visto e de acordo

MÁRCIO LUIZ DE ANDRADE NETTO 26/02/88

DESENVOLVIMENTO DE UM SISTEMA INTEGRADO DE COMPUTAÇÃO SIMBÓLICA

AUTOR: Eng. José Dutra de Oliveira Neto

ORIENTADOR: Prof. Dr. Márcio Luiz de Andrade Netto

Dissertação apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, em cumprimento às exigências para obtenção do Grau de Mestre.

Fevereiro - 1988

Aos meus pais e a Dri, que sempre acreditaram em mim

AGRADECIMENTOS

Ao professor Márcio, pela orientação e confiança.

A UNICAMP, pela oportunidade de desenvolvimento deste trabalho.

A CAPES e a FAPESP, pelo apoio financeiro.

Ao CTI, pela utilização dos micros.

Aos colegas da pós-graduação, pela amizade neste período, em especial ao Evandro e Ivan.

A Dri, pelo apoio nas fases mais difíceis.

E a todos que direta ou indiretamente contribuiram para a realização deste trabalho.

SUMÁRIO

RESUMO

O objetivo principal deste trabalho é mostrar a viabilidade de um sistema de manipulação algébrica em ambientes computacionais de pequeno porte. O mesmo foi desenvolvido em **prolog**, acrescido de funções especialmente desenvolvidas para manipulação de derivadas, matrizes e integrais.

ABSTRACT

The main objective of this work is to show the viability of an algebraic manipulation system in microcomputers. The system was written in PROLOG language extended with statements especially developed to deal with differentiation, matrices, symbolic integration and algebraic manipulation.

PREFÁCIO

A computação simbólica têm como principal característica, a manipulação de expressões matemáticas formais tendo como resultado expressões matemáticas equivalentes. Difere da computação numérica pelo fato de realizar todo o processo com símbolos literais e posteriormente substituí-los, se for desejado, por valores numéricos.

Esta área vem, juntamente com a manipulação numérica, oferecer uma alternativa precisa e eficiente para a análise algébrica nos casos em que haja uma grande manipulação de fórmulas matemáticas. Existem muitos casos em que a manipulação simbólica apresenta ampla vantagem em relação aos métodos numéricos. O ambiente ideal teria a disponibilidade de ambos os métodos , com a utilização dependente da aplicação específica.

A área de manipulação numérica está num estágio bem desenvolvido e já possui um grande número de trabalhos e ferramentas disponíveis, o mesmo não ocorrendo com a manipulação simbólica que tem um potencial enorme a ser desenvolvido. Já existem alguns "softwares" comerciais disponíveis, tais como Mumath / muMATH 80/, Maple / MONARD e NAKANISH - 1984/ e Macsyma /MOSES -1984/, bem como algumas máquinas simbólicas, tais como LISP MACHINE.

Este trabalho tem por objetivo principal mostrar a viabilidade de um sistema integrado de manipulação simbólica em sistemas computacionais de pequeno porte. O presente trabalho consta de uma abordagem teórica sobre o assunto, sua implementação e uma aplicação prática do mesmo.

No primeiro capítulo descrevemos o histórico da manipulação simbólica, as técnicas aplicadas, as áreas de atuação, os ambientes de programação, as linguagens utilizadas e os principais sistemas existentes.

No segundo propomos a especificação de um sistema de computação simbólica.

No terceiro apresentamos a implementação de um sistema denominado SICS.

No quarto apresentamos uma interface especialmente desenvolvida para o sistema SICS. Ela foi desenvolvida para facilitar o desenvolvimento de programas em **prolog** nas aplicações utilizando o interpretador expandido.

No quinto temos um exemplo de aplicação do SICS e algumas experiências realizadas com o mesmo.

No último capítulo propômos algumas sugestões para desenvolvimentos futuros , comentários e conclusões sobre o trabalho.

Finalmente temos os apêndices referentes às especificações dos programas, bem como orientações gerais para o uso adequado do sistema.

CAPÍTULO I

I.1- UM HISTÓRICO SOBRE MANIPULAÇÃO SIMBÓLICA

Os primeiros trabalhos na área de manipulação simbólica foram de Kahrimaniam e de Nolan em 1953 .Estes realizavam diferenciação de forma tabular onde ainda existiam transformações manuais.

Em seguida tivemos um grande avanço com o projeto Dartmouth em 1953 que além da diferenciação tinha a integração, solução de equações diferenciais de primeira ordem e a simplificação. Neste projeto as transformações já eram automáticas.

No ramo específico da integração simbólica tivemos um grande impulso a partir da década de 60 com a divisão do ramo em três sub-áreas: Inteligência Artificial, Matemática e Algébrica. Na área da Inteligência Artificial temos o sistema pioneiro SIN(Slagle) que em 1961 empregava técnicas que permitiam igualar o comportamento médio de um calouro de ciências exatas no processo de integração. Já na manipulação algébrica surgiu o Mathlab (1968) que foi o embrião de um grande sistema existente ainda hoje que é o MACSYMA. No ramo da matemática surgiram novos algorítmos eficientes que foram sendo incorporados aos sistemas existentes. A linguagem utilizada era LISP, a qual permitiu um grande avanço na manipulação de símbolos.

Apesar dos avanços nesta área, poucos sistemas tiveram êxito pois a área despertava pouco interesse nos meios comerciais e acadêmicos que persistiam no uso exclusivo dos métodos numéricos. Com o desenvolvimento das áreas de Inteligência Artificial, máquinas simbólicas, algorítmos matemáticos eficientes e acesso/armazenamento de informações, criaram-se condições favoráveis ao êxito destes sistemas.

Surgiram Mumath /muMATH 80 /, Macsyma (versão otimizada)/MOSES 1974/ e Maple /MONARD e NAKANISH - 1984/ e outros hoje disponíveis comercialmente.

Como paradigma de programação adotou-se, oriunda do projeto Japonês de quinta geração(administrado pela ICOT), a linguagem PROLOG, que faz manipulação simbólica de maneira eficiente. Juntamente com LISP, são as duas linguagens mais utilizadas nesta área.

Recentemente surgiram máquinas simbólicas que aumentam a eficiência dos programas simbólicos. São máquinas com múltiplos paradigmas de programação, arquitetura específica para tratamento de símbolos e ainda possuem os recursos do cálculo numérico. A máquina SM45000 (Arquitetura tipo tag) da Integrated Inference Machine Inc. tem estas características. Sua memória real pode chegar a 240Mbytes e a virtual a 20Gbytes sendo também compatível com o PC-DOS a nível de arquivos. Outras máquinas simbólicas existentes são as Lisp Machines da Lisp Machine Inc., Symbolics , LMI, família XEROX 1100 , NTT Corp. e as máquinas de inferência baseadas na linguagem prolog como PSI-II da ICOT, Mitsubishi Eletric Corp. e Oki Eletric Industry Co.

I.2-SISTEMAS DE MATEMÁTICA SIMBÓLICA

Embora os sistemas computacionais sejam mais utilizados para processar dados numéricos , dispõe-se de linguagens e programas para manipulação algébrica há mais de 20 anos. O histórico sobre sistemas de matemática simbólica está bem descrito em /SUNDBLAD - 1974/.

Não há sistemas computacionais bons para todos os tipos de cálculos e sim ferramentas boas para cada um deles. As linguagens dos sistemas simbólicos são boas para cálculos algébricos.

Os sistemas de matemática simbólica são comumente denominados de Sistemas de Computação Algébrica (C.A.S), apesar de trabalharem com simplificação trigonométrica, cálculos e outras operações não algébricas.

Aplicações:

- Álgebra linear
- Equações diferenciais
- Equações diferenciais parciais
- Probabilidades e Estatística
- Teoria dos números
- Combinatória
- Álgebra moderna
- Usuários de rotinas específicas como integração e derivação automática
- Máquina de calcular para auxiliar em pequenos problemas algébricos (Equações de segundo grau, funções primitivas, etc).

Analisaremos as principais características dos sistemas de computação simbólica :

Recursos de memória

Os recursos de memória necessários para os sistemas computacionais simbólicos variam de 200kbytes de memória para uso moderado de cálculos, podendo atingir valores da ordem de dezenas Mbytes.

Há também os sistemas modulares em que o usuário utiliza somente os recursos necessários para a solução de seu problema específico. Neste caso 44kbytes de memória são suficientes para um sistema como o Mumath.

Portabilidade

A maioria dos sistemas atuais foram desenvolvidos nas linguagens PROLOG ,LISP e "C" disponíveis na maioria dos computadores utilizados. A portabilidade existe a nível de sistema operacional.

Facilidade de Manutenção

O usuário deve ter condições de executar pequenas correções no sistema, sem que haja necessidade da regeração do mesmo.

Sintaxe Algébrica

A sintaxe da linguagem usada na interação com a máquina deve facilitar a utilização dos recursos disponíveis do sistema. Sua notação deve ser semelhante à utilizada pelos livros-texto de cálculo ou linguagens clássicas de computação.

Depuração

O objetivo da depuração é otimizar o tempo e o esforço do programador. Neste sentido o sistema deve ter um controle eficiente sobre os erros e a sintaxe do programa. Estes recursos de depuração devem ser opcionais por razões de eficiência. Um bom sistema de mensagem de erro deve estar disponível.

Documentação

Uma boa documentação deve ser de fácil leitura, com introdução motivadora; tendo descrição completa da linguagem.

Alternativas de Respresentação

Um sistema deve conter mecanismos de representação de fórmulas equivalentes de modo a possibilitar a melhor escolha num dado contexto. A transformação, o armazenamento e a recuperação destes formatos deve ser uma característica destes sistemas.

Exemplo de fórmulas equivalentes:

		melhor representação para:
cos2x+cosx		-> Análise por métodos de Fourier
2cos(x)^2 + cosx - 1		-> Análise através de polinômios trigonométricos

Saída

O sistema algébrico deve possuir uma boa formatação de saída de modo a ter uma representação, a mais natural possível.

Uma característica importante é a saída bi-indexada para melhor visualização dos índices nas fórmulas matemáticas.

Precisão

Os sistemas algébricos possuem uma boa precisão em aritmética racional , mas para os casos onde grande precisão é necessária, o melhor procedimento é o cálculo da fórmula pelo sistema algébrico e sua avaliação através de sistemas numéricos. Estes últimos já possuem ferramentas numéricas eficientes e por isto farão uma avaliação mais precisa dos resultados.

Algorítmos

A utilização de algorítmos algébricos e analíticos está em crescente desenvolvimento, sendo que grande parte dos sistemas atuais já têm alguns incorporados.

Estrutura de dados

A representação interna das fórmulas deve ser de tal forma que torne o seu armazenamento e manipulação eficientes.

Interpretação e compilação

Vários sistemas são interpretados facilitando com isto a interação e a depuração dos programas a serem desenvolvidos.

A compilação deve ser considerada com o intuito de otimizar e ganhar velocidade de execução em um programa já depurado.

I.3 - DIFERENCIAÇÃO SIMBÓLICA

Um dos primeiros trabalhos nesta área foi desenvolvido por Kahrimanian em 1953 /HANSON, CAVINESS e JOSEPH - 1962 /, escrito para o sistema UNIVAC I. A entrada deveria ser cuidadosamente preparada e então transformada em uma forma tabular onde teríamos uma sequência de funções elementares. As funções elementares eram diferenciadas pelas regras elementares de diferenciação. O resultado obtido deveria então ser transformado manualmente da forma tabular para a forma analítica.

Outro trabalho nesta área foi o de NOLAN em 1953 /HANSON, CAVINESS e JOSEPH - 1962 / que tinha a mesma forma de entrada, mas diferindo na transformação para a forma tabular, onde o sentido da transformação era do parêntese mais externo para o mais interno. O processo consistia na introdução de um operador de diferenciação na forma tabular resultante e, através de aplicações sucessivas de regras elementares, teríamos a forma resultante da derivação, que deveria ser transformada manualmente para a forma analítica. Este mesmo processo foi utilizado por SLAGLE em seu sistema SAINT de Slagle.

Podemos ressaltar também o Projeto Dartmouth /HANSON, CAVINESS e JOSEPH - 1962 /que além da diferenciação era capaz de executar integração simbólica, resolver equação diferencial de primeira ordem e efetuar simplificações . O processo de diferenciação era similar ao de NOLAN ,diferindo apenas na forma de entrada e saída. A entrada era simbólica, mas

com restrições de hierarquia entre os operadores, acarretando com isto, excessivos parênteses nas expressões. A saída já era na forma analítica.

O trabalho desenvolvido pelo grupo de Hanson em 1962 permitia uma notação matemática natural, utilizando os operadores usuais como +,-,*,/. Era permitida a utilização de funções trigonométricas, hiperbólicas, exponenciais e logarítmas, mas com o limitante de primeiro grau. A expressão era transformada em árvore e representada através de uma tabela de triplas, onde cada tripla consistia de dois nós e um operador da árvore. As regras de diferenciação eram aplicadas a cada tripla. Isto criava uma matriz com diversas linhas para dada tripla . Novas linhas que necessitassem de simplificação eram eliminadas assim que criadas. Na saída a forma tabular era convertida em forma analítica.

No projeto FORMAC /SAMMET - 1966/ a diferenciação é realizada em uma expressão construída pelos operadores FORMAC. O usuário pode especificar uma ou mais variáveis de diferenciação e a ordem num único comando. O processo consiste na distribuição e execução do operador de diferenciação ao longo da árvore formada pela expressão matemática. O resultado é então simplificado pelo sistema.

Os sistemas atuais já tratam a diferenciação de uma maneira mais natural. Um exemplo típico são os sistemas Mumath e Macsyma.

No caso do Mumath/ muMATH - 1980/ podemos derivar expressões simbólicas contendo polinômios e/ou funções racionais e trancendentais. Sua notação de entrada é a própria expressão matemática e a sua variável de derivação.

Caso o sistema desconheça a regra de derivada para uma dada função, seu procedimento poderá seguir dois caminhos:

- A resposta será zero se os argumentos ou operadores não contiverem a variável de diferenciação

  DIF(F(Y),X)-> 0

- A resposta não é avaliada nos outros casos

  DIF(F(X),X) -> DIF(F(X),X).

Já no Macsyma/ FATEMAN - 1980/ MOSES - 1974/ a diferenciação é realizada sobre polinômios, funções racionais e transcendentais sem restrição de tamanho ou complexidade. Sua gama de regras de diferenciação engloba um domínio insuperável dentre os sistemas existentes.

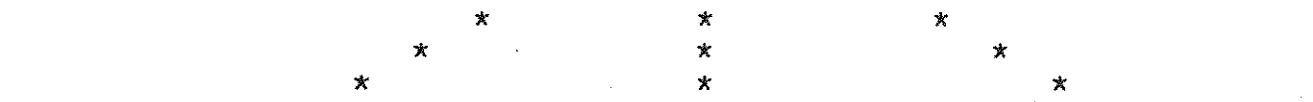
I.4- INTEGRAÇÃO SIMBÓLICA

Integração simbólica é considerada como um programa de resolução de problemas. São usadas técnicas de Inteligência Artificial, técnicas algébricas e matemáticas para resolver estes problemas.

A área de integração simbólica teve um grande impulso a partir da década de 60, quando se dividiu em três grandes grupos, conforme a estratégia de solução dos problemas / MOSES - 1971/:

Início: Slagle(1960)

3 RAMOS PRINCIPAIS



Os primeiros trabalhos desenvolvidos nesta área utilizavam métodos de Inteligência Artificial, tentando simular o processo humano através de técnicas heurísticas. Nesta área surgiu além do programa de Slagle (SAINT) o sistema SIN de Moses.

Na área de manipulação algébrica encontramos o sistema para o cálculo de integrais de funções racionais de Manove (Sistema Mathlab) e dos trabalhos de Horowitz e Tobey.

No ramo da matemática podemos ressaltar os trabalhos de Risch para o cálculo da integral de funções elementares. Estes métodos tiveram um grande sucesso no cálculo de integrais de funções transcendentais e foram anexados ao sistema SIN em 1969 por Moses.

Nos sistemas atuais como o Mumath / muMATH - 1980/ podemos calcular integrais definidas e indefinidas. Devemos fornecer a expressão matemática e a variável de integração e assim obter o resultado. Caso o sistema não consiga realizar a integral de partes da expressão, a resposta conterá partes avaliadas e outras com o símbolo da integral INT , indicando o insucesso da integração.

O sistema utiliza em seu processo a distribuição sobre somas, extração de fatores independentes, integrais elementares, regras de redução e a regra deriva-divide. Sua abrangência é reduzida, mas suficiente para um grande número de aplicações.

O Macsyma / MOSES - 1974/ é um dos sistemas existentes mais completos para a solução de integrais. Possui entre outros módulos o de integração de funções racionais, integração indefinida, algorítmo de integração de Risch, integração definida,etc.

Seu sucesso não se restringiu ao meios acadêmicos, e hoje o sistema em uma versão otimizada está sendo comercializado pela SIMBOLICS INC e engloba vários algorítmos eficientes.

I.4.1- SAINT (Simbolic Automatic INTegrator)

Desenvolvido por James R. Slagle no Massachussets Institute of Technology como tese de doutoramento, tendo como orientador M. L. Minsky /SLAGLE - 1961/. O seu objetivo principal foi o de solucionar problemas elementares de integração simbólica.

O programa tenta chegar à solução do problema, percorrendo a árvore de solução a partir do nó inicial até o nó terminal, expandindo ou não os nós encontrados. A estratégia de busca utilizada que foi posteriormente incorporada ao SIN por Moses em 1969, é determinada por técnicas heurísticas.

Abragência:

Integração indefinida,definida e múltipla.

Tipo de integrando:

Funções elementares de variáveis reais.
Definições de funções elementares:
1- constante é uma função elementar
2- variável é uma função elementar
3- soma e produto de funções elementares são funções elementares
4- função elementar elevada a uma potência (de função elementar) é função elementar.
5- função trigonométrica de uma função elementar é uma função elementar.
6- logarítmos ou função trigonométrica inversa de uma função elementar é função elementar.

Nível de solução:
Universitário iniciante, da área de ciências exatas.

Para solução dos problemas, o sistema dispõe:
26 formas padrões
18 transformações algorítmicas
10 transformações heurísticas

Organização:

Similar ao "Lógic Theoristic" de Newell, Shaw e Simon. /NEWELL L., J. SHAW AND SIMON - 1957/

Linguagem utilizada:

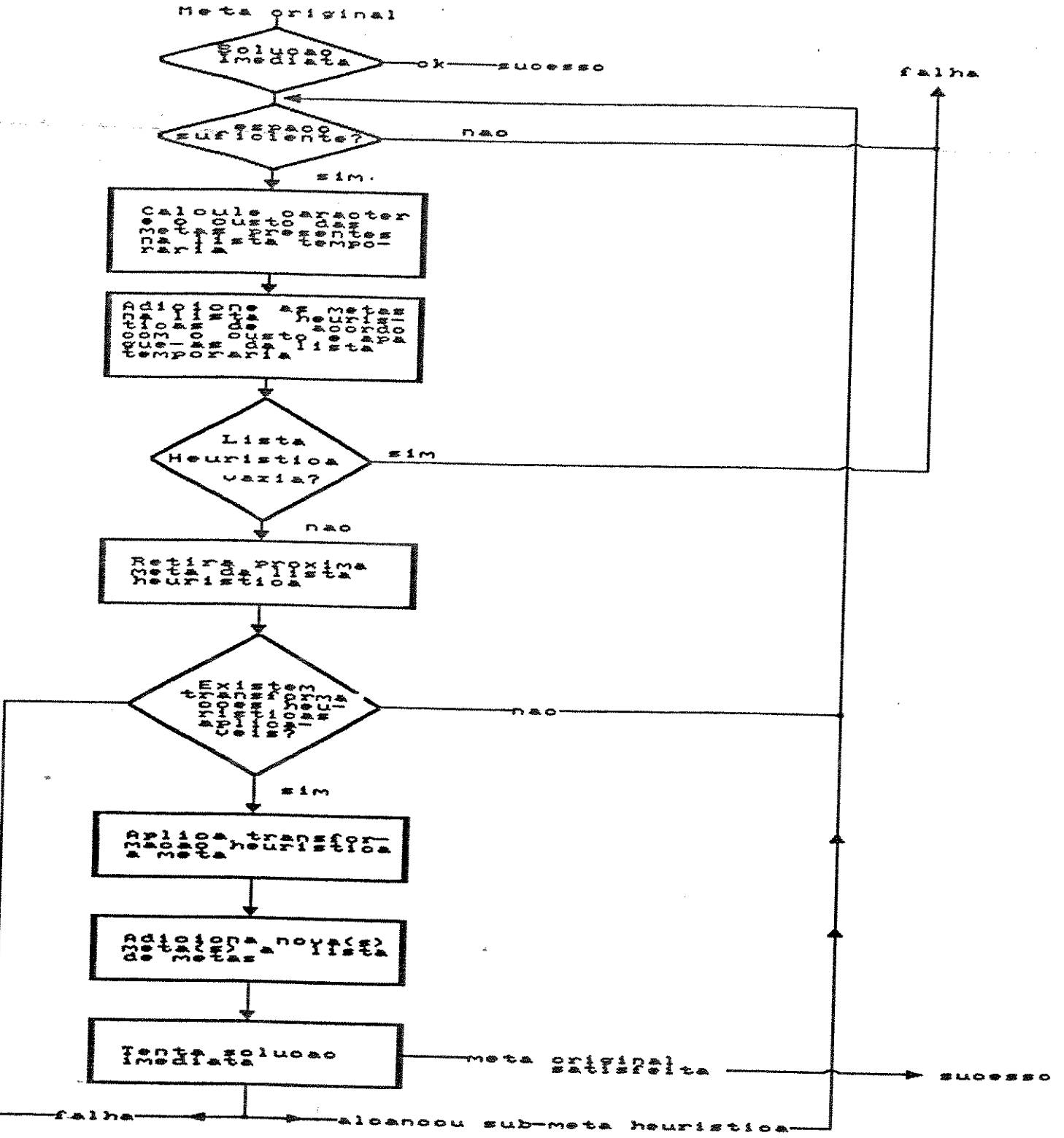
Foi utilizada uma versão simplificada do LISP de McCarthy com algumas funções e procedimentos específicos, especialmente desenvolvidos para manipulação simbólica.

Equipamento utilizado:

IBM 709
7090
32k memória.....

. 1/3 Sistema LISP
. 1/3 Programas específicos
. 1/3 SAINT
(sendo 3k de área de trabalho)

Organização executiva:



Definições:

Meta:

É o problema a ser solucionado. No presente caso é o integrando ou resultado de transformações no integrando.

Forma padrão:

Refere-se à tabela de integrais imediatas. Quando um integrando é desta forma, atinge-se a solução pela substituição simples e direta.

Eis algumas das formas padrões:

$$a- \int c.dv = c.v$$

$$b- \int e^v.dv = e^v$$

$$c- \int c^v.dv = c^v/\ln(c)$$

Transformações algorítmicas:

Nos casos em que não se aplica a forma padrão tenta-se aplicar as transformações algorítmicas. Estas, se aplicáveis, são quase sempre mais apropriadas.

Eis algumas transformações algorítmicas:

$$a- \int c.g(v).dv = c\int g(v).dv$$

$$b- \int -g(v).dv = -\int g(v).dv$$

$$c- \int \sum g_i(v).dv = \sum \int g_i(v).dv$$

Transformações heurísticas:

Quando uma meta não se enquadra na forma padrão ou uma transformação algorítmica não é aplicável, tenta-se como última alternativa a aplicação de transformações heurísticas, que são transformações ou substituições para se chegar à solução dos problemas , porém com possibilidade de falha.

Exemplo:

a) Substituição por uma sub-expressão cuja derivada divide o integrando

$$\int x \cdot e^{x^2} dx$$

tente $u = x$

b) Integração por partes

$$\int G \cdot h \, dv = G \cdot H - \int \frac{dG}{dv} H \, dv$$

Exemplo:

$$\int \frac{x \cdot e^x}{(1+x)^2} dx = -\frac{x \cdot e^x}{1+x} - \int -e^x dx$$

c) Outras transformações

C1 - Transformação de função elementar de função trigonométrica

$$\int \frac{dx}{\sec x} \longrightarrow \int \cos x\, dx$$

C2 - Substituição por uma função trigonométrica

$$\int \frac{\sec^2 t}{1+\sec^2 t - 3\operatorname{tg} t}\, dt \qquad \text{tente} \quad x = \operatorname{tg} t$$

Algorítmo de solução imediata:

Assim que uma meta é gerada, tenta-se métodos diretos para solucioná-la. Nesta tentativa pode-se:

- Adicionar a meta à lista temporária
- Adicionar sub-metas à lista temporária
- Se a meta é satisfeita, tenta-se satisfazer a meta original.

Árvore hierárquica:

Quando se aplicam as transformações heurísticas ou algorítmicas criam-se metas e sub-metas que possuem uma hierarquia representada por grafo ou árvore. As relações entre metas e sub-metas podem ser do tipo "AND" e "OR".

Ao aplicar-se o procedimento de solução imediata sobre uma meta e caso esta não esteja na forma padrão ou não se enquadre nas transformações algorítmicas, então adiciona-se a mesma à lista temporária.

Lista de metas:

É a lista de metas ordenada de acordo com seu custo , que no presente caso foi o comprimento do integrando.

Característica da meta:

Ao retirar uma meta da lista temporária, calcula-se seu 'caracter', isto é, uma lista de características úteis para estimar custos e selecionar transformações heurísticas apropriadas.

Algumas destas características são:

a) Profundidade -> nível máximo de parênteses (espelha a dificuldade).

b) Função trigonométrica -> TRUE ou FALSE (Se é ou não uma função trigonométrica)

Ao analisar um integrando e o mesmo for uma função trigonométrica, deve-se aplicar sobre ele os métodos heurísticos específicos para funções trigonométricas.

Exemplos

$$\int \frac{x^4}{(1-x^2)^{5/2}}\,dx$$

$x = \operatorname{sen} y$

$y = \operatorname{arc\,sen} x$

$dx = \cos y\,dy$

$$\int \frac{\operatorname{sen}^4 y}{\cos^9 y}\,dy$$

subst. trig.

$$\int \operatorname{tg}^4 y\,dy$$

$$\frac{z^4}{(1+z^2)^2}$$

$$\left(-1 + z^2 + \frac{1}{z^2+1}\right)dz$$

$$z \quad z^2 dz \quad \frac{z^3}{3}$$

subst. trig.

$$\int \cot g^{-4} y\,dy$$

$y = \operatorname{arc\,cotg} z$

$dy = \frac{1}{1+z^2}\,dz$

$$\frac{dz}{1+z^2}$$

$z = \operatorname{tg} w$

$dz = \sec^2 w\,dw$

dw

w

sucesso

$$\int \cot g^{-4} y\,dy$$

$z = \cot g\, y$

$y = \operatorname{arc\,cotg} z = -\frac{1}{1+z^2}\,dz$

$$-\frac{dz}{z^4(1+z^2)}$$

$$-\frac{dz}{z^4(1+z^2)}$$

subst. trig.

$$\int \operatorname{tg}^4 y\,dy$$

$\operatorname{tg} y = 2z/1-z^2$

$z = \operatorname{tg} y/2 \quad dy = \frac{2}{1+z^2}\,dz$

$$\int \frac{32\,z^4}{(1+z^2)(1-z^2)^4}\,dz$$

$$32 \int \frac{z^4}{(1+z^2)(1-z^2)}\,dz$$

Convenções

- TRANSF PADRÃO
- TRANSF. HEURÍSTICA
- TRANSF. ALGORÍTMICA
- EQUIVALENTES

$$-z + z^3/3 + w$$

$$-\operatorname{tg} \operatorname{arcsen}(x) + 1/3 \operatorname{tg}^3 \operatorname{arcsen}(x) + \operatorname{arc\,sen}(x)$$

1.4.2- SISTEMA Symbolic INtegration - SIN

Desenvolvido por Moses em 1967-67 através da linguagem LISP /Moses - 1971/. Foi implementado inicialmente para os sistemas IBM 7094, PDP-10 e IBM -360.

Estratégia Geral

Estágio 1 -> Tentativa por métodos simples(Ex: Versão do método deriva-divide)

Estágio 2 -> Tentativa pelos 11 métodos específicos (Trigonométricas, Exponenciais,Radicais, etc)

Estágio 3 -> Possui dois métodos gerais para os casos que não se enquadram nos estágios anteriores. (Ex: Integração por partes, Algorítmo de Risch).

Primeiro estágio

É o processo que engloba as integrais mais comuns. Neste estágio, verifica-se inicialmente se a integral é do tipo em que a derivada de sub-expressões do integrando divide o resto do integrando:

$$\int c \cdot op(u(x)) \cdot u'(x) \cdot dx$$

c ->constante
u(x) -> função elementar
u'(x) -> derivada
op -> operador elementar (sen, cos, tg, cosec, arcsen, arcsec, arctg, e funções exponenciais u(x), u(x)^ -1, u(x)^d e d^u(x) sendo d=constante.

Caso a integral seja de um destes tipos, procura-se na tabela através de "op" e substitue-se todo "x" da expressão encontrada na tabela por u(x).

exemplo:

$$\int x \cdot e^{x^2} \cdot dx = \frac{1}{2} \cdot e^{x^2}$$

sendo:

$op(u) = d^{u(x)}$ $\quad c = 1/2 \quad u(x) = x^2 \quad u'(x) = 2x$

Existem ainda duas regras de transformação:

. regra de soma de integrais

$$\int \sum A_i \, dx \quad \longleftrightarrow \quad \sum \int A_i \, dx$$

. desenvolvimento multinomial de soma elevada a uma potência inteira

$$\int (x + e^x)^2 \, dx \rightarrow \int x^2 \, dx + \int 2 . x . e^x \, dx + \int e^2 . x \, dx$$

resolvidas no 1º 2º 1º estágio

Após a aplicação das regras de transformação, aplica-se recursivamente o SIN.

Neste primeiro estágio foram resolvidos 45 dos 86 problemas propostos na tese de Slagle(Programa SAINT) com um tempo três vezes mais rápido. (Já levando em consideração o fato de o programa do Slagle ser interpretado).

Segundo Estágio

É aplicado para os casos que não se enquadram no primeiro estágio

Descrição:

. 11 métodos aplicáveis;

. Rotinas "Form" que determinam qual o método a ser aplicado (Se o método escolhido não for aplicável, tenta-se outra alternativa). Se por exemplo, a rotina "Form" encontra uma expressão e^x em um integrando, então o método adotado será o que lida com funções exponeciais;

. Tempo de processamento: 2 segundos em média;

. Linguagem Schatchen para reconhecimento de padrão. Nesta linguagem podemos por exemplo declarar um padrão para expressões quadráticas Ax^2 +Bx + C e obter para a expressão x^2+2x+3 os seguintes valores *A=1, B=2 e C = 3);

. Alguns métodos são de decisão, isto é, encontra-se a resposta se a mesma existir em termos de funções elementares, caso contrário falham.

A maioria dos métodos transforma o integrando em um problema mais simples e aplica-se recursivamente o SIN desde o estágio 1.

Terceiro Estágio

Possuia na implementação original os seguintes métodos:

I- Integração por partes

$$\int g \cdot h \cdot dx = g \cdot H - \int g' H \cdot dx$$

$$H(x) = h \cdot dx$$

Neste método faz-se uma pesquisa para determinar uma boa partição do integrando (técnica esta não muito bem definida)

II- EDucated GuEss -EDGE (Tentativa condicionada)

É um método heurístico, usado para fazer tentativas da solução da integral baseado na estrutura do integrando, com embasamento teórico na Teoria de integração de Liouville. O método é o seguinte:

A partir de uma tentativa, diferencia-se a mesma e seus coeficientes desconhecidos são obtidos unificando-se a derivada com o integrando. Desta forma é possível resolver uma grande classe de problemas.

A tentativa é baseada na procura de um termo singular no integrando. É singular o termo que não está contido em outros fatores e suas derivadas e nem pode ser obtido dos outros fatores ou suas derivadas por operações racionais.

EXEMPLOS

$f(x) = \int x\, e^x$

fonte singular: e^x

EDGE faz tentativa

$f(x)\,dx = \int x \cdot e^x \cdot dx = \int a(x)\, e^x + b(x)$

$a(x)$ não é função de e^x

derivando a tentativa

$$a(x)\, e^x + a'(x) \cdot e^x + b'(x) = x \cdot e^x$$

EDGE escolhe $a(x)$ unificando os termos:

$$a(x)\, e^x = x \cdot e^x \longrightarrow a(x) = \frac{x \cdot e^x}{e^x} = x$$

obtemos: $a'(x) = 1$

$\therefore \quad b'(x) = -e^x \iff b(x) = -e^x\, dx$

aplica-se EDGE novamente para $-e^x \cdot dx$

faz-se a tentativa $b = -e^x = a_1 \cdot e^x + b_1$

derivando $-e^x = a_1 e^x + a_1' \cdot e^x + b_1'$

$$a_1\, e^x = -e^x$$

$$a_1 = -1 \qquad a_1' = \emptyset$$

portanto $b_1 = -0\, e^x = \emptyset = \underline{cte}$

e. $b = -e^x \cdot dx = a_1 \cdot e^x + b_1 = -e^x + \underline{cte}$

$f(x)\,dx = a(x) \cdot e^x + b(x) = x \cdot e^x - e^x + \underline{cte}$

Os sistemas SIN e SAINT embora não obtiveram sucesso comercial, contribuiram de forma efetiva para o desenvolvimento de grandes sistemas, como por exemplo, o MACSYMA.

I.5 - SIMPLIFICAÇÃO ALGÉBRICA

É um dos suportes dos sistemas algébricos que exercem um papel fundamental no desempenho destes sistemas.

Inicialmente surge o problema da definição de "simples", que depende basicamente do contexto do problema. Os critérios possíveis seriam: o comprimento da expressão, significado no contexto, fatorar ou não, denominador comum, frequência de uso, etc. Não há um consenso neste sentido, pois o conceito de "simples" varia com a aplicação.

As simplificações mais comuns são aquelas de termos que contém 0 e 1 . Neste caso as transformações são do tipo:

$$A + \phi = A$$

$$A * \phi = A$$

$$A^{\phi} = 1$$

$$A^{1} = A$$

$$1^{A} = 1$$

$$A * 1 = A$$

Um dos trabalhos pioneiros na área de simplificação foi o do Projeto Dartmouth em 1953 / SAMMET - 1966/ que permitia em seu sistema a produção e o manuseio de expressões de ponto flutuante e de funções transcendentais (exceto exponencial). A sua forma canônica consistia de soma de produtos, onde cada produto continha no máximo três fatores: constante, variável elevada a uma potência e uma função exponencial contendo somente variáveis. A rotina de simplificação consistia em duas partes: a primeira transformava a fórmula em uma forma padrão, mas não realizava nenhuma simplificação de somas. A propriedade distributiva era aplicada sempre que havia um produto cujo argumento era uma soma. A segunda etapa era a simplificação de somas e a combinação de termos similares.

