ESTUDO E CRITICA DE UM MODELO PARA SISTEMAS ESPECIALISTAS ADAPTATIVOS

Jayme Bentes[1]

Tese submetida ao corpo docente da Coordenação dos Programas de Pós-Graduação em Engenharia da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para a obtenção do grau de mestre em Engenharia de Sistemas e Computação.

Aprovada por:

Antonio de Almeida Pinho

Prof. Antonio de Almeida Pinho, D. Sc.
(Presidente)

Sheila Regina Murgel Veloso

Profa. Sheila Regina Murgel Veloso, D. Sc.

Claudia GR do Valle

Profa. Cláudia Guerreiro Ribeiro do Valle, D. Sc.

Rio de Janeiro, RJ - Brasil
Maio de 1992

---

[1] Engenheiro Eletricista,Analista de Sistemas - PETROBRÁS

Bentes, Jayme

Estudo e Critica de Um Modelo para Sistemas Especialistas Adaptativos [Rio de Janeiro] 1992

VII, 120p. 28cm (COPPE/UFRJ, M.Sc. , Engenharia de Sistemas, 1992)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Inteligência Artificial

I. COPPE/UFRJ II. Título (série).

Índice

Prefácio

Os Sistemas Especialistas, apesar de já não despertarem tanta curiosidade acadêmica, continuam sendo uma área fértil para a busca de soluções criativas para problemas de processamento de dados não convencionais.

Um ramo pouco explorado e com resultados ainda pouco convincentes, é o que combina Aprendizado por Máquina com Sistemas Especialistas.

A possibilidade de um computador poder vir a adquirir conhecimento por conta própria, sem a interferência direta do ser humano, é uma idéia que fascina a todos, pelas inúmeras e inimagináveis consequências que acarretaria.

Estamos longe, porém, de tal realidade, e o pouco que se conseguiu até agora neste sentido só serviu para nos mostrar o quão distante está este objetivo.

As Redes Neuronais constituem um ramo de pesquisa que vem obtendo alguns bons resultados na busca por Sistemas Especialistas Adaptativos capazes, em alguns casos, de aprender a partir de exemplos.

Alguns pesquisadores, porém, tem envidado esforços no sentido de dotar Sistemas Especialistas baseados em Regras com algum poder de adaptação.

Um modelo de aprendizado, baseado na habilidade adquirida por sucessivas repetições da mesma tarefa, é o ponto focal deste trabalho, onde procuramos avaliar as possibilidades e limitações que esta abordagem pode proporcionar quando aplicada a Sistemas Especialistas baseados em regras.

Jayme Bentes
COPPE/UFRJ

Resumo

ESTUDO E CRITICA DE UM MODELO PARA SISTEMAS ESPECIALISTAS ADAPTATIVOS

Jayme Bentes
Maio de 1992

Orientador: Prof. Antonio de Almeida Pinho, D.Sc.
Programa : Engenharia de Sistemas e Computação

Este trabalho apresenta o estudo e a crítica a um modelo de aprendizado aplicável a Sistemas Especialistas baseados em regras. A filosofia de aprendizado por trás do modelo é o aprimoramento da habilidade pela prática repetida. Na sua concepção foi também utilizada uma analogia econômica, baseada nas forças de mercado, para a seleção da regra mais promissora a cada ciclo de inferência.

Para permitir uma abordagem dos aspectos práticos da utilização do modelo, foi feita uma implementação do mesmo, na linguagem de programação em Lógica - PROLOG. Por fidelidade à proposta original, empregou-se aí um motor de inferência com encadeamento progressivo.

A execução de uma aplicação simples e ilustrativa serviu de base para as conclusões sobre o desempenho do modelo.

Abstract

STUDY AND CRITICISM OF A MODEL FOR ADAPTATIVE EXPERT SYSTEMS

Jayme Bentes

1992, May

Thesis Supervisor: Antonio de Almeida Pinho, D.Sc.

Department : Systems Engeneering and Computation

This work presents the study and critical considerations about a learning model suitable for Rule-Based Expert Systems.

The learning philosophy behind the model is the skill refinement through practice. Its conception, an economic analogy based on market strengths, was used to guide the selection of the most promising rule in each recognize-act cycle.

To enable practical considerations about the model, an implementation in PROLOG - a Programming Language in Logic - was made.

We used a forward chaining inference engine to adhere to the original plan.

Our conclusions about the model performance were helped by the use of a simple and illustrative aplication.

1. Introdução

Chamamos **Ciclo de Vida** de um Sistema de Informação o período que compreende a concepção, implementação, manutenções corretivas e extensões e finalmente a descontinuação do produto, quando este deixa de atender às necessidades do usuário, ao nível de satisfação desejado.

No Sistema Especialista, onde o Conhecimento da área problema, representado pela Base de Conhecimento, acha-se desvinculado da lógica de controle da aplicação, tem-se uma expectativa de vida semelhante aos sistemas tradicionais, porém mais vinculada à disponibilidade de Conhecimento sobre a área problema. A evolução do sistema se refletirá em novas peças de Conhecimento adicionadas ou modificadas na Base de Conhecimento original.

Sistemas Especialistas Adaptativos apresentam vantagem em relação a Sistemas Especialistas tradicionais justamente pelo seu poder de adaptação às mudanças que se fizerem necessárias em sua Base de Conhecimentos, sem revelar impactos significativos em seu desempenho global. É de se esperar, como consequência, que tais sistemas tenham uma vida útil superior à dos demais.

Antes de apresentarmos o Modelo de Aprendizado baseado no conceito de Refinamento de Habilidade, conforme proposto por Pi-Sheng Deng e outros e cujo estudo, implementação e crítica constituem o escopo deste trabalho, faremos, no Capítulo 2, uma breve recapitulação do que vem a ser um Sistema Especialista, segundo o enfoque tradicional.

No Capítulo 3 tecemos algumas considerações sobre Aprendizado, onde buscamos delinear que características são fundamentais para caracterizar um sistema como sendo capaz de aprender.

As principais diferenças entre Sistema Especialista e Sistema Especialista Adaptativo são colocadas nesse capítulo.

O Capítulo 4 apresenta o modelo particular de aprendizado por máquina e sua descrição matemática. Utilizamos este modelo para implementar um Sistema Especialista Adaptativo, conforme veremos no Capítulo 5.

Uma aplicação prática é o assunto do Capítulo 6, onde são comentadas as adaptações necessárias para tornar adequada a representação do Conhecimento, via regras de produção, ao sistema que implementamos.

Os resultados obtidos e as conclusões que daí pudemos extrair estão expostos no sétimo e último capítulo. Os Anexos complementam este trabalho, com ênfase na documentação da implementação do modelo.

2. Sistemas Especialistas Tradicionais

A maioria, senão todas as publicações sobre Sistemas Especialistas existentes atualmente, dedicam um tópico, ou mesmo um capítulo, à definição do que seja um Sistema Especialista, o que mostra a polêmica que o assunto ainda sugere.

Uma definição abrangente, aprovada pelo comitê formado pelo grupo especializado em Sistemas Especialistas da Sociedade de Computação Britânica, enuncia:

"Um Sistema Especialista é entendido como a materialização em computador, via componente com Base de Conhecimentos, da habilidade de um especialista, de tal forma que o sistema pode oferecer suporte inteligente ou tomar uma decisão inteligente sobre uma função de processamento. Uma característica adicional, que muitos consideram fundamental, é a capacidade do sistema justificar, sob demanda, sua própria linha de raciocínio de uma maneira direta e inteligível pelo usuário. O estilo adotado para atender a estas características tem sido a programação baseada em regras."

Em "Build your own Expert System" [01] encontramos toda uma argumentação subjetiva a respeito, que encerra com o texto :

"... Um programa de computador, em suma, que irá fazer algo que você não tinha imaginado que pudesse ser feito, todavia, por um computador. Um programa de computador que faz alguma coisa a qual você pensava que realmente precisasse dos serviços de um especialista humano para ser realizada.

Mas ainda, no final das contas, um programa de computador."

O que faz, então, um Sistema Especialista diferir dos demais tipos de programas de computador ?

"Só no final dos anos 70 é que os cientistas de IA começaram a perceber algo muito importante: A capacidade de resolução de problemas de um programa vem do conhecimento que ele possui, não apenas dos formalismos e esquemas de inferência que emprega. Este salto conceitual pode ser enunciado de maneira bastante simples:

- Para fazer um programa inteligente, abasteça-o com quantidades de conhecimento específico e de alta qualidade sobre alguma área problema.

Essa descoberta levou ao desenvolvimento de programas de computador de propósitos específicos, sistemas que eram especializados em alguma área problema limitada. Esses programas receberam o nome de Sistemas Especialistas e um novo campo teve inicio." [02]

Podemos concluir que um Sistema Especialista é um programa de computador com Base de Conhecimentos capaz de realizar uma tarefa executável até então só por um especialista humano, desde que esta atividade não envolva habilidades outras que não as intelectuais.

Esta conclusão subentende que a tarefa em questão não deve ser uma para a qual se conheçam algoritmos eficazes e capazes de fornecer respostas invariávelmente corretas. Torna, também, mutáveis os problemas suscetíveis de serem representados por Sistemas Especialistas, pois uma vez que se consiga conhecer com precisão a natureza de um determinado problema, algoritmizando-o, este passa a não pertencer mais à classe dos problemas de interesse de IA, tornando-se um problema solúvel por processamento de dados convencional.

