verdadeiro do tipo de dado lógico
falso	Valor falso do tipo de dado lógico
leia( )	Instrução para entrada de dados
escreva( )	Instrução para a saída de dados

A análise morfológica (léxica) é a primeira etapa de um compilador, e identifica os itens léxicos de uma sentença. Ela efetua a análise da cadeia de caracteres do programa fonte (e é chamado também de *scanner* por este motivo), da esquerda para a direita, agrupando as seqüências que possuam significado coletivo e determinando sua classe. Os *tokens* são separados uns dos outros por caracteres denominados separadores, que são espaços, tabulações, quebras de linhas e operadores, e o analisador morfológico também desconsidera comentários, pois os mesmos não tem valor mediante a análise do algoritmo. Cada *token* é representado por três características:

- Classe do *token*: o que o *token* representa no algoritmo. Pode-se citar identificadores, cadeias numéricas, operadores, palavras reservadas. Esta

informação é de suma importância para a realização da análise sintática, posteriormente.

- Valor do *token*: de acordo com a classe do *token*, o mesmo pode ter um valor, como por exemplo, um número inteiro.
- Posição do *token*: indica o local (mais especificamente a linha) aonde o *token* foi encontrado.

Dado o algoritmo da figura 16, o CIFluxProg classificaria os *tokens* conforme mostrado na Tabela 3.

```
programa teste;
declaracoes
  inteiro a, b, c;
inicio
  leia(a);
  leia(b);
  c = a + b;
  escreva (c);
fim
```

Figura 16: Algoritmo de soma de duas variáveis

No CIFluxProg, através da aplicação de algoritmos sobre esta análise, buscou-se dados como identificação e criação de variáveis, contagem de linhas, armazenamento de constantes e identificação por tipo dos valores intrínsecos no portugol. Quanto as variáveis, o CIFluxProg guarda seus respectivos nomes, valores e tipos em uma estrutura de dados, para que as mesmas possam ser utilizadas nas análises posteriores.

Já a análise sintática verifica se as palavras inseridas no código fonte estão gramaticalmente corretas. Tratando-se do CIFluxProg, a análise sintática possibilitou uma maior preparação para a interpretação do algoritmo. Com os dados necessários identificados e estabelecidos, criou-se uma árvore de estrutura de dados para interpretação do algoritmo, que irá estabelecer a seqüência em que o algoritmo deve ser executado.

Uma pequena árvore é montada a cada estrutura analisada no portugol, contendo a seqüência e as informações necessárias para interpretação. Cada nó está diretamente relacionado com um determinado tipo de aplicação, como constantes,

identificadores ou operadores. As constantes são armazenadas para serem atribuídas como valores necessários nas estruturas do portugol. A estrutura da árvore é processada por um algoritmo que realiza a interpretação, e após todas as operações que necessitem desta árvore, por outro que destrói sua estrutura. Na Figura 17, um exemplo de uma árvore montada referente à linha de comando a = a + 10, baseado na gramática definida.

Tabela 3: Classificação de *tokens* de algoritmo dado

VALOR	CLASSE	POSIÇÃO
programa	Palavra Reservada	Linha 1
teste	Identificador	Linha 1
;	Caracter Especial	Linha 1
declaracoes	Palavra Reservada	Linha 2
inteiro	Palavra Reservada	Linha 3
A	Identificador	Linha 3
,	Caracter Especial	Linha 3
B	Identificador	Linha 3
,	Caracter Especial	Linha 3
C	Identificador	Linha 3
;	Caracter Especial	Linha 3
inicio	Palavra Reservada	Linha 4
leia()	Palavra Reservada	Linha 5
A	Identificador	Linha 5
leia()	Palavra Reservada	Linha 6
B	Identificador	Linha 6
C	Identificador	Linha 7
=	Atribuição	Linha 7
A	Identificador	Linha 7
+	Operador	Linha 7
B	Identificador	Linha 7
;	Caracter Especial	Linha 7
escreva ()	Palavra Reservada	Linha 8
C	Identificador	Linha 8
fim	Palavra Reservada	Linha 9

.

A Figura 18 apresenta um exemplo da estrutura de dados montada para o armazenamento dos comandos em Portugol. Nesta figura, os ponteiros realizam a identificação do que será necessário para a resolução do problema (condições, cálculos,

etc) e o que vem logo após na estrutura, caso haja. Como exemplo, um desvio condicional deverá saber quais as instruções caso a condição do fluxo seja verdadeira e quais as condições no caso do fluxo ser falso.

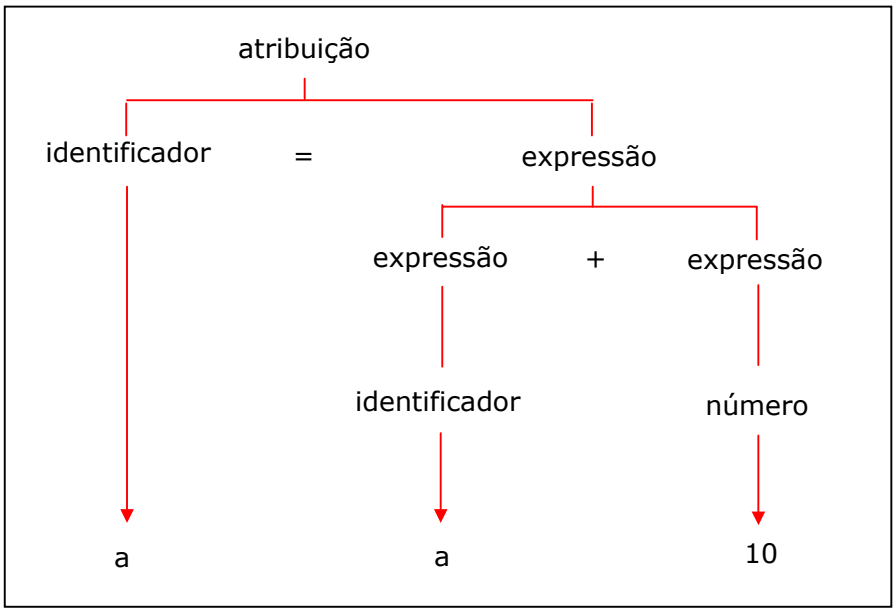

Figura 17: Árvore de derivação da linha de comando a = a + 10

Figura 18: Estrutura de dados de armazenamento de comandos em Português Estruturado

Após a construção da árvore, que contém os passos para a interpretação, um algoritmo realiza a execução do código em portugol. Para que isso ocorra, realizou-se a identificação de todas as estruturas presentes no portugol – laços de repetição, desvios condicionais, operadores de atribuição, entradas, saídas – e, a cada passo da execução, foi-se analisando cada nó da árvore de interpretação e cada área que deve haver uma execução. O algoritmo aloca os recursos necessários e executa o fragmento de portugol.

A realização do reconhecimento do Português Estruturado e a geração da execução é demonstrado na Figura 19.

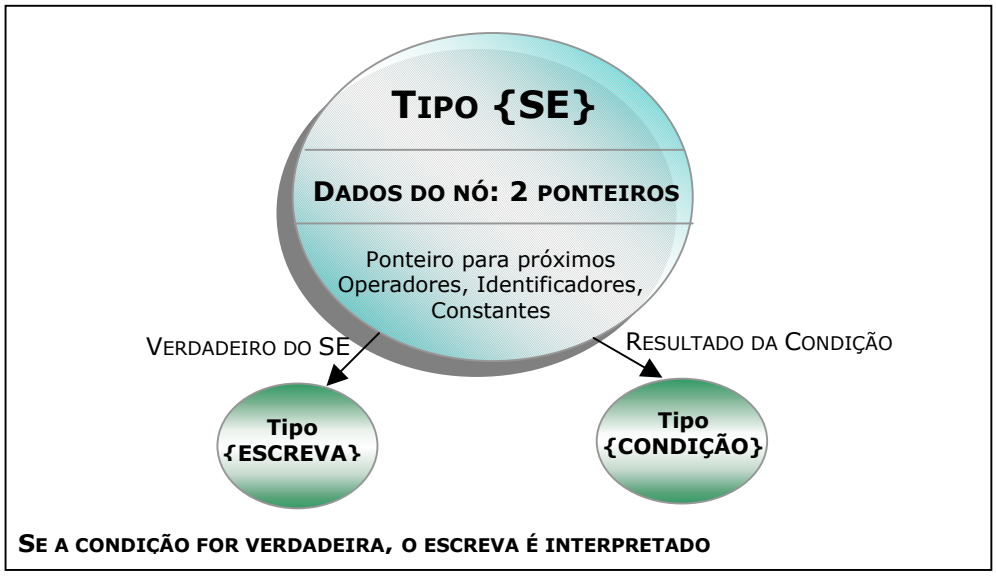

Figura 19: Visão da árvore de Estrutura de Dados e o tratamento do Interpretador

A árvore de interpretação não é montada inteira. São montados pequenos pedaços, que são analisados, e logo após é montada outra parte, e assim sucessivamente, até o final do algoritmo.