Um dos primeiros sistemas de simplificação escrito em LISP foi a Tese de Goldberg / SAMMET 1966/ onde primeiro simplificava-se cada argumento da expressão algébrica e posteriormente chamava-se programas especiais de simplificação para os seguintes operadores: PLUS, MINUS, TIMES e POWER. Termos similares eram combinados , transformações de expressões com 0 e 1 eram realizadas, funções especiais para eliminar parênteses na soma de produtos, termos de uma soma eram fatorados,etc.

Outro programa de simplificação em LISP foi o de Hart em 1961 / SAMMET - 1966/. Muitas funções eram executadas, mas os argumentos deviam estar ordenados canonicamente. As expressões deveriam ser colocadas na forma padrão. O procedimento de simplificação consistia, entre outras tarefas, em:

- Combinação de termos numéricos;
- Colocação de termos sob um operador de produto;
- Fatoração de números e sub-expressões comuns;
- Achar o denominador comum;
- Mudança de sinal;
- Simplificação de expressões com 0 e 1.

Konsvold desenvolveu em 1961 / SAMMET - 1966 /outro programa de simplificação em LISP. Os operadores disponíveis eram: PLUS, TIMES, MINUS, EXPT, LN ,etc. O usuário pode controlar o processo de simplificação através de respostas à consultas feitas pelo programa. A rotina de simplificação normalmente converte a expressão de entrada na forma canônica, executa a transformação de expressões com 0 e 1 e combina termos similares. A partir daí o usuário controla a expansão de produtos de somas, a ordem das variáveis, mudança de expressões da forma polinomial para a soma de produtos e vice-versa. O usuário pode definir novas regras de simplificação antes ou durante o processo.

O sistema desenvolvido por Wooldridge em 1963 /SAMMET 1966/ tem como características básicas a transformação de expressões com 0 e 1, combinação de termos similares e produtos similares bem como o cancelamento de inversas. Polinômios são manuseados separadamente para aproveitar a vantagem de sua forma especial. Termos podem ser fatorados especificando-se as variáveis. O usuário deve especificar exatamente onde ele deseja que a tarefa seja realizada.

Outro trabalho de simplificação foi o realizado em FORMAC pelo grupo de Tobey em 1965 / SAMMET - 1966/. A idéia era executar os comandos de simplificação mais comuns como combinação de termos similares, cancelar inversas, transformações de expressões com 0 e 1, etc. Havia a possibilidade de o usuário comandar expansões e fatorações. O esquema consistia inicialmente da aplicação de um conjunto de tranformações que realizasse as modificações das expressões com 0 e 1 e removesse os sinais negativos e parênteses. A partir daí o resultado era enviado a uma rotina que o coloca na forma canônica e durante o processo combina e cancela os termos e fatores. Expressões já processadas não são simplificadas novamente.

Os sistemas atuais como o Maple /MONARD e NAKANISH - 1984/, Macsyma /MOSES - 1984/ e Mumath/ muMATH 80 / possuem execelentes algorítmos de simplificação, além da possibilidade de se efetuar a simplificação manual através de flag manipulada pelo usuário.

I.6 - AMBIENTES DE PROGRAMAÇÃO

Os ambientes de programação /BOBROW, IAMSOM AND STEFIK - 1986D86O/ têm por finalidade básica o auxílio no desenvolvimento de sistemas computacionais. Este auxílio abrange as fases de digitação, teste, depuração, e documentação.

No caso específico de inteligência artificial, existem programas muito extensos que devem ser desenvolvidos e testados em diversos módulos separadamente. Outra característica é a modificação frequente do programa fonte, o que requer uma ferramenta apropriada para facilitar o trabalho de depuração do programador.

As principais características de um ambiente de programação são:

- Linguagem interativa;
- Bom editor;
- Facilidade de depuração;
- Boas rotinas de E/S;
- Ferramenta de análise e refinamento do desempenho do programa;
- Integração de ambientes diferentes.

O estilo de programação adotado no desenvolvimento de sistemas inteligentes depende de vários fatores entre os quais podemos ressaltar :

- aprendizado;
- depuração;
- mudança;
- execução;
- integração com outros paradigmas.

Com o crescimento do campo da Inteligência Artificial foram criadas ferramentas para desenvolvimento que deram origem a linguagens cujas características são voltadas para manipulação dos problemas específicos da área.

As primeiras idéias referentes às linguagens de Inteligência Artificial foram em manipulação de símbolos e processamento de listas. Sua primeira implementação foi a linguagem IPL (NEWELL,SHAW E SIMON - 1957) na área de resolução de problemas. Os elementos da linguagem eram símbolos (que podem representar qualquer coisa) e sua forma de associação são listas que permitem a construção de estruturas de dados sem restrição de forma ou tamanho.

A partir daí surgiram LISP e PROLOG que constituem as principais linguagens na área da Inteligência Artificial.

O desenvolvimento eficiente de sistemas nestas linguagens requer, além do conhecimento específico da mesma, a formalização do conhecimento, a representação do conhecimento e a manipulação da máquina de inferência.

I.6.1 - ESTILOS DE PROGRAMAÇÃO

Atualmente estão sendo usados vários estilos de programação em inteligência artificial. Analisaremos aqui as principais características de cada um deles.

Programação procedimental	PASCAL FORTRAN ADA, etc.
Programação orientada para objetos	SMALLTALK
Programação funcional	LISP
Programação lógica	PROLOG
Programação orientada para aplicações	M.1

Descreveremos agora com mais detalhes cada estilo de programação:

I.6.1.1- Programação procedimental

As linguagens imperativas (Fortran, Pascal, Basic, etc) são voltadas para a máquina, uma vez que especificamos no programa o fluxo de controle para solucionar o problema. São linguagens baseadas no modelo computacional do matemático John Von Newmann (1903-1957), criado na década de 40.

Estas linguagens tenderão a ser pouco utilizadas no futuro da Inteligência Artificial / TRELEAVEN E LIMA - 1985/ uma vez que apresentam dificuldades em relação à manipulação de símbolos e gerenciamento de memória. Já as linguagens concorrentes possuem aplicação nas áreas de resolução de problemas e busca heurística.

I.6.1.2-Programação orientada para objeto

É um ambiente de programação interativo desenvolvido pela XEROX- PALO ALTO /GOLDBERG - 1983/e é baseado em objetos ativos que se comunicam através do envio de mensagens.

Todo objeto pertence a uma determinada "classe" e é um exemplo típico desta classe. A classe define a representação detalhada de seu exemplo típico, as mensagens que podem ser respondidas e os métodos para cálculo das respostas adequadas.

Ex: SMALLTALK

No caso do Smalltalk, os principais elementos constituintes da linguagem são:

Objetos -> Componentes representado por:

- Memória particular
- Conjunto de operações

Exemplo de objetos: número, retângulos, programas, etc.

Mensagem -> Requisições para um objeto executar uma de suas operações. Especifica qual a operacao desejada e não como será realizada.

Exemplo: Adição

Envia a mensagem para um objeto que representa um número especificando a operação (adição) e os números a serem adicionados.

Receptor -> Objeto para o qual a mensagem é enviada e que determina como será realizada a operação.

Método -> Sequência de ações a serem executadas para realizar uma operação.

Protocolo -> Conjunto de mensagens que um objeto pode responder. A única maneira de interagir com um objeto é através do Protocolo.

Classe -> Uma descrição de um ou mais objetos que representam o mesmo tipo de componentes do sistema.

Exemplo -> Um objeto descrito por uma classe particular é denominado exemplo típico.

Um programa em Smalltalk consiste em criar classes, exemplos típicos de uma classe e especificar sequências de trocas de mensagens entre estes objetos.

1.6.1.3-Programação orientada para Lógica

A linguagem orientada para a lógica é uma variação do cálculo predicativo de primeira ordem (é possível expressar quantificadores sobre objetos específicos, mas não sobre predicados ou funções). A lógica de primeira ordem é consistente (não é possível provar uma afirmação falsa) e completa (toda afirmação verdadeira tem uma prova). Para expressar os problemas em lógica de primeira ordem devemos colocá-los na forma de axiomas.

Algumas das principais características da lógica são: naturalidade (é uma maneira natural de expressar conhecimento -McCarthy,Filman-),precisão (há métodos padronizados para determinar o significado das expressões) e flexibilidade (não faz distinção do tipo de processo que fará deduções).

PROLOG (PROgramação em LóGica)

É uma linguagem bastante utilizada no campo da Inteligência /CLOCKSIN - 1984/ Artificial pelas comunidades Européias e Japonesa, sendo implementada em 1972 por Colmerauer e Phillipe Roussel baseada em idéias de resolução desenvolvidas por Kowalski e Donald Kuehner. A consolidação da linguagem veio com a implementação de David Warren em 1975 na Universidade de Edinburgh. O compilador surgiu em 1977 em Edinburgh e Lisboa por David Warren, F. Pereira desenvolvida em MACRO-10 e Prolog.

É baseada num sub-conjunto do cálculo predicativo de primeira ordem, usada para representar relações lógicas, axiomas entre objetos e seus atributos na forma de cláusulas de Horn. Além dos axiomas, temos os teoremas a serem provados e a instanciação de variáveis (se possível) para tornar o teorema verdadeiro.

A programação em lógica soluciona problemas através da lógica, como formalismo computacional e é chamado de programação declarativa.

Um programa em lógica de primeira ordem é um conjunto finito de sentenças, isto é, fórmulas fechadas representando os axiomas de uma teoria. O programador é o próprio axiomatizador.

Programação em lógica significa:

- Verificar se uma sentença é consequência lógica de outras;
- Obter respostas e informações do programa;
- Permite construir respostas a partir de deduções.

O trabalho pioneiro na programação lógica foi o de Robinson e o da prova automática de teoremas em que se criou sistemas para decidir parcialmente se uma sentença é consequência lógica de outras, de forma eficiente. Posteriormente surgiram outros trabalhos como o de Kowalski e a linguagem PROLOG.

Características da linguagem Prolog:

- Boa utilização no processamento simbólico;
- Representa uma implementação da lógica como linguagem computacional;
- Apresenta uma semântica declarativa inerente à lógica em adição à semântica operacional usual das linguagens de programação tradicional;
- As entradas e saídas não precisam ser definidas a priori;
- Suporta mecanismos recursivos (Dispensa GOTO,etc);
- Permite escrever especificação e codificação numa mesma linguagem;
- Representa dados e programas no mesmo formalismo;
- Procedimentos podem ter várias entradas e saidas;
- Procedimentos podem gerar através do retrocesso ("backtracking") uma sequência de resultados alternativos;
- Pattern matching substitui o selector e construtor (da LISP) para operar em estruturas de dados;
- Estruturas de dados incompletas podem ocorrer (isto é, contendo variáveis livres) e podem ser, mais tarde, preenchidas por outros procedimentos.

SINTAXE

Alfabeto

- pontuação (,),.,'
- conectivos &
- letras do alfabeto A,... Z,a ..... z
- símbolos especiais +,*,...
- dígitos 0....9

Constante-> Cadeia de letras ou dígitos iniciando por uma letra minúscula, símbolos especiais ou numéricos.

Variável-> . Cadeia de letras e dígitos, iniciando com uma letra maiúscula.

. Símbolos "_" indicando variável anônima

Termos-> É o menor conjunto satisfazendo:

. Toda variável é um termo

. Toda constante é um termo

. Se t1,.....tn são termos e f é uma cadeia de letras e dígitos, iniciada com letras minúsculas então:

f(t1,...tn) é um termo e f é um símbolo funcional n-ário

Fórmula atômica(F.A.)-> É uma cadeia da forma:

p(t1,....tn)

onde:

p: símbolo predicativo n-ário

ti: termos

Cláusulas-> É o menor conjunto satisfazendo as condições:

. Se A é F.A. então A é uma cláusula (unitária);

. Se B1,...Bn e A são F.A. então A <- B1 & B2 ;

. Se C1,...Cn são fórmulas atômicas então <- C1& Cn é cláusula (objetiva).

CLÁUSULA DEFINIDA-> É uma cláusula unitária ou não.

Cláusula unitária-> Representa um fato verdade relativo a um determinado domínio do problema.

Ex:

p(t1,t2,...tn).

onde :

p é símbolo predicativo e ti são constantes

Cláusula não unitária-> Expressa implicação entre relações existentes no domínio do problema. É um axioma em prolog(sentença).

P <- Q & Q1 & ... Qn

aplicam a constantes e/ou variáveis.

PROGRAMA PROLOG-> É uma sequência de cláusulas unitárias e não unitárias .

Cláusulas objetivas-> É uma consulta atômica ou conjuntiva e permite:
- certificar-se a respeito de um relacionamento entre objetos do domínio do problema;
- recuperar dedutivamente informação armazenada.

Exemplo:
:- depende(x,e)
existe um x que depende de e.

Consulta em prolog -> É uma cláusula objetiva prolog.
?- programa(X,prolog).
obteremos a seguinte resposta:
X = a
yes

para o caso em que estiver presente na base de dados a seguinte cláusula:
programa(a,prolog).

Resposta-> É **sucesso** quando a cláusula objetiva pode ser provada ou **falha**, em caso contrário. No caso de sucesso são mostradas as instâncias das variáveis livres (se houver).

Sintaxe dos operadores em prolog

prefixa *(a,b)
sufixo (ab)*
infixo a*b
Declaração dos operadores
Ex:
op(nome,tipo,prioridade)
sendo:
nome: símbolo especial ou cadeia de caracteres ou dígito iniciando por caracter minúsculo
tipo: prefixo, sufixo ou infixo
prioridade: ordem de avaliação (menor número tem maior prioridade)
op(+,xfy,200)

SEMÂNTICA

As visões semânticas de um programa PROLOG são úteis na sua elaboração e entendimento. Analisaremos cada uma delas.

Visão declarativa: Esta interpretação independe da forma de execução do programa, permitindo ao programador definir "o que" e não "como". Permite ao programador visualisar o domínio do discurso através de assertivas e implicações existentes entre os objetos do domínio de discurso.

Visão procedimental: Esta interpretação descreve o método pelo qual a meta é executada para produzir instâncias verdadeiras da meta. A ordem das cláusulas aqui é importante.

LISTAS

Conjunto de termos denotando listas (notação Edinburgh). É um átomo na forma .(H,T) onde H é a cabeça e T a cauda de uma lista.

(1) [] Lista vazia
(2) [A|B] onde A é um termo e B é uma lista
(cabeça) (cauda)

[A | [B | C |[]]]
[A,B,C]

Exemplos:

[A,[B,C]] cabeça A
cauda [[B,C]]
[[A],B,C] cabeça [A]
cauda [B,C]

Exemplo de operação sobre listas:
Adotamos as seguintes convenções:
c1-> condição terminal
c2-> condição de recursão

Operação: Lista
Definição: Verifica se um termo dado é uma lista
1o argumento : termo dado
2o argumento : Lista dada
3o argumento : Lista resultante

c1: lista([]).
c2: lista(_|L):- lista(L).

. A cláusula c1 indica que [] é uma lista
. A cláusula c2 indica que o termo dado será uma lista se for possível decompor o mesmo em duas partes tais que a segunda parte seja uma lista.

Algumas operações básicas:
identificar lista vazia;
obter a cabeça de uma lista;
obter a cauda de uma lista.
Com o auxílio da unificação de listas podemos facilmente executar estas operações básicas.
[a]
[H|T]
unificação
H/a
T/[]

ORDENAÇÃO EM PROLOG

A ordem dos axiomas é significativa. O processo de execução tenta cada sub-objetivo da esquerda para direita dentro de um axioma de um objetivo, e em caso de falha tenta os axiomas alternativos obedecendo a uma ordem de cima para baixo. Dentro do processamento (recursivo) dos sub-objetivos segue a mesma ordem. Esta estratégia adotada pelo interpretador Prolog é denominada busca em profundidade ("depth first search").

A ordem de execução está sujeita a comandos extra-lógicos.

RETROCESSO

O retrocesso("backtracking") é uma forma de avaliar outras alternativas de uma solução. Nele o resultado obtido de uma alternativa é desfeito para se tentar outra, e é aplicado tanto no caso de insucesso com o de sucesso, o que permite satisfazer um mesmo objetivo de várias modos. Isto é útil, e tem sido comparado a uma forma de alto nível de interação, pois atingindo o objetivo de vários modos obtem-se de várias respostas.

RESOLUÇÃO NA LINGUAGEM PROLOG -> É a regra de inferência baseada na Resolução Linear onde são permitidas apenas axiomas expressáveis como cláusulas de Horn (admite no máximo um literal positivo para que tenhamos uma única representação e assim facilitar a implementação em sistemas computacionais). Este tipo de resolução é um refinamento dos princípios de resolução introduzidos por Robinson.

LINGUAGEM PROLOG EXPANDIDA

Possui comandos extra-lógicos e meta-lógicos para simular programação mais próxima das tradicionais

Extra-lógicos de comunicação -> Não são afetados pelo procedimento de refutação do PROLOG , portanto não têm retrocesso. São avaliados separadamente.

Alguns comandos de entrada e saída:
nl (nova linha);
tab (tabulação);
open(abrir arquivo);
create (criar arquivo);
read (ler arquivo);
write (escrever arquivo);
get (ler caracter de um arquivo);
halt (abandonar interpretador);
put (escrever caracter em um arquivo);
save (salvar base de dados);
restore (recuperar base de dados);
consult (carregar arquivo na base de dados);
reconsult (recarregar arquivo na base de dados);

Extra-lógicos de controle

Corte e falha permitem controlar a execução do programa pelo controle do procedimento de retrocesso.

! - corte
fail - falha
[! !] - snip

cut(!)

É um predicado com um único argumento cujo valor é sempre verdade. No caso do retrocesso, ele impede uma nova avaliação da cláusula que contém o corte, bem como de qualquer outra sentença com o mesmo predicado. É equivalente ao "go to" e altera o controle do programa. Se usado indistintamente altera a estrutura lógica do programa.

```
Início:- A,B,C.
B:- D,E,!,F.
```

fail

É um predicado de um único argumento cujo valor é sempre falso, forçando o retrocesso em busca de outras alternativas.

```
lê:- repeat,get(P),write(P),fail.
```

cut e fail

Provoca o insucesso da cláusula que o contém.

```
not(P):- P, !, fail.
not(P).
```

snip

Semelhante ao cut, exceto no retrocesso, não provoca falha do predicado que o contém, e somente das metas entre as chaves.

```
A:- B, [!C,D!],E.
```

Comandos extra-lógicos de processamento de cláusulas:

São comandos para manipular cláusulas da base de dados.

Exemplo:

assertz(Adiciona cláusula ao final do conjunto de cláusula de mesmo predicado)

asserta(Adiciona cláusula no inicio do conjunto de cláusula de mesmo predicado)

retract(Remove cláusula da base de dados)

abolish(Remove todas as cláusulas do tipo especificado)

listing(Lista toda a base de dados)

Comandos extra-lógicos de depuração

Permite controlar a execução do programa para fins de depuração.

Exemplos:
spy(sinaliza o início da depuração)
nospy(desmarca sinal de depuração)
trace(Aciona o modo de depuração)
notrace (Desliga o modo trace)

META-VARIÁVEL

Muito útil na definição de programas. Permite substituir uma meta variável por uma conjunção de fórmulas atômicas.

Exemplo:
```
executa(X) :- X.
```

NEGAÇÃO POR FALHA

Implementação da regra de negação por falha finita. Para implementação do mesmo foram utilizados os conceitos extra-lógicos corte, falha e meta-variável.

```
não(P) :- P, !, fail.
não(P).
```

Problemas da negação por falha:
- possibilidade de divergência na primeira cláusula , isto é, não achar o valor de não(P).
- É sensível a instanciação das variáveis e depende incorretamente da ordem das fórmulas atômicas no corpo das cláusulas.

Ex: (Kowalski - 1973)
```
gosta(maria,joão).
deseja(maria,bicicleta).

?- não(deseja(X,Y)),gosta(maria,X).   FALHA
?- gosta(maria,X), não(deseja(X,Y)). SUCESSO
```

A negação por falha torna-se correta para os casos em que não(P) é equivalente a:

não(P) = ∀ x1,x2,....∀xn(˜P)
˜(∃x1,...∃xn(P))

onde x1..xn são variáveis livres de P

O seu comportamento errado é:

não(P) = ∃x1,..∃xn(˜P) caso alguma variável x1,...xn for instanciada durante a avaliação de P

Em suma, o procedimento será correto se a avaliação de não(P) for realizada após a instanciação de todas as variáveis de P.

LIMITAÇÕES DA LINGUAGEM:
- Diferentes dialetos;
- Existem poucos ambientes de programação;
- Limitações do poder de expressão da cláusula de Horn;
- Trata a negação apenas como "negação por falha"

I.6.1.4-Programação funcional

O programa é uma função onde as entradas são os argumentos que podem ser funções e o resultado é a saída do programa. Seus componentes se resumem a definições de funções.

A principal característica da linguagem é a aplicação de funções a estruturas que são expressões matemáticas.

LISP

Linguagem desenvolvida por John McCarthy em 1959 e implementada inicialmente em um equipamento IBM 704 /FEIGENBAUM,ARON - 1981/PLESKUN AND THAZUTAUEETIL - 1987/.

É a linguagem pioneira e atualmente predominante no campo de Inteligência Artificial e tem como características básicas o processamento de listas, recursão e técnicas que permitem incrementar a linguagem criando novas funções.

O primeiro LISP desenvolvido chamado Lisp puro ou Lisp1 era de difícil programação e foi evoluindo através do Lisp 1.5, MacLisp, InterLisp, Scheme, Franz Lisp, T, etc. Surgiram também os Lisp-padrão como o Standard Lisp e o Commom Lisp.

Atualmente vários sistemas algébricos e sistemas especialistas estão desenvolvidos em Lisp dada a sua extensibilidade e flexibilidade. Podemos através de algumas funções primitivas e alguns tipos de dados construir grandes sistemas.

O programa em Lisp consiste em definições de funções que invocam outras funções. A execução de um programa em Lisp é o cálculo de um conjunto de funções encadeadas.

Ex:

$$\text{fatorial} \quad \begin{cases} N! = 1, & N = 1 \\ N(N-1)! & N > 1 \end{cases}$$

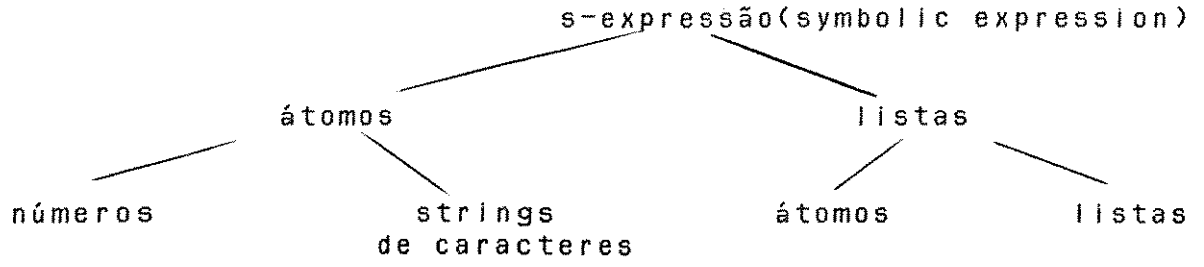
```
fatorial(N)
        (COND    ((EQUAL   N 1)
                 (fatorial(TIMES (DIFFERENCE N 1) N))
         )
```

A avaliação desta função em LISP é feita por substituições sucessivas baseadas no método denominado Cálculo Lambda /Church -1930/ .

Temos aqui um esboço do trace realizado para o cálculo fatorial.

```
fatorial(2)=TIMES(2(fatorial(difference(2 1))))
            TIMES(2(fatorial 1))
            TIMES(2(times(1(fatorial(difference(1
1))))))
            TIMES(2(times(1(fatorial 0))))
            TIMES(2(times(1 1)))
            TIMES(2)
           = 2
```

A estrutura de dados em LISP é a lista. Os dados em LISP são denominados s-expressões.



a- Átomos

Átomo é a menor forma de representação na estrutura de um dado. Átomos são números e nomes. Nomes são strings de caracteres.

Alguns exemplos de átomos são:

Eu Gosto

b- Listas

Lista é um conjunto de átomos e outras listas.

Sua notação é formada por uma sequência de zero ou mais s-expressões (separadas por espaço) entre parênteses. Uma lista vazia é também um átomo e é representado por () ou NIL.

Ex:

```
(José gosta (de Maria))
(1 2 3)
(SQRT (BxB))
```

A representação interna de uma lista é feita através de listas encadeadas de "células". Uma célula, é composta de dois ponteiros: o ponteiro CAR aponta para o conteudo da própria lista (átomo ou lista) e o ponteiro CDR que é a ligação com a próxima célula na lista. O átomo NIL termina a lista e é equivalente à lista vazia.

Há algumas funções primitivas para a manipulação de listas disponíveis na linguagem.

Construtor

CONS(x y) -> Retorna uma lista cuja cabeça é x e a cauda é y.

Ex:

CONS(a b) = (a b)

Seletores

CAR(x) -> Retorna a cabeça da lista

Ex:

CAR(a b) = a

CAR((a b) c) = (a b)

CDR(x)-> Retorna a cauda da lista

Ex:

CDR(a b) = (b)

CDR((a b) c) = (c)

CDR(a b (c d)) = ((c d))

E ainda algumas funções especiais denominadas predicados que dão como resposta somente o "T" ou "NIL". Eis alguns deles:

NUMBERP(N) -> T se N é número
NIL caso contrário
EX:
numberp(1) = T

EQUAL(X Y) -> T se átomos x e y são iguais
EX:
equal(a a) = T

ATOM(X) -> T se X é átomo
EX:
atom(a) = T

Os sistemas Lisp não são executados eficientemente nos sistemas mais comuns, dadas as diferenças semânticas entre a lingugem de manipulação de lista e a máquina de von Neuman. As principais áreas com problemas são:

- Chamada pouco eficiente de funções ;
- Manutenção do ambiente, manutenção das variáveis utilizadas nas funções;
- Representação e acesso a listas, pouco eficiente;
- Manutenção do pilha (identificação, manutenção e realocação de memória).

Surgiram então as Máquinas Lisp, cuja arquitetura é voltada para a solução destes problemas de maneira eficiente. As primeiras ídéias surgiram com as Máquinas Lisp da Xerox PARC e a máquina Lisp do MIT. A partir daí surgiram muitas máquinas, sendo algumas delas para o uso comercial.

As Máquinas Lisp foram divididas em três classes /PLESZKUN e THAZHVTHAVEETIL - 1987/:

Classe M -> Com processadores Lisp não especializados com primitivas em micro-códigos)

Ex: PARC Lisp Machine (XEROX)

Lisp Machine (MIT)

Classe S -> Máquinas com multiprocessadores, sendo cada um responsável por uma determinada função.

Ex: FAM-1 (Fairchild)
Lisp Machine (Keio University)

Classe P -> Sistema de multiprocessadores com cálculos concorrentes de diversas partes do prgrama com objetivo de obter um alto desempenho.

Ex: AHR Lisp Machine (Guzman)
EM-3
Eulis (University Osaka)

Estas máquinas propõem diversas alternativas para lidar com os problemas citados.

Para otimizar o uso da memória existe o "Garbage Collector" que é uma função automática que recupera as memórias não utilizadas pelo sistema. Esta alocação dinâmica de memória é imprescindível para estruturas tipo lista (que crescem imprevisivelmente). O tempo gasto pelo "Garbage Collector" é o preço pago pela flexibilidade da linguagem.

A linguagem é geralmente interpretada dadas as facilidades de interação durante o processo de desenvolvimento. Existem também compiladores disponíveis que aumentam consideravelmente a velocidade de execução dos programas.

I.6.1.5- PROGRAMAÇÃO ORIENTADA PARA APLICAÇÕES

São linguagens utilizadas na implementação de bases de conhecimento e podem ser divididos em:

- Sistemas esqueletos (Shells)
- Linguagens orientada para regras
- Linguagens baseadas em vários formalismos

Shells

Os Shells são sistemas peritos sem conhecimentos que já possuem uma estratégia de controle específico. No processo de desenvolvimento de sistemas peritos, o projetista deve representar um conhecimento específico através de regras. São aplicações limitadas a um certo contexto, dado que se referem a um domínio específico.

Ex: Emycin (shell)

As principais características dos shells são:

- encadeamento
  - . para frente(forward);
  - . para trás(backward).

- explicações de raciocínio
  - . durante a consulta;
  - . após a consulta.

- texto
  - . editor próprio de acordo com a estrutura da linguagem ;
  - . "display" automático e opcional.

- análise do desempenho do programa através de dados estatísticos.
- programas compilados ou interpretados.
- linguagens especializadas que facilitam o desenvolvimento.
- facilidade de depuração
  . durante a consulta;
  . após a consulta;
- interfaces especializadas.
- memória requerida
  . 128k a 512k (para versão de micros).
- preço
  . U$95 a U$80.000 (1985).

Linguagem orientada para regras

Possuem uma estratégia de controle mais geral aliado às operações de memória. Possuem ferramentas para explicar o comportamento do programa, responder questões e inserir novas regras. O mecanismo de inferência próprio é utilizado para solução dos problemas.

Sistema M.1.

Inicialmente o projetista desenvolve sua base de conhecimento em uma linguagem semelhante ao PROLOG e posteriormente depura-o no M1/WILLIANSON - 1986/. Durante este processo de depuração é possível manipular as regras e também dispor de "displays" para acompanhamento do processo.

São algumas características do sistema:
- retornar uma ou várias soluções
- aceitar perguntas durante a consulta
- calcular fatores para suas conlusões
- exibir texto durante a consulta

Linguagem de desenvolvimento: Prolog1
Sistema requerido: IBM-PC com 128k
Preço: U$ 10.000 (1986)

Leonardo

Desenvolvido pela Creative Logic (British) é um shell para sistemas especialistas baseado em regras para microcomputadores que proporciona um ambiente favorável para captar conhecimentos e questioná-los. Dispõe de múltiplos mecanismos de inferência (regras de encadeamento "forward", "backward").

Permite lidar com incerteza de varios modos:
- fatores de certeza;
- inferência bayesiana;
- fuzzy sets;
- acumuladores de evidência.

Outras características são:
- linguagem própria;
- acesso a arquivos e base de dados;

- dicionário;
- interrogações;
- "logging";
- trace;
- intercâmbio com as linguagens Pascal e C;
- preço: L 149.000 a L 1950.000 (1987).

KEE - Knowledge Engineering Environment software

É um sistema especialista que utiliza as técnicas de Inteligência Artificial para representar o raciocínio e fornecer explicação acerca do conhecimento. Pode interagir com o SQL relacional da IBM o que permite a interação com sistemas convencionais. Tem interface gráfica e utiliza "mouse" e menus para facilitar trabalho do usuário.

Cedar

É um ambiente para desenvolvimento e testes experimentais de sistemas computacionais. Foi desenvolvido pela XEROX e tem como descendentes os ambientes Smalltalk e Interlisp /SWINEHART e ZELLWEGER - 1986/.

Suas principais características são:

- Aumento da produtividade de programador através de diversas ferramentas como editores,compiladores e depuradores simbólicos bem como de gerenciadores automáticos de memória e operações concorrentes
- Integração de "software"
- "Software" de alta qualidade com o auxílio de processadores rápidos, programação concorrente, recursos compartilhados e interfaces especializadas.

Linguagem baseada em vários formalismos

Permite a integração de diferentes estilos de programação.

Ex: LOOPS (Permite a integração de programação procedimental, programação orientada para objetos e programação orientada para regras)

I.6.1.6- PROLOG X LISP

Tem-se discutido muito em torno da linguagem mais eficiente em relação à inteligência artificial, mas dificilmente chegaremos a um consenso. Podemos no entanto analisar cada uma das linguagens sob vários aspectos /WARREN, PEREIRA AND PEREIRA - 1985/WILIANSON - 1986/.

Enquanto LISP é uma linguagem pura na qual o programador constroi a máquina de inferência, PROLOG já possui seu próprio mecanismo de inferência.

LISP é baseado no formalismo matemático Lambda Calculus(Church) enquanto que PROLOG é um subconjunto do Cálculo Predicativo de primeira ordem. Ambas são apropriadas para o processamento simbólico.

O mecanismo de "pattern matching" do PROLOG é superior aos "selectors" e "construtores" do LISP.

Uma das grandes vantagens do PROLOG é sua utilização no desenvolvimento de protótipos, dada sua facilidade de desenvolver, testar e depurar. Outra vantagem é a definição de predicados através de cláusulas separadas ao invés de definições monolíticas de funções e procedimentos, encontradas no LISP. Este procedimento assemelha-se às regras de produção utilizadas em Inteligência Artificial.

Uma vantagem do LISP é o fato de que a ordem das informações é irrelevante, enquanto que no PROLOG é crítica.

Alguns fatores de desempenho:

Sintaxe: Lisp possui uma sintaxe inferior ao Prolog no que se refere à clareza do programa.

Tempo: Geralmente os programas em Prolog consomem menos tempo.

Memória: Programas em Prolog consomem menos memória, salvo os casos em que há excessivos retrocessos (backtrackings). Nestes casos utiliza-se do "CUT" para evitar buscas desnecessárias.

Recursos Gráficos e Manipulação de Tela: A maioria destes recursos já são inerentes ao LISP, portanto é uma opção superior. Em PROLOG poderiam ser desenvolvidas estas capacidades pelo usuário em uma linguagem de "baixo nível",em detrimento da viabilidade e da portabilidade do sistema.

Existem hoje máquinas simbólicas que permitem a integração de módulos de programas em diversas linguagens (inclusive Lisp ou Prolog).

I.7- SISTEMAS INTEGRADOS

I.7.1- MACSYMA

Projeto desenvolvido em 1968 por: C. Engelman
W. Martin
J. Moses

Sua implementação foi realizada em Julho de 1969 /MOSES - 1974/.

É um sistema de manipulação algébrica eficiente que engloba vários algorítmos específicos e otimizados.

Possui quatro representações de expressão visando alcançar uma maior eficiência em termos de tempo e espaço. São elas: Geral, Racional, Série de potências e Série de Poisson.

A representação geral é a "default" sendo a mais flexível e muito utilizada em interações com o usuário.

A racional é muito eficiente e é utilizada em vários algorítmos, como por exemplo no cálculo do máximo divisor comum.

A série de potências tem sua utilização na expansão de Taylor .

A série de Poisson é usada na manipulação de expressões polinomiais ou trigonométricas.

O custo de representações múltiplas para manipulação de problemas complexos não torna o sistema inviável dado o baixo custo de memória e o crescimento da velocidade das mesmas verificado ao longo dos anos.

O sistema é constituído de pacotes, sendo os principais:

- Linguagem e facilidades de interação;
- Representação geral;
- Representação de função relativa e algorítmos relacionados;
- Sub-sistema de integração;
- Sub-sistema de série de potências;
- Sistema MacLisp (Linguagem);
- Sub-sistema de tranformada de Laplace;
- Sub-sistema para solução de equações integrais;
- Sub-sistema para solução de equação diferencial ordinária;
- Sub-sistema de álgebra vetorial simbólica e cálculo vetorial.

O sistema já atingiu um bom grau de estabilidade principalmente devido ao "feedback" dos usuários.Os principais problemas eram oriundos das interações entre os pacotes.