2.1 A Arquitetura de um Sistema Especialista

Uma arquitetura comumente aceita apresenta um Sistema Especialista como sendo composto de [03] :

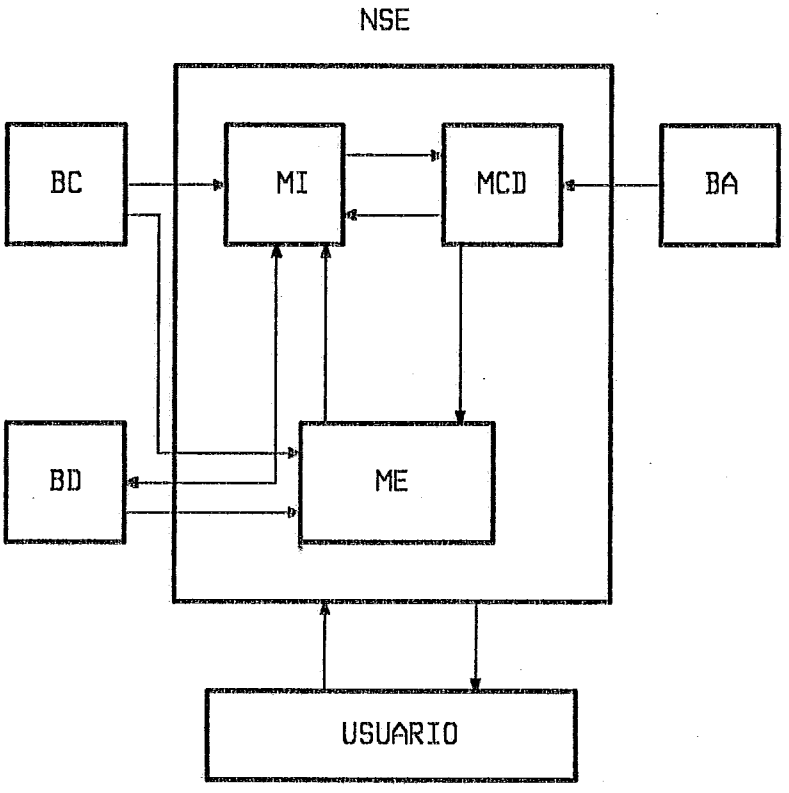
- Base de Conhecimentos
- Motor de Inferência
- Interface com o Usuário

A Base de Conhecimentos é o repositório do conhecimento real e heurístico no domínio da aplicação. O conhecimento heurístico é muitas vezes judicioso e não provado, mas é também, com frequência, uma parte vital desse campo do conhecimento. Por diversas vezes esse tipo de conhecimento é associado com fatores de confiança que indicam até que ponto a informação é confiável. É este tipo de conhecimento que torna os Sistemas Especialistas atrativos para serem usados em áreas às quais os enfoques algorítmicos não se aplicam ou não são adequados.

O Motor ou Máquina de Inferência é o processador de conhecimentos do Sistema Especialista, sendo sua função deduzir novos fatos a partir do conhecimento já existente e dos dados do problema.

A Interface com o Usuário tem duas funções básicas: a aquisição de novos conhecimentos e a justificação (ou explanação) do raciocínio utilizado para chegar aos resultados obtidos pelo Sistema Especialista.

Uma variação desta arquitetura, um pouco mais detalhada, é apresentada a seguir [04] e serviu de base para o trabalho de implementação do modelo:



FONTE: USP/ICMSC E ILTC - 1990

Figura 2.1: Núcleo de Um Sistema Especialista

Os componentes apresentados na figura são os seguintes:

BC Base de Conhecimento, onde se encontra armazenado o conhecimento do sistema. É fruto do trabalho do Engenheiro de Conhecimento, que transferiu para lá o saber de um especialista humano.

BD Base de Dados. Contém dados de entrada do problema, respostas fornecidas pelo usuário, etc. Serve ao objetivo de poder explicar os passos da linha de raciocínio do sistema.

Ba Base Algorítmica, uma divisão da BC, que carrega informação manipulada pelo MCD e necessária à obtenção de dados externos ao sistema.

NSE Núcleo do Sistema Especialista, composto de três módulos distintos: MI, MCD e ME

MI Motor de Inferência, que aqui utiliza a meta-interpretação como método de manipulação do conhecimento.

MCD Módulo Coletor de Dados, que ativado pelo Motor de Inferência, buscará dados externos ao sistema para permitir novas asserções.

ME Modulo de Explicação ou Justificador, é responsável pela explicação ***como*** o Motor de Inferência chega a certas conclusões e ***porque*** faz determinadas perguntas. É ativado por solicitação do usuário.

USUÁRIO É a pessoa que está realizando alguma consulta ao Sistema Especialista.

Uma documentação, com descrição detalhada, destes módulos consta da literatura sugerida na Bibliografia deste trabalho [04], portanto não desceremos aqui a um grau maior de detalhe.

A forma como o Motor de Inferência processa o conhecimento da BC - a linha de raciocínio que utiliza para propor alguma solução para o problema - pode ser executado segundo uma das seguintes estratégias, quando se utiliza regras como forma de representação do conhecimento:

- Encadeamento regressivo (backward chaining)
- Encadeamento progressivo (forward chaining)

Em linguagem simples, o **encadeamento regressivo** consiste em supor verdadeira uma provável solução e tentar confirmá-la através dos dados

disponíveis, enquanto o **encadeamento progressivo** parte dos dados disponíveis em busca de possíveis conclusões.

A implementação realizada pela USP [04] é de um Motor de Inferência com encadeamento regressivo, porém, como veremos mais adiante, os autores do modelo de aprendizado que estudamos optaram por implementar a inferência com encadeamento progressivo.

Para tornar nosso estudo do modelo o mais fiel possível ao proposto originalmente, utilizamos a arquitetura da USP adaptada à inferência forward chaining.

3. Sistemas Especialistas Adaptativos

O ser humano é naturalmente dotado de capacidade de adaptação a novas situações, decorrente de sua inteligência que lhe permite aprender com a experiência.

A seguir abordamos os pré-requisitos necessários a procedimentos que procurem reproduzir, num sistema de computação, parte das características humanas que conduzem ao aprendizado e faremos um paralelo com o modelo em estudo.

3.1 Aprendizado por Máquina

O termo Aprendizado por Máquina, no contexto deste trabalho, refere-se à habilidade, exibida por um sistema de computação, de **aprender**. Esta habilidade pode ser aferida pela constatação de *qualquer acréscimo automático no desempenho de um sistema de computação, ao longo do tempo, como resultado da experiência* [05].

Para tanto, o sistema deve dispor de um **algoritmo de aprendizado**, que buscará realizar uma ou mais das seguintes coisas:

1. Cobrir uma faixa maior de problemas
2. Fornecer soluções mais acuradas
3. Obter respostas a um custo menor
4. Simplificar o conhecimento codificado

A simplificação do conhecimento armazenado possui um valor intrínseco, ou seja, expressões mais claras tornam o conhecimento mais acessível às pessoas, mesmo que isto não se traduza numa melhora de desempenho computacional.

O algoritmo empregado no modelo de aprendizado por Refinamento de Habilidade procura realizar o item três, ou seja, busca gerar planos, para o disparo das regras pelo motor de inferência, cada vez mais curtos, de maneira que a solução desejada possa ser alcançada com um menor esforço computacional.

Qualquer sistema projetado para modificar e melhorar o seu próprio desempenho (Figura 3.1) deve incluir os seguintes componentes principais:

1. Um conjunto de estruturas de informação que codifiquem o nível presente de especialização do sistema *(a Base de Conhecimento, B.C.)*

2. Um algoritmo que use as regras para guiar as suas atividades *(o Executor)*

3. Um módulo de **feedback** (realimentação) que compare os resultados reais obtidos com os resultados desejáveis *(o Crítico)*

4. Um mecanismo de aprendizado que use o feedback fornecido pelo crítico para corrigir as regras *(o Aprendiz)*

No algoritmo de aprendizado que implementamos podemos observar nítidamente os itens um - Base de Conhecimento e dois - o Executor ou Motor de Inferência. Já os itens três e quatro apresentam algumas diferenças: não se conhece, a princípio, o resultado desejável (o plano ótimo para solucionar o problema). O que se faz é, então, ao se chegar à uma solução do problema, premiar as regras disparadas que se encontram no caminho solução do problema (um plano), modificando seu peso para uma próxima rodada. Procura-se, desta forma, favorecer o disparo destas regras numa tentativa subsequente de se alcançar um caminho solução que seja mais curto - se possível, o ótimo.

Ao *Crítico*, no nosso caso, caberia verificar se o último plano gerado é satisfatório ou se deve ser buscado um novo plano para resolver o problema.

Além disso, o modelo não modifica o conhecimento presente nas regras da Base de Conhecimento. Esta prerrogativa é reservada a outros modelos de aprendizado, fugindo ao escopo deste trabalho.

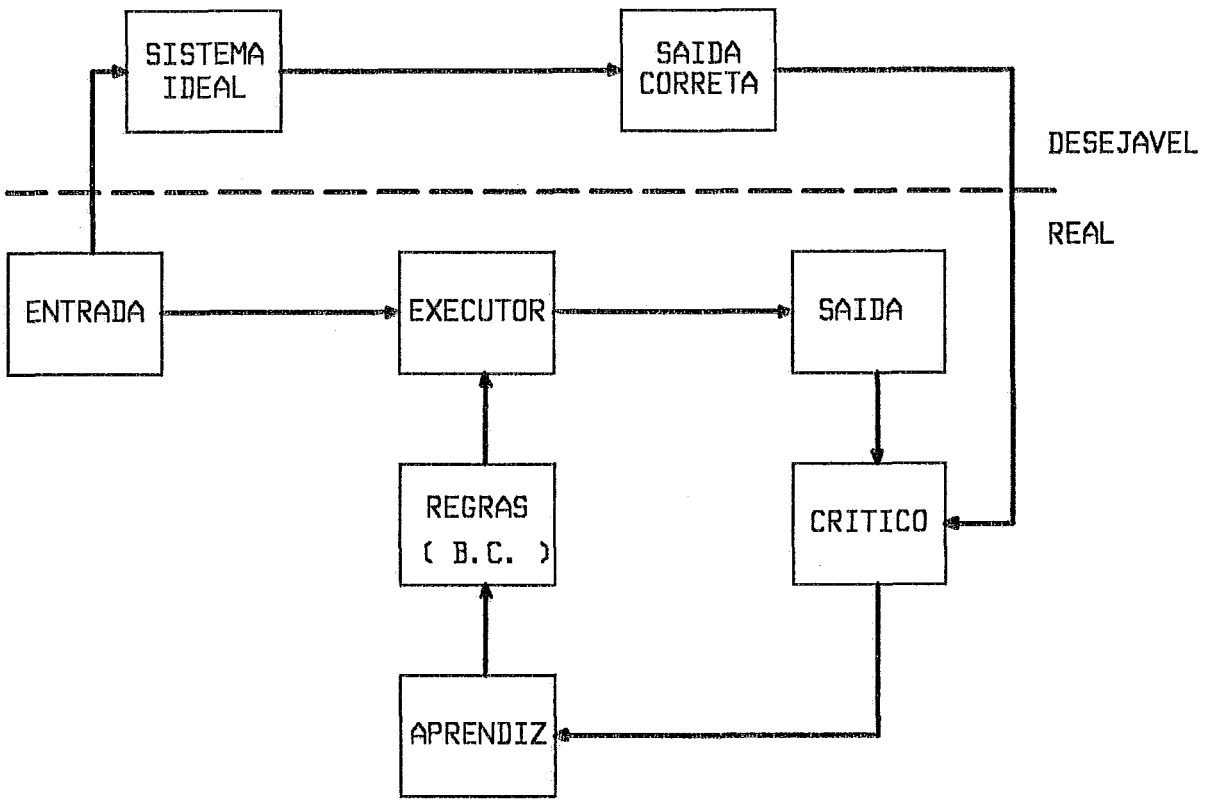


Figura 3.1: Componentes de Um Sistema Adaptativo

Os Sistemas Especialistas não adaptativos ficam muito limitados na sua potencialidade por não contarem com mecanismos de aprendizado. Falando das limitações atuais da tecnologia de Sistemas Especialistas, Stephen F. Smith [06] declara:

A construção de Sistemas Especialistas ainda permanece muito próxima de uma arte. Embora tenham aparecido ferramentas e metodologias para fornecer uma ajuda considerável a esta atividade, o processo de **elicitar, representar e refinar o conhecimento** *utilizado pelo especialista da área, permanece mal definido e consome muito tempo.*

A aplicação desta tecnologia permanece restrita ao domínio estreitíssimo dos problemas auto-contidos e o desempenho geralmente se degrada gravemente à medida que o sistema se aproxima dos limites de seu conhecimento. Há pouca habilidade para adaptar ou reorganizar o conhecimento, em função dos requisitos de desempenho que mudam ao longo do tempo.

Provavelmente, o maior potencial para fornecer soluções de longo prazo para este problema se situe na área de aprendizado por máquina

Entre as pesquisas correntes, que sugerem uma larga faixa de enfoques promissores para amenizar estes problemas, o mesmo autor cita:

Metodologias para auxílio à operacionalização

Aprendizado por Analogia

Generalização

Descobrimento

O modelo de aprendizado, baseado no Refinamento de Habilidade, que exploramos neste trabalho, se situa dentre a gama de pesquisas que procuram dar um maior poder ao Sistema Especialista, dotando-o de capacidade própria de adaptação às mudanças apresentadas em sua Base de Conhecimento.

3.2 Sistema Especialista e Aprendizado

Reproduzimos, a seguir, o que dizem os autores [07] do modelo de aprendizado que implementamos, sobre Sistemas Especialistas e capacidade de aprendizado.

"Os Sistemas Especialistas atuais tem evoluído bastante no suporte aos tomadores de decisão, todavia não podem simular totalmente os especialistas humanos. A habilidade de aprender é uma das características humanas mais importantes; carentes de uma capacidade substancial de aprendizado, os sistemas especialistas ficam criticamente limitados.

O aprendizado envolve dois fenomenos básicos - aquisição de conhecimento e refinamento de habilidade.

A aquisição de conhecimento se refere à fixação de uma nova informação simbólica (isto é, conhecimento descritivo). Isto pode também envolver a apreensão de conhecimento sobre o raciocínio e os procedimentos que permitam aos aprendizes aplicar a informação nova de maneira efetiva. Vários enfoques tem sido propostos para a aquisição automática de conhecimento.

O Refinamento de Habilidade é um tipo comum e importante de aprendizado, especialmente do ponto de vista gerencial. Um tipo de refinamento do conhecimento, o refinamento de habilidade usualmente se refere a um processo não simbólico, no qual a habilidade motora ou cognitiva gradualmente se aprimora através da prática repetida. O fenômeno da curva de aprendizado dá ilustrações bem conhecidas. Em qualquer organização os especialistas (por exemplo, gerentes ou técnicos) são normalmente gente experiente - que acumulou muito de sua experiência realizando tarefas repetidamente ao longo do tempo. Através da prática, são aprendidos os métodos de solução de problemas mais efetivos e eficientes, tornando-se parte da experiência de alguém. Refinamento

de Habilidade é um comportamento essencial para uma clara aquisição de conhecimento. Sua importância deve ser destacada.

Embora os Sistemas Especialistas sejam destinados a simular o comportamento de assessoria dos especialistas humanos, eles (diferentemente dos humanos) geralmente não aprendem a partir da experiência. Em outras palavras, os Sistemas Especialistas atuais não estão equipados com poder de refinamento de habilidade. Eles dependem inteiramente de Engenheiros de Conhecimento para alimentá-los com cada peça de conhecimento de raciocínio, habitualmente na forma de regras. Em geral, um melhor solucionador de problemas depende do Engenheiro de Conhecimento fornecer um conhecimento de raciocínio melhor ou mais abrangente. Entretanto, a inteligência do Sistema Especialista pode aumentar automatizando-se pelo menos parte do ajuste do conhecimento.

A tecnologia de I.A. tem feito apenas um modesto progresso sobre o problema do refinamento de habilidade. Embora alguns modelos simbólicos para o refinamento de habilidade tenham aparecido, nenhum deles utilizou os ciclos de reconhecimento-ação dos Sistemas Especialistas para refinar as habilidades do Sistema. Este trabalho descreve nossa pesquisa em equipar um sistema especialista baseado em regras com o comportamento de refinamento de habilidade através da utilização do mecanismo de controle de reconhecimento-ação - um passo em direção a sistemas especialistas que sejam mais inteligentes em virtude de uma habilidade de aprender a partir da experiência."

A capacidade de aprender com a prática torna o Sistema Especialista mais versátil, na medida em que ele fica capaz de se adaptar a uma nova configuração de regras na Base de Conhecimento. Para Bases de Conhecimento pequenas, esta faculdade pode não trazer muito ganho, uma vez que o próprio Engenheiro de Conhecimento estará habilitado a estruturar as Regras de maneira a obter um melhor desempenho do sistema. Já para Bases de Conhecimento grandes, a complexidade da tarefa de estruturação das regras pode ter um custo proibitivo. Aí, o ganho de desempenho adquirido com a prática, pode se tornar interessante.

4. Um modelo de Aprendizado

No modelo de aprendizado proposto, os autores [07] fazem uma analogia econômica, associando cada regra a um agente econômico e o conjunto de regras à sociedade econômica. A medida da riqueza de cada agente econômico tem seu correspondente na fôrça de cada regra.

Dois mecanismos interdependentes caracterizam este modelo :

- Um mecanismo de seleção de regras, usado no processo de resolução de conflitos - quando mais de uma regra tem condições de ser disparada - que utiliza um processo de seleção múltipla incorporando critérios de preferência. Tal mecanismo tem sua analogia no processo de escolha exibido por um consumidor, frente a várias opções, na hora da compra.

- Um mecanismo de atribuição de crédito baseado em economia (correspondendo a um procedimento de realocação de capital) que usa uma inferência da experiência da máquina para atualizar a potencialidade de cada regra que participa do processo de resolução do problema.

Estes dois mecanismos atuam em consonância com os ciclos de reconhecimento-ação do motor de inferência, que compreendem a verificação de qual regra tem condições de ser disparada e a atualização do estado dos descritores, ou fatos, na Base de Conhecimento, com a parte ação da regra selecionada.

Uma regra é considerada apta a ser disparada quando seus requisitos são todos possíveis de ser atendidos pelos fatos já existentes inicialmente ou deduzidos posteriormente e armazenados na Base de Conhecimento. Ao conjunto de todas as regras nestas condições, num determinado ciclo de inferência, chamamos de **conjunto de discórdia.**

Para escolher que regra, entre aquelas que integram o conjunto de discórdia, deve ser a escolhida, o mecanismo de seleção leva em consideração os seguintes fatores:

- Os valores dos atributos P, C, U e O de cada regra. Estes atributos, que podem variar de acordo com o desejo do Engenheiro de Conhecimento que constrói a Base de Conhecimento, têm os seguintes significados:

  P prioridade da regra, um indicador da importância da regra no domínio da aplicação.

  C custo da regra, uma medida do dispêndio envolvido no disparo da regra.

  U número de variáveis não definidas na cauda da regra.

  O ordem da regra no conjunto de regras que compõem a Base de Conhecimento.

- As faixas arbitradas para os critérios de preferência, ou seja, dentro de que valores de P, C, U e O devem estar as regras mais promissoras, na visão do Engenheiro de Conhecimento.

- O valor da força de cada regra no ciclo corrente. A fôrça de cada regra dá a medida em que cada regra contribuiu para o desempenho da tarefa a que o sistema se propõe realizar. No início da execução do sistema, todas as regras possuem a mesma fôrça. O mecanismo de realocação de capital redistribui estes créditos. Assim, regras que contribuirem mais para se atingir o objetivo do sistema, devem ter, no final, uma fôrça maior que as demais que participarem do processo.

O capital global do sistema permanece constante ao longo de toda execução do sistema, porém é realocado segundo as normas abaixo:

- Quando uma regra é selecionada, dentre as regras possíveis de serem as escolhidas em determinado momento pelo mecanismo de seleção de regras, ela paga uma taxa calculada sobre a sua potencialidade às demais regras que contribuiram para satisfazer as suas condições (cauda da regra). Se nenhuma regra contribuiu, então as condições foram satisfeitas por fatos já conhecidos inicialmente e o pagamento é feito ao **ambiente**.

- Ao final do processo, quando a tarefa do sistema é concluída, a regra que levou o sistema a atingir seu objetivo final recebe um pagamento equivalente a uma parte do capital acumulado pelo ambiente. O saldo de capital do ambiente é distribuído equitativamente entre as regras que contribuíram para o sucesso do sistema.

Ao conjunto das regras que levaram o sistema a alcançar o seu alvo, denomina-se **plano**.

No final de cada execução, avalia-se o plano e caso este não seja satisfatório, procede-se a uma nova execução do sistema, agora já com as forças das regras envolvidas modificadas pela rodada anterior. A repetição deste processo deve levar a uma solução mais econômica da tarefa, supondo-se que a cada rodada o sistema adquira mais habilidade para tratar o mesmo problema.