4.3.2 Análise Semântica

A análise semântica trata da interpretação que se pode atribuir ao conjunto de todas as suas sentenças, e deve garantir que o programa fonte respeite todas as restrições impostas pela linguagem. Ao realizar a análise semântica do CIFluxProgII, manteve-se o mesmo padrão descrito anteriormente: o sistema trabalhará com algoritmos simples, somente com a estrutura de seleção SE ... SENÃO e os laços de repetição PARA ... FAÇA, ENQUANTO ... FAÇA e FAÇA ... ENQUANTO.

A análise sintática e a análise semântica ocorrem em sincronia, conforme a árvore sintática vai sendo montada. O analisador semântico utiliza a árvore sintática montada e realiza tarefas como identificar operadores e operandos de expressões, reconhecer erros semânticos, realizar verificações de compatibilidade de tipo e analisar

o escopo das variáveis. A análise realizada no CIFluxProgII refere-se aos seguintes itens:

- Identificadores usados devem estar declarados.
- Identificadores não podem ser declarados mais do que uma vez.
- Os identificadores só poderão ser utilizados de acordo com a categoria na qual foram declarados.
- As expressões de teste em laços de seleção e repetição devem ser do tipo lógico.
- Ao lidar com operadores de atribuição, o identificador do lado esquerdo deve ser uma variável.
- O tipo da expressão deve ser igual ao tipo da variável do lado esquerdo da atribuição - excetuando-se que real aceita inteiro e cadeia aceita caracter, mas o contrário não é possível.
- Quando se lida com operadores relacionais, os operandos devem ser do mesmo tipo (excetuando-se real e inteiro, cadeia e caracter).
- Relacionado a operadores aritméticos, podem ser aplicados aos tipos inteiro e real.
- Ao se realizar uma operação aritmética que possuam operadores inteiros e reais, o resultado será real, assim como o uso do operador / (divisão) resulta em um real.
- Quando utilizados operadores lógicos, os operandos devem ser do tipo lógico também.

Para a verificação semântica, foi utilizada a tabela de símbolos elaborada durante a análise morfológica (léxica). A tabela de símbolos segue a estrutura mostrada na Figura 20.

```
struct tabela_símbolo {
   char nome[100];
   int tipo;
   int dimensao;
   int usada;
   struct tabela_simbolo *prox;
};
```

Figura 20: Estrutura da tabela de símbolos

É uma estrutura de nome tabela_símbolo, que guarda informações como nome da variável ou constante, qual o tipo de valor armazenado (Inteiro, Real, Lógico, Caracter, Cadeira de Caracteres ou Constante) , dimensão (normal ou vetor) e vezes em que a variável foi usada no código.

Esta estrutura é utilizada para guardar informações sobre os nomes declarados em um programa, e pode-se denomina-la, agora por diante, como Tabela de Símbolos. Ela é criada durante a etapa morfológica (léxica), mas os dados podem ser modificados durante as análises posteriores. A pesquisa na tabela de símbolos é realizada cada vez que um nome é encontrado no programa fonte, e alterações e exclusões sempre que um novo nome ou nova informação sobre um nome existente é obtida. Como vantagens encontradas durante a implementação pode-se citar a simplicidade para manutenção da mesma, e a boa performance em questão de inserção/remoção de identificadores.

```
void insereSimbolo(char nome[100], int tipo, int dimensao){
 int jadeclarada = 0;
 tab_aux = tab_simb;
 if (tab_aux != NULL){
  while (tab_aux != NULL){
    if (strcmp(tab_aux->nome,nome) == 0){
       yyerror("");
       printf ("Variavel %s ja declarada!\n", nome);
       jadeclarada=1; }
    tab_aux = tab_aux->prox;
  } }
if (jadeclarada==0){
 tab_aux = tab_simb;
 if (tab_aux != NULL){
   while (tab_aux->prox != NULL){
     tab_aux = tab_aux->prox; }
   novo_simbolo = (struct tabela_simbolo *) malloc (sizeof (struct
   tabela_simbolo));
   tab_aux->prox = novo_simbolo;
   strcpy(novo_simbolo->nome,nome);
   novo_simbolo->tipo = tipo;
   novo_simbolo->dimensao = dimensao;
   novo_simbolo->prox = NULL; }
 else {
   novo_simbolo = (struct tabela_simbolo *) malloc (sizeof(struct
   tabela_simbolo));
   strcpy(novo_simbolo->nome,nome);
   novo_simbolo->tipo = tipo;
   novo_simbolo->dimensao = dimensao;
   novo_simbolo->usada = 0;
   novo_simbolo->prox = NULL;
   tab_simb = novo_simbolo; } } }
```

Figura 21: Código de inserção de um símbolo na tabela de símbolos

A análise semântica, no CIFluxProgII, foi sendo realizada em paralelo com a análise sintática. Conforme a árvore sintática foi sendo montada, foi-se realizando pesquisas à tabela de símbolos para poder validar os tipos dos dados. Isso foi realizado atribuindo ações semânticas às produções da gramática, o que é conhecido como gramática de atributos. A principal função de uma gramática de atributos é incluir restrições semânticas às construções sintaticamente corretas, estabelecendo a dependência semântica entre todos os símbolos da gramática.

À medida que o analisador sintático reconhece um símbolo, o analisador semântico verifica se o símbolo se encontra ou não na tabela de símbolos e, não estando, o símbolo é armazenado. Caso contrário, o analisador procura verificar a compatibilidade entre os atributos do símbolo inserido e a forma coerente do seu uso. A Figura 12 exibe o código para inserção de um símbolo na tabela de símbolos.

4.3.3 Análise Pragmática

A Análise Pragmática trata das relações dos significados com o contexto do enunciado, ou seja, como o contexto influencia a interpretação do significado. Quando se trata de avaliação de algoritmos, a análise pragmática deve verificar se o que foi solicitado pelo professor realmente condiz com o desenvolvimento realizado pelo aluno. Como exemplo, o professor solicitou que o acadêmico desenvolvesse um algoritmo que exibisse a média aritmética de três avaliações, e o acadêmico elaborou o algoritmo exibido na Figura 22.

```
programa media;
declaracoes
   real nota1,nota2,nota3,med;
inicio
   leia(nota1);
   leia(nota2);
   leia(nota3);
   med = nota1 + nota2 + nota3;
   escreva(med);
fim.
```

Figura 22: Algoritmo para cálculo de média com erro no cálculo

Houve um erro no desenvolvimento do algoritmo da Figura 22. O acadêmico deveria realizar a média das notas, não a soma das mesmas. A análise pragmática deve indicar um erro neste caso, pois a operação apresentada não condiz com o que foi solicitado, portanto, há um erro de interpretação.

Para que pudesse ser efetuada a análise pragmática foi implementado um novo módulo, explicado anteriormente, em que o professor entra com o enunciado do algoritmo, e com informações relevantes a respeito do mesmo. Estes dados serão salvos em um banco de dados, para poderem ser utilizados pelo aluno posteriormente. Por exemplo, no algoritmo acima o professor informaria que serão efetuadas 3 leituras, haverá 1 saída, não existem laços de repetição, nem laços de seleção, e a operação a ser realizada é cálculo de média de 3 notas.

O acadêmico, ao entrar no módulo do acadêmico, selecionará o tipo de exercício que deseja realizar, e o programa automaticamente selecionará um dos exercícios cadastrados pelo professor naquela categoria. Tão logo o usuário insira o algoritmo, e peça para ser realizada a avaliação, o interpretador entrará em ação. Após a realização das análises léxica, sintática e semântica, será a vez da análise pragmática.

O processo de análise pragmática é composto por duas partes: a geração de entradas e saídas e a procura por pontos-chave explicitados pelo professor.

Para realizar a análise pragmática, em um primeiro momento, serão analisadas as entradas e saídas do algoritmo. O professor, ao fazer o cadastro do enunciado, cadastra também informações relevantes ao algoritmo. Dentre estas informações, constam 3 possíveis entradas e quais as três saídas que seriam geradas por esta entrada. Por exemplo, se o algoritmo em questão fosse um algoritmo para geração de fatorial, uma das formas de resolução poderia ser a demonstrada na Figura 23.

Como entrada, tem-se a leitura de um valor inteiro. O professor, por exemplo, cadastraria como possíveis entradas os valores inteiros 4, 5 e 7, e as saídas seriam, respectivamente, 24, 120 e 5040. Quando o sistema percorre o algoritmo do acadêmico,

ele irá executar o programa do aluno com as entradas indicadas pelo professor, e capturar as saídas que obtiver. Serão comparadas as saídas, para verificar se são as mesmas ou não. Caso sejam, o algoritmo será considerado correto em questão pragmática. Caso não sejam, o algoritmo do acadêmico será percorrido em busca das informações cadastradas como relevantes ao algoritmo pelo professor. Como exemplo, se o professor indicar que o algoritmo possui um laço de repetição que varia de 1 a 10, e que esse laço pode ser um laço para ... faça ou um laço enquanto .... faça, o sistema percorrerá o algoritmo a procura de um desses laços. Caso encontre, irá passar para o próximo item cadastrado, e percorrerá todos os itens cadastrados, até verificar todos. Portanto, a análise pragmática verifica as saídas geradas pelo sistema, e em caso de não encontrar problemas, verifica itens considerados importantes na correção pelo professor.

```
programa fatorial;
declarações
  inteiro fat,num, i;
inicio
  fat = 1;
  leia(num);
  para i = 1 ate num passo 1 {
  fat = fat * num;
  }
  escreva(fat);
fim
```

Figura 23: Algoritmo para geração de fatorial

Estes itens foram levantados juntamente com professores da disciplina de Algoritmos e Programação. Foram aplicados exercícios para a turma de 2º período da disciplina de Algoritmos e Programação, e em cima destes exercícios feito um levantamento dos principais erros praticados pelos acadêmicos. Verificou-se os seguintes erros pragmáticos:

- Leituras não efetuadas, ou não foram efetuadas todas as saídas necessárias.
- Sem instruções de saída, mesmo quando solicitado pelo algoritmo, ou erro nos dados de saída.
- Falta de laços de repetição ou seleção, necessários ao algoritmo.
- Valores de variação de laço de repetição incorretos.
- Condição lógica do laço de seleção incorreta, ou incompleta
- Cálculos aritméticos incorretos.