A seguir descreveremos algumas características importantes do sistema:

- Existe um "parser" de entrada e uma forma adequada de saída de dados;
- A linguagem de programação é um interpretador LISP modificado para manipulação algébrica. Podemos interromper o programa ou fazer uma busca em funções definidas pelo usuário. Existe a opção também de compilar certas funções para ganhar velocidade em detrimento da interação;
- Existe um módulo que controla todo processo de entrada/saída. Podemos armazenar ou recuperar módulos ou informações intermediárias obter resultados em dispositivos quaisquer de armazenamento;
- O sistema possui também facilidades para controle estatístico e depuração;
- Outra facilidade é um editor específico para manipulação de expressões simbólicas;
- Existem alguns recursos gráficos para visualização de funções.

O **Macsyma** possui vários algorítmos para manipulação de polinômios e funções racionais, sendo que os usuários podem fazer a opção do algorítmo desejado.Um destes algorítmos é o RADCAN que têm como função básica a simplificação de expressões algébricas, sendo muito utilizado no algorítmo de integração de Risch.

Sub-sistema de integração:

- integração de funções racionais
- progama de integração SIN
- algorítmo de integração de Risch
- programa Limite

Este é um sistema bastante complexo e tem como base vários algorítmos (Por ex: RISCH), linguagens especializadas para reconhecimento de padrões (SCHATCHEN) e sistemas específicos para o cálculo de determinadas funções.

Exemplo de utilização do Macsyma

obs: "*" refere-se ao comando do usuário

Integração, diferenciação e simplificação

* integrate($\frac{1}{x^3+2}$, x);

Resposta :

$$-\frac{\log(x^2-2^{1/3}x+2^{2/3})}{6(2^{2/3})}+\frac{\arctan((2x-2^{1/3})/2^{1/3}\sqrt{3})}{2^{2/3}\sqrt{3}}+\frac{\log(x+2^{1/3})}{3(2^{2/3})}$$

* dif($\frac{1}{x^3+2}$, x);

Resposta :

$$\frac{1}{3\left(\left(\frac{2x-2^{1/3}}{3.2^{2/3}}\right)^2+1\right)}-\frac{2x-2^{1/3}}{6.2^{2/3}(x^2-2^{1/3}+2^{2/3})}+\frac{1}{3.2^{2/3}(x+2^{1/3})}$$

* rat_simp(%);

Resposta : $\frac{1}{x^3+2}$

Solução de equação polinômial

Resultados

* solve ($x^6 - 1$);

$x = \left(\frac{\sqrt{3}i + 1}{2}\right)$

$x = \left(\frac{\sqrt{3}i - 1}{2}\right)$

$x = -1$

$x = -\left(\frac{\sqrt{3}i + 1}{2}\right)$

$x = -\left(\frac{\sqrt{3}i - 1}{2}\right)$

$x = 1$

Tentaremos substituir a primeira solução para testar a validade da resposta

* ev($x^6 - 1$, % [1]);

Resposta:

$x = \left(\frac{\sqrt{3}i + 1}{64}\right)^6 - 1$ (primeira solução)

* expand (%);

Resposta = $\emptyset$

Comparemos com o resultado numérico(inexato)

```
*trial;ev(first(sol)),numer,expand
tempo = 400ms
Resposta:
          x = 0.866025

*x^6 - 1,trial,expand;
tempo = 166ms
Resposta:
          5.204e-17i+8.32e-17
```

fatoração

*Factor($x^{30}+1$);

Resposta:

$$(x^2+1)(x^4-x^2+1)(x^8-x^6+x^4-x^2+1)(x^{16}+x^{14}-x^{10}-x^8-x^6+x^2+1)$$

matriz

*determinant(

$$[x^3, x^2, x, 1]$$

$$[y^3, y^2, y, 1]$$

$$[z^3, z^2, z, 1]);$$

Resposta:

$$x(-y^2(x^3-w^3)+w^2z^3\ldots$$

O sistema MACSYMA é o único sistema integrado de computação algébrica desenvolvido na década de 60' que se encontra disponível na forma comercial. O sistema foi muito otimizado, desde a sua criação, obtendo assim, um código muito eficiente. Novos algorítmos foram incorporados, aumentando a abrangência do sistema.

I.7.2- REDUCE

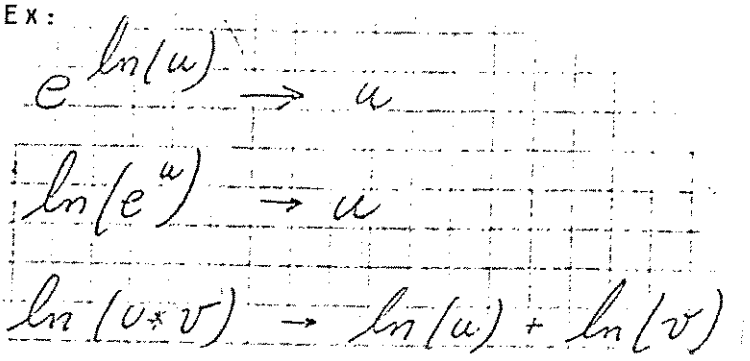
Implementado pelo grupo de Anthony Hearn /STOUTMEYER - 1979/, foi desenvolvido e implementado inicialmente para os sistemas: PDP-10
IBM360
CRAY-1

Memória requerida: 450k (IBM 360)
Linguagem de desenvolvimento: RLISP
Linguagem de utlização: ALGOL
Eis algumas características do sistema:

Precisão: Boa precisão em aritmética racional

Transformações algébricas automáticas ou manuais (controladas por flags)

Ex:



Simplificações exponenciais, logarítmicas e trigonoméricas

Diferenciação e integração simbólica: Módulo bem complexo onde são utilizados algorítmos como o de de Risch-Norman que garante a solução se a mesma existir.

Manipulação matricial: Aqui se encontram diversas rotinas desenvolvidas por Hearn para manipulação matricial. Alguns dos comandos disponíveis são:

UNIMAT A(2D,B(3) -> Gera matrizes unitárias de dimensão 2,3...

VCONCMAT(M1,M2) -> Efetua a concatenação entre matrizes.

MATEXTC -> Extrai uma coluna de uma matriz.

I.7.3- MUMATH

Desenvolvido por Albert D. Rich e David R. Stoutemyer da Soft Warehouse / STOUTEMEYER - 1979/muMATH - 80/.

O sistema necessita somente de 48k de memória graças a sua modularidade, que permite ao usuário usar só os módulos específicos para uma dada aplicação.

Eis algumas características do sistema:

. **Simplificação algébrica automática**
- agrupamento de termos similares
- agrupamento de fatores similares
- propriedades de identidade 0 , 1 como 1*u -> u

. **Possui boa precisão racional aritmética**

$$\frac{\sqrt{18}-\sqrt{8}}{\sqrt{6}} = \frac{1}{\sqrt{3}}$$

. **Transformações algébricas opcionais:**

- expansão de soma de potência inteiras
- expansão de produto de somas
- fatoração
- cálculo do denominador comum
- distribuição dos termos do denominador sobre termos do numerador correspondente.

Estas opções são controladas por variáveis para aplicação específica do usuário. Eis alguns exemplos resultantes das transformações:

$$\frac{2a^2x^2 - a^2bx - a^2b^2 - ax^3 + axb^2 - x^4 - bx^3}{a^2x^2 - a^2b - ax^2 + 2axb - ab^2 - b^2x} \longrightarrow$$

$$\frac{2ax + ab + x^2}{a + b}$$

. **Transformações trigonométricas**

automáticas e opcionais:
- remoção do sinal negativo dos argumentos;
- cálculo exato para ângulos que são múltiplos de pi/2;
- expansão angular múltipla;
- expansão da soma angular ;

- conversão de produtos trigonométricos em soma de ângulos ;

- conversão de potências trigonométricas em ângulos múltiplos.

. Regras para diferenciação e integração das funções e dos operadores do sistema.

Permite-se incluir regras para funções definidas pelo usuário.

Exemplos:

```
1/2 + 1/3;
@: 5/6

solve(x^2+5*x+6 == 0,X) ->
{x == 2 , x == 3}.
```

$$Int\left(\frac{1}{x^2+5x+6}, x\right) \rightarrow ln\left(\frac{-4-2*x}{6+2x}\right)$$

Para criar, por exemplo, a função cosecante:

```
CSC(u) ->  1/SIN(u)
FUNCTION(CSC(u)
        when u=0, undefined exit,
        1/SIN,
        ENDFUN;
```

CAPÍTULO II

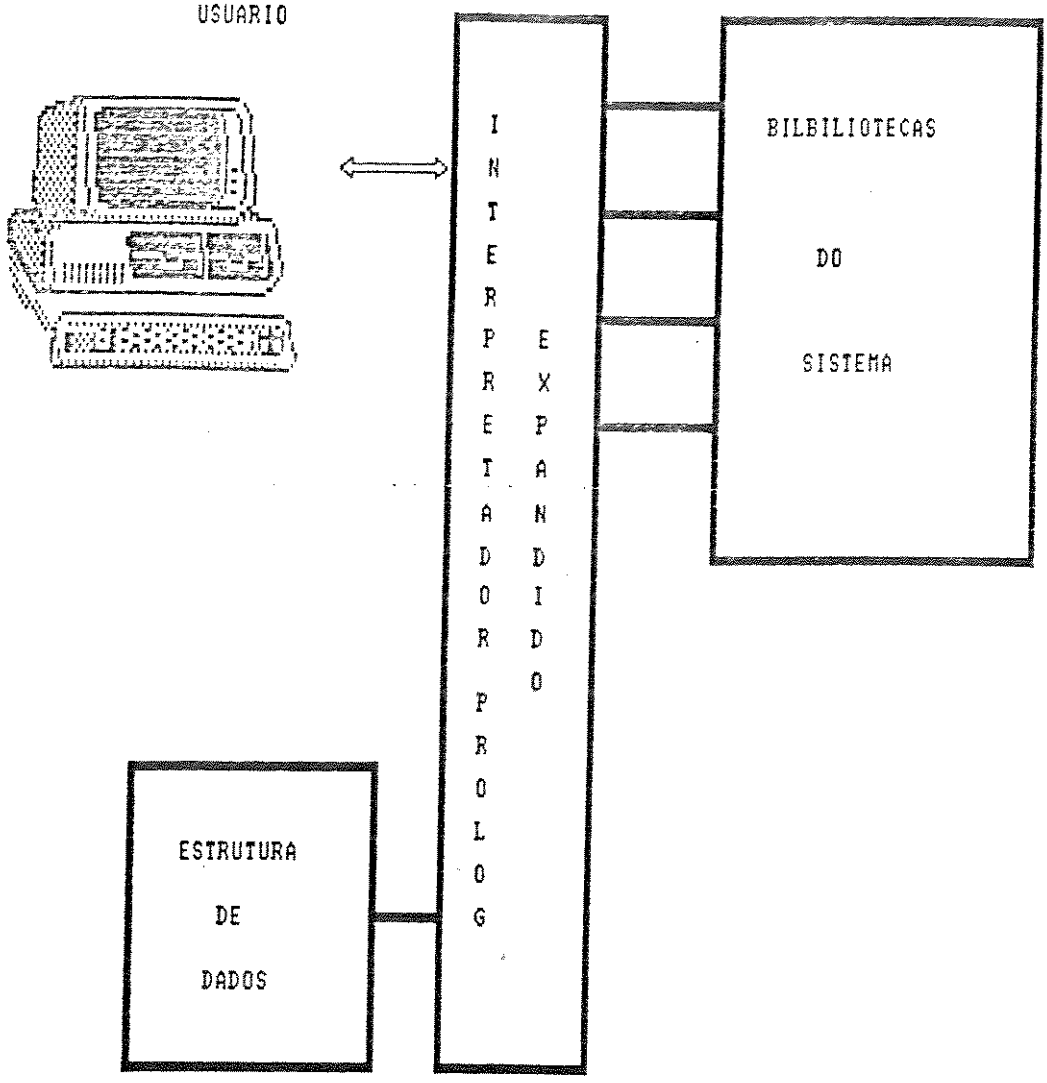
PROPOSTA DE UM SISTEMA INTEGRADO DE COMPUTAÇÃO SIMBÓLICA - SICS

II.1- OBJETIVO

O objetivo principal deste trabalho é mostrar a viabilidade de um sistema de manipulação algébrica em ambientes computacionais de pequeno porte. Este sistema foi desenvolvido tendo como suporte o interpretador PROLOG acrescido de funções especialmente desenvolvidas para manipulação de derivadas, matrizes , integrais e simplificação algébrica.

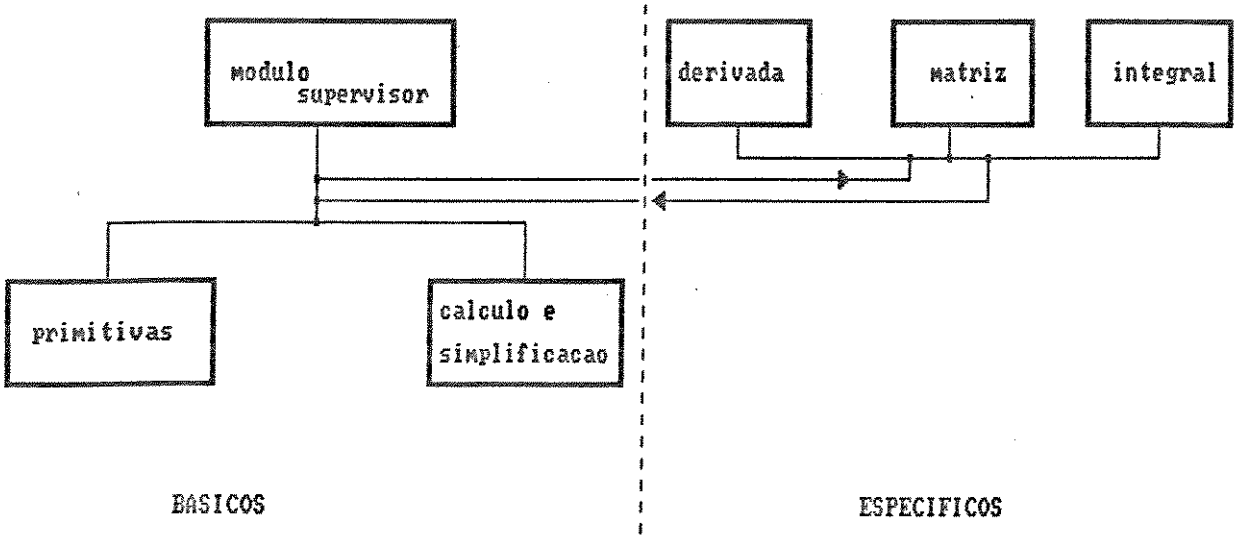
II.2- DESCRIÇÃO GERAL DO SISTEMA

- Interpretador PROLOG expandido
- Bibliotecas do sistema
- Estrutura de Dados



Interpretador Expandido-> É um interpretador PROLOG ao qual foram acrescentadas funções para manipulação de derivadas, matrizes e integrais.

Bibliotecas do Sistema -> São módulos que aumentam os recursos do sistema. Aqui se encontram os módulos de derivada, matriz , integral, primititivas, supervisor , cálculo e simplificação.



Os módulos serão divididos em módulos básicos e específicos.

O módulo das funções primitivas, o módulo supervisor e o módulo de cálculo e simplificação constituirão os módulos básicos do sistema. O módulo supervisor será responsável pela supervisão geral do sistema SICS. Terá diversas atribuições, entre as quais, podemos destacar: gerenciamento de arquivos, gerenciamento da base de dados, interfaceamento com os comandos dos DOS, etc. O módulo de funções primitivas conterá todas as primitivas a serem utilizadas durante o processo de desenvolvimento do sistema, bem como outras funções primitivas igualmente importantes para o desenvolvimento de sistemas computacionais utilizando o SICS. O módulo de cálculo e simplificação terá como atribuição, uma das tarefas mais importantes do sistema, uma vez que é o responsável pela avaliação de expressões matemáticas, retornando o resultado simplificado. Sua atuação abrangerá os diversos módulos do sistema.

Os módulos específicos são: módulo de derivada, módulo de matriz e módulo de integral. Estes módulos tornam o sistema mais abrangente, permitindo assim efetuar cálculos em áreas específicas como por exemplo a derivada e a integral.

módulo de derivada:

Este módulo será responsável pela derivada de expressões matemáticas, retornando a expressão simplificada. A derivada poderá ser simples, composta, de ordem superior e parcial. A simplificação do resultado será realizada pelo módulo de cálculo e simplificação.

módulo de matrizes:

Este módulo deverá conter um conjunto de funções básicas para manipulação de matrizes de forma simples e eficiente. Utilizaremos a lógica combinatória como uma forma alternativa de armazenamento e manipulação de matrizes.

módulo de integrais:

Este módulo será responasável pela integração de certas expressões matemáticas. A representação a ser utilizada para o problema da integral será a de grafos e/ou e a estratégia a ser utilizada será a best-first e/ou. Adotaremos o algorítmo utilizado por Slagle em seu programa SAINT, que tem na sua hierarquia de solução, três estágios principais:

- integrais imediatas- São as integrais encontradas nas tabelas dos livros texto de cálculo integral. Neste caso a solução é retornada imediatamente com o auxílio destas tabelas.

- integrais algorítmicas- Caso o integrando não se enquadrar nas integrais imediatas, tenta-se efetuar uma transformação no integrando de tal forma que resulte em um integrando de menor complexidade.Os resultados desta transformação serão integrados recursivamente através desta hierarquia de solução.

- integrais heurísticas- Nesta caso tenta-se efetuar uma transformação baseada na característica do integrando(isto é, informações a respeito do integrando). Caso a integral se enquadre neste caso, as possibilidades de sucesso são excelentes. Um exemplo seria uma função sen(x)^2 que se enquadraria no grupo de transformações trigonométricas.

Não se enquadrando em nenhum destes casos, o sistema retornará o integrando original.

Estrutura de Dados-> É a área de trabalho do Interpretador PROLOG e do usuário. O interpretador utiliza esta área para dados, pilhas, tabelas de variáveis, constantes, etc. Já o usuário utiliza esta área para carregar seus programas e efetuar suas manipulações. Esta área é portanto, única e pode ser dimensionada pelo próprio usuário.

II.3- DERIVADA, INTEGRAL E MANIPULAÇÃO DE MATRIZES

Faremos uma análise superficial do cálculo diferencial e integral de determinadas funções e também da manipulação de matrizes.

DERIVADA E INTEGRAL

Descreveremos a seguir o procedimento a ser efetuado para cada um dos tipos de funções.

A) Funções Algébricas - Uma função é algébrica se satisfaz a equação:

$$P_0(X)y^n + P_1(X)y^{n-1} + \ldots + P_{n-1}(X)y + P_n(X) = 0$$

sendo n inteiro e positivo
Pi(X) polinômio em X

A1) Polinômios

Definição de monômio: É um termo da forma C.X^n

com C -> constante

n -> inteiro positivo ou nulo

Definição de polinômio: É a soma de um número finito de monômios.

Ex:

$x^6 + 3x$ em X

$t^2 + t$ em t

Se os valores de uma função satisfazem uma equação $y=f(x)$ onde $f(x)$ é um polinômio em x então f é uma função polinomial.

Definição da derivada:

$$f'(x) = \lim_{Dx \to 0} \frac{f(x+Dx) - f(x)}{Dx}$$

f é diferenciável em x

Notação: $f'(x) = \frac{d(x)}{dx}$

Derivadas	Integrais
1- $\frac{d(c)}{dx} = 0$	$\int c\,du = c\int du$ $\quad c = c\underline{te}$
2- $\frac{d(c \cdot x^n)}{dx} = c n x^{n-1}$	$\int u^n du = \frac{u^{n+1}}{n+1}$ $\quad n \neq -1$
3- $\frac{d(c \cdot u)}{dx} = c\frac{du}{dx}$	$\int c \cdot du = c \cdot \int du$
4- $d(\text{soma}) = \text{soma das derivadas}$	$\int (du + dv) = \int du + \int dv$

O produto de dois polinômios diferenciáveis em X é também um polinômio.

$$5 - \frac{d(u.v)}{dx} = \frac{u.dv}{dx} + \frac{v.du}{dx}$$

A potência de um polinômio sendo inteira positiva gera um novo polinômio (u^n sendo u=g(x)).

$$6 - \frac{d(u^n)}{dx} = n.u^{n-1}\frac{du}{dx} \qquad \int u^n.du = \frac{u^{n+1}}{n+1}$$

Já o quociente de dois polinômios pode não gerar um polinômio e neste caso se enquadra nas funções racionais.

A2) Funções racionais -> São funções que se enquadram na forma:

f(x) = P(x) / Q(x) onde P(x) e Q(x) são polinômios

$$1 - \frac{d(u/v)}{dx} = \frac{v.\frac{du}{dx} - u.\frac{du}{dx}}{v^2} \qquad v \neq \emptyset$$

$$2 - \frac{d(u^{p/q})}{dx} = \frac{p}{q}u^{p/q-1}\frac{du}{dx} \qquad p \text{ e } q \text{ inteiros e } q \neq \emptyset$$

B) Funções transcendentais -> São funções que não se enquadram nas funções algébricas.

São elas:

- Funções trigonométricas
- Funções trigonométricas inversas
- Funções Logarítmicas
- Exponenciais

B1) Trigonométricas

derivada	integral
$\frac{d(\operatorname{sen} u)}{dx} = \cos u \frac{du}{dx}$	$\int \cos u \, du = \operatorname{sen} u + C$
$\frac{d(\cos u)}{dx} = -\operatorname{sen} u \frac{du}{dx}$	$\int \operatorname{sen} u \, du = -\cos u + C$
$\frac{d(\operatorname{tg} u)}{dx} = \sec^2 u \frac{du}{dx}$	$\int \sec^2 u \, du = \operatorname{tg} u + C$

compostas:

$$\frac{d(\operatorname{cotg} u)}{dx} = -\operatorname{cosec}^2 u \frac{du}{dx} \qquad \int \operatorname{cosec}^2 u \, du = \sec u + C$$

$$\frac{d(\sec u)}{dx} = \sec u \operatorname{tg} u \frac{du}{dx} \qquad \int \sec u \operatorname{tg} u \, du = \sec u + C$$

$$\frac{d(\operatorname{cosec} u)}{dx} = -\operatorname{cosec} u \operatorname{cotg} u \frac{du}{dx} \qquad \int \operatorname{cosec} u \operatorname{cotg} u \, du = -\operatorname{cosec} u + C$$

B2) Trigonométricas inversas -> São as funções do tipo ARCSEN(U), ARCCOS(U), etc. Não serão abordadas neste trabalho.

B3) Logarítmas
Logarítma neperiano -> Esta função é definida por:

- $\frac{d \ln x}{dx} = 1$ para x positivo
- $\ln(1) = 0$

sendo sua derivada e sua integral

$$\frac{d \ln|x|}{dx} = \frac{dx}{x} \qquad \int \frac{du}{u} = \ln|u| + c$$

tendo como propriedades:

$$\ln a \cdot x = \ln a + \ln x \quad \text{com } a = cte$$

$$\ln \frac{x}{a} = \ln x - \ln a$$

$$\ln x^n = n \cdot \ln x$$

$$\ln u \cdot v = \ln u + \ln v$$

$$\ln a^u = u \cdot \ln a$$

$$\ln v/v = \ln v - \ln v$$

B4) Função $y = \log_a u$

sendo u positivo e "a" diferente de "1", dizemos que y é o logarítmo de "u" na base "a" desde que $a^y = u$

$$\frac{d \log_a(u)}{dx} = \frac{1}{u \ln a}\frac{du}{dx}$$

tendo como propriedades:

$$\log_a(u) = \frac{\ln u}{\ln a}$$

$$\log_a(u.v) = \log_a(u) + \log_a(v)$$

$$\log_a(u/v) = \log_a(u) - \log_a(v)$$

$$\log_a(u^v) = v.\log_a(v)$$

B5) Função exponencial -> É o inverso do logarítmo natural e é definida por:

$y = e^x$ SSE $x = \ln y$

É uma função contínua, uniforme de x definida para todo x real.

sendo e = 2,71828...

sua derivada

$$\frac{de^x}{dx} = e^x$$ (invariante em relação à diferenciação)

sendo a regra mais geral, com a regra da cadeia implícita.

$$\frac{d\,e^u}{dx} = \frac{d\,e^u du}{du\,dx} = e^u \frac{du}{dx}$$

sua integral é

$$\int e^u du = e^u + d$$

Tendo como propriedades:

$$e^a e^b = e^{a+b}$$

$$e^{-a} = 1/e^a$$

$$e^0 = 1$$

B6) Função a^u
Sendo "a" real positivo qualquer sua derivada é:

$$\frac{d\,a^u}{dx} = a^u \cdot \frac{du}{dx} \ln a$$

e sua integral:

$$\int a^u\,du = \frac{1}{\ln a}\,da^u = \frac{a^u}{\ln a} + C$$

e suas propriedades são:

$$a^0 = 1$$

$$a^1 = a$$

$$a^u \cdot a^v = a^{u+v}$$

$$a^{-u} = 1/a^u$$

$$(a^u)^v = a^{uv}$$

$$(a \cdot b)^u = a^u\,b^u$$

Métodos gerais para derivada:

C1) Derivada de funções compostas

Duas funções y(u) e u(x) podem ser combinadas de diversas maneiras como por exemplo a função composta que é representada por y(u(x)). Um exemplo deste tipo de função é:

$y(u) = u^8$
$u(x) = x^2 + 1$

Para derivar adotamos o seguinte critério:
1o- Diferencie y(u) e substitua u(x) por x no resultado.
2o- Multiplique então pela derivada de u(x).
Este método é mais conhecido como regra da cadeia.

Exemplo:

$y = u^8 \rightarrow y'(u) = 8u^7$
$u = x^2 + 1 \rightarrow u'(x) = 2x$
$y'(x) = y'(u).\ u'(x) = 8.u^7.2.x = 16.u^7.x =$
$= 16.\ (x^2 + 1)^7.x$

C2) Derivada de ordem superior

Derivada segunda

$$f''(x) = \frac{d}{dx} f'(x) = \underline{\frac{(d^2)f(x)}{dx}} = \underline{\frac{d^2 f(x)}{dx^2}} = \underline{\frac{d\,[df(x)]}{dx\,dx}}$$

Derivada terceira

$$f'''(x) = \frac{d}{dx}\left[\frac{d}{dx}\left[\frac{d\,f(x)}{dx}\right]\right]$$

e assim por diante.

C3) Derivada parciais

Para funções com múltiplas variáveis
f(x,y) em relaçãp a y f_y

$f(x,y)=5.x^3y^2$

$$\frac{\partial f}{\partial x} = 15.x^2y^2$$

Procedimento para o cálculo de uma integral:

A solução da integral pode ser resumida em dois métodos principais de integração:

- Método de substituição

Com o auxílio de "tabelas" e métodos de transformações. Alguns destes métodos são:

. Tabela de integrais de funções elementares
. Integrais de potências de funções trigonométricas

. Integrais que envolvem
a.x^2+b.x+c
. Integrais pelo método de frações parciais

- Integração por partes

Nos casos que não se resolvem pelo método de substituição recorre-se ao método de integração por partes:

$$\int a.dv = u.v - \int v.du + C$$

MATRIZES

Matriz- É um arranjo retangular de elementos distribuidos segunda a forma:

$$A = \begin{bmatrix} a11 & a12 & \dots & a1m \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ an1 & an2 & \dots & anm \end{bmatrix} = aij \qquad \begin{matrix} i = 1,....m \\ j = 1,.... n \end{matrix}$$

Definiremos a seguir alguns conceitos básicos acerca de matrizes:

Igualdade de matrizes: Quando os elementos correspondentes forem iguais. Devem ser de mesma dimensão.

Vetor: É uma matriz de apenas uma coluna ou linha

$$\begin{bmatrix} x1 \\ x2 \\ . \\ . \\ xn \end{bmatrix}$$

vetor coluna (vetor n-dimensional)

[x1,x2,..... xn] **vetor linha**

Matriz quadrada: É uma matriz onde o número de linhas é igual ao número de colunas. É dita de ordem "n" se possuir "n" linha e "n" colunas.

Matriz diagonal: Quando todos os elementos, com exceção da diagonal principal, são nulos.

$aij = 0$ onde $i \neq j$

Matriz identidade (I): É uma matriz diagonal com os elementos da diagonal principal iguais a hum.

Matriz nula: Quando todos os elementos são zeros.

Matriz triangular superior: Quando uma matriz quadrada possui os elementos acima da diagonal principal nulos.

$aij = 0$ pra $j>i$

São ditas **triangulares superiores** quando $aij = 0$ para $i > j$.

Matriz tridiagonal: Possui todos os elementos, com exceção da diagonal principal, da subdiagonal e da superdiagonal, nulos.

Determinante de uma matriz-> Toda matriz quadrada possui um determinante.

$det(A) = |A|$

As suas propriedades são:

- Se duas linhas(colunas) quaisquer são permutadas o determinante muda de sinal.
- Se qualquer linha(coluna) possuir apenas elementos zeros, o valor do determinante é zero.
- Se os elementos correspondentes de duas linhas(colunas) são iguais ou proporcionais, o valor do determinante é zero.
- Se um múltiplo escalar de uma linha (coluna) de A é somado a outra linha(coluna) , o valor do determinante não muda.
- O determinante do produto de duas matrizes quadradas A e B é o produto dos determinantes ou:

$|A.B| = |A|.|B|$

- O determinante de uma matriz triangular (superior ou inferior) é:

$|A| = a1.a22....ann$

Matriz Singular: Quando o determinante associado é zero.

Matriz transposta: Se as linhas e colunas de uma matriz Anxn são trocadas, a matriz resultante A'mxn é denominada transposta de A.

$(A')' = A$
$(A.B)' = B'.A'$
$|A| = |A'|$
$|A.B| = |A|.|B|$

Matriz simétrica: Se uma matriz quadrada é igual a sua transposta.

$A = A'$

Matriz anti-simétrica: Se uma matriz quadrada é igual ao negativo de sua transposta.

$A = - A'$

Álgebra matricial

Adição e subtração de matrizes: Para efetuar adição ou subtração, as matrizes devem ter igual dimensão, isto é, mesmo número de linhas e colunas.

$A = aij \qquad B = bij$
$A + B = (aij + bij)$
$A - B = (aij - bij)$

Multiplicação de uma matriz por um escalar: Cada elemento da matriz é multiplicado pelo escalar obtendo a matriz resultante.

$$k.A = k(aij)$$

Multiplicação de duas matrizes: Só é possível entre duas matrizes conformes (onde o mesmo número de colunas da primeira matriz é igual ao número de linhas da segunda).

$$A_{nxm}$$
$$B_{mxp}$$
$$C = (AB) = \sum_{k=1}^{m} (aik.bkj) \qquad i=1,2...n \quad j=1,2...p$$
$$C_{nxp}$$

Propriedades:

$A(BC) = (AB)C$ -> Associativa
$A(B+C) = AB + AC$ -> Distributiva
$(A + B)C = AC + BC$ -> Distributiva
$A + (B+C) = (A + B) + C$ -> Associativa
$A + B = B + A$ -> Comutativa
$A + 0 = A$ -> Identidade sob adição

Potência de uma matriz->A nésima potência de uma matriz A é definida por:

$$A^k = \underbrace{A.A...A}_{k}$$

para uma matriz diagonal $A^k = diagonal(a^k11, a^k22, ..a^knn)$

propriedades:

$(A + B)' = A' + B'$
$(AB)' = B'A'$

Inversão de matrizes: Dada uma matriz quadrada A, existe uma matriz B denominada inversa de A se:

$BA = AB = I$
$B = A^{-1}$

Ela existe se o determinante de A for não nulo ou A for não singular. Se existir será única.

Matriz adjunta: É a transposta da matriz cujos elementos são os cofatores de A.

$B = (bij) = (Aji) = adjA$

com a seguinte propriedade:

$A(adj\ A) = |A|\ I$

Cofator Aij: O cofator Aij do elemento aij da matriz A é definido pela equação:

$Aij = (-1)^{i+j} Mij$

Por definição, a inversa de uma matriz é:

$$A^{-1} = \frac{adj\ A}{|A|}$$

propriedades:

$$A \cdot A^{-1} = A^{-1} A = I$$

$$(A^{-1})^{-1} = A$$

$$(A^{-1})^{*} = (A^{*})^{-1}$$

$$(A \cdot B)^{-1} = B^{-1} A^{-1}$$

sendo "*" a transposta de uma matriz

Diferenciação de matriz

A derivada de uma matriz AmXn é a matriz das derivadas dos elementos de AmXn. Para isto, todos os elementos devem ter derivada em relação a variável de derivação.

$$\frac{dA(t)}{dt} = \left(\frac{d.aij(t)}{dt}\right) = \begin{bmatrix} \frac{d.a11(t)}{dt} & \cdots & \frac{d.a1m(t)}{dt} \\ \vdots & & \vdots \\ \frac{d.an1(t)}{dt} & \cdots & \frac{d.anm(t)}{dt} \end{bmatrix}$$

Podemos afirmar também:

$$\frac{d[A(t).B(t)]}{dt} = \frac{d.A(t).B(t)}{dt} + A(t).\frac{dB(t)}{dt}$$

$$\frac{dA\hat{}-1(t)}{dt} = -A\hat{}-1(t).\frac{dA(t)}{dt}.A\hat{}-1(t)$$

Integração de uma matriz

A integração de uma matriz AmXn é uma matriz das integrais dos elementos de AmXn.

$$\int A(t).dt = \int aij(t)dt = \begin{bmatrix} \int a11(t)dt & \cdots & \int a1m(t)dt \\ \vdots & & \vdots \\ \int an1(t)dt & \cdots & \int anm(t)dt \end{bmatrix}$$

II.4- AMBIENTE COMPUTACIONAL

Micro utilizado: Micro de 16 bits da linha de compatíveis com o IBM PC-XT, dada sua padronização nos meios científicos. Sua memória real é de 640k e winchester de 10M.

Linguagem: Prolog

Memória: É otimizada sua utilização devido à modularidade do sistema

II.5- INTERFACE COM USUÁRIO

Esta interface foi desenvolvida sobre o interpretador prolog para facilitar o desenvolvimento de programas do usuário do SICS. Foi projetada segundo as características da linguagem, o que simplifica a tarefa do usuário. Para maiores detalhes desta ferramenta vide capítulo 4.

II.6 - DESCRIÇÃO DA META-LINGUAGEM

As expressões serão descritas em uma meta linguagem resumida abaixo:

Operando ou operação	Representação
Adição	+
Subtração	-
Divisão	/
Exponenciação	^
Parênteses	()
Variável de diferenciação	qquer carac. alfabet.
Constante	letra minúscula
Matriz	[[a,b],[c,d]]
Traço da matriz	mt
Transposta da matriz	#
Funções transcendentais	nome(X)
Integral	int(integrando,var)
Integral definida	int(integ,[var,lim1,lim2])
Derivada	der(exp,var)

onde nome poderá ser:

Função	Nome
Exponencial	^
Logaritmo	log
Seno	sen
Coseno	cos
Tangente	tg

Classe de expressões:

. Uma constante ou uma variável é uma expressão
. Se E é uma expressão -E ou +E são expressões
. E1 e E2 sendo expressões então:
E1+E2; E1-E2, E1*E2, E1/E2 são também expressões
. Se E1 e E2 são expressões então E1^E2 é uma expressão
. Se E1 é uma expressão e "Nome" representa uma função transcendental dentre as definidas acima, então Nome(E1) é uma expressão
. Os ítens acima são as únicas expressões permitidas como argumentos da funções definidas ao longo do sistema SICS.