O critério de parada não está claro na proposição original do modêlo. Sugerimos o seguinte critério:

Um plano será considerado satisfatório se seu comprimento for menor ou igual ao comprimento do último plano satisfatório gerado pelo sistema. O primeiro plano é sempre satisfatório.

Um plano satisfatório será ótimo quando contiver apenas as regras necessárias e suficientes para a solução do problema - ou seja, não conterá regras repetidas.

Observamos, ao longo dos testes realizados com a implementação do modelo, que podem ocorrer oscilações no comprimento do plano gerado, ou seja, a rodada seguinte não leva, invariávelmente, a uma solução melhor que a rodada anterior. O que se observa é uma melhora ao longo de um certo número de execuções, número este que depende da taxa **c** de realocação de capital do sistema utilizada.

A Figura 4.1, extraída da monografia em que baseamos este trabalho, delineia o modelo conceitual que apresentamos acima.

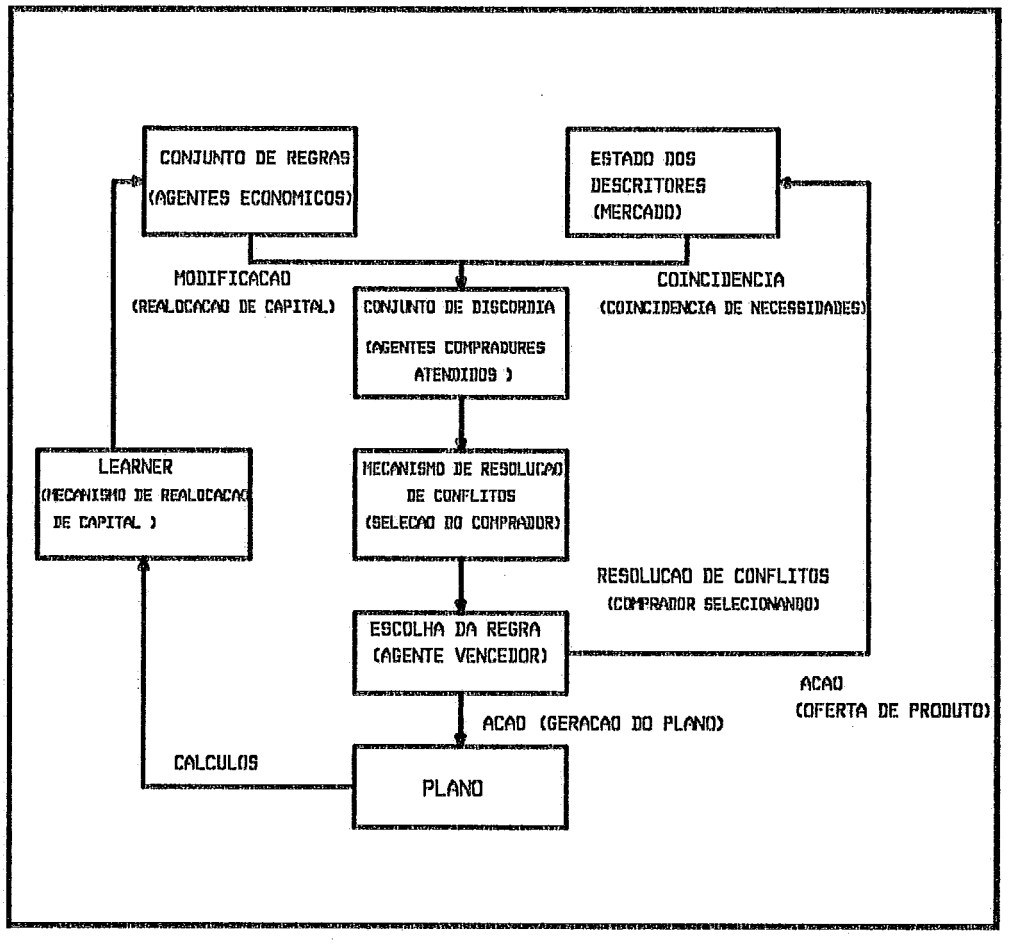


Figura 4.1. Um quadro conceitual para aprendizado por um Sistema Especialista

4.1 Descrição Matemática do Modelo

Seja o conjunto de discórdia gerado no t-ésimo ciclo de reconhecimento-ação denotado por CON(t) e o conjunto de regras por RS(t). Cada regra tem uma força, que é uma medida de sua contribuição histórica ao sistema. Seja a força de uma regra no t-ésimo ciclo denotado por S_t. Baseados na força, nós podemos definir um índice de importância relativa para cada regra no conjunto de regras, no ciclo t, como sendo

$$P_t(r^{(K)}) = \frac{S_t(r^{(K)})}{\sum_{r^{(i)} \in RS(t)} S_t(r^{(i)})}$$

onde

$r^{(K)}$: a regra K

$P_t(r^{(K)})$: o índice de importância relativa ou a probabilidade a priori da regra K no ciclo t

$S_t(r^{(K)})$: a força da regra K no ciclo t.

O índice de importância relativa pode ser também entendido como a probabilidade de cada regra sendo selecionada. Como este índice é definido sobre o conjunto inteiro de regras, nós o usamos como a probabilidade inicial de uma regra, no início de um certo ciclo de reconhecimento-ação.

Cada regra tem um valor para cada critério de seleção, que é considerado como um atributo desta regra. Assumimos que temos M critérios e portanto M valores de atributo para cada regra. Denotamos a lista de valores de atributos de uma regra por um vetor $A_K = (a_{K,1}, a_{K,2}, \ldots, a_{K,M})$, onde $a_{K,i}$ é o valor do atributo i para a regra K.

Podemos especificar uma faixa para cada critério como uma condição para a seleção de regras do conjunto de discórdia. A especificação da faixa para cada critério reflete, então, nossas preferências sobre estas M estratégias de seleção face

a um certo conjunto de discórdia. Pode-se representar a distribuição de preferência sobre estas M estratégias usando o conceito de probabilidade condicional. Seja cada estratégia denotada por CR(i), para i=1,2,...,M. A distribuição da preferência sobre estas M estratégias, face ao conjunto de discórdia no ciclo t, pode ser formulada como

$$P(CR(k) \mid \mathbf{CON}(t)) = \frac{\sum_{i \in \mathbf{CON}(t)} a_{ik}}{\sum_{i \in \mathbf{CON}(t)} \sum_{j=1}^{M} a_{ij}} \quad \text{, para k = 1, 2, ..., M} \qquad (1)$$

Cada estratégia irá gerar um subconjunto de regras a partir do conjunto de discórdia. Uma vez que assumimos M critérios, M subconjuntos irão ser gerados a partir do conjunto de discórdia, num ciclo t, baseados nestes M critérios. Seja cada subconjunto, num dado ciclo t, denotado por **RULE**(t,i) onde i = 1,2, ... ,M. Como podem existir algumas regras que não se enquadram dentro de nenhuma faixa especificada para algum critério, a união de todos estes subconjuntos gerados pode não ser a mesma coisa que o conjunto de discórdia. Em outras palavras,

$$\mathbf{CON}(t) \supseteq \bigcup_{i=1}^{M} \mathbf{RULE}(t,i)$$

e

$$\mathbf{RULE}(t,i) \cap \mathbf{CON}(t) = \mathbf{RULE}(t,i),\ i = 1,2, \ldots,M$$

Neste sistema, quando

$$\mathbf{CON}(t) \supseteq \bigcup_{i=1}^{M} \mathbf{RULE}(t,i)$$

e

$$\mathbf{CON}(t) \neq \bigcup_{i=1}^{M} \mathbf{RULE}(t,i)$$

o sistema irá invocar a estratégia de seleção default para gerar um subconjunto tal que estes dois subconjuntos irão ser idênticos. Além do mais, os M subconjuntos não são necessariamente disjuntos. Em outras palavras,

$$\mathbf{RULE}(t,i) \cap \mathbf{RULE}(t,j) \neq \phi, \text{ onde } i \neq j$$

pode muito bem ser verdade. Algumas regras que aparecem em um certo subconjunto podem também aparecer em outros subconjuntos.

O sistema gera cada um dos subconjuntos descritos acima, a partir de um certo conjunto de discórdia, baseado em certos critérios. Assim, podemos medir a ocorrência de cada um destes subconjuntos em têrmos da probabilidade condicional a posteriori

$$P(\mathbf{RULE}(t,k) \mid \mathbf{CON}(t), CR(k)), \text{ para k} = 1,2, \ldots ,\text{M}.$$

como se segue:

$$P(\mathbf{RULE}(t,k) \mid \mathbf{CON}(t), CR(k)) = \frac{P(\mathbf{RULE}(t,k) \cap \mathbf{CON}(t), CR(k))}{P(\mathbf{CON}(t), CR(k))}$$

$$= \frac{P(\mathbf{RULE}(t,k), CR(k))}{P(\mathbf{CON}(t), CR(k))}$$

Como $P(x,y) = P(x \mid y) \times P(y)$, a fórmula acima pode ser escrita como

$$\frac{P(CR(k) \mid \mathbf{RULE}(t,k)) \times P(\mathbf{RULE}(t,k))}{P(CR(k) \mid \mathbf{CON}(t)) \times P(\mathbf{CON}(t))} \quad \text{para k} = 1,2, \ldots ,\text{M} \qquad (2)$$

Nas fórmulas acima, podemos deduzir $P(CR(k) \mid \mathbf{RULE}(t,k))$ por um método semelhante ao utilizado para $P(CR(k) \mid \mathbf{CON}(t))$. Como a cada regra é atribuída uma prioridade inicial no início de cada ciclo, podemos escrever a prioridade inicial de **CON**(t) e **RULE**(t,k) como

$$P(CON(t)) = \sum_{r^{(i)} \in \mathbf{CON}(t)} P_t(r^{(i)})$$

e

$$P(RULE(t,k)) = \sum_{r^{(i)} \in \mathbf{RULE}(t,k)} P_t(r^{(i)})$$

Com base nestas informações, a fórmula 2 pode ser calculada.