- Atribuições, quando necessárias, incorretas ou inexistentes.

E tomando por base estes erros encontrados na avaliação dos exercícios, foram levantados os itens que constam na tela do professor, durante o cadastro de cada enunciado.

4.3.4 Tratamento de Erros

Aqui será explicado o tratamento realizado quando o sistema encontra erros referentes a qualquer uma das análises realizadas. Ao encontrar erros morfológicos (léxicos), sintáticos ou semânticos, o sistema exibe o erro e a linha onde foi encontrado o erro para o usuário, grava as mesmas informações em um arquivo texto, e espera que o usuário corrija o erro. Tão logo o mesmo seja corrigido, continua a análise até o final, ou até encontrar novo erro. A análise pragmática só é realizada após o sistema efetuar as três análises anteriores, sendo realizada através da análise de entradas e saídas, e detalhes pertinentes ao algoritmo informados pelo professor.

Um diferencial importante entre o CIFluxProg e o CIFluxProg II são as mensagens de erro. No CIFluxProg simplesmente era mostrada uma mensagem de erro e a linha onde se encontrava. No CIFluxProg, além de serem gravados os erros para serem usados posteriormente, é exibida uma mensagem de erro que especifica claramente o tipo de erro encontrado pelo usuário (vide Figura 24), para que o mesmo possa ter noção do tipo de erro cometido. Na Tabela 4 podem ser visualizados os erros detectados no CIFluxProgII, código e classificação do erro.

```
if (strcmp(s, "ST01") == 0) {
   fprintf (stderr, "%s: ", s);
   printf ("Erro Sintatico na linha %d ", linha);
   printf (" * Falta de ponto e virgula\n");
}
```

Figura 24: Regra para exibição de tipo de erro

Tabela 4: Erros detectados no CIFluxProgII

CÓDIGO	ERRO	CLASSIFICAÇÃO
L01	Caracter desconhecido (não reconhecidos pela gramática)	Léxico
L02	Identificador inválido	Léxico
ST01	Falta de ponto e vírgula	Sintático
ST02	Parêntese não foi aberto	Sintático
ST03	Parêntese não foi fechado	Sintático
ST04	Palavra reservada com nome incorreto	Sintático
ST05	Estrutura de Seleção incompleta e/ou incorreta	Sintático
ST06	Estrutura de Repetição incompleta e/ou incorreta	Sintático
ST07	Colchete não foi aberto	Sintático
ST08	Colchete não foi fechado	Sintático
ST09	Palavra Reservada Faltante	Sintático
SM01	Identificador não declarado	Semântico
SM02	Identificador já declarado	Semântico
SM03	Tipos Incompatíveis	Semântico
SM04	Atribuição incorreta	Semântico
SM05	Operadores aritméticos aplicados de forma incorreta	Semântico
SM06	Uso do operador divisão: resultado deverá ser real	Semântico
SM07	Expressão de laços incorreta	Semântico
SM08	Constantes não recebem atribuição	Semântico
PRG01	Laço de repetição não condiz com o pedido	Pragmático
PRG02	Laço de seleção não condiz com o pedido	Pragmático
PRG03	Operação Incorreta	Pragmático
PRG04	Inicialização Incorreta	Pragmático
PRG05	Tipo de variável não condiz com o solicitado pelo enunciado	Pragmático
PRG06	Encadeamento entre laços incorreto	Pragmático
PRG07	Instrução faltante	Pragmático
PRG08	Saídas Incorretas	Pragmático
PRG09	Algoritmo não condiz com enunciado	Pragmático

A respeito dos erros léxicos, esse tratamento foi realizado diretamente na gramática léxica. Um exemplo de gramática de tratamento de erros pode ser visualizado na Figura 25.

```
[0-9][a-z0-9_]* { yyerro("L02"); return (" ");}
.              + yyerro("L01");
```

Figura 25: Gramática para tratamento de erros léxicos

Na primeira linha foi o tratamento realizado caso alguma variável inicie por valor numérico, o que não é permitido no CIFluxProg. Caso isto aconteça, será repassado o código do erro encontrado (L02) para a função yyerro, que se encontra no módulo de análise sintática. Já a segunda linha trata de caracteres inválidos durante o programa, como a seguinte linha de instrução: d = d # 2. O analisador léxico percorre todo o programa, e ao encontrar o caracter inválido envia o código do erro (L01) para a função yyerro. Nesta função o erro será gravado em um arquivo texto, e logo após o programa dará continuidade a análise a partir da linha seguinte.

Quando trata-se de erros sintáticos, foram realizadas alterações na gramática sintática. Um exemplo de tratamento de erro sintático é ilustrado na Figura 26.

```
program: dec_program VARIABLE ';'
         dec_declara lista_decl
         dec_inicio function dec_fim     { exit(0); }

         | dec_program VARIABLE { yyerro("ST01");}
          dec_declara lista_decl
          dec_inicio function dec_fim
          ;
dec_program:    PROGRAMA
         |       {yyerro("ST09");}
         ;
dec_inicio:     INICIO
         |        {yyerro("ST09");}
                 ;
dec_fim: FIM
         |        {yyerro("ST09");}
                 ;
dec_declara:    DECLARACOES
         |        {yyerro("ST09");}
```

Figura 26: Exemplo de tratamento de erro sintático

No caso acima, o erro tratado é o de código ST09, que é um erro referente à falta ou erro de palavra reservada. O escopo geral de um programa no CIFluxProg é:

PROGRAMA <Nome do Programa>;
DECLARACOES
<Declaração de Variáveis>
INICIO
<Corpo do Programa>
FIM

Caso alguma destas palavras seja escrita de forma errônea, ou não sejam incluídas no programa no local adequado, o programa detecta como sendo um erro sintático, e o código ST09 é enviado ao arquivo texto. Como exemplo da palavra PROGRAMA, ou o acadêmico digita PROGRAMA, ou o programa detecta como erro e envia o código do erro. Os erros sintáticos são tratados desta forma.

Os erros semânticos são tratados diretamente na gramática sintática, conforme mostrado na Figura 27.

```
valores:
          VARIABLE  { if (retornaExiste(nome_ident)){
                          $$ = id(nome_ident);
                      } else{
                          yyerro("SM01");
                      }}
```

Figura 27: Exemplo de erro semântico tratado na gramática sintática

Este erro trata da não declaração de algum identificador. Foi atribuída uma ação semântica à gramática sintática. Caso o valor tenha sido declarado, o mesmo é atribuído a árvore sintática. Senão, é chamada a função de erro, e gravado o erro no arquivo texto. O tipo de erro SM01 acontece quando o acadêmico gera a solução de seu algoritmo, e gera uma série de variáveis no corpo do trabalho, mas esquece de declara-las ou declarar alguma na área reservada para declarações.

Os erros léxicos, sintáticos e semânticos são todos tratados diretamente na gramática do programa. Os erros pragmáticos são tratados de forma diferente.

Durante as análises léxica, sintática e semântica, conforme os erros vão sendo encontrados vão sendo informados ao acadêmico para que o mesmo os corrija, e possa dar prosseguimento a análise. A análise pragmática só é realizada após todos os erros encontrados na outra análise terem sido arrumados. A verificação pragmática foi separada em duas partes: a análise de entradas e saídas, e dos itens importantes.

Em uma primeira fase, é realizada uma análise de entradas e saídas do algoritmo. O professor, ao cadastrar as informações pertinentes ao algoritmo, informa quantas entradas o algoritmo possui, e quantas saídas devem ser geradas. Também é solicitado para o professor cadastrar 3 valores fictícios para cada uma das entradas, e informar quais seriam os valores de saída para estas três entradas. Quando o sistema realiza a execução do algoritmo do aluno, e chega a análise pragmática o algoritmo do acadêmico é executado e é realizada uma comparação entre a saída do professor e a saída do aluno. Caso seja constatado que as saídas produzidas nos três exemplos foram iguais as do professor, é constatado que o algoritmo não possui erros pragmáticos. Caso as saídas não sejam iguais, o sistema percorre o algoritmo do acadêmico a procura dos detalhes cadastrados pelo mesmo, como, por exemplo, se o sistema possui laço de repetição, qual a variação. Caso o sistema encontre algum erro referente as informações, é retornado erro, e gravado no arquivo texto o código e linha do mesmo. Neste caso, o acadêmico não deverá realizar as correções, mas os erros podem ser visualizados pelo mesmo.