CAPÍTULO III

IMPLEMENTAÇÃO DE UM SISTEMA INTEGRADO

O sistema SICS compreende vários módulos que serão descritos a seguir:

III.1 - MÓDULO SUPERVISOR DO SISTEMA (M.S.S.)

Arquivo: MOD_SIS.ARI

Este módulo é o responsável pela interação do usuário (por intermédio da linguagem PROLOG) com os recursos do sistema.

Os comandos aqui presentes são para manipulação de arquivos e de acesso aos recursos do DOS . Alguns comandos disponíveis originalmente no PROLOG foram criados de forma diferente, de modo a facilitar o trabalho do usuário. Outros comandos não disponíveis, mas de grande utilidade foram desenvolvidos. Os comandos em geral são DOS-LIKE, isto é, assemelham-se ao DOS.

Uma das grandes vantagens deste supervisor é que o mesmo foi desenvolvido em PROLOG, o que pode ser facilmente modificado ou mesmo implementado com novas funções que atendam especificamente um dado usuário.

As funções atualmente disponíveis são:

```
a(mudar o drive corrente para a:)
abrir (abrir arquivo)
all_primi(lista toda a base de dados)
b(mudar o drive corrente para b:)
bak(deletar arquivos "backup")
c(mudar o drive corrente para c:)
criar(criar arquivo)
d(lista arquivo fonte)
del(deletar arquivo)
dia
dir(lista diretório corrente)
disco(mudar drive corrente)
disk(mostra o drive corrente)
hora
l(listar a base de dados)
lg(editor  com armazenamento automático da  sessão
   de trabalho)
ne(invocar editor Norton)
nsp(nospy)
nt(notrace)
primi(ajuda on-line do usuário)
r(reconsult)
salva_cláusulas(salvar cláusulas da base de dados)
sp(spy)
tex(interface especializada)
tr(trace)
ws(invoca temporariamente o wordstar)
```

Exemplos de aplicações de algumas destas funções são:

?- a. (muda o drive corrente para a)

?- bak. (deleta arquivos .bak e .~ do drive corrente)

?- del arquivo. (deleta arquivo)

?- dir. (diretorio normal)

?- tex. (Aciona interface especializada que contém a maioria das funções deste módulo. Vide capítulo IV)

Para maiores informações sobre as funções aqui citadas, vide apêndice D - Descrição das funções implementadas.

III.2 - MÓDULO DE MANIPULAÇÃO DE DERIVADA (M.M.D)

Arquivo: MOD_DER.ARI

Este módulo é o responsável pela manipulação das regras de derivada sobre funções simples, compostas, de ordem superior e parciais.

Dada as características da linguagem PROLOG, o sistema de derivada sem a simplificação, fica extremamente facilitado. As regras aqui disponíveis atualmente são as derivadas elementares e a regra da cadeia.

Incialmente daremos alguns exemplos de sua aplicação como forma de dar uma idéia concreta de sua atuação:

- derivar

$$\frac{d(x^2+3x)}{dx}$$

em notação prolog:

```
?- avalia(der(x^2+3*x,x),Resposta).
Resposta = 3*x + 3
```

- derivar

$$\frac{dd(x^2*x+3+e^{(2*x)})}{dxdx}$$

em notação prolog:

```
?- avalia(der(x^2+3+e^ (2*x),x),Resposta).
Resposta = 2*(x+e^ (2*x))
```

```
?- avalia(der(x,y),Resposta).
 Resposta = 0

?- avalia(der(x^2,x^2),Resposta).
 Resposta = 1

?- avalia(der(sen(x*y),[x,y]),Resposta).
 Resposta = cos(x*y) - y*(x*sen(x*y))
```

```
?- avalia(der([[x,x],[x,x]],x) +
  int([[x,x],[x,x]],x),Resposta).
  Resposta = [[1+x^2/2,1+x^2/2],[1+x^2/2,1+x^2/2]]
```

Sua área de atuação é: derivada simples, composta, de ordem superior e parcial sobre funções elementares de variáveis reais. A definição de função elementar é a seguinte:

I- Constante é uma função elementar.

II- Variável é uma função elementar.

III- Soma ou produto de funções elementares são funções elementares.

IV- Função elementar elevada a constante é uma função elementar.

V- Função trigonométrica de função elementar é função elementar.

O nível da solução dos problemas está limitado às funções elementares descritas, visando uma aplicação didática, mas podendo ser complementado com a simples adição de regras de derivação.

O procedimento adotado no cálculo da derivada de funções elementares é a redução da expressão a subproblemas até chegar a uma expressão que contenha a derivada imediata.

O procedimento geral é:

1- A expressão se enquadra nas derivações imediatas? . Caso positivo, faça a transformação segundo uma "tabela de derivadas imediatas" e retorne o resultado. Caso contrário continue.

2- É aplicável alguma transformação de redução na expressão ? . Se não, retorne à expressão como resultado. Se sim, continue.

3- Faça a transformação obtendo uma nova expressão reduzida.

4- Retorne ao item 1.

Sendo:

Derivadas imediatas: São expressões em que se pode-se obter a sua derivada através de uma simples transformação.

Neste caso existe uma "tabela de derivadas imediatas" que faz a conversão imediata. Esta tabela é encontrada em livros de cálculo diferencial básico.

Transformação de redução: As funções não atômicas elementares são desmembradas em funções de menor complexidade.

Exemplos

I- $\frac{d(x-1)}{dx}$

transformação de redução

$\frac{d(x)}{dx} - \frac{d(1)}{dx}$

derivada imediata

$1 - \emptyset$

simplifica

1

II- $\frac{d(sen(x) * cos(x))}{dx}$

transformação de redução

$cos(x) * \frac{d(sen(x)}{dx} + sen(x) * \frac{d(cos(x))}{dx}$

imediata

$cos(x) * cos(x) + -sen(x) * sen(x)$

simplifica

$cos(x)^2 - sen(x)^2$

O predicado principal responsável pela derivada de expressões matemáticas é o der:

```
der(X,V,Resp):-
        deriva_matriz(X,V,Resp);
        deriva_expr(X,V,Resp).
```

Inicialmente verifica-se se o termo é uma matriz. Caso positivo, será feito um desmembramento da matriz e aplicada a derivada a cada um dos seus elementos recaindo na aplicação recursiva de derivada de expressões não matriciais. Caso a expressão não seja matriz será aplicado o procedimento descrito a seguir:

```
a1. der(X,[V],Resp):- der(X,V,Resp).
a2. der(X,[V|T],Resp):- deriva(X,V,Inter),
                        simplifica(Inter,Simp),
                        der(Simp,T,Resp).
a3. der(X,V,Resp):- deriva(X,V,Inter),
                    simplifica(Inter,Resp).
```

sendo o predicado simplifica responsável pela simplificação de expressões matemáticas.

As duas cláusulas iniciais são responsáveis pela solução de derivadas compostas, de ordem superior e parciais de expressões matemáticas, onde no lugar da variável de derivação colocamos uma lista de uma ou mais variáveis. Neste último caso, efetua-se a derivada simples em relação à primeira variável. Simplifica-se e o resultado intermediário é derivado em relação à segunda variável obtendo-se o resultado final. Se houver mais variáveis de derivação, segue-se o mesmo raciocínio.

A terceira cláusula é responsável pela derivada simples com a imediata simplificação do resultado.

As regras básicas de derivação implementadas são as derivadas imediatas e a regra da cadeia.

Algumas destas regras são:

```
deriva(X,X,1).
deriva(T,X,0).
deriva(U+V,X,DU+DV):- deriva(U,X,DU),
                      deriva(V,X,DV).
```

Exemplos:

$$\frac{-d(x)}{x} = 1$$

em notação PROLOG:

```
deriva(x,x,1).
```

$$\frac{-d(U+V)}{dx} = \frac{d(U)}{dx} + \frac{d(V)}{dx}$$

em PROLOG:

```
deriva(U+V,X,DU+DV):- deriva(U,X,DU),
                      deriva(V,X,DV).
```

Para adicionar novas regras à base de dados o procedimento é muito simples. Basta incluir no arquivo "regra.ari" a nova regra de acordo com a formatação ali presente e então atualizar o sistema em relação a esta nova regra. Para isto, recorre-se ao apêndice C -(Expandindo o interpretador) e realize as operações ali descritas a partir do ítem 2.

III.3 - MÓDULO DE MANIPULAÇÃO DE INTEGRAIS (M.M.I)

Arquivo: MOD_INT.ARI

Este módulo é o mais complexo do sistema e é responsável pelo cálculo de integrais. A solução do problema da integral foi baseada no sistema SAINT /SLAGLE 1961/ acrescido de idéias dos sistemas SIN / MOSES 1969/ e / KANOUI 1976/ e tem como característica básica a aplicação de técnicas de Inteligência Artificial.

O domínio de atuação do módulo de integral é o da integral indefinida e integrais definidas que sejam extensões simples de integrais indefinidas. Os integrandos devem ser funções elementares definidas como segue:

a- Qualquer constante é uma função elementar.

b- Uma variável é uma função elementar.

c- O produto ou a soma de duas funções elementares são funções elementares.

d- Uma função trigonométrica de uma função elementar é uma função elementar.

O nível de solução é o mesmo de um curso básico de ciências exatas.

Existem duas grandes estratégias utilizadas para solução de integrais: As Heurísticas e os Algorítmos. As primeiras são métodos que auxiliam na solução dos problemas através de tentativas do caminho a ser adotado, mas com possibilidade de falha. Já os algorítmos são métodos que decidem infalivelmente o que fazer para uma certa classe de problemas. Geralmente são adotadas ambas as técnicas uma vez que os algorítmos podem gerar uma árvore de busca muito grande tornando o sistema inviável. Nos casos em que a solução está bem clara, então recomenda-se utilizar diretamente os algorítmos. Já nos casos em que a solução não está muito clara recorre-se primeiramente às heurísticas para diminuir o espaço de busca , e se necessário aos métodos algorítmos.

Neste trabalho procurou-se utilizar basicamente técnicas encontradas no ramo da Inteligência Artificial, que foi o objetivo inicial desta dissertação de mestrado. Cabe aqui ressaltar que para o sistema tornar-se completo deve-se implementar alguns métodos algorítmos.

A representação adotada foi a da busca em grafo. Um dos grandes problemas deste tipo de representação é a explosão combinatória, que foi combatida com eficiência através de técnicas heurísticas, graças aos conhecimentos específicos acerca do domínio em questão para direcionar a busca, evitando com isto caminhos desnecessários . No presente caso, temos o domínio da integral bem definido, de onde podemos extrair informações necessárias para escolher o caminho mais adequado.

O tipo de grafo adotado foi o de grafos e/ou. A escolha foi devida a esta ser adequada a problemas bem definidos que podem ser divididos em sub-problemas independentes como é o caso da integração simbólica.

A estratégia de busca adotada foi a best-first e/ou onde na expansão do integrando, a escolha do melhor candidato é feita através de uma função avaliadora. É um

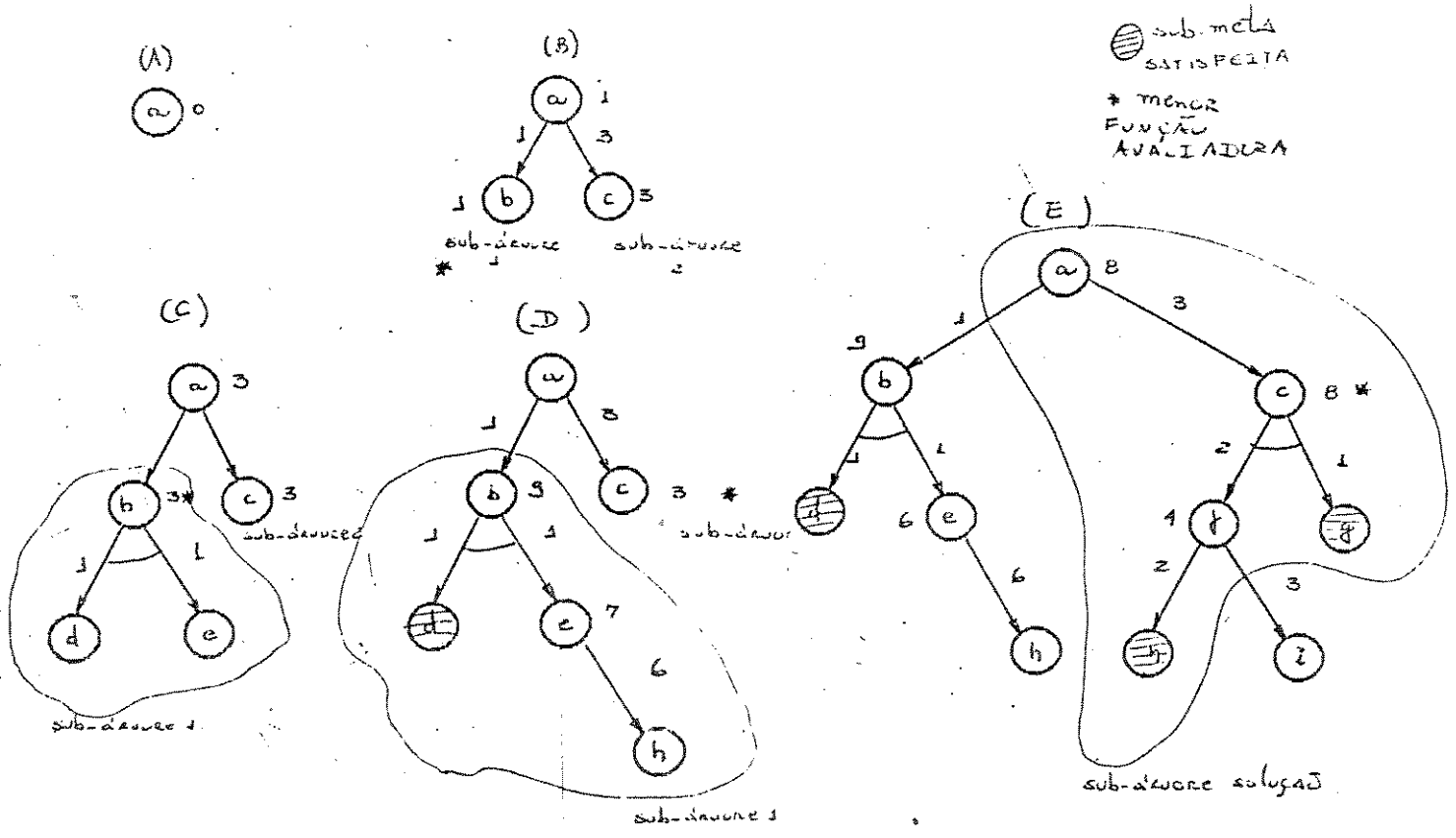
refinamento da busca em largura. Usando esta função avaliadora chegaremos ao algorítmo A* /NILSON - 1980/.

Esta função além de evitar a explosão combinatória do espaço de busca gerada na representação dos grafos, permite percorrer caminhos mais promissores em relação a meta a ser atingida. O objetivo desta estratégia é chegar a uma árvore solução com custo mínimo. Durante o processo de expansão é selecionada sempre a árvore solução mais promissora.

O algorítmo utilizado é nada mais que um algorítmo de busca best-first generalizado , uma vez que utilizamos um algorítmo e/ou em que foram incluido os custos. Este algorítmo foi adaptado do algorítmo desenvolvido por Bratko/Bratko 1986/.

Descrição do algorítmo

O processo de busca consiste em um número de sub-árvores, cada uma explorando uma alternativa. Somente uma sub-árvore está ativa por vez: a que tem a alternativa mais promissora (menor função avaliadora). As sub-árvores restantes devem esperar até que o valor da função avaliadora mude de tal modo que a alternativa atual deixe de ser a mais promissora. Neste ponto a atividade é chaveada para a sub-árvore com a menor função avaliadora. Enquanto uma sub-árvore está ativa (menor valor de função avaliadora) esta sofre continuamente o processo de expansão, podendo retornar à solução se a meta original for satisfeita . A atualização do valor da função avaliadora é necessária a cada nível de busca, para que possa chavear o processo para a sub-árvore mais promissora.



A → $F(a) = \phi$

B → $F(b) = C_{ab} + H_b = 1$
$F(c) = C_{ac} + H_c = 3$
$\Rightarrow F(a) = \min\left(F(b), F(c)\right) = 1$
∴ sub-árvore 1 escolhida

C → $F(d) = C_{bd} + H_d = 1$
$F(e) = C_{be} + H_e = 1$
$\Rightarrow F(b) = C_{ab} + \Sigma\left(F(d) + F(e)\right) =$
$= 1 + 2 = 3$
∴ sub-árvore 1 escolhida

D → $F(h) = C_{eh} + H_h = 6$
⇓
$F(e) = 7$
$F(d) = 1$
$\Rightarrow F(b) = 1 + (7 + 1) = 9$
∴ sub-árvore **2** escolhida

E → $F(h) = C_{fh} + H_h = 2$
$F(i) = C_{fi} + H_i = 3$
$\Rightarrow F(f) = 2 + \min(2,3) = 2 + 2 = 4$
$F(g) = C_{cg} + H_g = 1$
⇓
$F(c) = 3 + (4 + 1) = 8$
retornando como solução
$h + g$

A estratégia é botton-up uma vez que partimos do nó incial (integrando) e com a expansão dos nós mais promissores vamos gerando a árvore de busca até chegarmos à árvore solução.

O SICS utiliza atualmente para solucionar seus problemas 16 formas padrões, 15 transformações algorítmicas e 15 transformações heurísticas.

Inicialmente definiremos alguns conceitos básicos utilizados neste módulo.

Meta original -> É a expressão a ser integrada.

Nó ou meta -> É o integrando ou expressão gerada a partir de transformações no integrando.

Função Avaliadora -> É uma função utilizada na ordenação dos nós a serem expandidos.Sempre que uma meta é expandida, é calculado a função avaliadora das sub-metas geradas, sendo também atualizado o valor da função avaliadora da sub-árvore corrente. Assim podemos selecionar a meta mais promissora a ser expandida conseguindo com isto, uma solução otimizada.

Esta função avaliadora é necessária para determinarmos o custo das árvores-solução e então selecionarmos a árvore-solução com o menor custo.

Esta função avaliadora é constituída de dois fatores de custo. O custo do nó ancestral até o nó atual e o custo do nó atual até à solução.

Quando um nó é encontrado, o caminho do nó inicial até o nó presente já é conhecido e seu custo (soma dos custos do caminho) pode ser calculado.

O outro termo, que se refere ao custo do nó presente até o nó terminal, não sabemos e deve ser baseado em estimativas heurísticas, através do conhecimento do algorítmo acerca de um domínio específico. Portanto não existe um método universal para o seu cálculo.

A definição da função avaliadora é:

F(N) = C(M,N) + H(N) onde
C(M,N) -> Custo do nó de M a N
H(N) -> Estimação heurística do custo do nó corrente até à solução.

Nesta definição se enquadram os nós ainda sem sucessores ("tip nodes").

Para o caso dos nós interiores, onde um dado nó N tem como ancestral M e sucessores Ni teremos a definição segundo o tipo de nó encontrado:

Nó OR
F(N) = C(M,N) + min(F(Ni))
sendo o M ancestral de N , Ni os filhos de N e C(M,N) o custo do nó M a N.

Nó AND
F(N) = C(M,N) + Somatória(F(Ni))

Neste caso, não utilizamos o valor de H(N) diretamente, uma vez que dispomos de algumas informações sobre este nó, isto é, já conhecemos os seus sucessores.

F(N) estimará o custo do melhor caminho a partir do nó inicial para o nó terminal, com a condição de que este caminho passe pelo nó N.

Neste algorítmo, o valor do custo C(M,N), cresce de uma unidade a cada nível na árvore-solução , sendo que o primeiro nível (o integrando original) tem o valor zero para o C(M,N). A medida que os nós vão sendo expandindo, o valor do custo vai crescendo.

Já o valor de H(N) utiliza um critério para estimar a dificuldade do problema. Neste sistema foi utilizado o número de funções presentes no integrando, como uma forma de selecionar o menor integrando dentre os existentes, e escolher o melhor caminho até a solução, evitanto com isto, expandir nós menos promissores ou desnecessários.

Lista de sub-árvores -> Lista de árvores a serem expandidas. São ordenadas de acordo com o valor crescente da função estimadora(F). As árvores resolvidas são colocadas no final da lista.

Característica -> Sempre que uma meta é gerada, calcula-se sua característica que é um conjunto de informações sobre o integrando a ser utilizada na ordenação da lista de metas e na aplicação da transformação heurística. Eis algumas das informações pertinentes a característica atualmente disponíveis:

a- Comprimento

O comprimento de uma expressão é o número de funções elementares que tenta espelhar a dificuldade do problema a ser resolvido.

```
comp(sen(x),2).
comp(x,1).
```

b- Função elementar de sen ou cos
c- Função elementar de sec ou tg
d- Função elementar de cosec ou cotg
e- Função elementar de função trigonométrica (sen, cos, tg, cotg, sec e cosec)
f- Somas não constantes
g- Potências inteiras positivas de somas não constantes
h- Expressão polinomial
i- Expressões compostas
j- Expressões que envolvem suas derivadas
k- Integral por partes

Após a análise do integrando, será retornada uma lista de informações sobre o integrando em relação aos ítens acima. Por exemplo se o integrando é sen(x) então obteremos a lista de características com os ítens "a=1(comprimento)" , "b=1(setado) e "e=1(setado). Os ítens restantes estarão resetados.

A partir destas características, podemos reconhecer um determinado integrando e então aplicar sobre o mesmo, uma regra com grande possibilidade de êxito.

Se a expressão é função elementar de sen ,será aplicada sobre a mesma uma das regras do grupo das funções

trigonométricas.

Após a transformação do integrando, obteremos um ou mais sub-integrandos, que devem ser anexados a lista de metas para expansão, conforme o valor de sua função avaliadora.

No presente trabalho, o critério utilizado na função avaliadora para o termo de H(estimativa heurística) foi o comprimento da expressão (ítem a), isto é, o número de funções presentes no integrando, tentando espelhar a dificuldade do integrando. Novos critérios devem ser implementados futuramente, visando obter um algorítmo mais eficiente.

Expansão dos nós

Para a expansão dos nós o sistema usa um procedimento que tem como objetivo achar uma solução eficiente, sem, no entanto efetuar mudanças radicais no integrando, a menos que seja necessário. A estratégia deste procedimento obedece uma hierarquia de soluções. Esta hierarquia pode ser dividida em três estágios e foi baseada no sistema SAINT desenvolvido por Slagle.

Primeiro estágio

TENTATIVA PELOS MÉTODOS DIRETOS(TRANSFORMAÇÕES IMEDIATAS)

Sempre que um nó é gerado, inicialmente tenta-se verificar se o mesmo se enquadra nas integrais imediatas, que são satisfeitas através de uma simples substituição. As regras atuais estão descritas a seguir:

a. $\int c.dx = c\ x$
b. $\int 1/x.dx = ln(x)$
c. $\int x^{-1}.dx = ln(x)$
d. $\int x.dx = x^2/2$
e. $\int x^n.dx = x^{n+1}/n+1$
f. $\int e^x.dx = e^x$
g. $\int c^x.dx = c^x/ln(c)$ sendo c uma constante
h. $\int ln(x).dx = x.ln(x)-x$
i. $\int sen(x).dx = -cos(x)$
j. $\int cos(x).dx = sen(x)$

Caso o integrando original se enquadre neste primeiro estágio, a resposta será imediata.

Exemplo:

$$\int \frac{1}{x}\,dx = ln(x)$$

A ordem das regras é importante, uma vez que o

PROLOG efetua a busca de cima para baixo. Podemos observar que não especificamos o valor de "n" (<> -1) na regra "e" uma vez que a regra "c" já enquadrou este caso.

Segundo estágio

TENTATIVA PELOS MÉTODOS ESPECÍFICOS (TRANSFORMAÇÕES ALGORÍTMICAS)

Para os casos que não se enquadram nos métodos diretos, tenta-se efetuar uma transformação do integrando a problemas menos complexos e mais perto da solução.

a) Fator constante

$$\int c \cdot f(x)\,dx = c \cdot \int f(x)\,dx$$

$c = cte$

Exemplo:

$$\int 5 \cdot \operatorname{sen}(x)\,dx = 5 \int \operatorname{sen}(x)\,dx$$

b) Somatória

$$\int \sum g_i(x)\,dx = \sum \int g_i(x)\,dx$$

Exemplo:

$$\int (x+2)\,dx = \int x\,dx + \int 2\,dx$$

c) Expansão

$$\int (x+1)^2 dx = \int x^2 dx + \int 2x\, dx + \int 1\, dx$$

d) Expressões trigonométricas

$$\int tg(x)\, dx \quad - \int -\frac{d\cos(x)}{sen(x)}\, dx$$

$$\int tg^2(x) \quad - \int (sec^2(x) - 1)\, dx$$

$$\int cotg^2(x) \quad - \int (cosec^2(x) - 1)\, dx$$

Terceiro estágio
TENTATIVA POR MÉTODOS HEURÍSTICOS

Nos casos que não se enquadram na forma padrão ou na transformação algorítmica, tenta-se como último recurso, a transformação heurística. As transformações heurísticas são aquelas que, se aplicadas, podem chegar mais próximas da solução, mas com possibilidade de falha.

Neste estágio são efetuadas transformações

radicais no integrando como última alternativa de se obter uma resposta.

É uma transformação inteligente, uma vez que antes de efetuar a aplicação da mesma, é feito um reconhecimento do integrando para se determinar qual a transformação mais adequada a ser aplicada. Existem diversos grupos de regras, cuja aplicação é mais apropriada para um determinado tipo de integrando. Há diversos grupos de transformações entre os quais podemos citar: grupos de funções trigonométricas, grupo de polinômios, grupo de expressões que envolvem sua derivada, etc. Se o integrando for uma função trigonometrica, então será aplicado sobre o mesmo, uma regra de tranformação do grupo da funções trigonométricas.

Algumas destas transformações retornam à solução da integral e outras geram sub-problemas de menor complexidade.

Eis alguma das transformações heurísticas atualmente disponíveis:

a- Um integrando que não foi gerado por esta regra e é função elementar de funções trigonométricas, isto é, da forma (sen(x),cos(x),tg(x),cotg(x),sec(x),cosec(x)) é transformado em uma ou mais das seguintes formas:

a1-

$$tg(x) \rightarrow \frac{sen(x)}{cos(x)} \qquad sec(x) \rightarrow \frac{1}{cos(x)}$$

$$cotg(x) \rightarrow \frac{cos(x)}{sen(x)} \qquad cosec(x) \rightarrow \frac{1}{sen(x)}$$

e o integrando não é função elementar de sen(x) e cos(x)

a2-

$$sen(x) \rightarrow \frac{tg(x)}{sec(x)} \qquad cotg(x) \rightarrow \frac{1}{tg(x)}$$

$$cos(x) \rightarrow \frac{1}{sec(x)} \qquad cosec(x) \rightarrow \frac{sec(x)}{tg(x)}$$

e o integrando não é função elementar de sec(x) e tg(x)

a3-

$$\operatorname{sen}(x) - \frac{1}{\operatorname{cosec}(x)} \qquad tg(x) - \frac{1}{\operatorname{cotg}(x)}$$

$$\cos(x) - \frac{\operatorname{cotg}(x)}{\operatorname{cosec}(x)} \qquad \sec(x) - \frac{\operatorname{cosec}(x)}{\operatorname{cotg}(x)}$$

e o integrando não é função elementar de cosec(x) e cotg(x) .

Exemplo:

$$\int \frac{dx}{\sec(x)} \quad - \quad \int \cos(x)\, dx$$

b- Expressão que envolve sua derivada:

$$\int c \cdot op(u(x))\, u'(x)\, dx$$

sendo $c = c\underline{te}$

$op(u(x)) = u(x)$

$u(x)^{-1}$

$u(x)^{d}$ $d \neq -1$

$d^{u(x)}$ $d\ c\underline{te}$

$$\int x \cdot e^{x^2} = \frac{1}{2} e^{x^2}$$

$$\int \operatorname{sen}(x) \cdot \cos(x) \cdot dx = \frac{1}{2} (\operatorname{sen}(x))^2$$

c- Integral por partes

$$\int u\,dv = uv - \int v\,du$$

Exemplo

$$\int x \cdot \cos x\,dx = x \cdot \operatorname{sen}(x) - \int \operatorname{sen}(x)\,dx$$

d- Expansão de potências inteiras de soma não constantes

$$\int x(x+1)^2 dx - \int (x^3 + 2x^2 + x)\,dx$$

e- Distribuição de somas não constantes

$$\int 2x(x^2 + x)\,dx - \int (2x^3 + 2x^2)\,dx$$

f- polinomios

$$\int (P1(x) * P2(x))\,dx = \int P(x)\,dx$$

$$P(x) = P1(x) * P2(x)$$

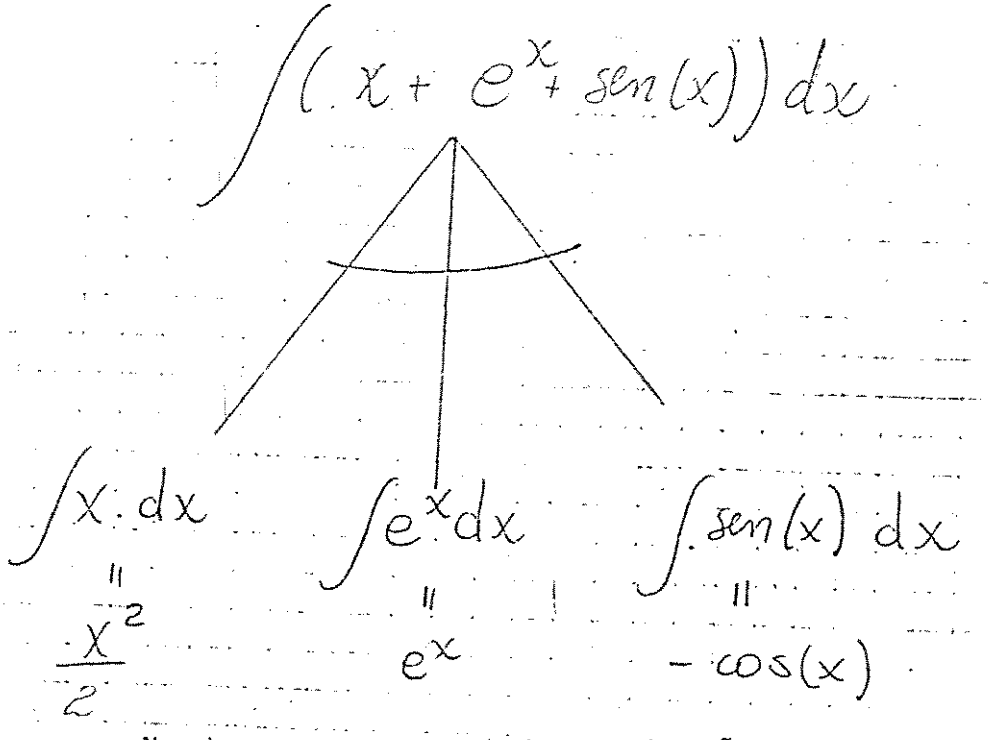
$$Pi \text{ — polinomio}$$

Quando uma transformação heurística ou uma transformação algorítmica é aplicada nas metas, elas geram novas metas, que por sua vez podem gerar outras metas, criando uma hierarquia. Esta é representada através de grafo e/ou.

As metas geradas a partir de um ancestral são chamadas sub-metas e são inter-relacionadas de duas maneiras:

a) Relacionamento do tipo E

Para o caso em que a expansão de uma meta gera duas ou mais sub-metas, para satisfazer a meta ancestral, deve-se satisfazer todas as sub-metas.

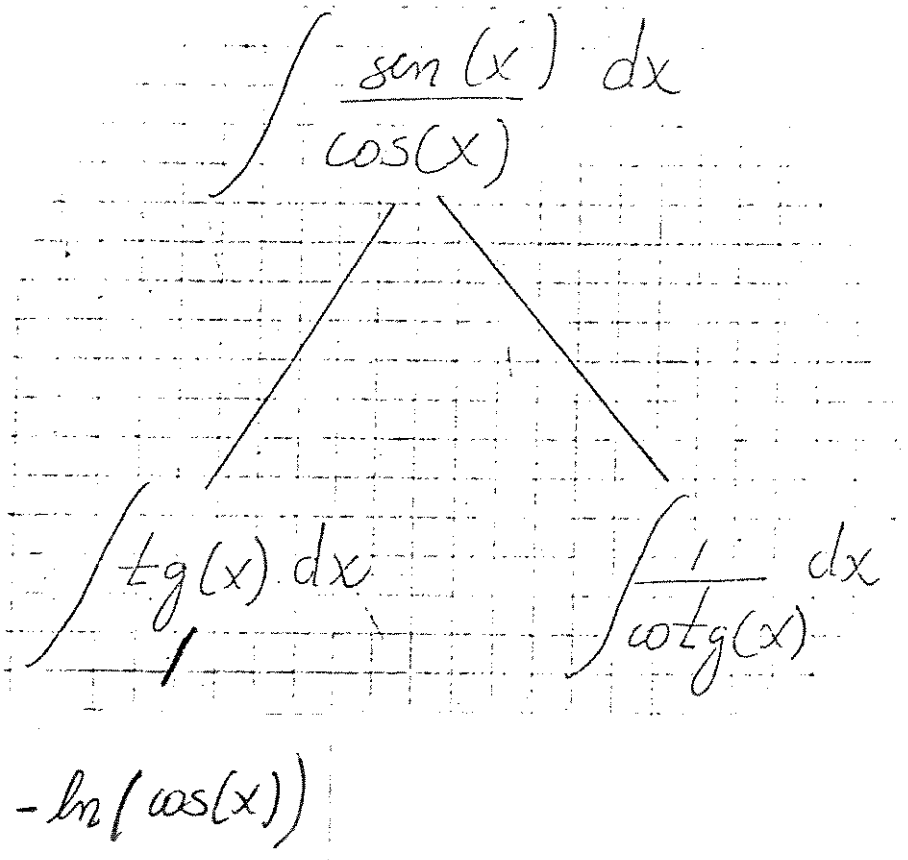


Neste caso, o integrando não se enquadrou na transformação imediata, mas sim nas transformações algorítmicas gerando novas metas (x.dx, .dx e sen(x).dx),Estas se enquadram nas transformações imediatas, conseguindo então satisfazer todas as sub-metas. Com isto atingiu-se todas as sub-metas e a satisfação da meta original.

b- Relacionamento tipo OU

Neste caso a satisfação do nó ancestral é obtida com a satisfação de apenas uma da sub-metas geradas.

Neste caso tenta-se inicialemente satisfazer a primeira sub-meta (menor função avaliadora). A sub-meta escolhida é a tg(x) que é integrada, obtendo como resultado -ln(cos(x)). Sendo esta meta satisfeita, satisfaz-se a meta original retornando -ln(cos(x)) como resultado da integral original.