Em seguida, a probabilidade de que uma regra seja escolhida a partir de um certo subconjunto (que foi obtido de algum conjunto de discórdia baseado em um certo critério) em um determinado ciclo pode ser expressa como

$$P(r^{(i)}_{\mathbf{RULE}(t,k)} \mid \mathbf{CON}(t), \mathbf{RULE}(t,k), CR(k)) = \frac{P(r^{(i)}, \mathbf{CON}(t), \mathbf{RULE}(t,k), CR(k))}{P(\mathbf{CON}(t), \mathbf{RULE}(t,k), CR(k))}$$

$$= \frac{P(r^{(i)} \cap \mathbf{CON}(t) \cap \mathbf{RULE}(t,k), CR(k))}{P(\mathbf{RULE}(t,k) \mid \mathbf{CON}(t), CR(k))}$$

$$= \frac{P(r^{(i)}, CR(k))}{P(\mathbf{RULE}(t,k) \mid \mathbf{CON}(t), CR(k))}$$

$$= \frac{P(CR(k) \mid r^{(i)}) \times P(r^{(i)})}{P(\mathbf{RULE}(t,k) \mid \mathbf{CON}(t), CR(k))}, \text{ para } k = 1,2, \ldots, M \qquad (3)$$

Na fórmula acima, obtemos $P(\mathbf{RULE}(t,k) \mid \mathbf{CON}(t), CR(k))$ via fórmula 2 e podemos calcular $P(CR(k) \mid r^{(i)})$ como fizemos na fórmula 1. Como descrito anteriormente, $P(r^{(i)})$ é o índice de importância relativa derivado da fôrça da regra. Então, formulamos a probabilidade de uma regra ser a escolhida dos M subconjuntos num ciclo t como

$$P(r^{(i)}{}_{\mathrm{RULE}(t,1),\ \mathrm{RULE}(t,2),\ldots,\mathrm{RULE}(t,M)}) = \sum_{j=1}^{M} P(r^{(i)}{}_{\mathrm{RULE}(t,j)} \mid \mathbf{CON(t),RULE(t,j)}, CR(j)) \qquad (4)$$

Definimos também a potencialidade de uma regra, num ciclo t, em termos de seus critérios de valores e sua fôrça, como se segue :

$$potencialidade_t(r^{(i)}) = \sqrt{a^2_{i,1} + a^2_{i,2} + \ldots + a^2_{i,M} + S^2_t(r^{(i)})} \qquad (5)$$

Podemos escrever, então, a potencialidade esperada de cada regra num ciclo t como

$$potencialidade\_esperada_t(r^{(i)}) = P(r^{(i)}{}_{\mathrm{RULE}(t,1),\ \mathrm{RULE}(t,2),\ldots,\mathrm{RULE}(t,M)}) \times potencialidade_t(r^{(i)}) \qquad (6)$$

A máquina de inferência irá escolher a regra r^*, que tem a maior potencialidade esperada entre as regras do conjunto de discórdia **CON**; assim

$$potencialidade\_esperada_t(r^*) = \max_{r^{(i)} \in \mathrm{CON}(t)} potencialidade\_esperada_t(r^{(i)}) \qquad (7)$$

A ação de selecionar a regra r^*, irá atualizar o descritor de estado. Esta, por sua vez, precisará pagar pela atualização e este pagamento será deduzido de sua força para o próximo ciclo. Neste ponto, fazemos a quantia deste pagamento ser alguma porção da potencialidade da regra r^*:

$$c \times potencialidade_t(r^*) \ , \text{ onde } 0 \leq c \leq 1$$

A força da regra r^* no próximo ciclo reconhecimento-ação, em t+1, será

$$S_{t+1}(r^*) = S_t(r^*) - c \times potencialidade_t(r^*) \ , \ 0 \leq c \leq 1 \qquad (8)$$

Este pagamento, por outro lado, será compartilhado entre aquelas regras que criaram o descritor de estado e permitiram que a condição da regra r^* fosse satisfeita. Elas são premiadas por estabelecer uma situação útil para a regra r^*. Estas regras são as fornecedoras de informação da regra r^*, e suas forças no próximo ciclo de reconhecimento-ação, t+1, ficam sendo

$$S_{t+1}(r') = S_t(r') + \frac{c \times potencialidade_t(r^*)}{| r' |} \text{, onde } 0 \leq c \leq 1 \qquad (9)$$

onde r' é o conjunto de fornecedores de informação da regra r^* no ciclo t.

Embora não esteja muito claro na enunciação do modelo, podemos inferir que quando a regra r^* se utiliza apenas de fatos já existentes na Base de Conhecimentos ao iniciar-se a tarefa, a regra efetua o pagamento estipulado acima ao "ambiente", que posteriormente (quando o sistema concluir a tarefa) irá redistribuir o capital acumulado proporcionalmente entre as regras que ajudaram a atingir a meta do sistema.

Neste fenômeno econômico, cada regra selecionada é considerada como um intermediário que paga seus fornecedores de informação e recebe pagamentos de seus consumidores de informação. A força de uma regra significa o registro de sua contribuição para o sucesso do sistema, enquanto sua potencialidade indica a habilidade potencial da regra de fazer uma contribuição no futuro. Usamos potencialidade aqui como uma espécie de mecanismo de compensação para aquelas regras com pouca força, de forma que o processo de seleção de regras tenha uma base mais equitativa.

Após o processo de modificação da força, o sistema modifica adequadamente o índice de importância relativa (ou probabilidade a priori) de cada uma das regras no conjunto de regras. A repetição dos ciclos de reconhecimento-ação resultam em uma cadeia de regras que constituem um **plano** para atingir a meta do sistema.

Neste momento, o ambiente terá acumulado um capital N, do qual $c \times N$ será deduzido para pagar a última regra selecionada pelo sistema (aquela que levou o sistema a atingir o objetivo desejado).

Todas as demais regras pertencentes ao conjunto P, que constitui o plano gerado, receberão um pagamento de

$$\frac{N \times (1 - c)}{| P |}$$

do ambiente.

5. Implementação de um SE Adaptativo

Para implementarmos o protótipo de SE adaptativo, baseado no modelo proposto, utilizamos como plataforma de hardware:

- Micro PC, compatível com o modelo IBM XT, com clock de 8MHz
- Disco rígido de 40 Megabytes
- Unidade de disco flexível de 5 ¼"
- Impressora matricial

A plataforma de software foi composta de :

- Sistema Operacional MS-DOS da Microsoft, versão 3.30
- Interpretador/Editor Arity PROLOG, versão 5.1
- Núcleo de Sistema Especialista PROLOG, do ICMSC-ILTC, modificado.

O produto final consistiu de quatro módulos principais, contituindo cada um deles um arquivo independente:

RODA.ARI Programa Principal. Prepara o ambiente para a chamada do Motor de Inferência, abriga os procedimentos básicos para operações com listas (utilizados pelos demais módulos) e simula as operações com janelas da Interface com o Usuário (ICU). A modificação deste programa, para realizar operações reais com janelas (o ARITY/PROLOG [08] dispõe de um produto denominado Dialog Boxes para este fim), pode ser feita com relativa facilidade.

O programa oferece as seguintes opções:

- Escolha do **objetivo** do sistema que se deseja alcançar
- Valor da constante de realocação de capital **c** que se pretende utilizar
- Refinamento de um plano já existente para o objetivo escolhido
- Saída no vídeo ou na impressora acoplada ao micro
- Informações adicionais para depuração ou elucidativas dos passos percorridos pelo Motor de Inferência, em particular as que dizem respeito ao algoritmo de seleção com aprendizado.

MIFC.ARI Motor de Inferência Forward Chaining, com algoritmo de aprendizado embutido. Elabora o processo de inferência, baseado no modelo proposto, e gera um plano para a realização da tarefa desejada. Este plano é armazenado, e caso se deseje um refinamento posterior, o MIFC toma como ponto de partida o último plano gerado para a tarefa.

MAUX.ARI Módulo de Rotinas Auxiliares do MIFC. Conjunto de procedimentos para obtenção de dados externos para o MIFC, acessos à Base de Conhecimentos, justificação de resultados obtidos (como e porque), tradução de variáveis e relações da Base de Conhecimentos e realização de outras funções necessárias ao funcionamento do Motor de Inferência.

BCEX.ARI Base de Conhecimentos do Problema Exemplo. Contém as regras, os fatos pré-estabelecidos, as perguntas e os procedimentos de crítica necessários à obtenção de dados externos ao MIFC e as traduções das relações que compõem a Base de Conhecimentos.

O foco de interesse deste trabalho está centrado no MIFC, portanto vamos nos limitar a descrever apenas este módulo com mais detalhes. Os demais módulos,

apesar de importantes para o trabalho como um todo, não apresentam nenhuma inovação digna de nota.

5.1 O Motor de Inferência Forward Chaining com Aprendizado

O texto do código PROLOG que implementa o MIFC é o seguinte:

```
forward :- verifica_goal,              % < tarefa terminada ? >
           paga_ultima_regra,          % paga regra que completou a tarefa
           paga_plano,                 % paga regras que pertencem ao plano
           mostra_plano.               % exibe o plano gerado


forward :- inicia_ciclo,               % comeca o ciclo de reconhecimento-acao
           testa_con,                  % testa se foi gerado conjunto discordia
           probabilidade_priori,       % calcula prob. a priori das regras
           gera_subcon,                % gera subconjuntos por criterios sel.
           testa_uniao,                % testa se uniao igual conj discordia
           estrategia_selecao,         % aplica estrategia de selecao
           potencialidade,             % determina potencialidade cada regra
           potencialidade_esperada,    % idem, potencialidade esperada
           seleciona_regra,            % seleciona regra max potenc. esperada
           modifica_forca,             % regra selecionada paga sistema
           dispara_regra,              % dispara a regra selecionada
           paga_regras,                % paga regras que contribuiram
           novo_ciclo,                 % identifica novo ciclo
           !,                          % ------------------------------------
           forward.                    % chama MIFC recursivamente
```

O primeiro parágrafo do procedimento **forward** é na realidade o último a ser *bem sucedido*, pois se constitui no **critério de parada** do procedimento, que é **recursivo**.

Vamos começar a descrever o procedimento pelo segundo parágrafo, que deve ser *bem sucedido* até que o objetivo do sistema seja alcançado. Cada cláusula, no parágrafo, constitui um procedimento PROLOG e pode ser vista como um subprograma do procedimento principal **forward**.