Logo após esta etapa, é realizada a contagem dos erros encontrados, e a análise dos mesmos pelo Sistema Especialista desenvolvido.

4.3.5 Sistema Especialista

Tão logo sejam realizadas as análises morfológica (léxica), sintática, semântica, pragmática, e tenham sido gravados os erros encontrados no algoritmo do acadêmico em um arquivo texto, o sistema percorre o arquivo texto e realiza a contagem dos erros encontrados por categoria. Os respectivos códigos são analisados por um Sistema Especialista, que gera uma nota de acordo com os erros encontrados, e fornece ao usuário a nota recebida bem como comentários a respeito do erro encontrado.

A base de conhecimento é o coração de um Sistema Especialista, e contém o domínio particular da aplicação. No CIFluxProgII foi utilizada uma base de conhecimento baseada em Regras. Foram geradas 744 regras, e quanto ao seu

encadeamento são do tipo *forward chaining*. Logo após, foi aplicado um filtro para verificação das regras, e as mesmas chegaram ao número de 273 regras.

A geração das notas é realizada conforme os tipos de erros encontrados, a quantidade encontrada e a análise de quanto cada erro inviabiliza o algoritmo. Para realizar esta análise foram realizadas reuniões com os professores da disciplina de algoritmos e programação do curso de Ciência da Computação da UNIVALI – Campus Itajaí para que os mesmos relatassem os erros encontrados em provas de alunos e realizassem um ranking dos mesmos, referente a importância de cada erro. Chegou-se a conclusão que o ranking de acordo com a categoria do erro fica desta forma:

- Erros Pragmáticos: estes foram classificados como erros de maior importância, erros que possuem uma incidência maior sobre os algoritmos. Os erros pragmáticos tratam de erros de compreensão do problema, e ao não compreender o acadêmico poderá realizar uma resolução completamente diferente do que foi solicitado. Dependendo do tipo de erro, o algoritmo completo poderá ser considerado errôneo.
- Erros Semânticos: este tipo de erro possui uma incidência menor que os erros pragmáticos, mas são erros de importância dentro de um algoritmo. Na grande maioria tratam de tratamento errôneo para com as variáveis. Na classificação, fica em segundo lugar a nível de incidência no algoritmo.
- Erros Sintáticos: ao realizar um erro sintático, o acadêmico está realizando erros que vão contra a gramática utilizada pela linguagem. São erros que incidem sobre o algoritmo, mas possuem uma importância menor sobre a avaliação, a não ser que sejam em grande número.
- Erros Léxicos: assim como os erros sintáticos, este tipo de erro possui uma incidência menor. Irão detectar tokens que não fazem parte da linguagem. Desde que não sejam realizados em grande número, são erros relevantes em uma avaliação. Podem ser realizados por distração ou pressa do acadêmico, assim como os erros sintáticos.

Levando-se em consideração as avaliações acima, foram realizadas ponderações a respeito dos erros detectados pelo sistema, e de acordo com a quantidade, e tipos de erros realizados pelos acadêmicos, foram geradas as notas e os comentários pertinentes ao algoritmo desenvolvido. Por exemplo, um acadêmico que produz um erro de código PRG09 (vide códigos e descrição de erros na Tabela 4), que trata de um erro pragmático em que o algoritmo não condiz com o enunciado, o acadêmico receberá nota mínima, e será exibido um texto sugerindo que o acadêmico leia o enunciado com mais cuidado, pois ele não compreendeu o que realmente foi solicitado. Como forma de exemplo, segue uma das regras realizadas:

```
if (l01==0) && (l02==0) && (ST01==0) && (ST02==0) && (ST03==0) &&
(ST04==0) && (ST05==0) && (ST06==0) && (ST07==0) && (ST08==0)
&& (ST09==1) && (SM01==1) && (SM02==0) && (SM03==0) &&
(SM04==0) && (SM05==0) && (SM06==0) && (SM07==0) && (SM08==0)
&& (PRG01==0) && (PRG02==0) && (PRG03==0) && (PRG04==0) &&
(PRG05==0) && (PRG06==0) && (PRG07==0) {
    printf("Nota: 9,00");
    printf("Você esqueceu alguma palavra reservada, como também de
    declarar um identificador. Procure cuidar com estes erros no próximo
    algoritmo.");
}
```

Quando o sistema percorre o arquivo texto em busca dos códigos de erros encontrados, ele vai armazenando os erros em variáveis, para ter-se uma contagem de quantos erros por código foram encontrados no algoritmo. Foram criadas variáveis para acumular a contagem dos erros com nomes iguais aos códigos dos erros, e assim facilitar a montagem das regras de produção. Por exemplo, na regra mostrada acima, as variáveis ST09 e SM01 possuem o valor 1, ou seja, foram encontrados 1 erro de código ST09 e 1 erro de código SM01, sendo estes erros, respectivamente, Palavra Reservada Faltante e Identificador não declarado. Ao encontrar estes dois erros o acadêmico perdeu um ponto, sendo sua nota 9,00. É escrito na tela a nota que o acadêmico tirou, e uma consideração a respeito dos erros encontrados.

As outras regras são formadas desta forma, sendo contados os erros e realizadas as inferências de acordo com a quantidade de erros encontrados.

4.4 TELAS DO SISTEMA

O acadêmico, ou mesmo o professor, ao acessar o CIFluxProgII entra em uma tela onde é solicitado seu login, senha e o módulo a ser acessado. Tão logo o usuário entre com estes dados, é realizada uma verificação no banco de dados para certificação de que o usuário pode acessar o módulo que deseja. Por exemplo, se o professor selecionar o módulo professor ou mesmo módulo aluno, é verificado o banco de professores para verificar se um professor quer entrar ou não. Caso não seja, é exibida mensagem informando que o usuário não possui direito a acessar aquele módulo. Se for um aluno acessando, será liberado somente o acesso ao módulo aluno (Figura 28).

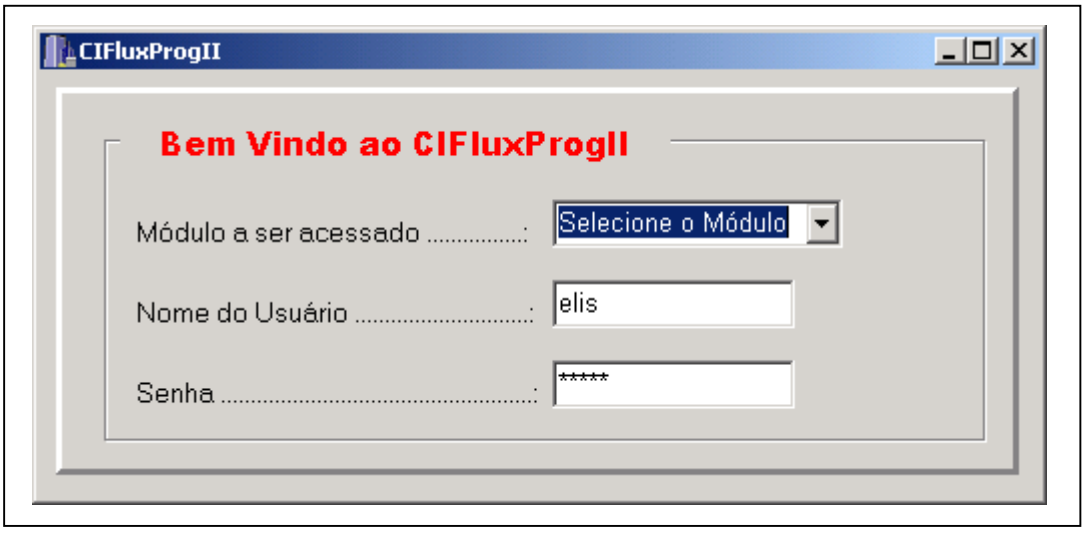

Figura 28: Tela de Login

Quando o acadêmico entra no módulo do aluno, a primeira tela que surge é a tela de seleção de exercícios, conforme mostra a Figura 29. Através dela o sistema sorteará o enunciado do exercício a ser gerado pelo acadêmico, dependendo da classificação que o professor fornecer ao exercício.