Descrição da organização para expansão das metas

a- verifique se o integrando é do tipo de integral imediata. Caso positivo retorne o resultado.

b- Verifique se o integrando se enquadra em alguma transformação algorítmica. Caso positivo efetue a transformação algorítmica, gerando novas expressões menos complexas.

c- Tente como última alternativa a transformação heurística do integrando. Caso alguma transformação seja aplicável, execute-a gerando novas expressões menos complexas ou resultado da integral.

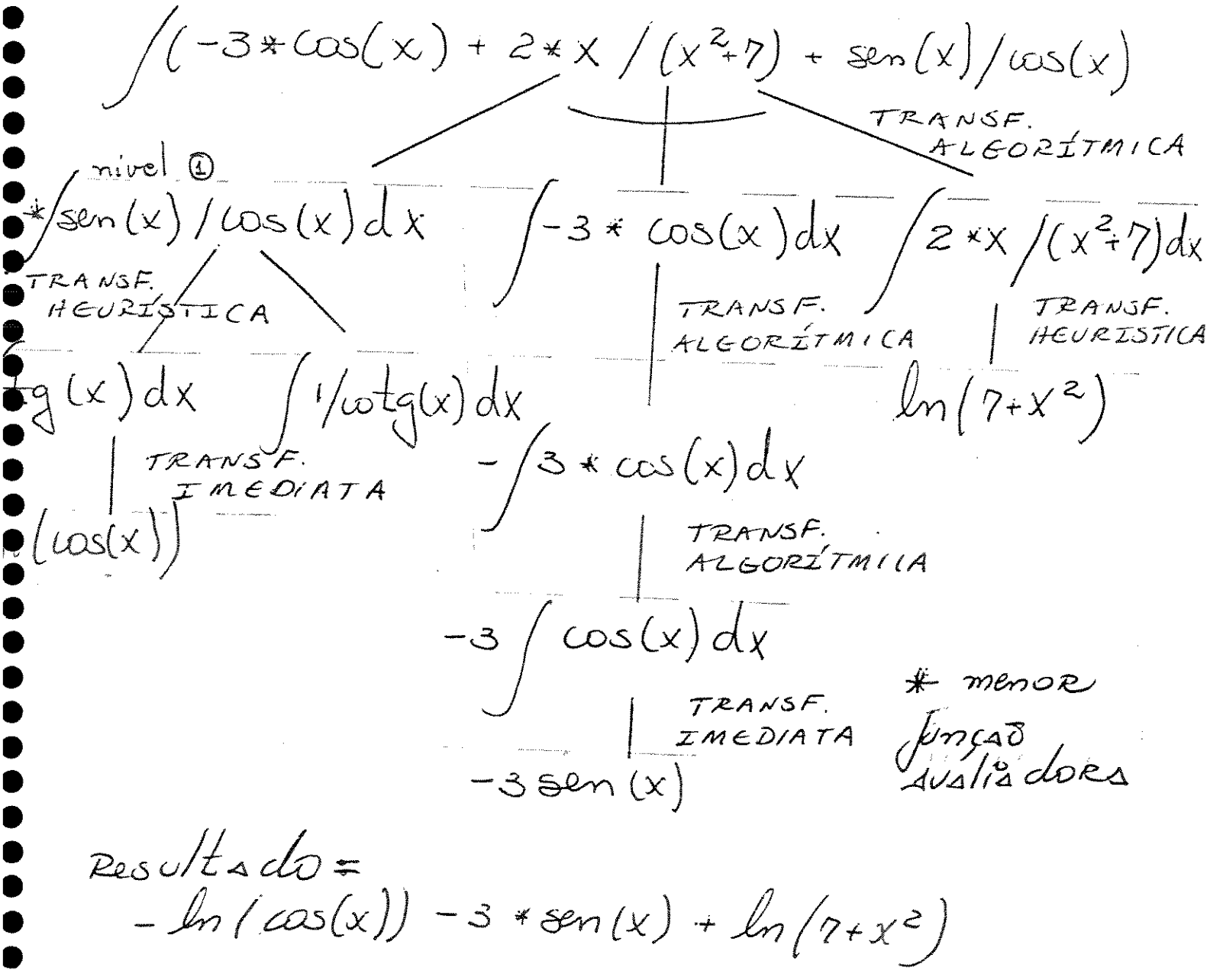
Satisfação da meta original

Sempre que uma meta é satisfeita, tenta-se satisfazer a meta original. Este procedimento é esquematizado como segue:

a- Se a meta satisfeita é a meta original, então o problema está solucionado.

b- caso contrário, tente satisfazer seu ancestral aplicando recursivamente este procedimento até atingir a meta original. Caso não seja satisfeita uma meta intermediária, tente outros caminhos alternativos.

Exemplo de aplicação:



Resultado =

$$-\ln(\cos(x)) - 3 * \operatorname{sen}(x) + \ln(7+x^2)$$

Uma característica importante deste algorítmo é que ele permite a expansão de novas regras imediatas, regras heurísticas e algorítmicas.

Aprendizado

O SICS incorpora opcionalmente, após a realização de integrais, a forma do integrando e sua solução, permitindo com isto incorporar um conhecimento adquirido a sua base de dados.

Integral definida

O cálculo da integral definida é realizado inicialmente através do cálculo da integral indefinida e posterior substituição dos limites de integração.

Exemplo:

$$\int_{\pi/6}^{\pi/3} \cos(x)\,dx = \left[\int \cos(x)\,dx\right]_{\pi/6}^{\pi/3} - \left[\int \cos(x)\,dx\right]_{\pi/6}^{\pi/3}$$

Inserção e/ou alteração de novas regras

Para alterar as regras existentes ou incluir novas regras na base de conhecimento acerca de derivadas e integrais no sistema SICS, o usuário deverá recorrer ao arquivo REGRAS.ARI . Nela deverá fazer as modificações desejadas de acordo com as regras ali presentes. Após esta modificação, o usuário deverá atualizar o sistema em relação a estas novas regras. Deverá recorrer ao apêndice C- Expandindo o interpretador - para maiores detalhes e realizar as operações ali descritas a partir do ítem 2.

Forma do integrando

1- Constante -> é o fator constante da integral
2- Integrando -> é a meta a ser satisfeita
3- Pai
4- Filhos
5- 0/1 -> Verifica se foi gerado pela expansão trigonométrica
6- Característica
7- Heurística a ser aplicada
8- Resultado da integral

Exemplo

```
[1,3*sen(x),x,,,0,[0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],[a,b,c],-3*cos(x)]
```

Exemplos:

?- avalia(int(cos(x),[x, pi/6, pi/3]),Resp).

Resp = 0.366

yes

?- avalia(int(3*x^4 + cos(x)^2, x), Resp).

Resp = 0,6x^5 + 0,5(x + sen(x).cos(x))

?- avalia(int(x*cos(x), x), Resp).

Resp = x*sen(x) + cos(x)

?- avalia(int([[x, 1], [1, 1]], x), Resp).

Resp = [[x^2/2, x], [x, x]]

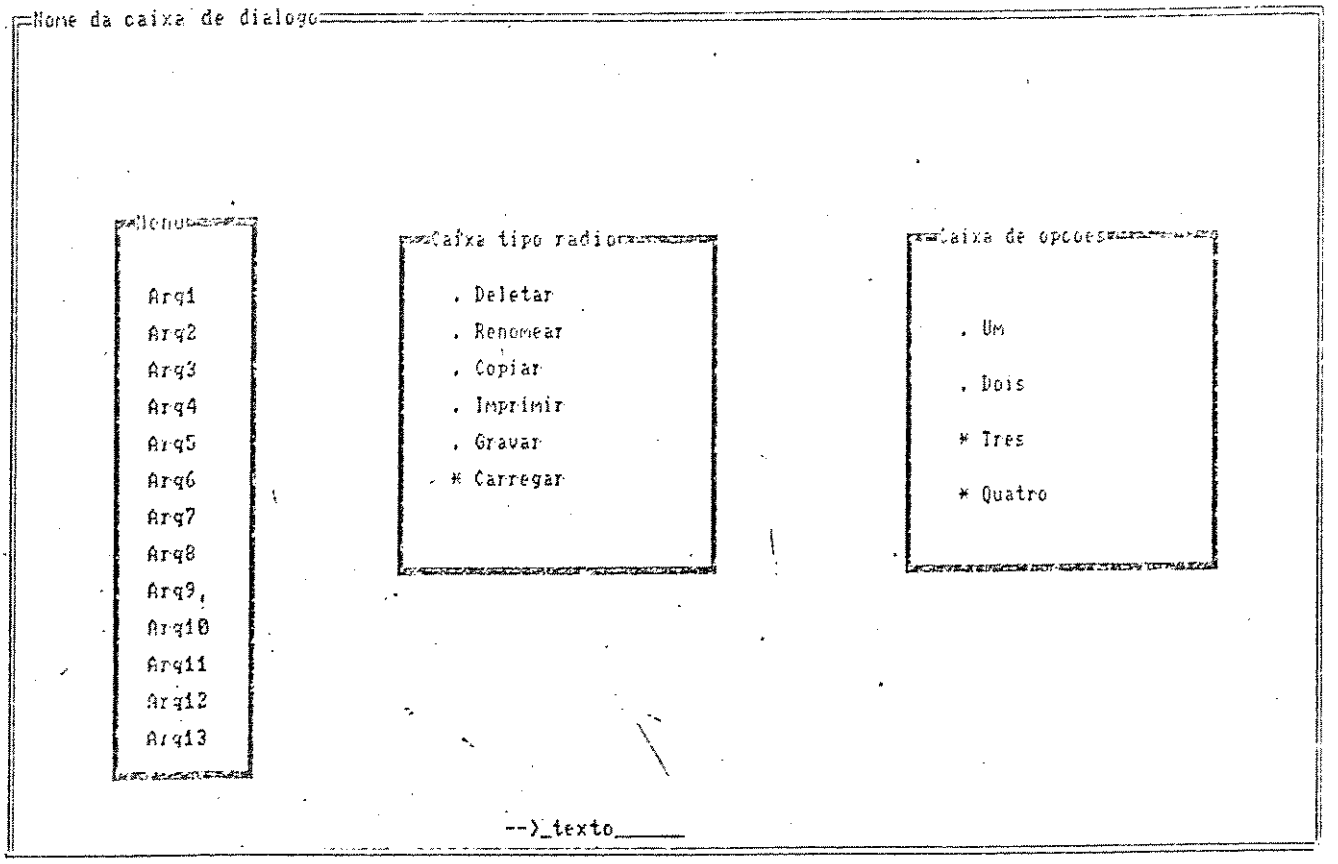
yes

III.4 - MÓDULO DE MANIPULAÇÃO DE TELA (M.M.T.)

ARQUIVOS: MOD_GRA.ARI

Este módulo é o responsável pela manipulação dos recursos de tela do sistema. Possui um conjunto de predicados que dispensam (mas não eliminam) a necessidade do usuário em conhecer os recursos de tela do micro utilizado.

Um exemplo típico de uma tela desenvolvida com os recursos deste módulo é:



O módulo contém vários arquivos e uma biblioteca com funções básicas de baixo nível que servem como base para os predicados de manipulação de tela presentes nos arquivos específicos. Os arquivos de manipulação de tela são passíveis de alteração para aplicações específicas do usuário.

Os recursos disponíveis são:

Menus -> Permite a construção de vários tipos de menus, sejam eles horizontais ou verticais.

Editor de linha -> Editor de linha muito útil na edição do interpretador PROLOG. Permite a utilização das teclas DEL, INS e setas de direção.

Caixa tipo rádio - É análoga à caixa de opções, exceto por permitir uma única opção.

Caixa de opções -> É uma caixa que contém uma lista circular de ítens a serem selecionados pelo usuário.

Tecla de confirmação -> É um controle que permite uma ação única.

Controles estáticos -> São ítens imutáveis que aparecem na caixa de diálogo. São usados para mostrar informações ao usuário.

Uma característica de grande importância é o fato deste estar quase inteiramente desenvolvido em PROLOG e a fonte estar disponível para o usuário para efetuar modificações e/ou implementações específicas.

Para maiores detalhes vide /ARITY - 1986/.

III.5 -MÓDULO DE FUNÇÕES PRIMITIVAS SIMBÓLICAS

Aqui estão presentes todas as funções referentes à lógica de combinadores/ BACKUS - 1978/ e outras muito úteis na elaboração de predicados a serem definidos pelo usuário.

O objetivo deste módulo foi o de reunir em um só arquivo todas as primitivas utilizadas pelo sistema SICS e outros igualmente importantes, para que o usuário disponha destes recursos adicionais e possa desenvolver os sistemas com maior facilidade.

As funções atualmente disponíveis para o usuário são:

abs(retorna número absoluto)
al(coloca elemento à esquerda da lista)
append(concatena listas)
aptodo(aplica um operador a todos elementos da lista)
ar(coloca o elemento à direita da lista)
avalia(avalia e simplifica expressões matemáticas)
bublesort(ordenação da lista pelo método quicksort)
comb(efetua combinação de listas)
confirma(menu de confirmação de ação)
consec(verifica se dois termos são consecutivos em uma lista)
consulte(armazena cláusulas de forma especial)
conta(efetua cálculo do número de elementos de uma lista).
denden(muda a flag DENDEN)
dl(distribui elemento à esquerda de cada elemento da lista)
dptodo(aplica um operador de derivada a todos elementos da lista)
dr(distribui elemento a direita de cada elemento da lista)
em_execução(emite mensagem de execução no centro da tela)
end_cro(pára o cronômetro previamente setado)
fat(cálculo do fatorial)
fib(calcula a função de Fibonacci)
final(mensagem de final de programa)
flags(verifica o status das flags de simplificação)
grav(muda flag de incorporação de resultados integrais)
intersecção(intersecção de duas listas)
last(último termo da lista)
lista_igual(Verifica a igualdade entre listas)
lista_reversa(retorna a lista revertida)

```
listp(verifica se termo é lista)
matriz(verifica se termo é matriz)
max(retorna o maior elemento de uma lista)
mdc(máximo divisor comum segundo algorítmo de
 Euclides)
member(verifica se um elemento pertence à lista)
mptodo(aplica um operador matricial a todos os
 elementos da lista)
notlistp(verifica se termo não é lista)
nulline(verifica se a lista é nula ou se seus
 elementos o são)
numnum(muda a flag NUMNUM)
nth(calcula a posição de um elemento da lista)
par(verifica se o número é par)
pause(pausa variável)
pegue_ate_n(pegue os primeiros "n" elementos da
 lista)
permutação(efetua a permutação dois a dois de uma
 lista)
primo(cálculo de números primos)
red(efetua uma operação de redução a todos
 elementos da lista)
rnd(retorna número randômico)
subtração(efetua subtração de duas listas)
start_cro(aciona o cronômetro)
subst(efetua a substituição de símbolos por
 valores numéricos e realiza a sua avaliação)
subconjunto(verifica se a lista é subconjunto de
 outra linha)
tira_elem(retira a primeira instância do elemento
 de uma lista)
tira_tudo(retira as todas instâncias de um
 elemento de uma lista)
trig(muda a flag trigonométrica)
união(união de duas listas)
Alguns exemplos de aplicação destas funções são:
fib(1,Resultado). % cálculo da função de Fibonacci
Resultado = 1

lista_reversa([a,b,c],Resultado). % reversão de lista
Resultado = [c,b,a]

tira_tudo(a,[a,b,c,a],Resultado). % tira elemento da lista
Resultado = [b,c]

last([a,b,c],Resultado). % retira último elemento da lista
Resposta = d

quicksort([1,2,3,5,4],Resultado). % ordena lista segundo o
Resultado = [1,2,3,4,5]      % método quicksort

bublesort([1,2,3,5,4],Resultado). % ordena lista segundo o
Resultado = [1,2,3,4,5]      % método bublesort
```

```
união([a,c],[m,n],Resultado). %  união de listas
Resultado = [a,c,m,n]

comb([1,2],[3,4],Resultado). % combinação de listas
Resultado = [1,2,3,4]

aptodo(max,[[1,2],[3,4],[2,4]],Resultado).%aplica uma função
Resultado = [2,4,4]      % a todos elementos de uma lista

member(a,[d,e,a,e]). % verifica se um termo é membro de
yes                  % uma lista
```

III.6 - MÓDULO DE CÁLCULO E SIMPLIFICAÇÃO (M.C.S)

ARQUIVOS: MOD_SIM.ARI

Este módulo é o responsável pelo cálculo e simplificação de expressões.

Este assunto já foi largamente discutido no ítem I.5 e é de grande complexidade. Os sistemas atuais são bem flexíveis quanto à simplificação do resultado.

Isto significa que o usuário deve direcionar a sequência de simplificação em uma dada expressão matemática. Por outro lado mostra o quanto ainda ainda estamos longe de um simplificador inteligente, cujos resultados práticos fatalmente virão da Inteligência Artificial.

No presente caso, optamos por uma simplificação automática reduzida, deixando para o usuário a opção de simplificação adicional orientada que possibilita obter uma resposta mais adequada às suas necessidades.

Este módulo é dividido em um procedimento principal e vários blocos de operadores, onde cada um deles executa o cálculo e simplificação com relação ao seu operador específico. No final de cada bloco de regras referente a uma determinada operação existe uma regra "catchall" que enquadra todos os casos não previstos, sem no entanto efetuar a simplificação.

No primeiro bloco temos o procedimento principal responsável pelo cálculo e simplificação das expressões matemáticas. O predicado "simplifica" consta de quatro cláusulas descritas como segue:

```
a1. simplifica(X,X):- atomo(X);numero(X).
a2. simplifica(X,Resp):-
        X =.. [Operador,X1,X2],
        simplifica(X1,X11),
        simplifica(X2,X22),
        aplica(Operador,[X11,X22],Resp).


a3. simplifica(X,Resp):-
        X =.. [Operador,X1],
        simplifica(X1,X11),
        aplica(Operador,X11,Resp).
a4. simplifica(X,X).
```

Sendo "aplica(Operador" responsável pelo tratamento diferenciado em relação a cada operador específico.

Por exemplo:

```
aplica(^,[X,0],1).  (bloco exponencial)

aplica(+,[0,X],X). (bloco adição)

aplica(-,X,-X). (bloco subtração)
```

O algorítmo de simplificação efetua o desmembramento recursivo das expressões segundo a precedência e associatividade dos operadores presentes até atingir um átomo ou número. A partir deste ponto, o retorno da recursão é efetuado fazendo sempre uma operação de cálculo ou simplificação por vez sobre os operandos, até obter a expressão simplificada.

A primeira cláusula é responsável pela parada da recursão, isto é, quando se atinge um átomo ou número.

A segunda cláusula efetua cálculo e simplificação sobre operadores binários.

A terceira cláusula realiza cálculo e simplificação sobre operadores unitários.

Na última cláusula recaem os casos não previstos, onde não é realizada operação alguma.

Seguidos do procedimento principal estão os blocos de operadores específicos.

O primeiro bloco contém uma série de operadores diversos, que realizam operações básicas sobre listas.

O segundo bloco é responsável pela derivada de expressões matemáticas.

O terceiro bloco efetua a integração de expressões matemáticas definidas no módulo de integrais.

O quarto bloco faz o tratamento da exponenciação possuindo várias cláusulas responsáveis para este fim. A função exponenciação exp(x) é representada por e^x.

O quinto bloco tem como objetivo o tratamento da divisão entre expressões elementares inteiras. Neste sistema ainda não foi implementada a manipulação de expressões racionais, mas deverá fazer parte de implementações futuras.

O sexto bloco faz o reconhecimento das funções trigonométricas sen,cos,tg e ln. Se o flag de simplificação trigonométrica estiver "setado", então o controle será desviado para o bloco de manipulação trigonométrica. Caso contrário, o sistema retorna à mesma função sem efetuar simplificação alguma.

No sétimo bloco temos o tratamento da subtração entre expressões elementares com algumas regras de fatoração. A propriedade comutativa é aplicada para este operador.

O oitavo bloco é responsável pelo tratamento da multiplicação entre expressões elementares. A propriedade comutativa é aplicada para este operador.

O nono bloco é composto de operações comuns à multiplicação e adição e também efetua aplicações das propriedades comutativa e distributiva.

O décimo bloco é comum à aritmética e efetua a operação entre dois valores numéricos inteiros ou flutuantes.

O último bloco é opcional e responsável pelo tratamento das funções trigonométricas sen,cos,tg e da ln em suas operações básicas de multiplicação, adição e subtração.

Para as opcionais temos disponíveis alguns flags para controle automático ou manual de simplificações. Por "default" o sistema realiza um número reduzido de simplificações. Caso o usuário sete algumas destas flags, então serão realizados outros tipos de simplificações.

Atualmente as flags disponíveis e seus valores "default" são:

trigonométrico -> Se setado, o sistema realiza simplificações trigonométricas.

numnum(0) -> Se setado, realiza simplificações no numerador.

denden(0) -> Se setado, realiza simplificações no denominador.

Estas duas últimas estão em fase de implementação.

III.7 - MÓDULO DE MANIPULACÃO DE MATRIZES (M.M.M.)

Arquivo: MOD_MAT.ARI

Neste bloco estão definidos os operadores para manipulação de matrizes. São eles:

- soma
  matriz + matriz

- subtração

  matriz - matriz

- multiplicação (escalar e vetorial)
  matriz * matriz

- divisão
  matriz / matriz

- exponencial de matrizes com expoente inteiro
  matriz ^ int

- transposta
  matriz #

- traço
  matriz mt

- derivada
  der(matriz,variável)

- integral indefinida
  int(matriz,variável)

- integral definida
  int(matriz,[variável,limiteinferior,limitesuperior)

O módulo compreende sub-grupos onde é feito o tratamento diferenciado em relação a cada operador. Nestes sub-grupos é feita uma tentativa de cálculo e simplificação das expressões. O processo é ánalogo ao encontrado no módulo de cálculo e simplificação, diferindo apenas com relação aos operadores específicos para manipulação de matrizes.

Antes de efetuar o cálculo de uma expressão matricial é feita a substituição dos operadores normais pelos operadores matriciais para efeito de otimização do sistema.

Nas operações em que as dimensões das matrizes devem ser verificadas, é feita uma análise antes de efetuar a avaliação e caso haja erro, a expressão não será avaliada e retornará à matriz envolvida de forma visível.

Para realizar as operacões entre matrizes temos uma poderosa ferramenta que é a lógica de combinadores /BACKUS 1978/, que oferece uma eficiente alternativa para a representacão, armazenamento, recuperação e manipulação de matrizes.

O módulo consiste em duas partes distintas: bloco de operadores e primitivas matriciais.

No bloco de operadores ocorre o tratamento diferenciado dos operadores disponíveis. São eles:

- soma
- subtração
- multiplicação
- exponenciação
- transposta
- traço
- derivada
- integral

Com relação às primitivas matriciais, atualmente dispomos dos seguintes predicados:

- deriva de linha
- integra_linha
- simplifica matriz
- matriz identidade
- gera identidade
- diagonal da matriz
- grava matriz
- recupera matriz
- monta matriz simétrica
- exibe elemento por linha
- exibe linha por linha
- inversa
- determinante
- solução de equações lineares

(Estes três últimos, resolvidos através do método de Gauss-Jordan).

Exemplo de utilização de algumas destas funções são:

```
?- avalia(der([[[x^2,x],[x,1]],x),Resposta).
Resposta = [[2*x,1].[1,0]]

?- det([[a,1],[1,d]],Resposta).
Resposta = a*d - 1

?- avalia([[a,b],[c,d]]+[[1,2],[3,4]]#,Resposta).
Resposta = [[1+a,3+b],[2+c,4+d]])

?- det([[a,b,c],[d,e,f],[g,h,i]],Inter),
subst(Inter,[[a,1],[b,2],[c,3],[d,4],[e,5],
                [f,6],[g,7],[h,8],[i,9]],Resposta).
Inter = i*(a*e) + g*(b*f) + h*(c*d) -c*(g*e) - h*(a*f) - i*(d*b)
Resposta = 0

?- solução([[2,1],[1,2]],[[0],[1]],Resposta).
Resposta = [[-0.33,0.66]]

?- avalia([[a,b],[1,2]] ^ -1,Resposta).
Resposta = [[1/a - b/(a^2*(2-b/a)),
-b/(a*(2-b/a))],[-1/(a*(2-b/a)),1/(2-b/a)]]
```

Para informações mais detalhadas sobre os predicados deste módulo, vide apêndice D - Descrição das funções implementadas.

III.8 - MÓDULO DE AJUDA AO USUÁRIO (M.A.U.)

Este módulo tem por finalidade prestar ajuda on-line ao usuário do sistema SICS. Esta ajuda se resume na explicação sucinta , do formato dos comandos presentes no interpretador PROLOG, bem como dos comandos desenvolvidos especialmentes para o sistema SICS e a sua localização no livro de referência ou programa fonte para informações mais detalhadas.

Uma das vantagens deste módulo é a de eliminar o tempo de consulta aos livros texto, acerca dos comandos disponíveis no interpretador expandido.

Este módulo não está disponível para o usuário, dado o grande espaço de memória requerido, portanto o comando solicitado é carregado quando a ajuda for solicitada.

Para a introdução de novos comandos criados pelo usuário, deve-se inserir os mesmos no arquivo fonte "help.ari" da mesma forma dos que ali estão presentes.

Para invocar a ajuda manualmente deve-se digitar:

all_primi -> exibe todos os predicados disponíveis (^s - congela tela ^q - continua)

primi -> para exibir um predicado específico. Não é necessário digitar o nome do predicado completo.

Podemos invocar a ajuda automaticamente com a interface especializada.

Exemplo:

?- primi.

Comando ?

ab <ret> (Não é necessário digitar o comando por completo)

abort

aborta o programa

Pag 15

A seguir o usuário deverá digitar:

- <return> para finalizar a operação
- <espaço> para efetuar a busca em outro comando que contenha os mesmos caracteres digitados.

CAPÍTULO IV - INTERFACE ESPECIALIZADA SICS

DESCRIÇÃO

Com a crescente popularização dos sistemas simbólicos, surgiu a necessidade de uma interface especializada em um sistema interativo que viesse facilitar a operação por parte do usuário .

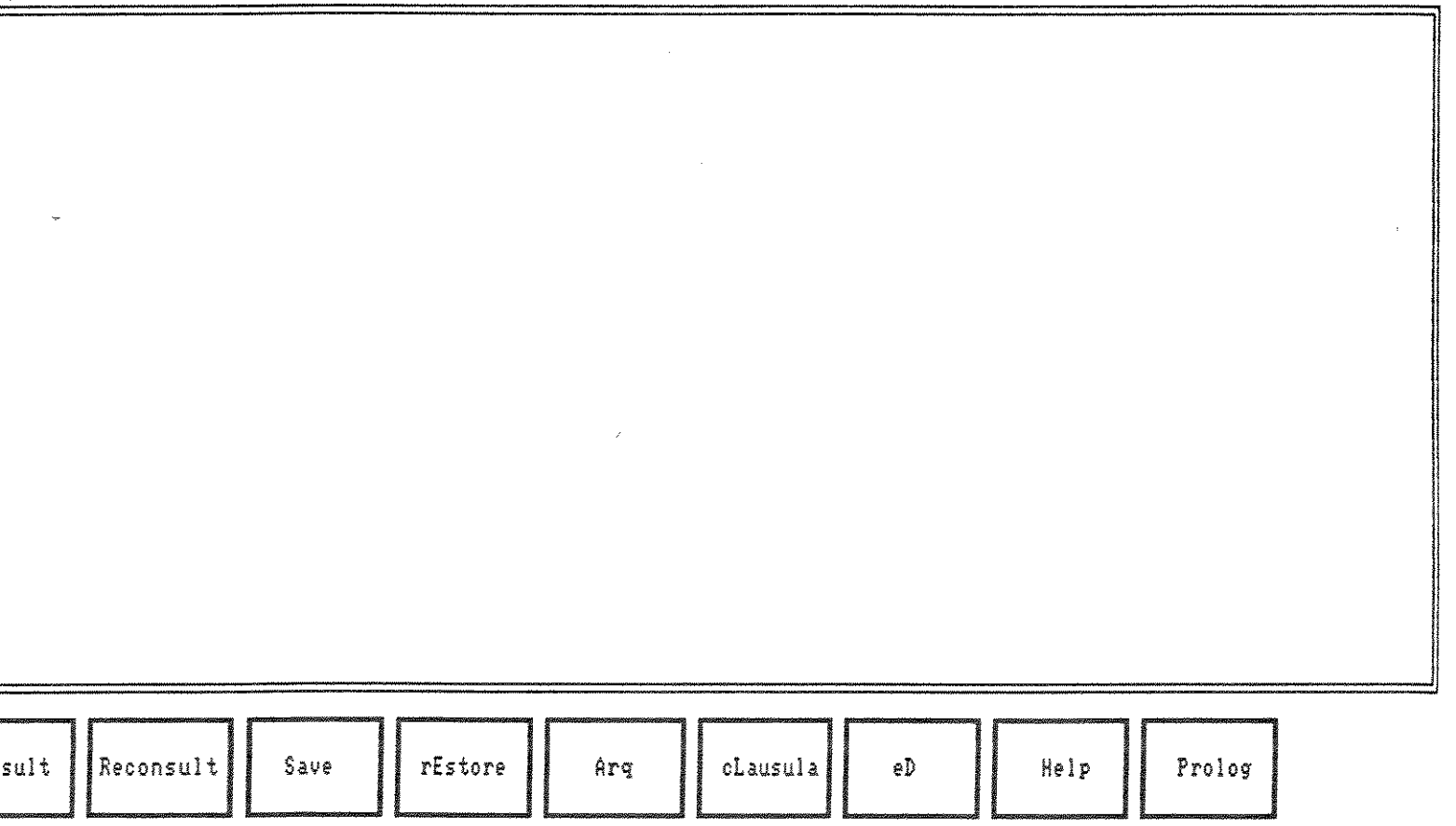
Uma das preocupações durante o desenvolvimento do sistema SICS, foi o de tornar o sistema, bastante acessível ao usuários da linguagem PROLOG. Existem algumas dificuldades por parte dos programadores de linguagens tradicionais com relação ao estilo declarativo da linguagem PROLOG. O objetivo desta interface foi o de reunir os comandos mais utilizados durante a elaboração de um programa em PROLOG e colocá-los a disposição do usuário de uma forma bastante amigável. Esta forma foi obtida com a elaboração de janelas e menus, onde são executados diversos comandos com a simples seleção de algumas teclas.

Esta interface foi especialmente projetada para o desenvolvimento de sistemas, utilizando a linguagem PROLOG. A mesma possui características da estrutura da linguagem através de operações sobre estruturas prolog ao invés de caracteres ou palavras. Um exemplo desta aplicação são as operações sobre cláusulas.

A utilização desta interface, embora tenha sido desenvolvida para utilização do sistema SICS, tem uma abrangência bastante ampla na área do aprendizado da linguagem PROLOG. Foi desenvolvida uma versão desta interface para utilização em micros que não disponham de "winchester".

Para ativar a interface digite "tex." a partir do interpretador prolog expandido.

A tela principal da interface é a seguinte:



Como podemos observar, os comandos disponíveis são os mais utilizados durante a etapa de desenvolvimento e por isto facilitam muito a tarefa do projetista.

Os comandos estão esquematizados de acordo com a estrutura a seguir:

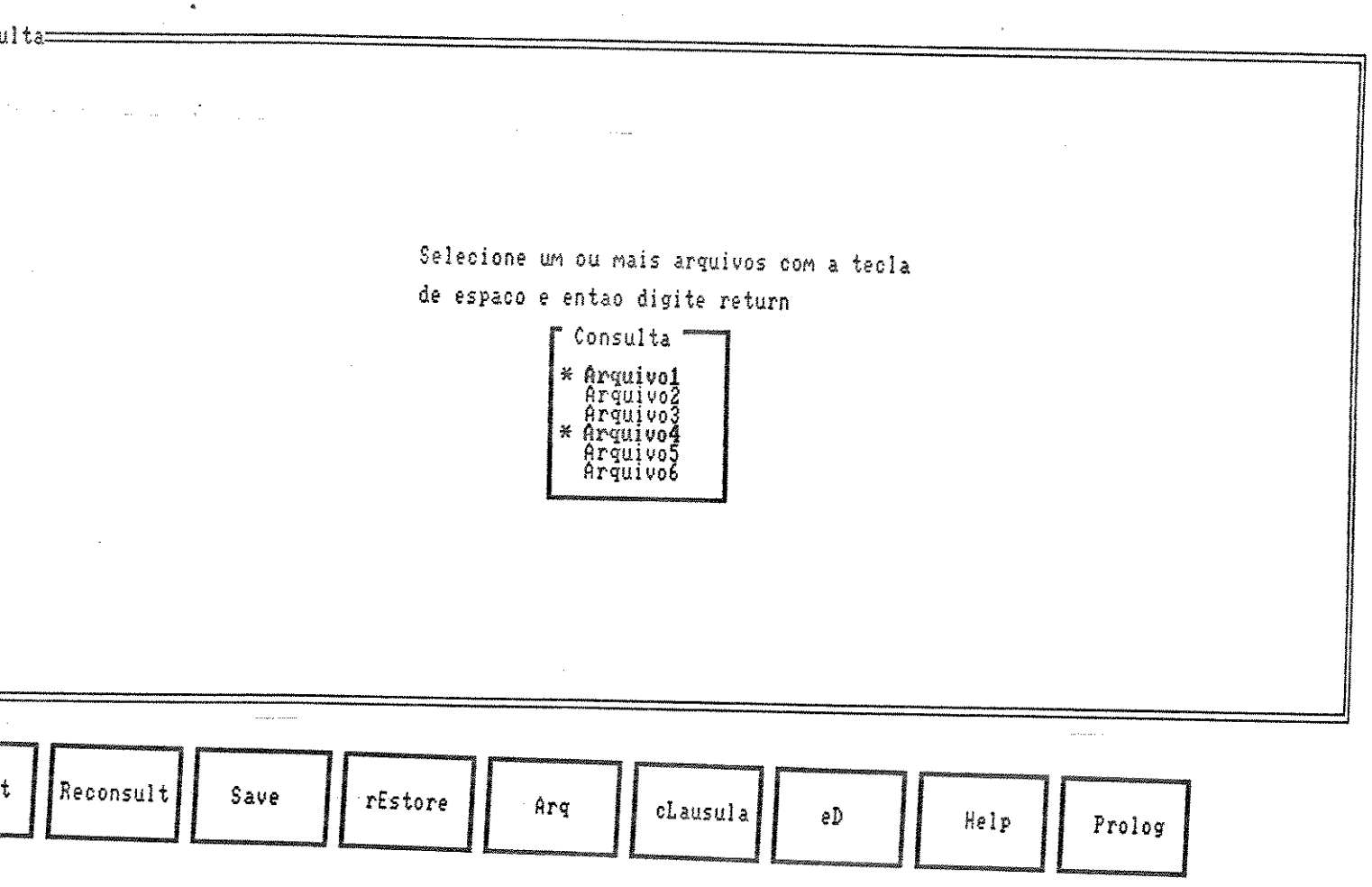
- Consulte
- Reconsulte
- Save
- rEstore
- Arq ---------- Coplar
- cláusuLa---------- Cláusula - Renomear
- Ne---------- Editor - Predicado - dIr
- Help-----Listar pred -Recons - Spy - Ari
- Prolog -Procurar pred - Nspy - discO
- spT - Bak
- tRace - Fonte
- notracE - Save
- Insere---Inicio
- retirAr -Final

Para selecionar uma das opções, o usuário utiliza as setas para posicionar o cursor na opção e em seguida deve digitar return. Outra maneira de selecionar é digitar a letra escrita em maiúscula na opção desejada.