Cada iteração do procedimento recursivo forward constitui um **ciclo de reconhecimento-ação** ou, como vamos nos referir daqui para a frente, um **ciclo** do MIFC.

5.2 Descrição do MIFC

A descrição que daremos a seguir segue a estrutura da nossa implementação, na linguagem PROLOG, porém independe desta linguagem para seu entendimento, pois é dada a nível de funções de cada módulo.

Apresentamos, a seguir, a descrição funcional de cada subprograma:

inicia_ciclo Tem por função verificar, frente aos descritores de estado do problema (os fatos na BC), que regras são *disparáveis* e gerar o conjunto de discórdia **CON(t)**. Tem, ainda, no ciclo inicial do processo de inferência, a tarefa de normalizar os valores atribuídos a cada regra, relativos a cada critério de seleção **(P, C, U e O)** adotado.

testa_con Verifica se o **Conjunto de Discórdia** gerado é não vazio. Se **CON(t)** for vazio, o procedimento para, pois não existirá nenhuma regra para ser disparada, face aos descritores de estado do problema, no momento.

probabilidade_priori Realiza o cálculo da probabilidade **a priori** para todas as regras pertencentes ao conjunto de regras **RS(t)**. Para obter este valor, divide a **força** de cada regra, no ciclo t, pelo somatório das forças de todas as regras, no mesmo ciclo.

gera_subcon Gera os subconjuntos de regras, a partir do conjunto de discórdia, segundo os critérios de seleção, por faixas,

especificados. Foram definidas regras para enquadrar os atributos de cada regra segundo a faixa de aceitação pré-determinada para cada critério (P, C, U ou O). Assim, temos:

CR(1,P) :- P > = 70 .

CR(2,C) :- C = < 20 .

CR(3,U) :- U = < 2 .

CR(4,O) :- O = < 5 .

Por exemplo, o critério CR(1,P) é satisfeito pela regra Rn se o atributo P desta regra possuir um valor maior ou igual a 70.

testa_uniao Verifica se a união de todos os subconjuntos, gerados no procedimento anterior, iguala o conjunto de discórdia **CON(t)**. Caso isto não aconteça, é gerado um novo subconjunto englobando as regras do conjunto de discórdia não pertencentes ao conjunto união.

estrategia_selecao Verifica a estratégia de seleção a ser aplicada.

- A estratégia de seleção *default* consiste em fazer a probabilidade de seleção de cada regra, no conjunto de discórdia, igual à sua probabilidade **a priori**. A estratégia de seleção *default* é aplicada quando a união dos subconjuntos gerados em **gera_subcon** não iguala o conjunto de discórdia, no ciclo t.

- A estratégia de seleção *não default* consiste em calcular a **probabilidade de seleção** de cada regra, no conjunto de discórdia, baseada na sua probabilidade

a posteriori, avaliada pelo procedimento. É aplicada quando a união dos subconjuntos gerados em **gera_subcon** iguala o conjunto de discórdia.

potencialidade Calcula a **potencialidade** de cada regra, que consiste em somar os quadrados dos argumentos P,C,U e O ao quadrado da força da regra e obter a raiz quadrada desta soma.

potencialidade_esperada Calcula a **potencialidade esperada** das regras pertencentes ao conjunto de discórdia. Este valor é obtido multiplicando-se a probabilidade de seleção da regra (obtido segundo uma das estratégias de seleção possíveis) pela sua potencialidade.

seleciona_regra Verifica qual, dentre as regras presentes no conjunto de discórdia, tem a **maior** potencialidade esperada. Esta será a regra escolhida pelo MI para ser disparada no ciclo.

modifica_forca A regra selecionada *paga* ao ambiente um valor equivalente a **c** vezes a sua potencialidade. A mesma regra é, também, incluída no **plano** que está sendo gerado.

dispara_regra A regra escolhida em seleciona_regra é **disparada** e sua parte **ação** (a cabeça da regra) atualiza o descritor de estado do problema (um novo fato é acrescentado à Base de Conhecimento). O conjunto **R′**, dos fornecedores de informação para a regra disparada, é atualizado.

paga_regras Efetua o pagamento às regras que contribuiram para a satisfação dos requisitos da regra selecionada. Se o conjunto **R′** for **vazio**, o pagamento, de **c** vezes o

potencial da regra disparada, é feito ao **ambiente**. Se o conjunto **R'** for **não vazio**, o pagamento é feito, em bases proporcionais, às regras pertencentes ao conjunto.

novo_ciclo Prepara o início de um novo ciclo de reconhecimento-ação do Motor de Inferência. O ciclo passa do valor **t** para **t + 1**.

forward Chamada **recursiva** do procedimento principal, para proceder a um novo ciclo de reconhecimento-ação.

O primero parágrafo do MIFC possui os seguintes subprogramas:

verifica_goal Verifica se o objetivo do problema já foi atingido. É o teste da condição de parada do procedimento recursivo **forward**.

paga_ultima_regra A regra que levou o MIFC a concluir com sucesso a sua tarefa, recebe um pagamento especial de **c** vezes o valor do capital acumulado pelo **ambiente**.

paga_plano O restante do capital acumulado pelo ambiente é rateado em partes iguais por todas as regras que compõem o plano gerado nesta rodada do MIFC.

mostra_plano O plano gerado é apresentado ao usuário, para que este julgue se está **satisfatório** ou não. Tal plano é constituído pelo conjunto de todas as regras que foram disparadas até a consecução do objetivo do sistema.

Ao leitor interessado em ver a programação PROLOG de cada um destes subprogramas, tal informação pode ser encontrada nos Anexos a este trabalho.

6. Uma Aplicação

A princípio, qualquer aplicação, representada em uma Base de Conhecimento, poderia ser usada para demonstrar e avaliar o funcionamento do modelo implementado pelo nosso protótipo. Porém, uma em particular atende a um requisito importante para uma melhor avaliação: a Base de Conhecimento do exemplo escolhido no artigo em que o modelo é apresentado. Usando a **mesma** Base de Conhecimento podemos fazer um batimento de resultados, os previstos no modelo e os obtidos pela nossa implementação.

O problema, no exemplo, consiste em determinar novas quotas trimestrais para vendedores de livros, baseando-se em influências diversas, tais como as vendas do vendedor no último trimestre, a perspectiva econômica, os investimentos em propaganda local, a taxa de desemprêgo e a linha de produto com a qual o vendedor trabalha.

Regras para recomendação de novas quotas trimestrais

REGRA: R1
SE: *SALES* > 1.15 × *QUOTA*
ENTÃO: *BASE* = *QUOTA* + (*SALES* − 1.15 × *QUOTA*)
EXPLANAÇÃO: Nos casos em que as vendas deste produto excedam a quota em mais do que 15%, a quantidade base para a nova quota é feita igual a quota anterior, mais a quantidade de vendas em excesso.
ATRIBUTOS: P = 70; C = 20; U = 2; O = 1

REGRA: R2
SE: *SALES* ≤ 1.15 × *QUOTA*
ENTÃO: *BASE* = *QUOTA*
EXPLANAÇÃO: A quantidade base para a nova quota é a mesma que a quota passada porque as vendas deste produto não ultrapassaram a quota passada em mais do que 15%.
ATRIBUTOS: P = 60; C = 10; U = 2; O = 2

REGRA: R3
SE: *ECONOMY* = "good" e *KNOWN*(**GROWTH**)
ENTÃO: *EFACTOR* = *GROWTH*
EXPLANAÇÃO: Quando a perspectiva econômica local é boa, o fator econômico iguala a taxa de crescimento antecipada da economia.
ATRIBUTOS: P = 80; C = 8; U = 2; O = 3

REGRA: R4
SE: *ECONOMY* = "fair" e *KNOWN*(**LOCALADS**) e *KNOWN*(**GROWTH**)
ENTÃO:
EFACTOR = *GROWTH*/3; *LAFACTOR* = *LOCALADS*/120000
EXPLANAÇÃO: Quando a perspectiva econômica local é regular, o fator econômico é um têrço da taxa de crescimento e o fator de propaganda local é 1/120000 da quantia negociada para propaganda local.
ATRIBUTOS: P = 60; C = 20; U = 3; O = 4

REGRA: R5
SE: *ECONOMY* = "poor" e *KNOWN*(**GROWTH**) e *KNOWN*(**UNEMPLOYMENT**)
ENTÃO:
EFACTOR = min(*GROWTH*, 0.085 − *UNEMPLOYMENT*)
EXPLANAÇÃO: Se a perspectiva econômica local é fraca, então o fator econômico deve ser o menor entre a taxa de crescimento e o resultado da diferença entre 8.5% e a taxa de desemprêgo.
ATRIBUTOS: P = 55; C = 25; U = 3; O = 5

REGRA: R6
SE: *GROWTH* ≥ 0.04 e *UNEMPLOYMENT* < 0.076
ENTÃO: *ECONOMY* = "good"
EXPLANAÇÃO: A perspectiva econômica é boa porque a taxa de desemprêgo projetada está abaixo de 7.6% e a taxa de crescimento prevista é de, pelo menos, 4%.
ATRIBUTOS: P = 85; C = 8; U = 2; O = 6

REGRA: R7
SE: *GROWTH* ≥ 0.02 e *GROWTH* < 0.04 e *UNEMPLOYMENT* < 0.055
ENTÃO: *ECONOMY* = "good"
EXPLANAÇÃO: A perspectiva econômica é boa porque a taxa de desemprêgo projetada está abaixo de 5.5% e a taxa de crescimento prevista está entre 2 e 4%.
ATRIBUTOS: P = 80; C = 12; U = 3; O = 7

REGRA: R11
SE: *ECONOMY* = "poor" e *LOCALADS* < 1500
ENTÃO: *LAFACTOR* = −0.015
EXPLANAÇÃO: Quando a perspectiva econômica é ruim e os gastos com propaganda local, para o produto, são modestos, então o fator de propaganda local é negativo.
ATRIBUTOS: P = 50; C = 10; U = 2; O = 11