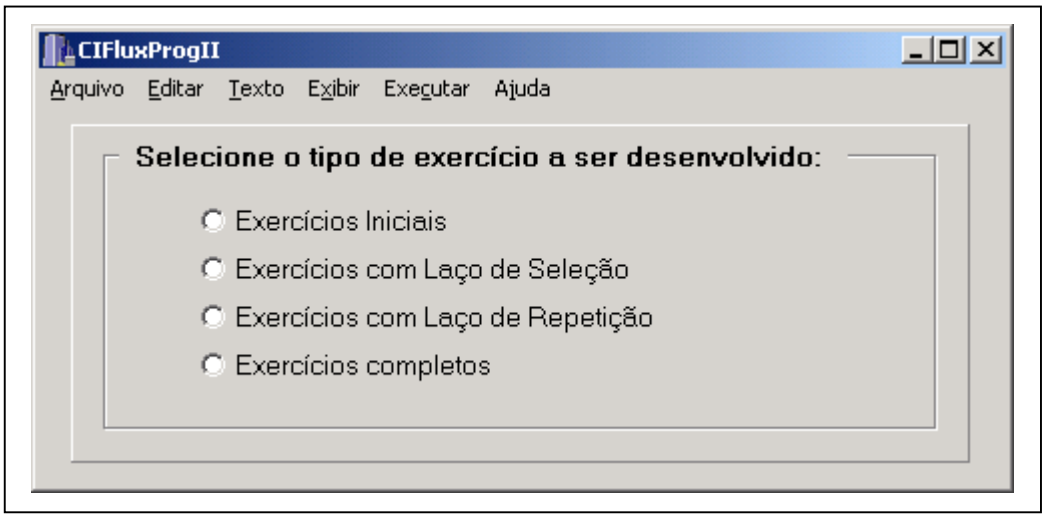

Figura 29: Menu de escolha de tipo de exercício a ser resolvido

Tão logo o acadêmico selecione o tipo de exercício que deseja resolver, a tela do módulo do aluno poderá ser visualizada (Figura 30). O usuário conta com um menu onde pode realizar funções normais em um sistema, como criar um novo algoritmo, abrir um algoritmo já existente, sair do Programa, copiar, colar, mudar textos, entre outras. Quando o usuário clica em executar, é gerada a série de analises referentes a análise do algoritmo, e exibido o resultado.

O acadêmico pode contar com um menu para auxílio ao desenvolvimento do algoritmo, onde pode-se encontrar os principais comandos e laços. Ao clicar sobre uma das opções, a mesma é inserida no algoritmo, na posição onde encontra-se o cursor. Para poder desenvolver seu algoritmo, o usuário digita-o em um campo memo, situado do lado esquerdo da tela, abaixo do menu. No lado direito, no enunciado do programa, consta o enunciado do algoritmo que o acadêmico irá desenvolver, selecionado entre aqueles cadastrados pelo professor e que pertençam ao tipo de algoritmo selecionado pelo acadêmico para desenvolvimento. Conforme vai realizando a análise, e encontrando algum erro, é mostrado o código do erro e o tipo de erro encontrado no memo Erros Encontrados. Tão logo sejam finalizadas as análises, o sistema especialista gera a nota e o comentário a respeito do algoritmo, que aparecem na parte inferior direita da tela.

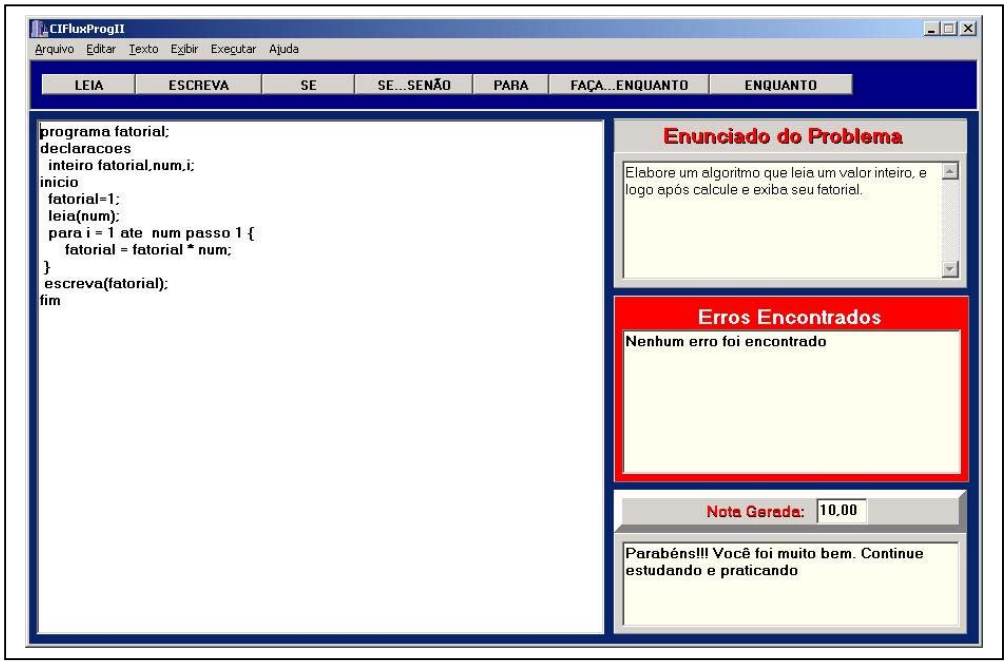

Figura 30: Tela do Módulo do Aluno

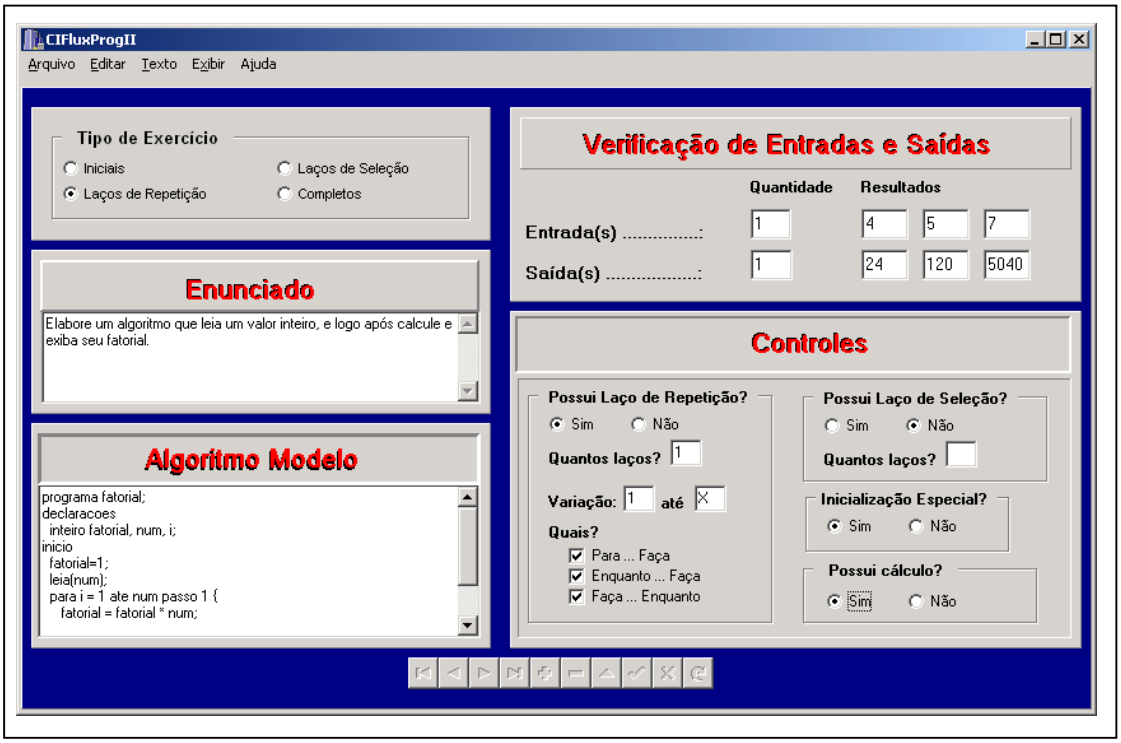

Figura 31: Tela do módulo do Professor

A Figura 31 mostra a tela do módulo do professor, que somente poderá ser acessada por usuários que possuam o login de professor. Nela, o professor irá cadastrar

todos os dados referentes ao enunciado que deseja disponibilizar aos alunos, com dados que serão utilizados na análise pragmática futuramente. Nesta tela o professor irá cadastrar o enunciado, em qual tipo ele se classifica, um algoritmo como modelo, a quantidade de entradas e saídas, e três valores aleatórios com as respectivas saídas – que servirão para testar as entradas e saídas, e informações a respeito do algoritmo, caso o algoritmo do aluno não apresente as saídas desejadas, levando-se em conta os valores identificados pelo professor.

5. CONCLUSÕES

A disciplina de Algoritmos e Programação é uma disciplina de importância vital no curso, e torna-se essencial que os professores efetuem as correções de exercícios e provas de forma coerente e mantendo um padrão quanto à avaliação. O CIFluxProgII foi elaborado com este intuito. A fim de avaliar o CIFluxProgII, três tipos de avaliações foram realizadas: avaliação do sistema junto aos alunos, avaliação das funcionalidades junto aos professores e avaliação de eficiência.

A avaliação do sistema junto aos alunos foi realizada em duas etapas: na primeira etapa foi apresentada a ferramenta aos alunos, e os mesmos desenvolveram dois algoritmos utilizando a ferramenta. Não foi efetuado nenhum tipo de treinamento anterior, pois dessa forma a análise ergonômica seria mais clara, pois uma das idéias do CIFluxProgII era o mesmo ser intuitivo. Após o uso da ferramenta, os acadêmicos responderam a um questionário de avaliação da ferramenta (vide Anexo II). Em um segundo momento, exatamente duas semanas após a primeira avaliação, os acadêmicos reavaliaram a ferramenta. Neste período de tempo foram resolvidos os problemas detectados na avaliação anterior, bem como incorporadas novas melhorias. Após a resolução dos exercícios propostos, os acadêmicos responderam ao mesmo questionário aplicado na primeira etapa, sendo assim possível verificar a melhoria do sistema e grau de satisfação dos acadêmicos em relação à ferramenta.