Analisaremos agora cada opção e seus respectivos menus.

Consulte

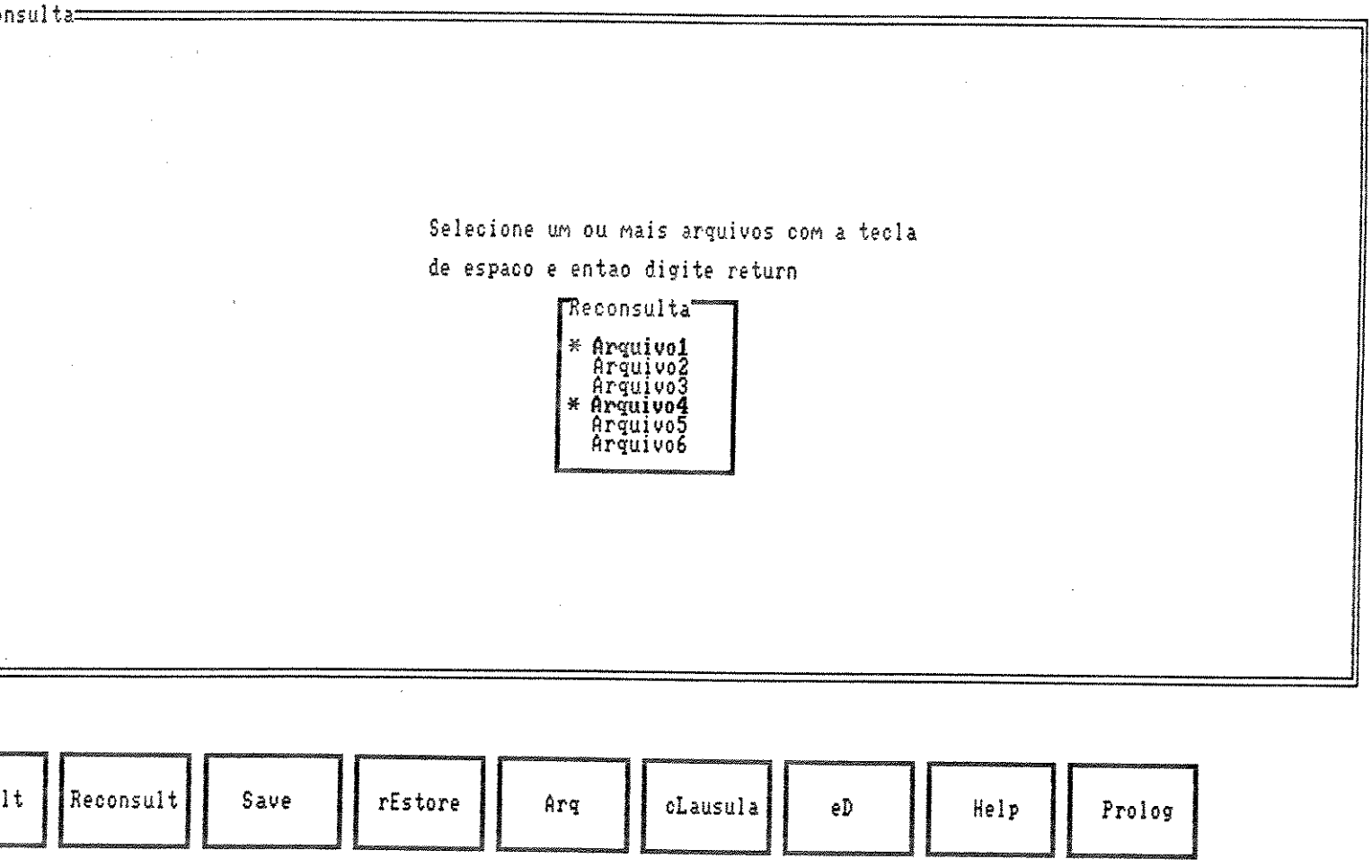
Foi desenvolvida uma tela com as características mostradas a seguir:



O usuário terá uma lista com os arquivos presentes no diretório corrente com extensão.ari e com o auxílio das setas e com a tecla de espaço poderá ativar ou desativar a seleção de arquivo(s) para consulta. Com as setas, a tecla home e a tecla end podemos percorrer toda a lista de arquivos. A posição corrente da lista é aquela em que o nome do arquivo aparece com uma cor mais intensa que a dos demais. Apertando a tecla de espaço aparecerá(ou desaparecerá) do lado esquerdo do arquivo um símbolo indicando sua seleção(*). Uma vez selecionado o(s) arquivo(s) (devem estar marcados com o símbolo (*)) desejados, o usuário deverá apertar a tecla return. Após cada consulta com sucesso, o sistema emite uma mensagem certificando o sucesso da operação. Ao final da operação teremos uma lista de arquivos consultados e eventualmente, dos arquivos que apresentarem problemas durante a consulta. Para desistir da operação, tecle ESC.

Reconsulte

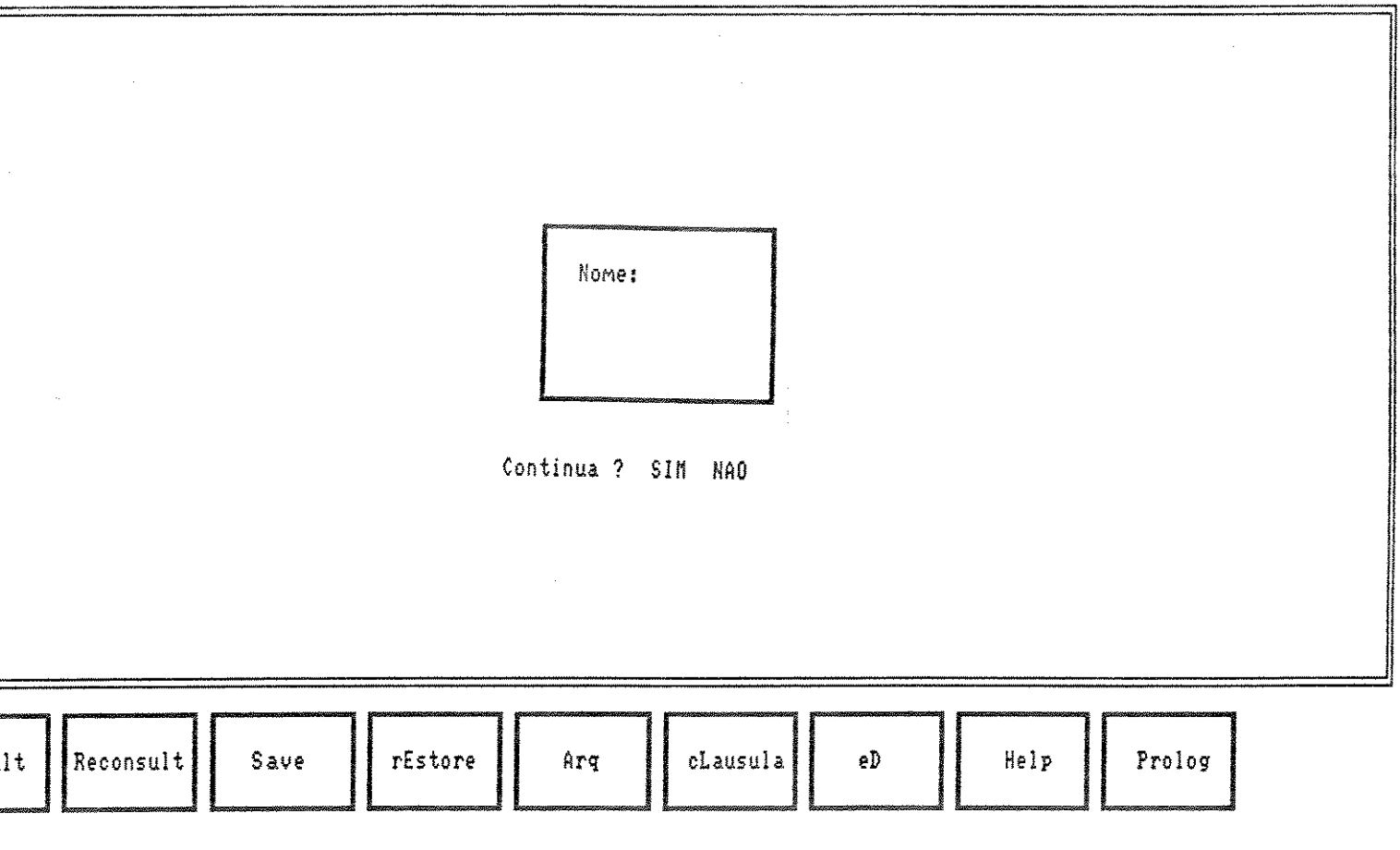
Foi desenvolvida uma tela com as características mostradas a seguir:



O usuário terá uma lista com os arquivos presentes na base de dados com extensão .ari e com o auxílio das setas e com a tecla de espaço poderá ativar ou desativar a seleção de arquivos para reconsulta. Com as setas, a tecla home e a tecla end podemos percorrer toda a lista de arquivos. A posição corrente da lista é aquela em que o nome do arquivo aparece com uma cor mais intensa que a dos demais. Apertando a tecla de espaço, aparecerá (ou desaparecerá) do lado esquerdo do arquivo um símbolo indicando sua seleção(*). Uma vez selecionado o(s) arquivo(s) (devem estar marcados com o símbolo (*)) desejados, o usuário deverá apertar a tecla return. Após cada consulta com sucesso o sistema emite uma mensagem certificando o sucesso da operação. Ao final da operação teremos uma lista de arquivos reconsultados e eventualmente, dos arquivos que apresentarem problemas durante a reconsulta. Para desistir da operação tecle ESC.

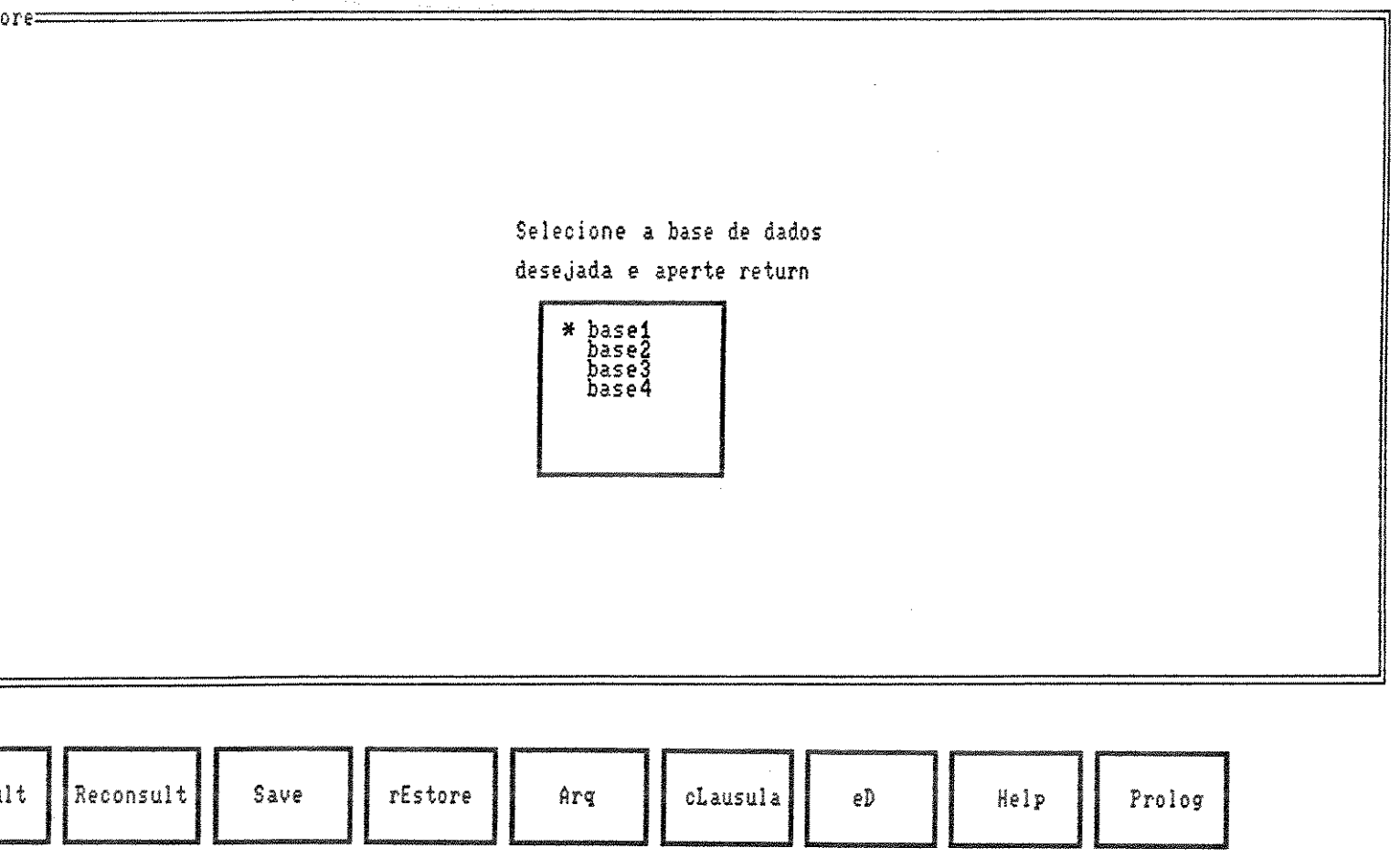
Save

Neste caso o sistema pedirá um nome para a base de dados a ser salva. Antes do usuário do sistema requisitar o nome da base a ser salva, é solicitada a confirmação da ação desejada.



Restore

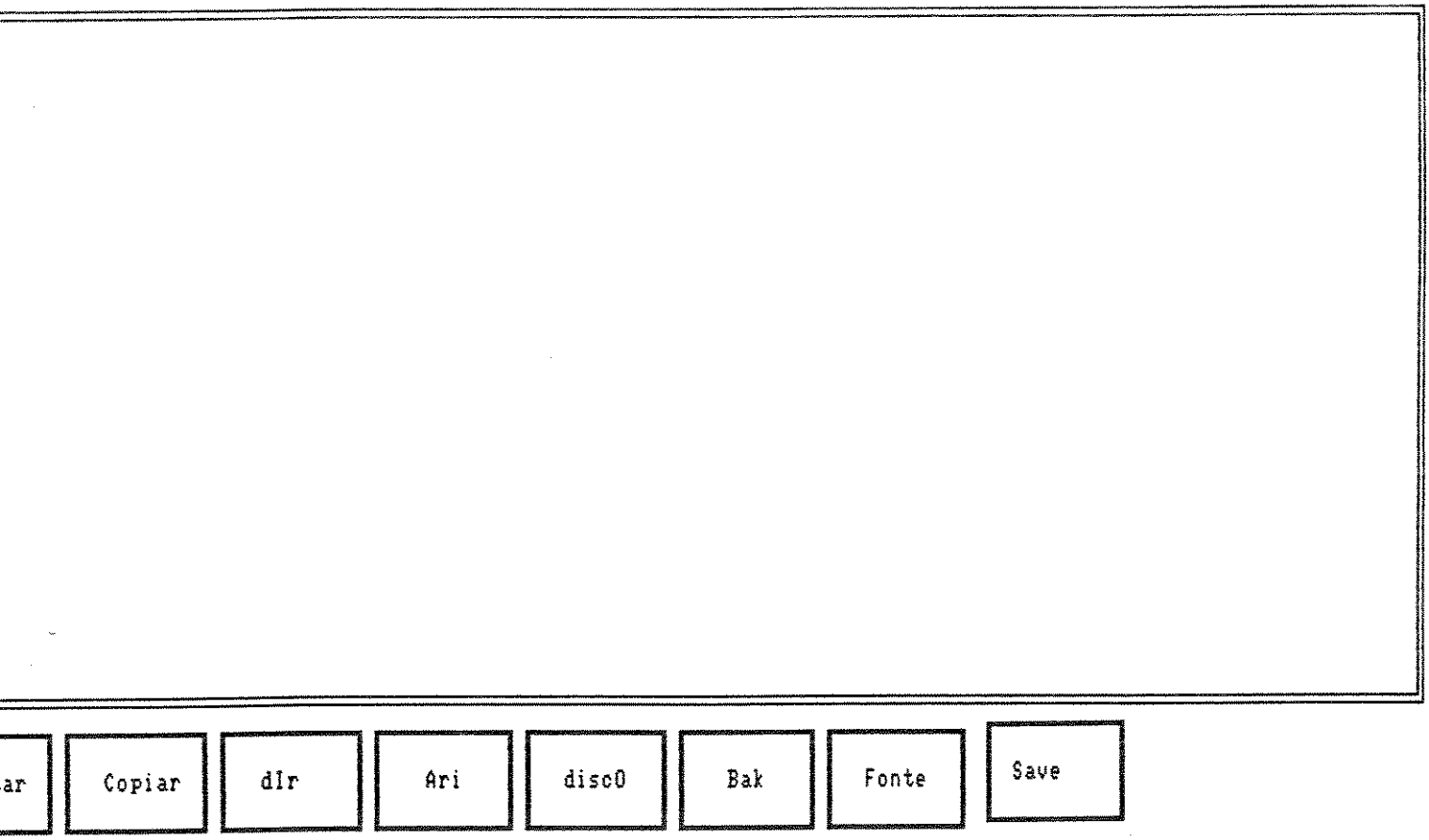
O usuário terá uma lista das bases de dados salvas através do comando "save" e assim poderá selecionar uma base desejada. Para desistir da operação tecle ESC.Uma vez carregada a nova base de dados, a interface retorna ao menu principal.



Arq

Aqui o usuário poderá manipular os arquivos presentes no diretório corrente. Ao selecionar esta opção aparecerá outro menu com as seguintes opções:

- Deletar
- Copiar
- Renomear
- dir
- Ari
- disc0
- Bak
- Fonte
- Save



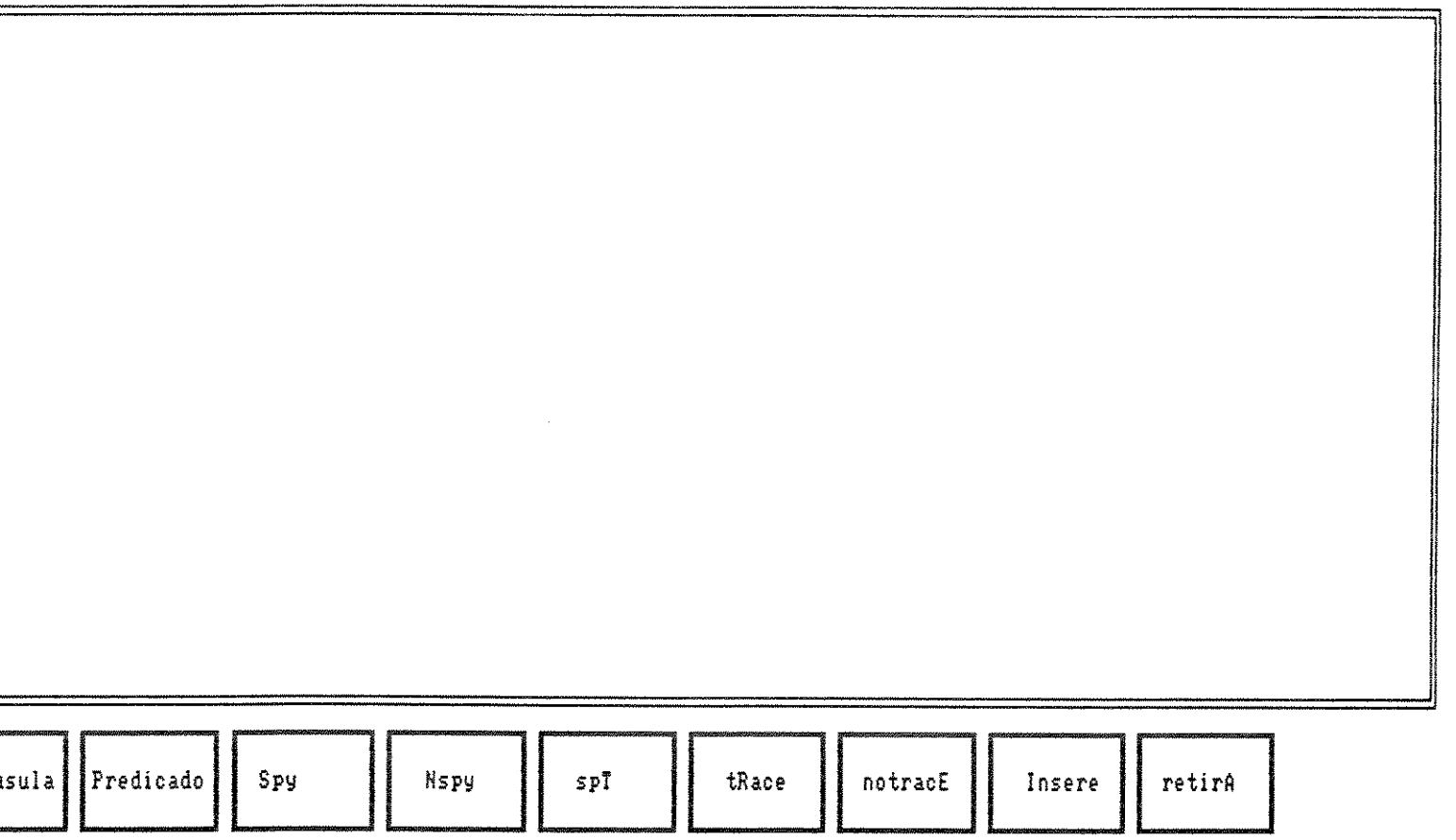
Escolhendo algumas destas opções, o usuário terá que digitar o que for solicitado na ação correspondente para executar a operação desejada. Para desistir, aperte a tecla ESC e voltará o menu principal. Eis a função de cada uma delas:

- Delete: Utilizado para deletar arquivos. Poderá deletar arquivos de qualquer drive. O formato é o seguinte: [drive:]Arquivo.ext
- Copie: Utilitário para cópia de arquivos. Deve ser fornecido o arquivo fonte e o arquivo destino. O formato do nome é o mesmo de Delete.
- Renomeie: Utilizado para renomear arquivos. O usuário deverá digitar o nome velho e o novo. Antes porém deverá confirmar a ação desejada.
- dir: Lista o diretório corrente. Para listar outro drive, deve-se inicialmente mudar o drive corrente e posteriormente dar o dir.
- Ari: Lista o diretório dos arquivos com extensão ari do drive corrente.
- disc0: Permite a mudança do drive corrente. O usuário deverá escolher o novo drive das opções existentes.
- Bak: Permite deletar todos os arquivos com extensão bak do diretório do drive corrente.
- Fonte: Permite visualizar a fonte de arquivos criados pelo usuário. Para arquivos com extensão ari no diretório do drive corrente não é necessário colocar a extensão. Para os outros casos deverá ser colocado o nome e, opcionalmente, o drive entre $$.
- Save: É uma maneira de armazenar a base de dados na forma de cláusula em um arquivo. O usuário deverá fornecer o nome do arquivo assim que for solicitado. Antes de colocar o nome, o usuário deverá confirmar a opção desejada.

Cláusula

Este é o comando para manipulação de cláusulas. temos os seguintes comandos disponíveis:

- Cláusula
- Predicado
- Spy
- Nspy
- spT
- traCe
- notracE
- Insere
- retirA



Sendo:

Cláusula: Listar toda a base de dados

Predicado: Listar predicado específico. O usuário deverá confirmar o desejo da operação antes de colocar o nome do predicado.

Spy: É o tradicional spy. O usuário deverá confirmar o desejo da operação antes de colocar o nome do predicado seguido de return.

Nspy: É a desativação do spy. Funciona de maneira análoga.

spT: É o spy normal com o comando trace sendo executado após o spy.

tRace: É o trace normal .

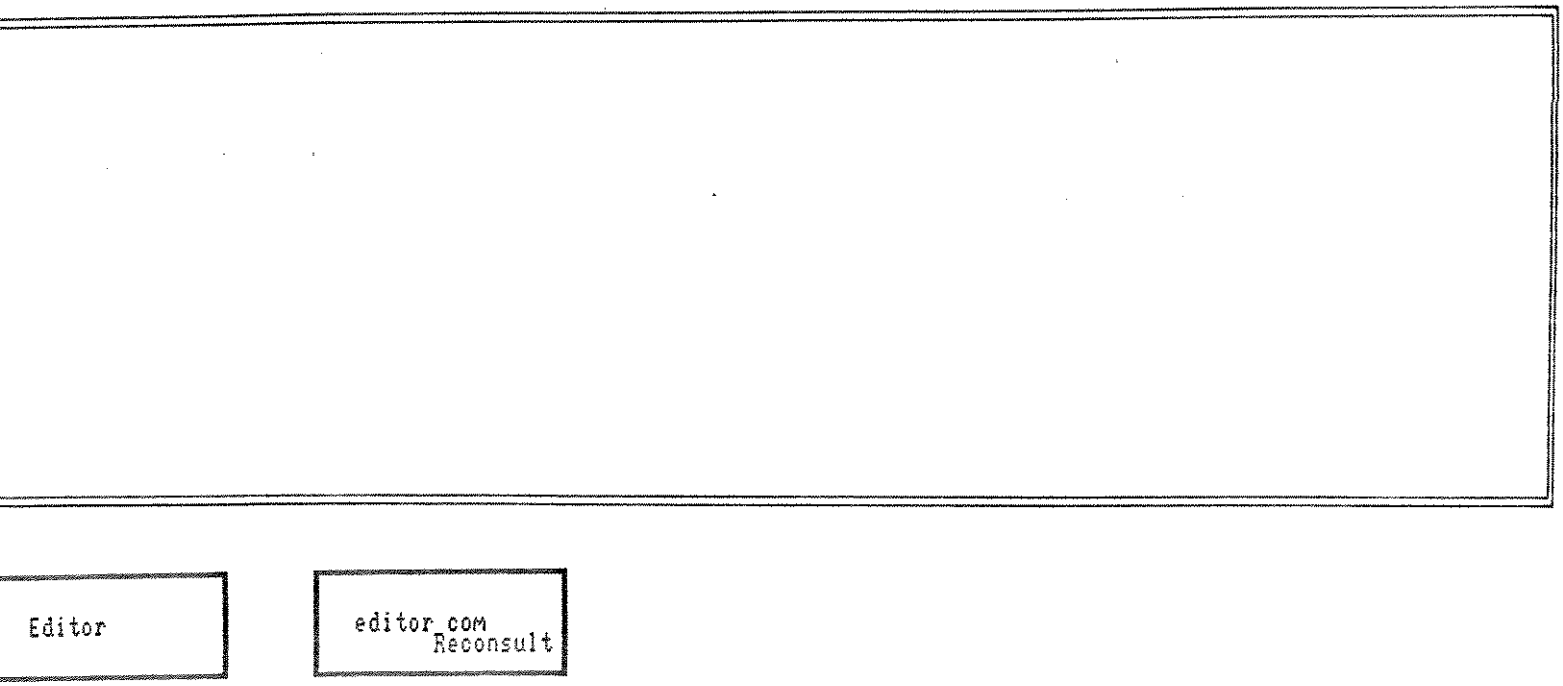
notracE: É a desativação do trace.

Insere: É uma maneira simples e direta de inserção de cláusulas simples na base de dados. Antes, o usuário deverá indicar se deseja que a cláusula seja colocada no início ou final do predicado. Não deve ser colocado ponto final na cláusula. Para desistir selecione "não".

retirA: É um comando para retirar cláusulas da base de dados. O usuário deverá fornecer o predicado e o sistema fornecerá a primeira cláusula com aquele predicado. Será solicitada a confirmação do procedimento de retirada da cláusula da base de dados. Desistindo, o sistema mostrará a próxima cláusula deste predicado.

eD

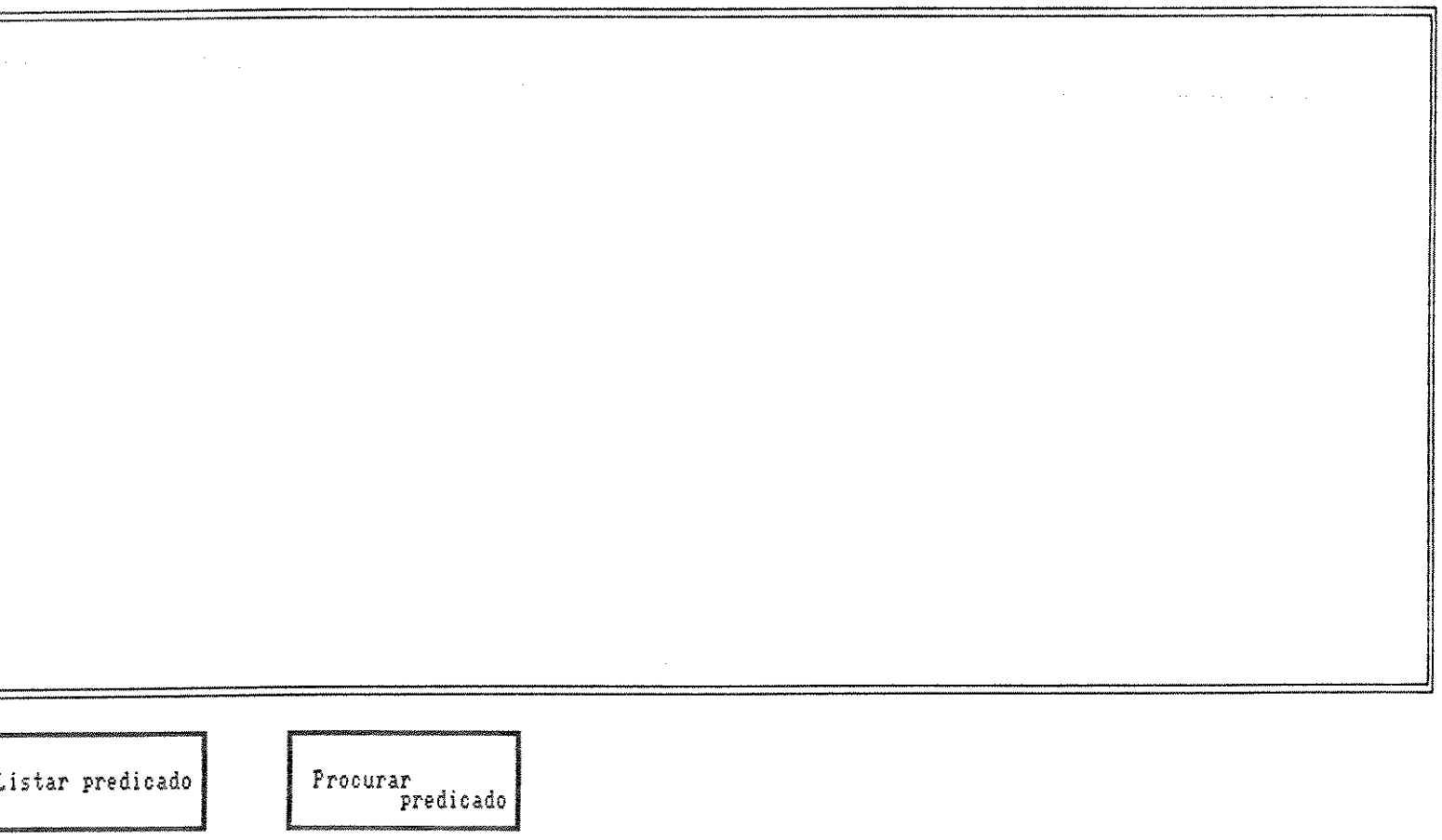
Esta tecla permite invocar o processador de texto temporariamente, de duas maneiras distintas. Ao selecionar esta opção aparecerá:



Se o usuário confirmar a escolha do editor com reconsulta , o arquivo a ser editado será reconsultado automaticamente após a saída do editor. Se o usuário não desejar reconsultar automaticamente um arquivo, deve escolher a opção editor e o sistema irá para o editor, sem que na sua volta para o PROLOG reconsulte arquivo algum. O sistema solicitará o nome do arquivo a ser editado antes de efetuar a operação. Caso o arquivo seja o último editado pelo editor, deve-se digitar apenas return sem nome algum.

Help

Após selecionar a opção HELP aparecerá o seguinte menu:



O usuário poderá selecionar uma das duas opções ou então digitar ESC para retornar ao menu principal.

. Listar comandos -> Esta opção permite listar os comandos disponíveis no interpretador expandido.

. Procurar comando -> Permite ao usuário ter uma breve ajuda a respeito de um determinado comando. O usuário deverá digitar o nome do comando desejado seguido de return e aparecerão as informações desejadas. Não é necessário digitar todo o comando por extenso, basta algumas letras. Neste caso, o sistema procurará o primeiro comando que contenha aquelas letras. Se o usuário desejar outros comandos com o nome digitado deverá apertar a tecla de espaço, caso contrário digitar <return>.

PROLOG

Selecionando esta opção o sistema voltará para o comando do interpretador .

CAPÍTULO V

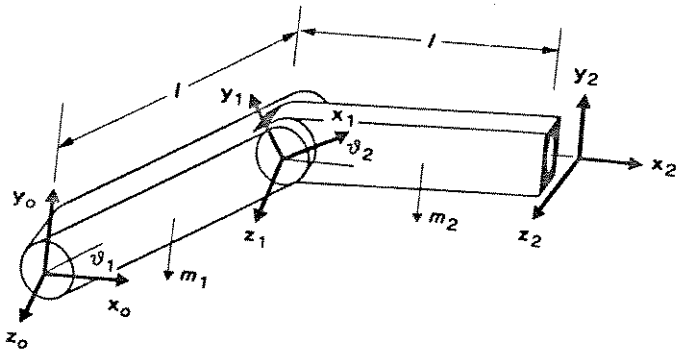
APLICAÇÕES DO SICS

V.1 - UMA APLICAÇÃO DO SICS

Esta aplicação tem por objetivo, dar uma visão de uma utilização do Sistema Integrado de Computação Simbólica (SICS).

A presente aplicação se enquadra no campo da Robótica, mais precisamente no cálculo da equação dinâmica de robô pelo método de Lagrange-Euler /LEE - 1983/ LOPES, PEREIRA e ALVES - 1986/, onde desenvolveremos o cálculo da equação que representa o efeito do momento de junta.

O exemplo aqui descrito é um braço de um robô de dois graus de liberdade como mostra a figura a seguir:



As ferramentas presentes no sistema fazem com que a tarefa de implementação desta equação, se torne extremamente facilitada.