REGRA: R12
SE: (*ECONOMY* = "poor" e *LOCALADS* ≥ 1500) ou (*ECONOMY* = "good" e *LOCALADS* < 2000)
ENTÃO: *LAFACTOR* = 0
EXPLANAÇÃO: O fator de propaganda local é desprezível devido à pouca propaganda numa economia boa ou à uma economia ruim aliada a uma substancial propaganda local para a linha de produto.
ATRIBUTOS: P = 40; C = 15; U = 2; O = 12

REGRA: R13
SE: *PRODUTO em* ["computer","romance","scifi"]
ENTÃO: *PFACTOR* = (*NEWTITLES* + *OLDTITLES*) / *OLDTITLES* − 1; *STRONG* = *TRUE*; *WEAK* = *FALSE*
EXPLANAÇÃO: Esta é uma linha de produto forte. O fator produto é baseado no crescimento do número de títulos desta linha.
ATRIBUTOS: P = 35; C = 30; U = 1; O = 13

REGRA: R14
SE: *PRODUTO em* ["reference", "biography", "psychology", "sports"]
ENTÃO: *PFACTOR* = 0.75 × ((*NEWTITLES* + *OLDTITLES*) / *OLDTITLES* − 1); *STRONG* = *FALSE*; *WEAK* = *FALSE*
EXPLANAÇÃO: Esta não é uma linha de produtos forte, nem fraca. O fator produto é proporcional a três quartos do crescimento do número de títulos nesta linha.
ATRIBUTOS: P = 35; C = 30; U = 1; O = 14

REGRA: R15
SE: *STRONG* e *KNOWN*(**BASE**) e *KNOWN*(**EFACTOR**) e *KNOWN*(**PFACTOR**) e *KNOWN*(**LAFACTOR**)
ENTÃO: Entre com o número RISE em "Entre o percentual de crescimento esperado das vendas devido ao aumento de interesse por" PRODUTO;
NEWQUOTA = *BASE* ×
(1 + *EFACTOR* + *LAFACTOR* + *PFACTOR* + *RISE*/100)
EXPLANAÇÃO: Esta é uma linha de produto forte. Quantidade base, fator econômico, fator produto e fator de propaganda local para o cálculo da nova quota, são todos conhecidos. Um palpite subjetivo sobre o crescimento esperado das vendas, devido ao crescente interesse geral no produto, é solicitado. A nova quota é então calculada.
ATRIBUTOS: P = 30; C = 40; U = 5; O = 15

REGRA: R16
SE: *WEAK* e *KNOWN*(**BASE**) e *KNOWN*(**EFACTOR**) e *KNOWN*(**PFACTOR**) e *KNOWN*(**LAFACTOR**)
ENTÃO: Entre com o número FALL em "Entre o percentual de retração esperado das vendas devido à diminuição de interesse por" PRODUTO;
NEWQUOTA = *BASE* ×
(1 + *EFACTOR* + *LAFACTOR* + *PFACTOR* + *FALL*/100)
EXPLANAÇÃO: Esta é uma linha de produto fraca. Um palpite subjetivo sobre o declinio esperado das vendas, devido ao decrescente interesse geral no produto, é solicitado. A nova quota é então calculada.
ATRIBUTOS: P = 30; C = 40; U = 5; O = 16

Regras para recomendação de novas quotas trimestrais

REGRA: R8
SE: *GROWTH* ≥ 0.02 e *GROWTH* < 0.04 e *UNEMPLOYMENT* ≥ 0.055 e *UNEMPLOYMENT* < 0.082
ENTÃO: *ECONOMY* = "fair"
EXPLANAÇÃO: A perspectiva econômica é regular devido à taxa moderada de crescimento e às expectativas de desemprêgo.
ATRIBUTOS: P = 75; C = 16; U = 4; O = 8

REGRA: R9
SE: *GROWTH* < 0.02 ou *UNEMPLOYMENT* ≥ 0.082
ENTÃO: *ECONOMY* = "poor"
EXPLANAÇÃO: A perspectiva econômica é fraca porque ou a taxa de crescimento é muito baixa ou a expectativa de desemprêgo é alta ou ambas as coisas.
ATRIBUTOS: P = 80; C = 8; U = 2; O = 9

REGRA: R10
SE: *ECONOMY* = "good" e *LOCALADS* > 2000
ENTÃO: *LAFACTOR* = *LOCALADS*/100000
EXPLANAÇÃO: Quando a economia está bem e a propaganda local ultrapassa os 2000 dólares, o fator de propaganda local é 1% de cada cem mil dólares dispendidos.
ATRIBUTOS: P = 60; C = 12; U = 2; O = 10

REGRA: R17
SE: *NOT*(*WEAK* ou *STRONG*) e *KNOWN*(BASE) e *KNOWN*(EFACTOR) e *KNOWN*(PFACTOR) e *KNOWN*(LAFACTOR)
ENTÃO: *NEWQUOTA* = *BASE* × (1 + *EFACTOR* + *LAFACTOR* + *PFACTOR*)
EXPLANAÇÃO: Esta não é uma linha de produto especialmente forte ou fraca. Sua nova quota é calculada a partir da quantidade base e dos fatores para a economia, propaganda local e expansão da linha de produto.
ATRIBUTOS: P = 35; C = 25; U = 6; O = 17

REGRA: R18
SE: NOT (*PRODUTO em* ["computer", "romance", "scifi", "reference", "biography", "psychology", "sports"])
ENTÃO: *PFACTOR* = 0.45 × ((*NEWTITLES* + *OLDTITLES*) / *OLDTITLES* − 1); *STRONG* = *FALSE*; *WEAK* = *TRUE*
EXPLANAÇÃO: Esta é uma linha de produto fraca. O fator produto é proporcional a menos da metade de seu crescimento em títulos.
ATRIBUTOS: P = 40; C = 45; U = 1; O = 18

Feita a escolha, algumas adaptações foram necessárias para tornar o exemplo adequado à implementação:

- Codificação das Regras

As regras na Base de Conhecimento são representadas pelo predicado

regra(Id, [P, C, U, O], Lista_Ação, Lista_Requisitos)

onde

Id — Cadeia de caracteres que identifica a regra.
Exemplo: *R01*

[P, C, U, O] — Atributos da regra, segundo cada um dos critérios P, C, U e O.
Exemplo: *[70, 20, 2, 1]*

Lista_Ação — Lista de cláusulas que compõem a cabeça da regra.

Exemplo: *[economia(fraca), investimento(baixo)]*

Lista_Requisitos Lista ou listas de cláusulas que compõem a cauda da regra. Pode ser uma lista, uma conjunção ou uma disjunção de listas.
Exemplo: *([propaganda(X), X>=1500] ; [propaganda(Y), Y<2000])*

- Codificação dos Descritores de Estado

Os fatos na Base de Conhecimento são representados pelo predicado

fato(X)

Exemplo: *fato(vendas(24000)).*

- Declaração das Relações Primitivas

Algumas relações que se achavam implícitas na definição da Base de Conhecimento do exemplo, foram declaradas como **primitivas**, atendendo à classificação dada no trabalho desenvolvido pelo ICMSC-ILTC [04].

São elas:

known(X) X é conhecido. Implementado por um procedimento que verifica se X é um fato na Base de Conhecimento ou uma relação provada pelo Motor de Inferência.

min(X,Y,Z) Z é o mínimo entre X e Y. Um procedimento que verifica qual dos dois valores é o menor, X ou Y, e atribui este valor ao argumento Z.

- Declaração das Relações Perguntáveis

Os valores, cuja obtenção dependia de intervenção do usuário, foram definidos como relações perguntáveis. São eles:

lerize(X) Indaga o valor da taxa de crescimento esperada para as vendas do produto X

lefall(X) Indaga o valor da taxa de diminuição esperada para as vendas do produto X

- Declaração das Relações Explicáveis

Este item era opcional, porém julgamos conveniente empregá-lo, para melhorar a apresentação dos resultados pelo sistema. Seu uso permite que as relações sejam exibidas *traduzidas* na interface com o usuário. Foram explicadas todas as relações na Base de Conhecimento e ainda alguns operadores.

- Declaração das Perguntas

Foram codificadas as perguntas relativas às relações perguntáveis e os procedimentos que efetuam a crítica dos valores dados como resposta pelo usuário.

O produto final destas adaptações constitui o módulo **BCEX.ARI**, que se encontra documentado na íntegra nos Anexos.

Uma descrição detalhada da aplicação pode ser encontrada sob o título *"Um exemplo de Aprendizado"*, constante do artigo de Pi_Sheng et alli, citado na bibliografia [07].

6.1 Exemplo de uma execução do sistema

Uma vez carregada a Base de Conhecimento com os fatos e as regras pertinentes à aplicação que se quer executar, o sistema estará apto a fornecer as respostas desejadas, com a exibição do plano gerado até a sua obtenção.

Logo que começa a executar, o sistema faz as seguites indagações ao usuário:

- Qual a resposta desejada ?

  Responder com a meta a ser alcançada pelo sistema. Por exemplo newquota(X), para obter a nova cota de um vendedor, no problema exemplo.

- Qual a constante de Realocação de Capital ?

  Informar o valor de **c**, entre zero e um, que será considerado.

- Refinamento de Plano (s / n) ?

  Responder com **s** (sim) caso voce queira que o sistema parta dos valores de forças das regras obtidos em uma execução anterior do sistema. O 'default' é **n**, condição que leva o sistema a partir dos valores iniciais atribuidos para as forças das regras. No exemplo apresentado, todas as regras tem um valor inicial de força igual a 100.

- Saída em impressora (s / n) ?

  Responder com **s** (sim) caso voce queira que o sistema exiba suas respostas na impressora. O 'default' é **n**, o que faz com que todas as respostas sejam exibidas no vídeo.

- Depuração ativada (s / n) ?

A resposta **s** (sim) fará com que sejam exibidos no dispositivo de saída (vídeo ou impressora) todos os passos intermediários dos algoritmos de seleção de regras a cada ciclo de reconhecimento-ação do Motor de Inferência. A finalidade desta opção é permitir um melhor entendimento do modelo.

Uma vez respondidas estas perguntas, o sistema inicia o processo de inferência. É exibida uma tela composta das quatro seguintes janelas:

Pergunta Quando, para prosseguir a inferência, for necessário solicitar algum dado adicional ao usuário, a pergunta pertinente é exibida nesta janela.