Após avaliação dos questionários, verificou-se um crescente aumento de satisfação dos acadêmicos na segunda etapa. Pode-se verificar, pelas respostas dadas aos questionários, que os alunos conseguiram compreender melhor problemas clássicos de programação e sua lógica de solução. Pelo fato de poderem obter uma avaliação do algoritmo logo após tê-lo desenvolvido possibilitou aos alunos um desempenho melhor na solução dos erros cometidos. Outro ponto bastante favorável foi que 100% dos alunos conseguiram acharam a navegação do sistema fácil e intuitiva.

A avaliação da ferramenta juntos aos professores deu-se em duas etapas: uma

avaliação anterior ao desenvolvimento do sistema e outra após a conclusão do sistema. Na primeira etapa, foi realizada uma reunião informal junto aos professores da disciplina de Algoritmos e Programação, com o objetivo de levantar as informações consideradas importantes na avaliação de algoritmos. No dia anterior foi aplicada a acadêmicos da disciplina de Algoritmos e Programação alguns exercícios. Estes exercícios tinham por objetivo avaliar os problemas encontrados durante o desenvolvimento. Na reunião foram avaliadas as respostas dos acadêmicos e levantados os seguintes tópicos:

- avaliando-se as respostas dos acadêmicos, quais os maiores problemas encontrados;
- levando-se em consideração erros de análise (léxica, sintática, semântica, pragmática), enumerar o grau de importância dos mesmos na avaliação de provas de algoritmos;
- quais os tipos de erros léxicos, sintáticos, semânticos e pragmáticos devem ser detectados;
- quanto à análise da compreensão do enunciado pelo acadêmico, quais os itens considerados importantes para poder verificar a compreensão correta pelo aluno;
- se a avaliação de saídas, dadas respectivas entradas, seria significativa para avaliação da compreensão do algoritmo.

Através desta reunião foram levantados os itens importantes para a avaliação de algoritmos, e desenvolvido o sistema. Numa segunda etapa, os professores realizaram uma avaliação informal do sistema, com o intuito de verificar se todos os requisitos levantados na primeira reunião foram contemplados no sistema, e se realmente o sistema conseguia realizar a avaliação dos algoritmos e gerar nota e comentários relevantes. Os professores mostraram-se satisfeitos com o sistema, mas solicitaram que se verificasse uma maior quantidade de erros, e que na hora que o sistema estivesse realizando a verificação léxica, sintática e semântica, não houvesse necessidade de o sistema parar para o acadêmico encontrar o erro. Estas modificações serão incorporadas em uma versão futura.

A terceira e última avaliação, a avaliação da eficiência, foi realizada por um conjunto de 5 acadêmicos e dois professores da disciplina. Os acadêmicos resolveram duas questões referentes a laços de repetição e laços de seleção, sendo que suas respostas também foram corrigidas pelos professores. O sistema mostrou-se eficiente quanto a avaliação léxica, sintática e semântica, mas poderia ser melhorada quanto a questão pragmática. A avaliação tomando-se por base pontos chaves cadastrados pelo professor, e a respectiva comparação com o algoritmo do acadêmico não mostrou-se eficiente para todos os casos. O que sugere-se para modificações futuras seria a utilização de outras técnicas para a avaliação pragmática, como a utilização de Redes Neurais Artificiais. Outra sugestão para futuras modificações seria utilizar o processamento de linguagem natural de forma que o sistema possa ir "conversando" com o acadêmico, e através de sugestões e questionamentos ir mostrando ao acadêmico seus erros, formas de resolver os problemas encontrados, e sugestões pertinentes ao problema. Desta forma, o sistema, além de avaliar o algoritmo do acadêmico, poderia auxiliar no processo de ensino de algoritmos.

As ferramentas Lex e Yacc mostraram-se satisfatórias no desenvolvimento da análise léxica e sintática, pois o desenvolvimento das análises foi realizada em menor tempo que se tivesse sido feito com código de programação, e sua elaboração mostrou-se relativamente fácil.

Quanto a técnica de Sistemas Especialistas, a mesma se mostrou eficiente somente para o universo abrangido nesta primeira versão, mas não seria uma técnica eficiente no caso de ampliação do sistema. A combinação dos itens a serem considerados nas regras de produção geraram 744 regras de produção, onde foi aplicado um filtro e finalizou-se em 273 regras. Uma ampliação do universo de abrangência aumentaria consideravelmente o número de regras, o que tornaria o sistema mais lento, menos eficiente e robusto. Uma sugestão para futuros trabalhos seria o estudo e utilização de outras técnicas para a geração de nota e parecer. Quanto ao parecer, poderia ser utilizado Processamento de Linguagem Natural, em que o sistema poderia manter um diálogo com o acadêmico referente aos erros encontrados e problemas que

os mesmos possam acarretar no desenvolvimento dos algoritmos.

Como sugestão recomenda-se a ampliação do universo a ser abrangido pelo sistema, incluindo vetores, matrizes, procedimentos, funções e ponteiros, o que seria de extrema valia, pois o sistema passaria a abranger toda a matéria passada na disciplina, e seria de grande auxílio ao professor.

O desenvolvimento do CIFluxProg mostrou-se válido para a correção e avaliação de algoritmos, e mostrou-se uma ferramenta eficaz no auxílio aos primeiros passos dos acadêmicos de computação, bem como ao professor, facilitando seu trabalho de análise e correção de algoritmos. Deve-se destacar que a ferramenta não substitui o trabalho do professor, mas auxilia para que o mesmo possa realizar suas avaliações de forma coerente.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, Alfred U.; SEIT, Ravi; ULLMAN, Jeffrey D. **Compiladotes, princípios, técnicas e ferramentas**. Editora LTC: Rio de Janeiro, 1995.

ALMEIDA, Eliana S. de; COSTA, Evandro de B.; SILVA, Klebson dos S. et al. **AMBAP: Um ambiente de apoio ao aprendizado de Programação**. Anais do XXII Congresso da Sociedade Brasileira de Computação. X Workshop sobre educação em computação. Universidade Federal de Santa Catarina – UFSC. Florianópolis, 15 a 19 de julho de 2002. ISBN: 85-88442-23-X.

ANDRÉ, Hildebrando A. de. **Gramática Ilustrada.** São Paulo: Editora Moderna, 1982. 3 ed.

ASCENSIO, Ana Fernanda Gomes; CAMPOS, Edilene Aparecida Veneruchi de. **Fundamentos da Programação de Computadores**. São Paulo: Prentice-Hall, 2002.

BARR, Avron; FEIGENBAUM, Edward A. **The Handbook of Artificial Intelligence**. Addison-Wesley Publishing Company, INC., 1986. vol. 1.

BARRETO, Jorge Muniz. **Inteligência Artificial no limiar do século XXI**. Florianópolis: J.M.Barreto, 1997.

BOOKSHEAR, J. Glenn. **Ciência da Computação: uma visão abrangente**. Porto Alegre: Bookman, 2000.

CASTRO, Eduardo Bernardes de; SÁ, Maria Auxiliadora Diniz de. **Habilidades, competências, valores e atitudes – um perfil para o profissional de computação e informática**. Anais do XXII Congresso da Sociedade Brasileira de Computação. X Workshop sobre educação em computação. Universidade Federal de Santa Catarina – UFSC. Florianópolis, 15 a 19 de julho de 2002. ISBN: 85-88442-23-X.

CASTRO, Thais Helena Chaves de; CASTRO JÚNIOR, Alberto Nogueira de; MENEZES, Crediné Silva de et al. **Utilizando Programação Funcional em disciplinas introdutórias de computação**. Anais do XXII Congresso da Sociedade Brasileira de Computação. X Workshop sobre educação em computação. Universidade Federal de Santa Catarina – UFSC. Florianópolis, 15 a 19 de julho de 2002. ISBN: 85-88442-23-X.

CHAVES, Eduardo O. C. **Informática e Educação**. Disponível em: [http://www.edutec.net/Textos/Self/EDTECH/cartgraf.htm]. Acessado em: (13 de novembro de 2003).

DAZZI, Rudimar Luís Scaranto. **Algoritmos**. Itajaí, 1998. Curso de Ciência da Computação, Universidade do Vale do Itajaí.

DELGADO, Carla; XEXEO, José Antonio Moreira; SOUZA, Isabel Fernandes de; CAMPOS, Marcio; RAPKIEWICZ. **Uma abordagem pedagógica para a iniciação ao estudo de algoritmos**. Anais do XXIV Congresso da Sociedade Brasileira de Computação. Universidade Federal da Bahia. Salvador, 31 de julho a 06 de agosto de 2004. ISBN: 85-88442-94-9.

DIWAN, Amer; WAITE, William M.; JACKSON, Michele H. **PL-Detective: a system for teaching programming language concepts**. Disponível em: [http://www-plan.cs.colorado.edu/diwan/sigcse04.pdf]. Acessado em: (18 de outubro de 2004).