Segundo /LEE - 1983/ a equação que define o efeito do momento de uma determinada junta em todos pontos de um determinado braço de um robô é:

$$U_{ij}^{\circ} \triangleq \frac{\partial A_0^i}{\partial v_j} \qquad (I)$$

sendo

$$A_{i-1}^{i} = \begin{bmatrix} \cos v_i & -\cos\alpha_i \operatorname{sen} v_i & \operatorname{sen}\alpha_i \operatorname{sen} v_i & a_i \cos v_i \\ \operatorname{sen} v_i & \cos\alpha_i \cos v_i & -\operatorname{sen}\alpha_i \cos v_i & a_i \operatorname{sen} v_i \\ 0 & \operatorname{sen}\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & l \end{bmatrix}$$

v_i - variáveis de junta

$d_i, \alpha_i^{\circ}, a_i$ - parâmetros do braço

A equação (1) pode ser facilmente implementada através do procedimento:

```
momento_de_junta([],[],RESP):- recupera(mat,RESP),!.
momento_de_junta([MTH|OMTH],[VJ|OVJ],RESP):-
    dr(VJ,[MTH|OMTH],INTER),
    mptodo(der_mat,INTER,MATRIZ),
    grava_matriz(mat,MATRIZ,_),
    momento_de_junta(OMTH,OVJ,RESP).
```

Para invocar este procedimento digitamos:

```
?-momento_de_junta(Lista_de_A,Lista_de_V,RESPOSTA).
```

Para um exemplo de um robô com dois graus de liberdade teremos:

```
Lista_de_A = [A ,A ]

Lista_de_V = [v1,v2]
```

E obteremos como RESPOSTA:

$$U_{11} = \frac{\partial A_0^1}{\partial v_1} \qquad U_{21} = \frac{\partial A_0^2}{\partial v_1} \qquad U_{22} = \frac{\partial A_0^2}{\partial v_2}$$

Após efetuar o cálculo simbolicamente, poderemos efetuar a substituição das variáveis por valores numéricos através do predicado sub:

```
subst(Resp_simbólica,Lista_variáveis_e_valores,Resp_numérica).
```

Para maiores detalhes a respeito das funções primitivas aqui utilizadas, vide apêndice D - Descrição das funções implementadas.

Este exemplo está implementado para um robô de dois graus de liberdade no arquivo "exemplo.ari" anexo ao sistema SICS. Para executá-lo, carregue-o no interpretador PROLOG e ative-o com o seguinte comando:

```
?- ativa_exemplo. <return>
```

V.2 - ALGUMAS EXPERIÊNCIAS REALIZADAS

COMPUTAÇÃO ALGÉBRICA NA EDUCAÇÃO

Surgem como opção de ensino, uma vez que permitem os estudantes concentrarem seus esforços em conceitos fundamentais e legar os trabalhos mecânicos ao computador. Outras vanta-

gens destes sistemas seriam:

- Dado que o computador permite executar operações rápidas e complexas, os limites do ensino básicos poderiam ser expandidos.
- O modo "trace" permitiria ao aluno analisar cada etapa do processo ao invés de somente os resultados.
- O armazenamento de uma sessão de trabalho para posterior análise
- O aluno poderia orientar o computador com relação aos procedimentos a serem executados, avaliando exaustivamente todas as suas idéias e conseguir um refinamento automático de seus conceitos.
- O recurso de questionamento para visualizar todas as etapas intermediárias
- Permite também a implementação de novos algorítmos e assim aumentar a abrangência do sistema.

Algumas destas idéias foram implementadas no sistema SICS. São elas: A simplificação orientada, o modo "trace" e o sistema aberto. Outras idéias estão sendo desenvolvidas e acredito que com a popularização destes sistemas algébricos, alcançaremos ótimos resultados na área de ensino.

Simplificação orientada: No âmbito da simplificação dispomos de alguns flags passíveis de alteração por parte do operador. Estas flags têm o objetivo de direcionar a simplificação em uma dada expressão matemática. Esta implementação objetiva mostrar a viabilização deste tipo de experiência, portanto está em aberto a implementação de novas flags para acompanhar a evolução do sistema.

As flags implementadas são:

TRIGONOMÉTRICA -> Efetua simplificações trigonométricas

NUMNUM E DENDEN -> São flags de simplificações de numerador e denominador. Estas flags estão em fase de implementação.

Existe um predicado que mostra os valores correntes das flags de simplificação.

O **modo trace** disponível é inerente à linguagem prolog, portanto já esta implementado. Este trace é diferente das demais linguagens pelo fato de o usuário visualizar todas as etapas de execução e seus valores, bem como alterar valores durante o decorrer do trace.

A **função log** que grava uma sessão de trabalho para posterior análise está implementada sob a forma de um editor de linha. Toda informação mostrada na tela será armazenada no arquivo "prolog.log".

O **recurso de questionamento** para o prolog também foi implementado, possibilitando ao usuário vizualizar todas as etapas realizadas até a solução encontrada.

E finalmente o **sistema é aberto.** Isto significa que qualquer pessoa que deseje particularizar, implementar, alterar o SICS, poderá fazê-lo sem muitos problemas. A documentação que segue nos anexos juntamente com a fonte do mesmo possibilita este recurso.

OTIMIZAÇÃO NO ARMAZENAMENTO DE INFORMAÇÕES

Para otimizar o acesso à informação foi desenvolvida uma função que carrega os programas de forma especial e permite a sua recuperação de forma eficiente. O ganho desta função em relação à velocidade de acesso é bastante significante.

consulte que tem como função o carregamento de arquivos para a base de dados do prolog. Este predicado é semelhante ao "consult" do interpretador prolog com diferença no tratamento diferenciado em relação a predicados específicos. O armazenamento destes predicados ocorre na forma de hash-table. Este predicado está presente no arquivo "Mod_pri.ari".

No caso do SICS esta função foi utilizada para armazenar as regras de derivada e integral.

Com estes predicado implementado podemos particularizar o interpretador ganhando com isto eficiência para o sistema SICS, sem no entanto perder a generalidade, uma vez que este novo predicado é na verdade uma extensão do consult (ainda disponível para o usuário na sua forma tradicional).

A formatação do predicado é:

consulte

consulte([{drive:}nome_arquivo{.extensão}{,outros}]).

Ex: consulte([arq1,$arq2.doc$,$a:arq3.$]).

* obs: Caso não for colocada extensão, o sistema assumirá uma extensão "default" .ari. Se o arquivo não tiver extensão ou tiver extensão diferente de .ari deve-se colocar toda a especificação do arquivo entre dois sinais "$".

CAPÍTULO VII - SUGESTÕES E CONCLUSÕES

SUGESTÕES PARA DESENVOLVIMENTOS FUTUROS

O sistema integrado de computação algébrica tem em sua base, um conhecimento acerca de algorítmos matemáticos para o cálculo algébrico. Para tornar o sistema mais inteligente e mais abrangente devemos incorporar novos conhecimentos à sua base. Além destes, o sistema deverá incorporar novos recursos que facilitem ainda mais a interação Homem-Máquina. A seguir, serão destacadas algumas idéias para futuras implementações:

- Novos algorítmos matemáticos;
- Algorítmos de simplificações;
- Manipulação de funções racionais;
- Desenvolvimento de novas flags de simplificação;
- O custo da função avaliadora de uma meta no cálculo de integrais deve ser função da média ponderada das características da meta com seus respectivos pesos, e não somente do comprimento.
- Transformações heurísticas para:
  . Racionalização do denominador
  . Integração por partes (aperfeiçoar)
  . Método de frações parciais
- Integrais definidas
- Equações diferenciais
- Transformada de Laplace
- Transformada z
- Reconhecimento de padrões
  . Aperfeiçoamento para reconhecer novas expressões
  . Utilizando os recursos do paralelismo
  . Generalização de problemas no procedimento de reconhecimento de padrões
- Aprendizado:
  . descobrir novos métodos e ajustar os velhos
  . aprendizado a partir de experiências
- Interface gráfica;
- Interface numérica;
- Facilidades da linguagem;
- Interface usuário (Linguagem natural);

COMENTÁRIOS E CONCLUSÕES

A primeira contribuição foi a de mostrar todas as etapas, técnicas e o processo de projetar um sistema simbólico. Nele estão as técnicas aplicadas, os problemas encontrados, as soluções adotadas e as idéias para desenvolvimentos futuros.

Outra importante contribuição foi o de mostrar a viabilidade da utilização do processamento simbólico na engenharia, no desenvolvimento de programação PROLOG e na educação (mais especificamente no aprendizado). Estes recursos são muito pouco utilizados no Brasil embora sejam indispensáveis á comunidade científica. Muitas apicações nesta área são realizadas através do processamento numérico que nem sempre tem a eficiência desejada.

O sistema SICS atingiu os objetivos propostos inicialmente no que se refere à viabilidade da implementação do sistema integrado de computaçao simbólica em micro-computadores através de:

- Manipulação de matrizes: Nesta área foi atingido o objetivo proposto, que foi o de desenvolver um conjunto de ferramentas básicas matriciais para o desenvolvimento de sistemas que envolvam manipulação matricial de maneira eficiente.
- Diferenciação - A manipulação de derivada foi extremamente facilitada com a utilização da lógica combinatória e foi desenvolvido um conjunto mínimo de primitivas de diferenciação que englobaram grande parte das aplicações. Novas regras de diferenciação podem e devem ser implementadas visando a atingir uma maior abrangência do sistema.
- Integrais - Neste módulo foi implementado um algorítmo básico para soluções de integrais que se mostraram eficazes na solução de uma certa classe de problemas. A representação adotada foi a de grafo e/ou e a estratégia de busca utilizada foi a best-first e/ou. O algorítmo é composto de três estágios: métodos diretos, métodos específicos e métodos heurísticos. Para alterar e/ou inserir novas regras existe um programa especialmente destinado a este fim.
- Simplificação - O método adotado foi o de implementar a simplificação orientada onde o usuário especifica a simplificação desejada através de flags.
- Reconhecimento de padrões - Um eficiente sistema de reconhecimento de integrandos é responsável pelo sucesso do sistema. No caso do SICS o sistema é recursivo na determinação de formas padrões, algorítmicas e heurísticas.
- Portabilidade - Atingir a portabilidade graças à utilização da linguagem PROLOG.
- Algorítmos e heurísticas - A perfeita sintonia entre a utilização de algorítmos e heurísticas é fundamental. No caso de domínios desconhecidos ou em casos de problemas de tempo e espaço, a heurística se torna obrigatória. Para outros casos deve-se tentar soluções algorítmicas e/ou soluções heurísticas. No caso do SICS deu-se ênfase às soluções heurísticas, uma vez que era o objetivo inicial deste trabalho. As regras heurísticas devem ser as mais precisas possíveis para evitar rumos indefinidos.
- Resolução de problemas - Os três estágios utilizados na solução de problemas de integrais no sistema SICS tiveram um desempenho importante na solução dos problemas de integral.
- Aprendizado - O SICS armazena opcionalmente a integral das funções aumentando assim seu desempenho em aplicações futuras ou mesmo durante o cálculo de uma integral, devido às suas chamadas recursivas. Outros recursos implementados foram:
  - Armazenamento do processo computacional para posterior análise;
  - A explicação da regras utilizadas;
  - O modo trace que permite a alteração dos procedimentos on-line durante o cálculo de uma expressão algébrica;

- Interface - A interface especializada desenvolvida para o sistema SICS atingiu o objetivo específico que era de facilitar o desenvolvimento de programas em PROLOG. Os seus objetivos iniciais estavam muito aquém do objetivo alcançado, o que dá uma motivação para implementações futuras com idéias inerentes ao shells. Foi desenvolvida também uma versão desta interface para micro-computadores com apenas dois discos e 640k bytes de memória.

A principal vantagem deste sistema em relação aos similares existentes é que o programa é aberto e portanto podemos alterar e/ou implementar novas técnicas aumentando a abrangência do domínio do sistema. Pode-se ainda particularizar o sistema para aplicações específicas, como por exemplo o aprendizado na educação ou mesmo utilizar a interface especializada para melhorar o rendimento no desenvolvimento de programas em prolog.

APÊNDICE A - INSTALANDO O SICS

O sistema SICS é composto de 6 discos distribuidos como segue:

1- Compilador:
   - APC.EXE (140.000)
   - APC.IDB (7.000)
   - APC.POO (23.000)

2- Interpretador:
   - API.EXE (244.000)
   - API.IDB (10.000)
   - API.POO (55.000)

3- Construtor:
   - ARITY.LIB (197.000)
   - API.LIB (17.000)
   - API.OBJ (3.000)
   - CODE.OBJ (9.000)

4- Utilitários:
   - CLONE.EXE (112.000)
   - ALINT.EXE (100.000)
   - ALINT.IDB (8.000)
   - ALINT.POO (23.000)

5- Programas fontes do SICS:
   - MOD_GRA.ARI (52.000)
   - MOD_SIS.ARI (29.000)
   - MOD_SIM.ARI (22.000)
   - MOD_INT.ARI (20.000)
   - MOD_MAT.ARI (11.000)
   - MOD_PRI.ARI (35.000)
   - HELP.ARI (24.000)
   - EXP_INT.BAT (1400)
   - N_REGRAS.BAT (1000)
   - PROLOG.INI (1000)
   - PROLOG.AUX (1000)
   - REGRAS.ARI (2000)

6- Base SICS gravada na forma binária e outros
   - SICS.IDB (10.000)
   - SICS.POO (88.000)
   - REGRAS.EXE (8.000)

Para o perfeito funcionamento do sistema deve-se utilizar o winchester com os diretórios conforme descrito a seguir:

diretório prolog:
Disco2
prolog.ini
diretório auxiliar:
Disco6
help.ari
ne.com

Para evitar problemas de abertura de arquivo, devemos alterar o arquivo CONFIG.SYS para 20 FILES. Com isto poderemos abrir até 6 arquivos ao mesmo tempo, já que o MS-DOS ocupa 5 e o ARITY 9.

Para ter um acesso automático a este diretório deve-se colocar no arquivo batch:

```
PATH=C:\;C:\Diretório usuário\prolog
SET FILEPATH=C:\Diretório usuário\auxiliar
SET EDITOR=C:\Diretório usuário\auxiliar
```

Caso não haja um no seu micro crie um com a especificação acima.

APÊNDICE B - INICIANDO COM O SICS

Após a instalação do sistema completo, estamos aptos a desenvolver sistemas computacionais que envolvam manipulação algébrica.

Há basicamente dois modos de operação: SICS interativo e SICS programável. No interativo, o usuário, após carregar o interpretador expandido, faz diversas perguntas ao interpretador obtendo sua resposta em seguida.

Já no modo programável, o usuário desenvolve todo o sistema na linguagem PROLOG e executa-o no interpretador. Para este modo dispomos de várias ferramentas com o intuito de auxiliar o programador em todas as etapas de desenvolvimento, tornando o trabalho simples e eficiente.

B.1 - Modo Interativo

Neste modo, iniciamos carregando o interpretador expandido prolog com o seguinte comando:

C> api <return>

Após o sistema ser carregado aparecerá a seguinte mensagem na tela:

```
ARITY/PROLOG INTERPRETER VERSION 4.0
COPYWRITE (C) 1986 ARITY CORPORATION
?-
inicializando o sistema...
```

A seguir será ativada a interface especializada.

A partir daí teremos o interpretador expandido para manipulação algébrica à disposiçao do usuário.

Podemos utilizar qualquer comando disponível no interpretador expandido, tendo como argumentos expressões matemáticas. O comando deve terminar com um ponto e <return>.

Após o return, o sistema executa o comando sobre argumentos, devolvendo o resultado seguido de "yes" caso a operação tenha sido realizada com sucesso e "no" caso contrário.

EXEMPLOS:

```
?- X is 3*8.
X = 24
yes

?- fat(3,Resposta).
Resposta = 6
yes
```

Para retornar ao controle do sistema operacional DOS digite:

```
?- ha.
C> (Retorna ao MS-DOS)
```

B.1.1 - CARREGAMENTO DOS PROGRAMAS DO USUÁRIO

Os programas criados pelo usuário em seu editor ou no editor disponível no sistema SICS, devem ser carregado na base do interpretador.

Para carregar os arquivos criados pelo usuário devemos utilizar o predicado "c" que têm o seguinte formato:

c([{drive:}arquivo{.extensão}{,outrosarquivos}]).

onde **drive** é o drive residente do arquivo a ser carregado. Se for o drive corrente não é necessário especificar.

.extensão é opcional. Se não for colocada, o sistema assume ".ari". Para nome de arquivos sem extensão devemos colocar o nome entre dois sinais "$" ($teste$). O mesmo é válido para arquivos com extensão diferente de ".ari".

.outros arquivos . Podemos num só comando carregar vários arquivos obedecendo às mesmas regras do primeiro arquivo. Os arquivos devem ser separados por vírgula (,).

Por exemplo se desejarmos carregar o arquivo "TESTE.ARI" devemos digitar:

?- c([teste]).

Outra maneira mais simples é a utilização da interface especializada. Vide capítulo IV para maiores detalhes.

Podemos sair do interpretador PROLOG temporariamente e executar os comandos disponíveis no DOS.

Existem três maneiras de executar os comandos DOS:

a)

?- shell <return>

e então aparecerá o "prompt" do DOS, permitindo então utilizar os comandos lá disponíveis. Para retornar ao PROLOG sem alterar a base de dados devemos digitar:

C> exit <return>

b)

?- shell($Comando DOS$).

o sistema irá para o DOS e executará o comando fornecido e então retornará automaticamente ao interpretador PROLOG.

c)

?- dos $Comando DOS$.

8.1.2 - SALVANDO A BASE DE DADOS

Quando os módulos utilizados pelo usuário são carregados e diversas alterações são realizadas sobre a base de dados, não podemos garantir que as mudanças estão salvas a menos que seja utilizado o comando save. Este comando salva toda a base de dados na forma compacta binária.

FORMATO:

save(nome) <return>

Para recuperar a imagem da base de dados em sua forma anterior, sem as mudanças realizadas após o último salvamento, devemos utilizar o predicado restore. Este comando é muito mais rápido do que o consult.

FORMATO:

restore(nome) <return>

Na interface especializada dispomos destes dois comandos .

B.1.3- Restrições em relação à memória

A manipulação algébrica requer uma quantidade muito grande de memória, portanto devemos carregar somente os arquivos estritamente necessários, evitando com isto problemas que possam surgir decorrentes da falta de espaço no ato da execução dos programas.

Caso haja insuficiência de memória, será emitida uma mensagem do tipo:

No global stack ! No local stack !
e o sistema retorna ao DOS

Neste caso há algumas sugestões que geralmente resolvem os problemas de falta de memória. São elas:

- Carregar somente os programas do usuário, estritamente necessários
- Executar o sistema por etapas
- Tentar otimizar a utilização de memória, analisando o programa através dos predicados statistics e garbage collector (gc)
- Tornar o sistema mais determinístico, utilizando para isto o "cut". Deve-se tomar cuidado na utilização do cut que é um comando extra-lógico.

B.2- Modo SICS programável

É o modo utilizado para desenvolvimento de sistemas computacionais que envolvam manipulação algébrica. As etapas deste processo são descritas a seguir:

Na **primeira etapa** o usuário codifica seus programas em PROLOG através do editor de texto e coloca um nome qualquer com extensão ari.

Na **segunda etapa** deve-se então eliminar os possíveis erros sintáticos. Neste caso, devemos carregar o programas diretamente no interpretador. Isto é feito como segue:

C> api ;invoca o interpretador

uma vez dentro do interpretador, devemos carregar o arquivo do usuário através do comando:

?- c([{drive:}arquivo{.ext}{,outros}]).

ou através da interface especializada.

Se houver algum erro sintático, o interpretador mostrará na tela as mensagens de erro juntamente com a cláusula que contém erros.

Na **terceira etapa** devemos alterar os possíveis erros do programa, se houver, com o editor de texto NORTON ou outro qualquer. Para utilizar o NE digite:

?- ne.

ou escolha a opção eD na interface especializada.

Aparecerá o editor de texto pronto para ser utilizado. Baseados nos erros mostrados na etapa anterior podemos efetuar as alterações necessárias.

Terminadas as modificações, o retorno ao interpretador PROLOG é automático, após a saída do editor.

Na **quarta etapa** devemos atualizar a base de dados e executar os programas.

Para atualizar a base de dados, com relação às mudanças efetuadas na etapa anterior, devemos utilizar o predicado "r" cujo formato é:

r([{drive:}Aquivo{.extensão},{,outros}]).

ou com a opção reconsult da interface especializada.

Se ainda houver erros deve-se retornar à terceira etapa.

Uma vez carregado o programa sem erros sintáticos, devemos rodá-lo para ver seu funcionamento. Antes disto devemos chavear para o mundo onde dispomos das ferramentas necessárias para execução do programa do usuário. Se não necessitar de ferramentas adicionais não há necessidade de fazer o chaveamento.

Se houver problemas na execução do programa, dispomos de uma excelente ferramenta para depuração que é o TRACE. Este executa o programa de forma orientada com a intermediação do usuário.

Caso algumas alterações sejam necessárias, devemos retornar à **terceira etapa.**

Se tudo estiver conforme o previsto, está terminada a etapa de desenvolvimento, restando agora a opção de compilar o programa, de forma a otimizar o código e ganhar velocidade de execução.

Outra opção no processo de desenvolvimento é a poderosa ferramenta ALINT que detecta os possíveis erros sintáticos e semânticos do programa e ainda fornece um mapa dos predicados globais e locais, excelente para otimizar os programas e muito útil quando estes programas são muito extensos.

Neste caso o programa só é carregado no interpretador PROLOG após eliminar os erros graves oriundos da execução do ALINT. A partir daí, segue a segunda etapa do processo normal.

O utilitário Lint(Disco número quatro) é invocado com segue:

C> alint ,arquivos,arquivoerros{,map}

onde **arquivos** são os nomes dos programas que o usuário deseja executar no interpretador.

arquivoerro é o arquivo onde serão armazenados os erros encontrados nos diversos arquivos. O sistema colocará extensão .LNT no nome do arquivo automaticamente.

O argumento **map** indica se o mapeamento dos predicados deve ser gerado.

EXEMPLO:

C> alint ,arq1+arq2,saida,map

O ALINT irá analisar os arquivos arq1.ari e arq2.ari .As mensagens de erro e o mapa dos predicados estarão no arquivo "saida.lnt".

As mensagens de erro do ALINT estão divididas em 3 categorias:

- erros graves ; devem ser corrigidos
- erros toleráveis ; devem ser analisados
- erros possíveis ; podem ser desprezados

As mensagens geradas são encontradas no livro "BUILDING ARITY/PROLOG APPLICATIONS"/ARITY - 1986/.

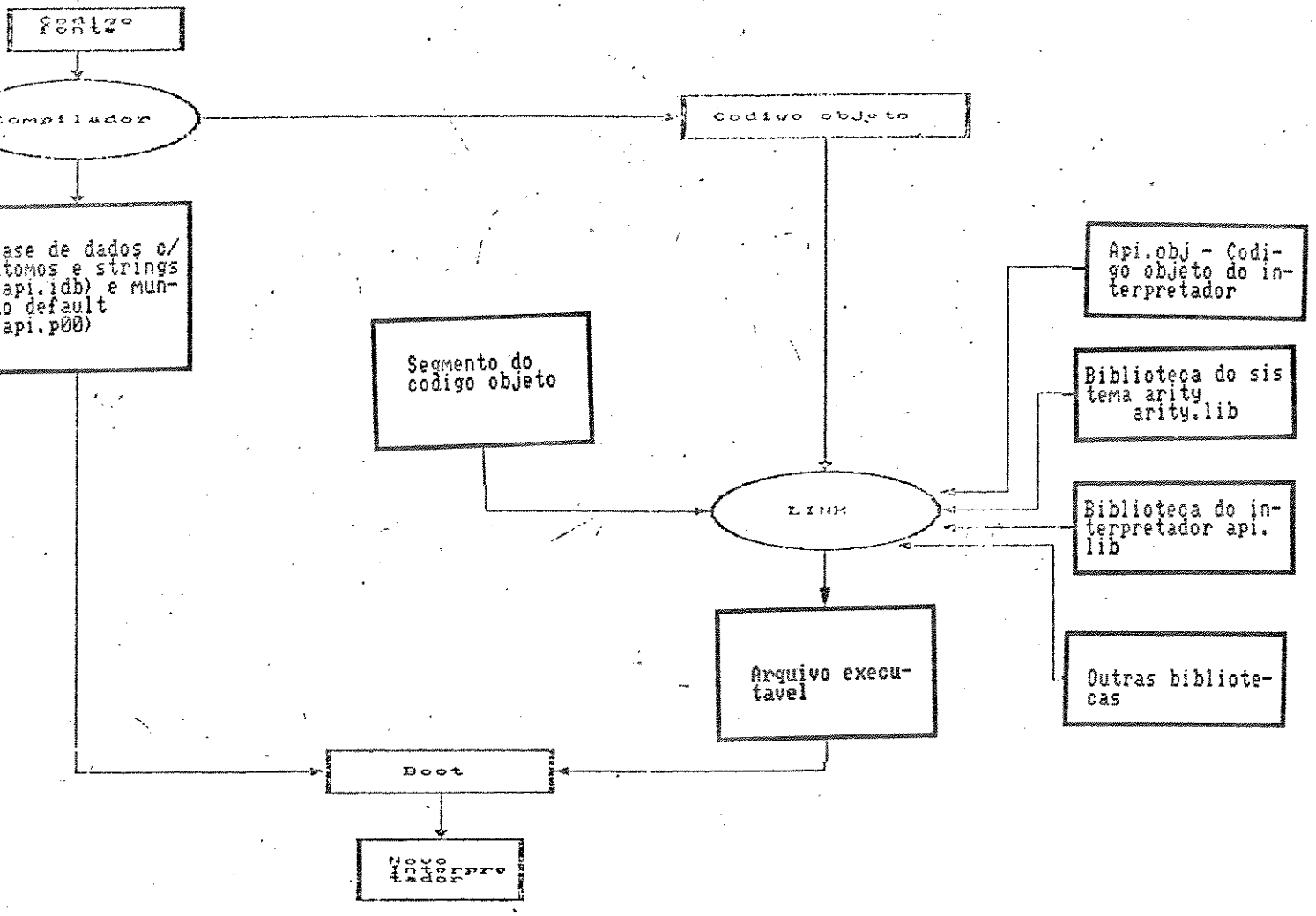
De posse de uma listagem do arquivo de erros, podemos, através do editor de texto, alterar o arquivo fonte.

Uma vez corrigidos os erros, devemos executar novamente o ALINT até que não haja mais erros do tipo grave. Este processo é necessário uma vez que o ALINT não detecta erros semânticos em cláusulas que contenham erros sintáticos.

APÊNDICE C- EXPANSÃO DO INTERPRETADOR

Uma das características mais importantes deste interpretador é a possibilidade de expansão e/ou alteração do mesmo. Sendo o sistema SICS um programa em desenvolvimento, devemos ter mecanismos fáceis para a alteracão do mesmo.

A expansão ou alteração do interpretador é esquematizada como segue:



O procedimento a ser executado está descrito a seguir:

1- Carregue os discos 1,2,3,4,5 e 6(incluindo os arquivos fonte SICS alterados ou não) num mesmo diretório do winchester, juntamente com o programa do usuário(se houver) a ser anexado ao interpretador.

É recomendável a utilização do utilitário ALINT com os programas fontes do sistema antes de efetuar a compilação.

Existe um arquivo tipo batch para facilitar o trabalho do usuário de gerar um novo interpretador. O usuário deverá alterar este arquivo colocando no seu início os seus programas para serem compilados (se houver) e colocar o nome destes arquivos também no link. O formato para compilação é o seguinte:

```
apc arquivosemextensão,,api
```

1- Ativar o programa batch:

```
C> exp_int <return>
```

2- digitar api para terminar o link do novo interpretador expandido.

```
C> api

  booting ... done

C>
```

3- Ativar o programa para inserir as regras de derivação e integração na base de dados.

```
C> n_regras <ret>
```

4- Para testar o novo interpretador faça:

```
C> api
e pronto ...
```

Os arquivos do novo interpretador são:

- api.exe
- api.idb
- api.p00
- prolog.ini
- sics.p00
- sics.idb

Para inserirmos e/ou alterarmos apenas as regras de derivação e integração devemos alterar a fonte do arquivo "regra.ari" e executar o procedimento acima descrito a partir da etapa 3.

APÊNDICE D - DESCRIÇÃO DAS FUNÇÕES IMPLEMENTADAS

As funcões atualmente disponíveis no módulo supervisor do sistema são:

OPERAÇÃO: **A**
OPERAÇÃO: **B**
OPERAÇÃO: **C**
ARGUMENTO: nenhum
DESCRIÇÃO: Mudar o drive corrente.
FORMATO:
    a.
    b.
    c.

OPERAÇÃO: **ALL_PRIMI**
DESCRIÇÃO: Lista uma breve descrição dos comandos disponíveis no SICS
ARGUMENTO: nenhum
FORMATO:
    all_primi.

OPERAÇÃO: **ARI**
ARGUMENTO: nenhum
DESCRIÇÃO:: Listar os arquivos com extensão .ari
FORMATO:
    ari.

OPERAÇÃO: **BAK**
ARGUMENTO: nenhum
DESCRIÇÃO: Deleta todos arquivos com extensão .bak do drive corrente.
FORMATO:
    bak.

OPERAÇÃO: **C**
ARGUMENTO: lista de nomes de arquivos
DESCRIÇÃO: Carregar arquivos na base do prolog.
FORMATO:
    c([Arq|Outrosarq]).
    c [Arq|Outrosarq].
EXEMPLO:
    c([Arquivo]) -> Carrega arquivo de nome arquivo.ari
    c [Arquivo] -> Carrega arquivo de nome arquivo.ari
    c([Arq1,Arq2]) -> Carrega arq1.ari e arq2.ari

OPERAÇÃO: **CRIAR**
ARGUMENTO: nome do arquivo
DESCRIÇÃO: Tem a finalidade de criar um novo arquivo
FORMATO:
    criar(Nome).
EXEMPLO:
    criar(teste) -> Cria arquivo de nome teste.ari

```
      criar($teste.sic$)  -> Cria arquivo de nome teste.sic
      criar($b:teste$)  -> Cria arquivo de nome teste no drive b:


OPERAÇÃO: DEL
ARGUMENTO:
      nome do arquivo
DESCRIÇÃO: Deleta arquivo específico.
FORMATO:
      del(nome). ou del nome.
EXEMPLO:
      del(teste)  -> deleta arquivo teste.ari do drive corrente


OPERAÇÃO: DIA
ARGUMENTO: dia
DESCRIÇÃO: Fornece a hora do sistema.
FORMATO:
      dia.
      dia(dia,mes,ano).
EXEMPLO:
      dia(10,6,1987). ; atualiza data


OPERAÇÃO: DIR
ARGUMENTO: nenhum
DESCRIÇÃO: Lista na tela o diretório corrente.
FORMATO:
      dir.
EXEMPLO:
      dir.    ;lista todo o diretório
      dir('*.ari') ; lista os arquivos com extensão .ari

OPERAÇÃO: DISCO
ARGUMENTO: drive
DESCRIÇÃO:  Muda o drive corrente. O predicado solicitará o drive
          desejado.
FORMATO:
      disk.

OPERAÇÃO: DISK
ARGUMENTO: nenhum
DESCRIÇÃO: Verifica qual o drive corrente.
FORMATO:
      disk.


OPERAÇÃO: HA
DESCRIÇÃO: Abandona o interpretador prolog
ARGUMENTO: nenhum
FORMATO:
         ha.
```

OPERAÇÃO: HORA
ARGUMENTO: hora
DESCRIÇÃO: Fornece a hora.
FORMATO:
    hora.
    hora(hora,minuto,segundo,centésimo)
EXEMPLO:
    hora(00,00,00,00). ; acerta o relógio do sistema

OPERAÇÃO: L
ARGUMENTO: nenhum
DESCRIÇÃO: Lista toda a base de dados.
FORMATO:
    l.
obs: As variáveis dos predicados presentes na base de dados terão nomes dados pelo interpretador.

OPERAÇÃO: LG
ARGUMENTO:nenhum
DESCRIÇÃO: Micro editor com opção de criar um arquivo log que é a imagem da saida da tela.
FORMATO:
    lg.
Obs: Para retornar ao interpretador normal digite fim <return>.

OPERAÇÃO: NOESC
ARGUMENTO: nome arquivo
DESCRIÇÃO: É o inverso de esc, isto é, desprotege o arquivo
FORMATO:
    noesc(nome).
EXEMPLO:
    noesc($teste$) -> Desprotege o arquivo teste.ari previamente protegido pelo predicado esc.