DOMINGUES, Maria José Carvalho de Souza. **Estimulando as inteligências múltiplas através da Realidade Virtual**. Disponível em: [http://www.lrv.ufsc.br/drv/artigos/maria_Jose/estimulando%20as%20IM%20pela%20RV.doc]. Acessado em: (05 de novembro de 2003 11:11)

ESMIN, Ahmed Ali Abdalla. **Portugol/Plus: uma ferramenta de apoio ao ensino de lógica de programação baseado no portugol**. Disponível em: [http://www.c5.c1/ieinvestiga/actas/ribie89/118.html]. Acessado em: (18 de maio de 2000 15:28)

FARACO; MOURA. **Gramática**. São Paulo: Editora Ática, 2000. 12 ed.

FERNANDES, Jorge Henrique Cabral. **Ensino Introdutório de Computação e Programação: uma abordagem concreta, construtivista, lingüística e histórica**. Anais do XXII Congresso da Sociedade Brasileira de Computação. X Workshop sobre educação em computação. Universidade Federal de Santa Catarina – UFSC. Florianópolis, 15 a 19 de julho de 2002. ISBN: 85-88442-23-X.

FERNANDES, Anita Maria da Rocha Fernandes. **Inteligência Artificial**. Disponível em: [http://www.cttmar.univali.br/~anita/ia/d105/index.htm]. Acessado em: (05 de abril de 2002).

GEVARTER, William B. **Artificial Intelligence Expert Systems Computer Vision and Natural Language Processing**. New Jersey: Noyes Publications, 1984.

GIRAFFA, Lucia M. M.; MARCZAK, Sabrina S.; ALMEIDA, Gláucio. **O ensino de algoritmos e programação mediado por um ambiente na *web***. Anais do XXIII Congresso da Sociedade Brasileira de Computação. Universidade Estadual de Campinas – UNICAMP. Campinas, 02 a 08 de agosto de 2003. ISBN: 85-88442-60-4.

INSITE. Disponível em: [http://lingüística.insite.com.br/nlp.phtml]. Acessado em: (22 de abril de 2002)

KOLIVER, Cristian; DORNELES, Ricardo Vargas; CASA, Marcos Eduardo. **Das (muitas) dúvidas e (poucas) certezas do ensino de algoritmos**. Anais do XXIV Congresso da Sociedade Brasileira de Computação. Universidade Federal da Bahia. Salvador, 31 de julho a 06 de agosto de 2004. ISBN: 85-88442-94-9.

LACERDA, José Neves de. **Generalização de fatos na compreensão de textos em linguagem natural.** Florianópolis, 1996. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina. Florianópolis, novembro de 1996.

LUGER, George F. **Artificial intelligence: structures and strategies for complex problem solving**. Grã-Bretanha: Person Education, 2002. 4th ed.

MANZANO, José Augusto N.G.; OLIVEIRA, Jayr Figueiredo de. **Algoritmos: lógica para desenvolvimento de programação**. São Paulo: Érica, 1996.

MARTIN, James H. **Introduction**. Disponível em: [http://www.cs.Colorado.edu/~martin/SLP/slp_ch1.pdf]. Acessado em: (24 de abril de 2002).

MAYERHOFER, Mário André. **Um ambiente para ensino de algoritmos introdutórios**. Doutorado - PUC/Departamento de Informática. Rio de Janeiro: 1995.

MENDES, Antonio José Nunes; GOMES, Anabela Jesus. **Suporte à aprendizagem da programação com o ambiente SICAS**. Disponível em: [http://www.c5.cl/ieinvestiga/actas/ribie2000/papers/319/]. Acessado em: (20 de setembro de 2004).

MENEZES, Crediné Silva de; NOBRE, Isaura Alcina Martins. **Um ambiente cooperativo para apoio a cursos de introdução a programação**. Anais do XXII Congresso da Sociedade Brasileira de Computação. X Workshop sobre educação em computação. Universidade Federal de Santa Catarina – UFSC. Florianópolis, 15 a 19 de julho de 2002. ISBN: 85-88442-23-X.

MIRANDA, Elisangela Maschio de Miranda. **Protótipo de um Sistema de Auxílio à Avaliação de Algoritmos**. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação, Universidade Federal de Santa Catarina. Itajaí, novembro de 2000.

OLIVEIRA, Álvaro Borges de. **Algoritmos**. Itajaí, 1998. Curso de Ciência da Computação, Universidade do Vale do Itajaí.

OLIVEIRA, Fabio Abreu Dias de. **Processamento de linguagem natural: princípios básicos e a implementação de um analisador sintático de sentenças da língua portuguesa**. Disponível em: [http://www.inf.ufrgs.Br/procpar/disc/cmp135/trabs/992/Parser/parser.html]. Acessado em: (22 de abril de 2002 14:15]

OLIVEIRA, Itamar Leite de. **Uma abordagem conexionista para resolução de anáforas pronominais.** Florianópolis, 1997. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina. Florianópolis, fevereiro de 1997.

OLIVEIRA, Alcione de Paiva. **Processamento da Linguagem Natural**. Disponível em: [http://www.ufv.br/dpi/alcione/ia/ln.htm]. Acessado em: [22 de abril de 2003 21:35]

PASSERINO, Liliana Maria. **Página de disciplinas ministradas**. Disponível em: [http://www.ulbra.tche.br/~lilianap]. Acessado em: [06 de dezembro de 2003 18:09]

PRICE, Ana Maria de Alencar; TOSCANI, Simão Sirineo. **Implementação de linguagens de programação: compiladores**. Porto Alegre: Instituto de Informática da UFRGS: Editora Sagra Luzzatto, 2001. 2 ed.

RABUSKE, Renato A. **Inteligência artificial**. Florianópolis: Editora da UFSC, 1995.

RAMOS, Ronaldo Fernandes. **Sistemas Especialistas - uma Abordagem Baseada em Objetos com Prototipagem de um Selecionador de Processo de Soldagem**. Florianópolis, 1995. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina. Florianópolis, fevereiro de 1995.

RICH, Elaine. **Inteligência Artificial**. São Paulo: Makron Books, 1994.

RUSSELL, Stuart; NORVING, Peter. **Artificial intelligence**. Prentice Hall, 2003.

SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madsen. **Algoritmos**. São Paulo: Makron Books, 1998.

SANTIAGO, Rafael de; DAZZI, Rudimar Luís Scaranto. **Ferramentas que auxiliam o desenvolvimento da lógica de programação**. Anais do XII Seminário de Computação. Universidade Regional de Blumenau – FURB. Blumenau, 05 a 08 de agosto de 2003.

SANTIAGO, Rafael de; DAZZI, Rudimar Luís Scaranto. **Interpretador de Portugol**. Artigo submetido e aprovado para o IV Congresso Brasileiro de Computação. Universidade do Vale do Itajaí – UNIVALI. Itajaí, 08 a 12 de outubro de 2004.

SANTOS, Regina Coeli Freitas dos; XEXÉO, José Antonio Moreira; RAPKIEWICZ, Clevi Elena et al. **Inovando a dimensão estratégica do ensino de Ciência da Computação**. Anais do XXIII Congresso da Sociedade Brasileira de Computação. Universidade Estadual de Campinas – UNICAMP. Campinas, 02 a 08 de agosto de 2003. ISBN: 85-88442-60-4.

SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. **Anais do II Curso de Qualidade de Cursos de Graduação da área de Computação e Informática.** Workshop sobre educação em computação. Pontifícia Universidade Católica do Paraná – PUCPR, Campus Curitiba. Editora Universitária Champagnat. Curitiba, 15 e 16 de julho de 2000. ISBN: 85-7292-052-8.

SONG, J. S.; HAHN, S. H.; TAK, K. Y.; KIM, J.H. **An intelligent tutoring system for introductory C language course**. Computers Education. Vol. 28 No. 2. pp. 93 – 102. 1997.

SOUSA, Almir Joaquim de; VIDAL, Fábio Silveira; COSTA, Fredson Vieira et al. **RAFF – Um compilador para facilitar o aprendizado de algoritmos**. Disponível em: [www.ulbra-to.br/ensino/43020/ artigos/anais2002/ Encoinfo2002/RAFF.pdf]. Acessado em: [27 de julho de 2004 20:35]

SOUZA, Eliane Moreira Sá de; GRANDI, Gilberto; DAZZI, Rudimar Scaranto et al.. **Reavaliando o ensino de algoritmos.** I Simpósio Catarinense de Computação, Itajaí, v. 1, p. 80-88, agosto 2000(a).

SOUZA, Eliane Moreira Sá de; GRANDI, Gilberto; DAZZI, Rudimar Scaranto et al. **Um novo enfoque para o processo ensino-aprendizagem de lógica e algoritmos em cursos de graduação.** I Simpósio Catarinense de Computação, Itajaí, v. 1, p. 69-79, agosto 2000(b).

ANEXOS

Anexo I

Léxico

```
[-()<>=+*/;{}.,] {
            return *yytext;
          }

[0-9]+              {    int aux = atoi(yytext);
                                ins_valor(&nconst, &aux, 1);  return (VALOR_INTEIRO); }

[0-9]+\.[0-9]+      {           float aux = atof(yytext);
                                ins_valor(&nconst, &aux, 2);     return (VALOR_REAL); }

\'[^\']*\'          {      char aux = *yytext;
                                ins_valor(&nconst, &aux, 3); return (VALOR_CARACTER);}

\"[^\"]*\"          {    char aux[256]; strcpy (aux, yytext);
                                ins_valor(&nconst, &aux, 5);  return (VALOR_CADEIA);}


">="          return GE;
"<="          return LE;
"=="          return EQ;
"!="          return NE;
"enquanto"            return WHILE;

"para"                return FOR;
"ate"                 return TO;
"passo"               return STEP;
"faca"                return FACA;
"se"          return IF;

"senao"       return ELSE;
"escreva" return PRINT;
"leia"                return SCAN;

"inteiro"          {return (INTEIRO);}
"real"            {return (REAL);}
"caracter"          {return (CARACTER);}
"logico"           {return (LOGICO);}
"cadeia"           {return (CADEIA);}


"defina"           return (DEFINA);
"definatipo"        return (DEFINATIPO);
"estrutura"        {return (ESTRUTURA);}
"fimestrutura"      {return (FIMESTRUTURA);}

"declaracoes"             {return(DECLARACOES);}
"inicio"       {return(INICIO);}
"fim"                     {return(FIM);}
"programa"                {return(PROGRAMA);}


"verdadeiro"        {  int aux = 1;
                                ins_valor(&nconst, &aux, 4); return (VALOR_LOGICO); }


"falso"           {  int aux = 0;
                                ins_valor(&nconst, &aux, 4); return (VALOR_LOGICO); }


[a-z][a-z0-9_]* { strcpy (yylval.nome, yytext); strcpy (nome_ident, yytext); return (VARIABLE);}
```

```
[0-9][a-z0-9_]* { yyerro("L02"); return ("  ");}

"\n"                    {linha++;}
[ \t+]      ;      /* ignore whitespace */

.           + yyerro("L01");
```

Sintático e Semântico

```
program:   dec_program VARIABLE ';'
                    dec_declara lista_decl
                    dec_inicio function dec_fim     { exit(0); }

          |dec_program VARIABLE { yyerro("ST01");}
                    dec_declara lista_decl
                    dec_inicio function dec_fim
      ;

dec_program:      PROGRAMA
                  |           {yyerro("ST09");}
                  ;

dec_inicio:       INICIO
          |     {yyerro("ST09");}
          ;

dec_fim: FIM
          |     {yyerro("ST09");}
          ;

dec_declara:      DECLARACOES
          |     {yyerro("ST09");}
          ;

lista_decl:         lista_decl decl
                  | decl
                  ;

decl:         dec_const
                  | dec_tipo
                  | lista_dec_var
                  ;

dec_const     : DEFINA ID_CONST valor ';' { if (busca_tab(nome_const) == NAO_ACHOU)

          insere_tab (nome_const, tipog);
                                                                                    else

          trata_erro (nome_const, 3);
                                                                                  }
                  ;
valor   :     VALOR_INTEIRO           { tipog = TIPO_INTEIRO;}
                  | VALOR_REAL        { tipog = TIPO_REAL;}
                  | VALOR_CARACTER    { tipog = TIPO_CARACTER;}
                  | VALOR_LOGICO              { tipog = TIPO_LOGICO;}
                  | VALOR_CADEIA              { tipog = TIPO_CADEIA;}
                  ;
dec_tipo: DEFINATIPO tipo ident ';'
                  | DEFINATIPO ESTRUTURA lista_dec_var FIMESTRUTURA ';'
```

```
                ;
tipo    :       INTEIRO         { tipog = TIPO_INTEIRO;}
                | REAL          { tipog = TIPO_REAL;}
                | CARACTER      { tipog = TIPO_CARACTER;}
                | LOGICO        { tipog = TIPO_LOGICO;}
                | CADEIA        { tipog = TIPO_CADEIA;}
                ;
ident   : VARIABLE {
                        insere_tab(nome_ident, tipog);
                        yylval.sIndex = retorna_endereco(nome_ident);}


                | ID '[' ID_CONST ']'
                | ID '[' ID_CONST ']' '[' ID_CONST ']'
                | ID '[' VALOR_INTEIRO ']'
                | ID '[' VALOR_INTEIRO ']' '[' VALOR_INTEIRO ']'
                ;
lista_dec_var   : lista_dec_var dec_var
                                |dec_var
                                ;
dec_var         : tipo lista_var ';'
                | tipo lista_var {yyerro("ST01");}
                                ;
lista_var       : lista_var ',' ident
                                | ident
                                ;

function:
        function stmt       {/*ex($2); freeNode($2);*/ }
      | /* NULL */
      ;

stmt:
        ';'                                                     { $$ = opr(';', 2, NULL, NULL);                 }

      | expr ';'                                                { $$ = $1;                                      }
      | PRINT abre_par expr fecha_par ';'                       { $$ = opr(PRINT, 1, $3);               }
      | PRINT abre_par expr fecha_par           { yyerro("ST01");}
              | SCAN abre_par expr fecha_par ';'                { $$ = opr(SCAN, 1, $3);                }
      | SCAN abre_par expr fecha_par            { yyerro("ST01");}
              | expr '=' expr                                   { yyerro("ST01");}
      | expr '=' expr ';'                                      { $$ = opr('=', 2,$1 , $3);              }
      | WHILE abre_par expr fecha_par '{' stmt '}' { $$ = opr(WHILE, 2, $3, $6);                         }
              | FOR expr TO expr STEP expr stmt          { $$ = opr(FOR, 4, $2, $4, $6, $7);         }
              | FACA stmt WHILE abre_par expr fecha_par  { $$ = opr(FACA, 2, $2, $5);                    }
      | IF abre_par expr fecha_par stmt %prec IFX { $$ = opr(IF, 2, $3, $5);                }
      | IF abre_par expr fecha_par  ELSE stmt       { $$ = opr(IF, 3, $3, $5, $7);              }
      | stmt_list                           { $$ = $1;                                      }
      ;

stmt_list:
        stmt                            { $$ = $1; }
      | stmt_list stmt                  { $$ = opr(';', 2, $1, $2); }
      ;

expr:
      VALOR_INTEIRO         { $$ = con(nconst); }
              | VALOR_REAL          { $$ = con(nconst); }
              | VALOR_CADEIA        { $$ = con(nconst); }
              | VALOR_LOstmtGICO        { $$ = con(nconst); }
              | VALOR_CARACTER      { $$ = con(nconst); }
      | VARIABLE            { if (retornaExiste(nome_ident)){
                                        $$ = id(nome_ident);
                                      }else{
```

```
                                        yyerro("Variavel nao encontrada");
                                    }
                                }
        | abre_par expr fecha_par { $$ = $2;                    }
        | expr '=' expr      { $$ = opr('=', 2,$1 , $3); }

    | '-' expr %prec UMINUS { $$ = opr(UMINUS, 1, $2);  }
    | expr '+' expr         { $$ = opr('+', 2, $1, $3); }
    | expr '-' expr         { $$ = opr('-', 2, $1, $3); }
    | expr '*' expr         { $$ = opr('*', 2, $1, $3); }
    | expr '/' expr         { $$ = opr('/', 2, $1, $3); }
    | expr '<' expr         { $$ = opr('<', 2, $1, $3); }
    | expr '>' expr         { $$ = opr('>', 2, $1, $3); }
    | expr GE expr          { $$ = opr(GE, 2, $1, $3);  }
    | expr LE expr          { $$ = opr(LE, 2, $1, $3);  }
    | expr NE expr          { $$ = opr(NE, 2, $1, $3);  }
    | expr EQ expr          { $$ = opr(EQ, 2, $1, $3);  }

    ;

abre_par: '('
                    |       {yyerro("ST02");}
                    ;

fecha_par:          ')'
                    |       {yyerro("ST03");}
                    ;

achave:   '{'
                    |       {yyerro("ST07");}
                    ;

fchave:   '}'
                    |       {yyerro("ST08");}
                    ;
```

Anexo II

Ficha de Avaliação do CIFluxProgII

Nome: _____ **Data**: ___/___/_____

É a primeira vez que faz a disciplina?

☐ Sim ☐ Não

Quanto a Ergonomia do Sistema:
1. O Sistema é fácil de usar e navegar?

☐ Sim ☐ Não
2. Os ícones são auto explicativos?

☐ Sim ☐ Não
3. As instruções estão em linguagem clara?

☐ Sim ☐ Não
4. A fonte das letras no texto é confortável?

☐ Sim ☐ Não
5. As cores das telas são cansativas?

☐ Sim ☐ Não

Quanto à aprendizagem:

1. A utilização do sistema lhe auxilia na compreensão da lógica de programação?

☐ Sim ☐ Não: qual sua crítica? _____
2. Você sentiu alguma evolução ao usar o sistema no que se refere ao raciocínio lógico?

☐ Nenhuma ☐ Pouca ☐ Indiferente ☐ Alguma ☐ Muita

Quanto ao desempenho:

1. O sistema apresentou algum erro de execução?

☐ Sim ☐ Não
2. O sistema apresentou o resultado de sua tarefa de forma correta?

☐ Sim ☐ Não