OPERAÇÃO: NSP
ARGUMENTO: nenhum
DESCRIÇÃO: Desativa um ponteiro de spy num determinado predicado.
FORMATO:
    nsp(predicado).
    nsp(predicado,aridade).
EXEMPLO:
    nsp(member).
    nsp(member,2).

```
OPERAÇÃO: NT
ARGUMENTO: nenhum
DESCRIÇÃO: Desliga o modo traco.
FORMATO:
      nt.

OPERAÇÃO: ABRIR
1o ARGUMENTO: nome do arquivo
2o ARGUMENTO: tipo de operação
DESCRIÇÃO:  Abre  um arquivo previamente criado com as  seguintes
opcões:
l - para ler
e -  para escrever
le - para ler e escrever
a - para acrescentar
la - para ler e acrescentar
FORMATO:
      ABRIR(nome,X).
sendo X:
      -l
      -e
      -le
      -a
      -la
EXEMPLO:
      abrir(teste,l)   -> Abre  arquivo de  nome  teste.ari  para
leitura.
      abrir($teste.sic$,le)   -> Abre arquivo de  teste.sic  para
leitura e escrita.
      abrir($b:teste$,a)  -> Abre arquivo de nome teste localizado
no drive b: para ser acrescentado.


OPERAÇÃO: PRIMI
DESCRIÇÃO: Help on-line acerca dos comandos disponíveis no SICS
ARGUMENTO: nenhum
FORMATO:
      primi.(A seguir digite o nome do comando e terá uma breve
descrição acerca do mesmo)



OPERAÇÃO: R
ARGUMENTO: lista de nomes de arquivos
DESCRIÇÃO: Recarrega arquivos na base do prolog.
FORMATO:
      r([Arq|Outrosarq]).
      r [Arq|Outrosarq].

EXEMPLO:
      r([Arquivo])  -> Recarrega arquivo de nome arquivo.ari
      r [Arquivo]  -> Recarrega arquivo de nome arquivo.ari
      r([Arq1,Arq2]) -> Recarrega arq1.ari  e arq2.ari
```

```
OPERAÇÂO: TEX
DESCRIÇÃO: Aciona a interface especializada.
ARGUMENTO: nenhum
FORMATO:
     tex.


OPERAÇÂO: TR
DESCRIÇÃO: Aciona o modo trace.
ARGUMENTO: nenhum
FORMATO:
     tr.


OPERAÇÃO: WS
ARGUMENTO: nenhum
DESCRIÇÃO: Vai para editor de texto (WS) provisoriamente.
FORMATO:
     ws.
```

COMANDOS DO MÓDULO MATRIZES

Eis o resumo dos principais predicados disponíveis para manipulação em geral de matrizes:

```
OPERAÇÃO: DERIVA_LINHA
1o ARGUMENTO: linha
2o ARGUMENTO: variável de derivação
3o ARGUMENTO: resposta
DESCRIÇÃO: Derivar uma linha de uma matriz .
FORMATO:
     mp(der_lin,[linha,variável],Resposta).
EXEMPLO:
     mp(der_lin,[[x^2,x,1,x],x],[2*x,1,0,1]).

OPERAÇÃO: DERIVA_MATRIZ
1o ARGUMENTO: matriz
2o ARGUMENTO: variável de derivação
3o ARGUMENTO: resposta
DESCRIÇÃO: Derivar matriz em relação a uma variável dada.
FORMATO:
     mp(der_mat,[matriz,variável],Resposta).
EXEMPLO:
     mp(der_mat,[[[x^2,x],[x,1]],x],[[2*x,1],[1,0]]).

OPERAÇÃO: DETERMINANTE
DESCRIÇÃO: Cálculo do determinante segundo o método de eliminação
          de Gauss-Jordan.
1o ARGUMENTO: matriz
2o ARGUMENTO: resultado
FORMATO:
     det(matriz,Resultado).
EXEMPLO:
     det([[1,2],[3,4]],Resultado).
     Resultado = -2
```

```
OPERAÇÃO: DIAGONAL
1o ARGUMENTO: matriz
2o ARGUMENTO: resultado
DESCRIÇÃO:  Retornar  uma  lista de elementos da diagonal  de  um
          matriz
FORMATO:
     diagonal(matriz,Resultado).
EXEMPLO:
     diagonal([[1,2],[3,4]],[1,4]).


OPERAÇÃO: EXIBE_ELEMENTO_POR_LINHA
DESCRIÇÃO:  Utilizado para listar na tela uma matriz de tal forma
          que fique um elemento por linha.  A matriz pode ser por
          referência ou por valor.
ARGUMENTO: referência ou matriz
FORMATO:
     exibe_elem_linha(Referência ou a matriz).
EXEMPLO:
     exibe_elem_linha([[a,c],[e,d]]).
     a
     b
     e
     d
     yes
     key(escola,Ref),exibe_elem_linha(Ref).
     joão
     josé
     maria
     yes

OPERAÇÃO: EXIBE_LINHA_POR_LINHA
DESCRIÇÃO:  Utilizado para listar na tela uma matriz de tal forma
          que  fique  uma linha da matriz por linha  da  tela.  A
          matriz pode ser representada por referência ou valor.
ARGUMENTO: referência ou matriz
FORMATO:
     exibe_linha_linha(Referência ou matriz).
EXEMPLO:
     exibe_linha_linha([[a,b],[c,d]]).
     [a, b]
     [c, d]
     yes
     key(escola,Ref),exibe_linha_linha(Ref).
     [josé, joão, maria]
     yes

OPERAÇÃO: GERA_IDENTIDADE
1o ARGUMENTO: matriz
2o ARGUMENTO: resultado
DESCRIÇÃO: Gerar a matriz identidade a partir de uma matriz dada.
FORMATO:
     gera_identidade(matriz,Resultado).
EXEMPLO:
     gera_identidade([[a,c],[w,e]],[[1,0],[0,1]]).
```

```
OPERAÇÃO: GRAVA
ARGUMENTO: matriz
DESCRIÇÃO:  Armazenar  listas de uma forma alternativa muito efi-
           ciente.
FORMATO:
     grava(chave,lista,Referência).
EXEMPLO:
     grava(escola,[joão,josé,maria],Referência).

OPERAÇÃO: IDENTIDADE
1o ARGUMENTO: matriz
2o ARGUMENTO: resultado
DESCRIÇÃO: Verificar se a matriz é identidade.
FORMATO:
     identidade(matriz).
EXEMPLO:
     identidade([[1,0],[0,1]]).
     ok

OPERAÇÃO: INTEGRA_LINHA
1o ARGUMENTO: linha
2o ARGUMENTO: variável de integração
3o ARGUMENTO: resposta
DESCRIÇÃO: Integrar uma linha de uma matriz .
FORMATO:
     mp(int_lin,[linha,variável],Resposta).
EXEMPLO:
     mp(int_lin,[[x,1,1,1],x],[x^2/2,x,x,x]).


OPERAÇÃO: INTEGRA_MATRIZ
1o ARGUMENTO: matriz
2o ARGUMENTO: variável de derivação
3o ARGUMENTO: resposta
DESCRIÇÃO: Integrar matriz em relação a uma variável dada.
FORMATO:
     mp(int_mat,[matriz,variável],Resposta).
EXEMPLO:
     mp(int_mat,[[[x,1],[x,1]],x],[[x^2/2,x],[x^2/2,x]]).


OPERAÇÃO: INVERSA DE UMA MATRIZ
DESCRIÇÃO:  O cálculo da inversa de uma matriz quadrada. Os
           elementos da matriz podem ser simbólicos ou numéricos.
           Convém simplificar a matriz após o cálculo, com o
           predicado específico.
1. ARGUMENTO: matriz
2. ARGUMENTO: resultado
FORMATO:
     inv([[1,2],[3,4]],[[-2,0,1.0],[1.5,-0.5]]).
```

```
OPERAÇÃO: QUADRADA
DESCRIÇÃO: Verificar se a matriz é quadrada
ARGUMENTO: matriz
FORMATO:
     quadrada(matriz).
EXEMPLO:
     quadrada([[1,2],[3,4]]).
     ok


OPERAÇÃO: RECUPERA.
DESCRIÇÃO: Recuperar listas gravadas pelo predicado grava
1o ARGUMENTO: chave
2o ARGUMENTO: lista
FORMATO:
     recupera(chave,Lista).
EXEMPLO:
     recupera(escola,[João,José,Maria]).



OPERAÇÃO: SIMPLIFICA_MATRIZ
1o ARGUMENTO: matriz
2o ARGUMENTO: resultado
DESCRIÇÃO: Simplificar matrizes em relação a operações básicas.
FORMATO:
     mp(simp_mat,matriz,Resposta).
EXEMPLO:
     mp(simp_mat,[[x+x,x],[1+0,1]],[[2*x,x],[1,1]]).


OPERAÇÃO: SOLUÇÃO
DESCRIÇÃO:  Solucionar  equações lineares simples pelo método  de
          eliminação de Gauss-Jordan.
1o ARGUMENTO: equação
2o ARGUMENTO: resultado
3o ARGUMENTO: Resposta
FORMATO:
     solução(equação,resultado,Resultado).
EXEMPLO:
     1*x+3*y=5
     2*x+4*y=6
     solução([[1,2],[3,4]],[[5],[6]],Resultado).
     Resultado = [-4,4.5]
```

FUNÇÕES PRIMITIVAS

OPERAÇÃO: **ABS**
DESCRIÇÃO: Cálculo do valor absoluto de um número
1o ARGUMENTO: Valor numérico
2O ARGUMENTO: Resultado do valor numérico
FORMATO:

```
abs(número,Resposta).
```

EXEMPLO:

```
abs(-5,Resposta).
Resposta = 5.
```

OPERAÇÃO: **AL**
DESCRIÇÃO: Colocar o termo como primeiro elemento de uma lista
1o ARGUMENTO: termo
2o ARGUMENTO: lista
3o ARGUMENTO: Resultado
FORMATO:

```
al(termo,lista,Resultado).
```

EXEMPLO:

```
al(2,[1,a,2],Resultado).
Resultado = [2,1,a,2]
```

OPERAÇÃO: **APPEND**
DESCRIÇÃO: Unir duas listas
1o ARGUMENTO: lista1
2o ARGUMENTO: lista2
3o ARGUMENTO: Resultado
FORMATO:

```
append(lista1,list2,Resultado).
```

EXEMPLO:

```
append([a,b],[c,d],Resultado).
Resultado = [a,b,c,d]
```

OPERAÇÃO: **APTODO**
DESCRIÇÃO: Aplicar uma determinada função especificada a todos elementos da lista
1o ARGUMENTO: função
2o ARGUMENTO: lista
3o ARGUMENTO: Resultado
FORMATO:

```
aptodo(função,lista,Resultado).
```

EXEMPLO:

```
aptodo(max,[[1,2],[3,4],[2,4]],Resultado).
Resultado = [2,4,4]
```

OPERAÇÃO: **AR**
DESCRIÇÃO: Colocar o termo como último elemento de uma lista
1o ARGUMENTO: termo
2o ARGUMENTO: lista
3o ARGUMENTO: Resultado
FORMATO:
    ar(termo,lista,Resultado).
EXEMPLO:
    ar(2,[1,a,2],Resultado).
    Resultado = [1,a,2,2]

OPERAÇÃO: **BUBLESORT**
DESCRIÇÃO: Ordenar uma lista segundo o método buble_sort
1o ARGUMENTO: lista
2o ARGUMENTO: resultado de uma lista
FORMATO:
    bublesort(lista,Resultado).
EXEMPLO:
    bublesort([1,2,3,5,4],Resultado).
    Resultado = [1,2,3,4,5]

OPERAÇÃO: **COMB**
DESCRIÇÃO: Combinar duas listas de igual dimensão
1o ARGUMENTO: lista1
2o ARGUMENTO: lista2
FORMATO:
    comb(lista1,lista2,Resultado).
EXEMPLO:
    comb([1,2],[3,4],Resultado).
    Resultado = [1,2,3,4]

OPERAÇÃO: **CONFIRMA**
DESCRIÇÃO: Confirmação de continuidade
ARGUMENTO: Nenhum
FORMATO:
    confirma.(A seguir será solicitado se o usuário deseja
    continuar)

OPERAÇÃO: **CONSEC**
DESCRIÇÃO: Checa se dois elementos são consecutivos em uma lista
1o ARGUMENTO: elemento1
2o ARGUMENTO: elemento2
FORMATO:
    consec(elemento1,elemento2).
EXEMPLO:
    consec(a,b,[m,n,a,b]).
    yes

OPERAÇÃO: **CONTA**
DESCRIÇÃO: Contar o número de elementos de uma lista
1o ARGUMENTO: lista
2o ARGUMENTO: Resultado
FORMATO:
conta(lista,Resultado).
EXEMPLO:
conta([a,b,c,d],Resultado).
Resultado = 4

OPERAÇÃO: **DENDEN**
DESCRIÇÃO: Permite alterar o estado da flag DENDEN
FORMATO:
denden.(A seguir será solicitado ao usuário digitar 0(resetar) ou 1(setar) a flag DENDEN)

OPERAÇÃO: **DL**
DESCRIÇÃO: Distribuir o termo à esquerda dos elementos da lista
1o ARGUMENTO: termo
2o ARGUMENTO: lista
3o ARGUMENTO: Resultado
FORMATO:
dl(termo,lista,Resultado).
EXEMPLO:
dl(a,[1,2],Resultado).
Resultado = [[a,1],[a,2]]

OPERAÇÃO: **DR**
DESCRIÇÃO: Distribuir o termo à direita dos elementos da lista
1o ARGUMENTO: termo
2o ARGUMENTO: lista
3o ARGUMENTO: Resultado
FORMATO:
dr(termo,lista,Resultado).
EXEMPLO:
dr(a,[1,2],Resultado).
Resultado = [[1,a],[2,a]]

OPERAÇÃO: **END_CRO**
DESCRIÇÃO: Parar o cronômetro e devolver o tempo em segundos.
1o ARGUMENTO: tempo
FORMATO:
end_cro(Tempo).

```
OPERAÇÃO: FAT
DESCRIÇÃO: Calcular o fatorial de um número
1o ARGUMENTO: número
2o ARGUMENTO: Resultado
FORMATO:
     fat(número,Resultado).
EXEMPLO:
     fat(3,Resultado).
     Resultado = 6

OPERAÇÃO: FIB
DESCRIÇÃO: Cálculo da função de Fibonacci
1o ARGUMENTO: Valor numérico
2o ARGUMENTO: Resultado do valor absoluto
FORMATO:
     fib(função,Resultado)
EXEMPLO:
     fib(1,Resultado).
     Resultado = 1



OPERAÇÃO: FINAL
DESCRIÇÃO: Colocação de mensagens de finais de programa no
           centro da tela
1o ARGUMENTO: mensagem, linha
FORMATO:
     final([mensagem,linha]).
EXEMPLO:
     final([$Final de programa$,12]).


OPERAÇÃO: FLAGS
DESCRIÇÃO: Mostra o estado das flags de simplificação
ARGUMENTO: Nenhum
FORMATO:
     flags.


OPERAÇÃO: GRAV
DESCRIÇÃO: Permite alterar o estado da "flag" INCORPORA
FORMATO:
     grav.(A   seguir   será   solicitado   ao   usuário   digitar
     0(resetar) ou 1(setar) a flag INCORPORA)

OPERAÇÃO: INTERSECÇÃO
DESCRIÇÃO: Intersecção de duas listas
1o ARGUMENTO: lista1
2o ARGUMENTO: lista2
3o ARGUMENTO: Resultado
FORMATO:
     intersecção(lista1,lista2,Resultado).
EXEMPLO:
     intersecção([a,c],[c,n],Resultado).
     Resultado = [c]
```

```
OPERAÇÃO: LAST
DESCRIÇÃO: Pega o último elemento da lista
1o ARGUMENTO: lista
2o ARGUMENTO: Resultado
FORMATO:
      last(lista,Resultado).
EXEMPLO:
      last([a,b,c],Resultado).
      Resposta = d

OPERAÇÃO: LISTA_IGUAL
DESCRIÇÃO: Verifica igualdade entre listas
1o ARGUMENTO: lista1
2O ARGUMENTO: lista2
FORMATO:
      lista_igual(Lista1,Lista2).
EXEMPLO:
      lista_igual([a,b,c],[a,b,c]).
      yes

OPERAÇÃO: LISTA REVERSA
DESCRIÇÃO: Reverte uma lista
1o ARGUMENTO: lista
2O ARGUMENTO: Resultado
FORMATO:
      lista_reversa(lista,Resultado).
EXEMPLO:
      lista_reversa([a,b,c],Resultado).
      Resultado = [c,b,a]

OPERAÇÃO: MATRIZ
DESCRIÇÃO: Verifica se termo é uma matriz
1o ARGUMENTO: termo
FORMATO:
      matriz(termo).
EXEMPLO:
      matriz(1).
      no

OPERAÇÃO: MAX
DESCRIÇÃO: Calcular o maior valor de uma lista de dois elementos
1o ARGUMENTO: Lista com dois elementos
2o ARGUMENTO: Resultado
FORMATO:
      max([elemento1,elemento2],Resultado).
EXEMPLO:
      max([2,3],Resultdao).
      Resultdao = 3
```

```
OPERAÇÃO: MDC
DESCRIÇÃO: Calcular o m.d.c. (máximo divisor comum) segundo o
           algorítmo de Euclides.
1o ARGUMENTO: numero1
2o ARGUMENTO: numero2
3O ARGUMENTO: Resultado
FORMATO:
     mdc(nume1,nume2,Resultado).
EXEMPLO:
     mdc(5,7,Resultdao).
     Resultado = 1

OPERAÇÃO: MENSAGEM
DESCRIÇÃO: Colocação de mensagens na tela
1o ARGUMENTO: coordenadas
2o ARGUMENTO: mensagem
FORMATO:
     mensagem(X:Y,$Mensagem$).
EXEMPLO:
     mensagem(10:12,$Digite seu nome$).


OPERAÇÃO: MEMBER
DESCRIÇÃO: Verifica se um elemento é membro de uma lista
1o ARGUMENTO: elemento
2o ARGUMENTO: lista
FORMATO:
     member(elemento,lista).
EXEMPLO:
     member(a,[d,e,a,e]).


OPERAÇÃO: MPTODO
DESCRIÇÃO: Aplica uma determinada função a uma matriz
1o ARGUMENTO: função
2o ARGUMENTO: matriz
FORMATO:
     mptodo(função,matriz,Resultado).
EXEMPLO:
     mptodo(first,[[1,2],[4,5]],Resultado).
     Resultado = [1,4]


OPERAÇÃO: NTH
DESCRIÇÃO: Calcula a posição de um elemento na lista
1o ARGUMENTO: elemento
2o ARGUMENTO: lista
3o ARGUMENTO: Resultado
FORMATO:
     nth(elemento,lista,Resultado).
EXEMPLO:
     nth(2,[a,b,c,2,d],Resultado).
     Resultado = 4
```

```
OPERAÇÃO: NUMNUM
DESCRIÇÃO: Permite alterar o estado da "flag" NUMNUM
FORMATO:
     numum.(A  seguir  será  solicitado  ao  usuário  digitar
0(resetar) ou 1(setar) a flag NUMNUM)

OPERAÇÃO: PAR
DESCRIÇÃO: Verifica se um número é par.
1o ARGUMENTO: número
FORMATO:
     par(número).
EXEMPLO:
     par(2).
     yes

OPERAÇÃO: PAUSE
DESCRIÇÃO: Pausa variável
1o ARGUMENTO: valor da pausa
FORMATO:
     pause(valor).
EXEMPLO:
     pause(100).



OPERAÇÃO: PEGUE_ATE_N
DESCRIÇÃO: Pega os primeiros "n" caracteres de uma lista
1o ARGUMENTO: lista
2o ARGUMENTO: Resultado
3o ARGUMENTO: n
4o ARGUMENTO: número total de elementos da lista
FORMATO:
     pegue_ate_n(lista,Resultado,n,ntotal de elem da lista).
EXEMPLO:
     pegue_ate_n([a,c,d],Resultado,2,3).
     Resultado = [a,c]



OPERAÇÃO: PERMUTAÇÃO
DESCRIÇÃO: Permutação de uma lista
1o ARGUMENTO: lista
2o ARGUMENTO: Resultado
FORMATO:
     permutação(lista,Resultado).
EXEMPLO:
     permutação([1,2,a],Resultado).
     Resultado = [1,2,a];
                 [1,a,2];
                 [2,1,a]

OPERAÇÃO: PRIMO
DESCRIÇÃO:  Calcula  os  números primos entre 0(zero) e o  valor
          fornecido.
1o ARGUMENTO: valor
```

```
2o ARGUMENTO: Resultado
FORMATO:
      primo(valor,Resultado).
EXEMPLO:
      primo(10,Resultado).
      Resultado = [1,3,5,7,9]

OPERAÇÃO: QUICKSORT
DESCRIÇÃO: Ordena uma lista segundo o método quicksort
1o ARGUMENTO: lista
2o ARGUMENTO: Resultado
FORMATO:
      quicksort(lista,Resultado).
EXEMPLO:
      quicksort([1,2,3,5,4],Resultado).
      Resultado = [1,2,3,4,5]



OPERAÇÃO: RED
DESCRIÇÃO:   Aplica   recursivamente   uma   função   a   cada   dois
          elementos de uma lista
1o ARGUMENTO: função
2o ARGUMENTO: lista
3o ARGUMENTO: Resultado
FORMATO:
      red(função,lista,Resultado).
EXEMPLO:
      red(+,[1,2,3],Resultado).
      Resultado = 6 (2+3=5 e 5+1=6)

OPERAÇÃO: RND
DESCRIÇÃO: Calcula um valor aleatório
1o ARGUMENTO: semente
20 ARGUMENTO: Resultado
FORMATO:
      rnd(semente,Resultado).
EXEMPLO:
      rnd(13,Resultado).
      Resultado = 0,1


OPERAÇÃO:START_CRO
DESCRIÇÃO:  Dispara cronômetro.  Utilizado para marcar tempo  de
          execução de programas.
ARGUMENTO: nenhum
FORMATO:
     start_cro.

OPERAÇÃO: SUBST
DESCRIÇÃO: Efetua a substituição em uma expressão simbólica
1o ARGUMENTO: expressão simbólica
2o   ARGUMENTO:   lista   cujos   elementos   são   listas   com   dois
          elementos, sendo o primeiro o símbolo a ser substituido
          e o segundo, o símbolo de substituição.
```

```
3o ARGUMENTO: Resultado
FORMATO:
      subst(Expressão,Lista,Resultado).
EXEMPLO:
      subst(2*x*y,[[x,1],[y,2]],Resultado).
      Resultado = 4


OPERAÇÃO: SUBCONJUNTO
DESCRIÇÃO: Verifica se uma lista é sub-conjunto da outra
1o ARGUMENTO: lista1
2o ARGUMENTO: lista2
FORMATO:
      subconjunto(lista1,lista2).
EXEMPLO:
      subconjunto([a,c],[m,n]).
      no


OPERAÇÃO: SUBTRAÇÃO
DESCRIÇÃO: Subtrai duas listas
1o ARGUMENTO: lista1
20 ARGUMENTO: lista2
3o ARGUMENTO: Resultado
FORMATO:
      subtração(Lista1,Lista2,Resultado).
EXEMPLO:
      subtração([a,b,c],[b,c],Resultado).
      Resultado = [a]


OPERAÇÃO: TIRA_ELEM
DESCRIÇÃO:  Tira a primeira instância de um elemento da lista
1o ARGUMENTO: elemento
2o ARGUMENTO: lista
3o ARGUMENTO: Lista resultante
FORMATO:
      tira_elem(elemento,lista,Resultado).
EXEMPLO:
      tira_elem(a,[a,b,a,c],Resultado).
      Resultado = [b,a,c]


OPERAÇÃO: TIRA_TUDO
DESCRIÇÃO: Tira todas as instâncias de um elemento de uma lista
1o ARGUMENTO: elemento
2o ARGUMENTO: lista
3o ARGUMENTO: Lista resultante
FORMATO:
      tira_tudo(elemento,lista,Resultado).
EXEMPLO:
      tira_tudo(a,[a,b,c,a],Resultado).
      Resultado = [b,c]
```

OPERAÇÃO: **TRIG**
DESCRIÇÃO: Permite alterar o estado da "flag" TRIGONOMÉTRICO.
FORMATO:
trig.(A seguir será solicitado ao usuário digitar 0(resetar) ou 1(setar) a flag TRIGONOMÉTRICO)

OPERAÇÃO: **UNIÃO**
DESCRIÇÃO: unir duas listas
1o ARGUMENTO: lista1
2o ARGUMENTO: lista2
3o ARGUMENTO: Resultado
FORMATO:
união(lista1,lista2,Resultado).
EXEMPLO:
união([a,c],[m,n],Resultado).
Resultado = [a,c,m,n]

No módulo da integral dispômos de dois predicados úteis para o usuário:

OPERAÇÃO: **INTEGRAL**
DESCRIÇÃO: Cálculo da integral de funções elementares.
1o ARGUMENTO: dint
2o ARGUMENTO: Lista com dois argumentos, sendo o primeiro o integrando e o segundo a variável de integração
3o ARGUMENTO: Resultado
FORMATO:
integral(integrando,variável,Resultado).
EXEMPLO:
integral(sen(x),x,Resultado).
Resultado = -cos(x)

No módulo da derivada o predicado para derivar é:

OPERAÇÃO: **DERIVADA**
DESCRIÇÃO: Cálculo da derivada de funções elementares.
1o ARGUMENTO: expressão a ser derivada
2o ARGUMENTO: variável de diferenciação
3o ARGUMENTO: resultado
FORMATO:
derivada(expressão,variável,Resultado).
EXEMPLO:
derivada(x,x,Resultado).
Resultado = 1

No módulo de cálculo e simplificação, temos o predicado de calcular expressões matemáticas.

OPERAÇÃO: **AVALIA**
DESCRIÇÃO: Cálculo de expressões matemáticas

```
1o ARGUMENTO: expressão
2O ARGUMENTO: resultado
FORMATO:
      avalia(expressão,Resultado).
EXEMPLO:
      avalia(3*4,Resultado).
      Resultado = 12
      ap(exec,[[1,2],[3,4]]+[[1,1],[1,1]],Resultado).
      Resultado = [[2,3],[4,5]]
```

ALGUNS PERIÓDICOS ESPECIALIZADOS NA ÁREA DE INTELIGÊNCIA ARTIFICIAL

Eis alguns dos periódicos especializados da área:

- Artificial Intelligence
- Communication of ACM
- IEEE Expert System
- IEE Transaction on Pattern Analysys and Machine INtelligence
- Machine Intelligence
- SIGART (ACM)
- SIGSAM (ACM)

ENDEREÇOS DE CENTROS DE PESQUISA NA ÁREA DE INTELIGÊNCIA ARTIFICIAL:

BUDAPESP
Nim Iguszi
1363 Budapest
P.U.B. 33
HUNGRIA

CTI
Centro Tecnológico para a Informática
Campinas - SP

EDINBURGH
Department of Artificial Intelligence University of Edinburgh
Edinburgh
8. Hope park Sq., Meadow Lane
Edinburgh EH8
Escocia

ESTOCOLMO
Departament of Computer Science University of Stockholm
106 91 Stockholm
Suécia

IME
Instituto Militar de Engenharia
Rio de Janeiro - R.J.

INPE
Instituto de Pesquisas Espaciais
Caixa Postal 515
São José dos Campos - S.P.

LEUVEN
Applied Mathematics and Programming Dept.
Katholieke Uni.
Leuven
Bélgica

LISBOA
Centro de Informática
Laboratório Nacional de Engenharia Civil
101 Av. do Brasil
Lisboa
Portugal

Departamento de Engenharia Informática
Universidade Nova de Lisboa
Seminário dos Cabeço
1800 Lisboa
Portugal

LONDRES
Departamento of Computing and Control Imperial College
Universty of London
180 Queens Gate
London SW7
Inglaterra

MARSEILLE
Group d'Intelligence Artificielle UER Scientifique de Luminy
70 route léon-lachaup
France

PARAIBA
Departamento de Sistemas e Computação
Universidade Federal da Paraiba
58100 - Campina Grande - Pb

UNICAMP
Departamento de Engenharia da Computação e Automação Industrial (DCA)
C.P. 6101
13081 - Campinas - S.P.

WATERLOO
Computer Science Departament University of Waterloo
Waterloo , Ontario
Canada

PUC/RIO
Prédio Rio Data Centro
R. Marques de S. Vicente 225
Gávea - Rio de Janeiro

BIBLIOGRAFIA:

/ALMEIDA, JUNIOR - 1985/ M.C.V. Almeida; R.M. Junior: SINEMA - Sistema integrado educacional de manipulação algébrica - **VIII Congresso Nacional de Informática**- São Paulo - Set. 1985.

/ARITY - 1986/ The Arity screen design toolkit - **Arity Corporation** - 1986.

/ARITY - 1986/ The Arity/Prolog Programming Language - **Arity Corporation** - 1986.

/ARITY - 1986/ Building Arity/Prolog applications - **Arity Corporation** - 1986.

/BACKUS - 1978/ Backus,J. - Can Programming be liberated from the Von Neuman style? A Functional Style and its algebra of programs - **Communications of ACM** - Aug. 78,pp. 613-641.

/BOBROW AND STEFIK 1986/ Bobrow, D.G. ; Stefik, M.J. - **Perspective on Artificial Intelligence Programming** - Science, vol 231 - Feb. 86,pp. 951-957.

/BRATKO - 1986/ Bratko, I. - Prolog programming for artificial Intelligence - Addison-Wesley .

/CALMET AND LUGIEZ - 1987/ Calmet, J. ; Lugiez, D. - A Knowledge Based System for Computer Algebra - ACM -**SIGSAM BULLETIN** - Vol 21 n# 1 - Feb. 1987.

/CAVINESS e COLLINS - 1972/ Caviness e Collins - Symbolic Mathematic Computation - **SIGSAM Bulletin** - N 23 - July 1972 - pg 25-28

/CASANOVA, GIORNO AND FURTADO - 1986/ Casanova, M. A.; Giorno, F. A. C.; Furtado, A. L. - Programação em Lógica - **V Escola de Computação** - Belo Horizonte - 1986.

/CLOCKSIN AND MELLISH - 1984/ Clocksin,W.F; Mellish,C.S. - **Programming in Prolog** . Springer Verlag 1984.

/COELHO, COTTA AND PEREIRA- 1980/ Coelho,H. ; Cotta, J.C. ; Pereira, L.M. - How to solve it with prolog - **Laboratorio Nacional de Engenharia Civil** - Lisboa - 1985.

/DAHL - 1983/ Dahl,V. - Logic Programming as Representation of Knowledge. - **Computer** - Oct. 83, pp. 106-111.

/DARLINGTON - 1985/ Darlington,J. - Program Transformation **Byte**- Aug. 85,pp. 201-216.

/DAVIS - 1985/ Davis,R.E. - Logic Programming and Prolog: A Tutorial - **IEEE Software** - Sept. 85,pp. 53-62.

/EISENBACH, SADLER - 1985/ Eisenbach,S.; Sadler C. - Declarative languages: An Overview -**Byte** - Aug. 85, pp. 181-197.

/ERNIE - 1985/ Ernie, Tello - The Languages of AI research.**PC Magazine** - April 85, pp. 173-189.

/FATEMAN - 1980 / Fateman,R.J. - Symbolic and Algebraic Computer Programming Systems.**International Conference of Mathematics Educators** - Berkley, California,pp. 21-32.

/FEIGENBAUM, AURON - 1981/ Feigenbaum E.A.; Barr Auron -**The Handbook of Artificial Intelligence**,1981.

/FURTADO - 1986/ Furtado, A. L. - Paradigmas de linguagens de programação - Editora UNICAMP - 1986.

/GENESERETH, GINSBERG - 1985/ Genesereth,M.R.; Ginsberg,M.L. - Logic Programming - **Communication of ACM** - Sept.

85,pp. 933-941.

/GOLDBERG / Goldberg, Adele - Smaltalk-80 .

/GOLDSTEIN, LAY e SCHNEIDER - 1981/ Goldstein,J.L. ; Lay, D.C. ; Schneider, D.I. - Cálculo e suas aplicações - Hemus Livraria Editora Ltda

/GUZMÁN e McINTOSH -- 1987/ - Guzmán, J. ; Mcintoshi, H. - **Communication of the ACM** - Vol 9 - N8 - Aug. 66.

/HANSON, CAVINESS E JOSEPH - 1962 / Hanson, J.W. ; Caviness, J. S. e Joseph, C. - Anaytic Differentiation by computer - **Communication of ACM** - vol 5 - n# 6 - Junho 1962.

/HARRISON, KHOSHNEVISAM - 1985/ Harrison, P.G.;Khoshnevisam,H. Functional Programming Using F.P. -**Byte** - Aug. 85,pp. 219-232.

/HEWWIT - 1985/ Hewwit,C. - The Challenge of Open Systems -**Byte**- April 85,pp. 226-242.

/KANOUI - 1976/ Kanoui, H. - (Group d'Intelligence Artificielle U.E.R. de Luminy - France) - Some aspects of symbolic integration via predicate logic programming - **SIGSAM bulletin (ACM)**

/KOWALSKI - 1985/ Kowalski,R. - Logic Programming - **Byte**- Aug. 85,pp. 171-177.

/LEE - 1983/ Lee, C.S.G. - Robot Arm Dynamic, Tutorial on Robotics, **IEEE Computer Society Press**, 1983

/LOMBARDI - 1984/ Lombardi, J.C. - Pascal - MP manipulador algébrico e numérico de polinômios - Dissertção de mestrado - INPE - Abril - 1984

/LOPES , PEREIRA e ALVEZ - 1986/ Lopes, J.S.N ; Pereira A.E.C e Alvez, J.B.M - Symbolic Computation Applied to Robot Dynamic Modeling

/MONARD E NAKANISHI - 1984/ - Monard, M. C. e Nakanishi, T. - MAPLE - A symbolic system from University of Waterloo, **Anais do VII Congresso Nacional da Sociedade Brasileira de Matemática Aplicada e Computacional**, Campinas 10-25, Setembro de 1984.

/MOSES - 1971/ Moses,J. -Symbolic Integration: The Stormy decade - **Communication of ACM** - Aug. 71,pp. 548-560.

/MOSES - 1974/ _____. - Macsyma - The fifth year - **SIGSAM Bulletin** - Vol. 8 - N3, Aug. 74,pp. 105-110.

/muMATH - 1980/ The reference manual.The Sotf Warehouse - USA - 1980

/NEWELL, SHAW AND SIMON - 1957/ Newell,A.;Shaw ,J.C.;Simon,H.A. - Empirical exploration of the logic theory machine.**Proc. Western Joint Comp. Conf.**, 1957,pp. 218-239.

/NILSSON - 1980/ Nilson, N. J. - Principles of Artificial Intelligence - Tioga Publishing - 1984

/O'KEEFE - 1983/ O'keefe, R.A. - Prolog compared with Lisp - **SIGPLAN notices** - V 18 5, May 1983

/PLESKUN AND THAZUTHAVEETIL - 1987/ Pleszkun A. R; Thazhuthaveetil M. J. - The Architecture of Lisp Machines - **Computer** - March 1987 - pp. 35-43.

/ROCHA e ROCHA - 1977/ Rocha, L.M. ; Rocha C.R. - Cálculo diferencial e Integral - Nobel - 1977

/SAMMET - 1966/ Sammet, J. E. - Survey of Formula Manipulation - **Communication of the ACM** - vol 9, n# 8 , Agosto 1966 - pg 555-569.

/SHOCHAT - 1982/ Shochat,D.D. - A Symbolic Mathematics System - **Creative Computing** - Oct. 82, pp. 26-33.

/SHOCHAT - 1982/ Shochat,D.D. - Experience with muSimp/muMATH-80 Symbolic Mathematics System - **SIGSAM bulletin** - Vol 16 - N3 - Aug - 1982 - pg 16-23

/SLAGLE - 1961/ Slagle,J.R. -A heuristics program that solves symbolic integration problems in freshman calculus: symbolic automatic integrator (SAINT).**Report A G-0001, MIT Lincoln Laboratory**, May 61.

/STEINBERG - 1982 / Steinberg - Mathematic and Symbol Manipulation - **SIGSAM Bulletin** - Vol16 - N3 - Aug 1982 - pg 11-15.

/STOUTMEYER -1979/ Stoutemyer, D. R. - Lisp based Symbolic Math Systems - **Byte** - Aug. 79, pp. 176-192.

/SUNDBLAD - 1974/ Sundblad, Y. "Symbolic Mathematical Systems Now and in the Future" - **SIGSAM Bulletin** - Vol.8, n# 3, Aug. 74, pp. 1-8.

/SWINEHART e ZELLWEGER - 1986/ Swinehart, D.C. ; Zellwegger, P.T. ; Ricahard, J.B. - A Structural View of the Cedar Programming Environment - **ACM transactions on Programming Languages and Systems** - Vol 8 - N4, Oct 86 - pp 419-490.

/ THOMAS - / Thomas, G.B. - Cálculo - Ao Livro Técnico S.A.

/TRELEAVEN - 1985/ Treleaven , P.C. ; Lima, Isabel Gouveia - Computers and programming languages for AI - **V Congresso da Sociedade Brasileira de Computação** - UFRGS - Porto Alegre - Julho - 1985.

/WARREN, PEREIRA AND PEREIRA - 1985/ Warren, D.H.D; Pereira, L.M.; Pereira F. - Prolog - The language and its implementation compared with Lisp, pp. 109-115.

/WILLIAMSON - 1986/ Williamson,M. - **Artificial Intelligence for Microcomputers** - Simon & Schuster Publishing - 1986.