Resposta Destina-se à entrada de dados, por parte do usuário, em resposta às perguntas feitas a ele.

Crítica/Prova Tem duas finalidades. A primeira, de crítica, é exibir faixas de valores aceitáveis para as respostas do usuário, quando estas forem recusadas pelo sistema. A segunda, de prova, é mostrar cada conclusão a que chegar o sistema, baseado nas regras e fatos da Base de Conhecimento, até o resultado final.

Mensagem Comunicação do sistema com o usuário: mensagens do tipo "Tecle enter para prosseguir", etc...

Finalmente, quando a resposta à consulta efetuada pelo usuário é alcançada, são exibidas as seguintes informações no dispositivo de saída:

- Plano gerado pelo sistema
- A resposta obtida
- Tempo de duração da consulta em minutos e segundos.

Um exemplo de saída é mostrado a seguir:

Plano **1** : **[R01, R01, R08, R04, R08, R13, R15]**

O resultado obtido para **newquota** foi: **27720.0**

Esta consulta levou **0** minutos e **30.26** segundos.

Neste momento, o usuário será solicitado a responder se deseja ver ou não a árvore de inferência gerada pelo sistema durante o processo de obtenção da resposta procurada. Se o usuário responder afirmativamente, será aberta uma janela no vídeo onde todas as conclusões serão justificadas, em linguagem o mais natural possível para que seja inteligível ao leigo.

Com pequenas alterações no código, pode-se programar o sistema para gerar N planos consecutivos, para parar a cada plano gerado e indagar ao usuário se é satisfatório ou não, para só exibir planos que sejam melhores ou iguais aos já gerados, etc... A escolha vai depender do uso que se dará ao sistema.

7. Resultados Obtidos e Conclusões

Nosso trabalho consistiu, até o momento, em descrever Sistema Especialista, comentar a limitação que a falta de aprendizado impõe a estes sistemas e apresentar um modelo que, teoricamente, fornece uma maneira de um Sistema Especialista aprender.

Como forma de avaliar o modelo proposto, fizemos a sua implementação em computador, conforme está descrita nos capítulos anteriores.

Adotamos a estratégia de rodar uma aplicação piloto, com o seguinte propósito:

- Se a aplicação piloto apresentasse bons resultados, nosso estudo deveria ser dirigido a uma averiguação do comportamento do modelo face a outros tipos de aplicações.

- Caso a aplicação piloto apresentasse resultados desfavoráveis, então voltaríamos nossos estudos para apontar as possíveis falhas do modelo e, se possível, mostrar onde e como o modelo deveria ser ajustado para corrigir estas falhas.

Os resultados obtidos com a aplicação piloto e as nossas conclusões serão expostas a seguir.

7.1 Resultados dos Testes Preliminares

Nossa primeira preocupação foi em verificar a fidelidade da implementação ao modelo em que nos baseamos. Daí a aplicação, por nós utilizada, ter sido a mesma sugerida como exemplo pelos idealizadores do modelo, apenas com as regras de produção adaptadas à implementação.

Em cinco rodadas sucessivas, obtivemos os seguintes resultados:

Plano 1: [R01, R01, R01, R08, R04, R08, R08, R04, R08, R08, R04, R08, R13, R15]

Plano 2: [R01, R01, R01, R08, R04, R13, R15]

Plano 3: [R08, R04, R08, R13, R08, R04, R08, R13, R08, R04, R08, R01, R15]

Plano 4: [R01, R01, R08, R013, R04, R15]

Plano 5: [R01, R08, R013, R04, R15]

Comparando estes resultados com os previstos no modelo, vimos que atingimos resultados semelhantes, para uma constante de realocação de capital de 1/100. No artigo tais resultados eram relacionados a um valor de 1/10 para a constante **c**. Este valor foi informado erroneamente, um provável erro tipográfico, pois daria resultados matematicamente conflitantes com o modelo.

Uma questão deixada em aberto pelos idealizadores do modelo, diz respeito à constante de realocação de capital, c, que pode assumir valores entre zero e um.

Para termos uma noção de como se comporta o modelo em relação às variações no valor arbitrado para a constante **c**, e avaliarmos o sistema em termos computacionais, rodamos o exemplo, refinando até o décimo plano, para diferentes valores dessa constante. Os resultados foram tabulados e se encontram na figura a seguir.

Valor da Constante c	Planos e Número de Regras por Plano										Número de Planos Ótimos
	01	02	03	04	05	06	07	08	09	10	
0.0000	*	*	*	*	*	*	*	*	*	*	0
0.0005	180	45	9	6	6	7	6	6	5	165	1
0.0010	90	26	9	5	6	5	84	24	88	27	2
0.0050	20	9	8	6	6	23	9	5	19	8	1
0.0100	14	7	13	6	5	12	6	13	6	5	2
0.0500	6	6	5	6	6	8	6	6	6	6	1
0.1000	6	5	6	6	5	6	6	8	5	6	3
0.2500	6	5	6	6	5	6	6	5	6	6	3
0.5000	6	5	6	5	6	5	6	5	6	5	5
0.7500	6	5	6	5	6	5	6	5	6	5	5
0.9500	6	6	6	7	6	7	6	9	6	9	0
1.0000	8	8	*	*	*	*	*	*	*	*	0

Figura 7.1: Influência da constante c no modelo

Os tempos de resposta obtidos oscilaram entre 12 minutos, para um plano com 180 regras e alguns segundos para planos próximos do ótimo, em torno de 5 regras.

Qual seria o valor de *c* ideal, aquele que levaria o sistema a atingir o seu objetivo com um mínimo de disparos de regras ? No exemplo rodado, valores de c na faixa de 0.5 a 0.75 apresentaram os melhores resultados. Já para valores muito próximos de zero ou de um (limites extremos de variação da constante) os resultados foram piores; em particular, para o valor zero e para o valor um. Isto era esperado, pois os mecanismos de seleção de regras - realocação de capital tornam-se ineficientes para valores limite da constante c.

O comportamento observado nesses dez refinamentos sucessivos parece se manter ao longo de refinamentos posteriores.

Para ilustrar, rodamos o modelo cem vezes seguidas, e obtivemos os seguintes resultados:

Valor da constante c arbitrado :	1/100
Número máximo de regras num plano :	16
Número mínimo de regras num plano :	5
Tempo máximo para gerar um plano :	2m 19.45s
Tempo mínimo para gerar um plano :	0m 45.31s
Número de planos ótimos obtidos :	16

Podemos observar que o número de regras em cada plano gerado ficou limitado ao intervalo [5, 16].

7.2 Avaliação do Modelo Frente aos Resultados

Se tomarmos como base de avaliação os resultados que obtivemos nas sucessivas rodadas do problema exemplo (Figura 7.1), podemos tecer as seguintes considerações:

- O plano 5 é um plano ótimo. Pode-se afirmar isso pois para que a regra R15 tenha seus requisitos satisfeitos, todas as demais regras disparadas ao longo do plano são necessárias. Além disso, nenhuma regra foi disparada mais do que uma vez, logo apenas as regras necessárias e suficientes foram disparadas. Isto elimina qualquer possibilidade de se alcançar o objetivo do sistema com um plano menor (com menos de cinco regras).

  O sistema foi capaz de chegar a um plano ótimo após ter feito ajustes nas forças das regras da Base de Conhecimento.

- Introduzimos uma regra "bôba" (que não contribui para a solução do problema), R0,

  REGRA: R0
  SE: SALES > 0
  ENTÃO: VENDEU
  ATRIBUTOS: P = 70; C = 20 ; U = 1 ; O = 0

na Base de Conhecimento e rodamos dez vezes a aplicação. Após duas rodadas sucessivas, a regra R0 não mais apareceu em nenhum plano, com a constante c fixada em 0.5.

O sistema abandona regras que não fazem parte do caminho solução do problema.

- Planos ora menores ora maiores foram gerados ao longo das sucessivas rodadas do sistema.

Esta constatação nos deixa perplexos face ao pressuposto de que o sistema exibiria um melhor desempenho na próxima vez que executasse, pois estaria refinando sua habilidade em resolver o problema. Seu comportamento parece mais o de uma busca dirigida heuristicamente.

O sistema poderia ser dotado de um módulo que faria o papel do "crítico", não aceitando as soluções que levassem a planos de comprimento superior ao melhor plano obtido até o momento. Esta seria uma solução de contorno, pois os planos insatisfatórios continuariam a ser gerados internamente pelo sistema.

A constatação de ter se obtido um plano ótimo pode não ser trivial quando se estiver lidando com uma Base de Conhecimento mais complexa. Os autores sequer fazem referência à questão da garantia de se atingir um plano ótimo no decorrer de sucessivos refinamentos - pensamos que não há certeza, para qualquer aplicação e valores distintos de c, disto acontecer sempre.

Mudanças em fatos na Base de Conhecimento, que afetam o caminho solução do problema, não alteraram significativamente o comportamento do modelo. Foram feitos testes com o problema exemplo, trocando-se produto para esporte, vendas para 20000 e taxa de crescimento da economia para 0.01.

7.3 Identificação dos Pontos Fracos

Seria interessante que se elaborasse um método para determinar a constante c que não fosse baseado na experimentação, pois uma escolha infeliz pode levar à geração de planos com um número elevado de regras e tornar mais difícil ou mesmo impossível, se alcançar um plano ótimo. Podemos adiantar, contudo, que não há uma resposta trivial, pois c, a taxa de transferência de capital, determina quanto uma regra paga ao ser disparada, mas também acarreta um pagamento proporcional a quem contribuiu para seu sucesso (outras regras ou o ambiente),

permanecendo inalterado o capital total no sistema. Isto implica numa interdependência entre as regras, com resultados difíceis de prever.

Exemplificando o que acabamos de citar, realizamos um teste com o modelo, alterando-o ligeiramente para que o disparo sucessivo de uma mesma regra fosse coibido. A alteração consistiu em penalizar a regra que estivesse sendo disparada novamente com um pagamento proporcionalmente maior ao número de vezes que ela já havia sido selecionada. A diferença entre o pagamento normal e este novo pagamento foi creditado ao ambiente, para não favorecer indevidamente às regras fornecedoras de informação.

Matematicamente, definimos uma nova constante, c' , segundo a fórmula abaixo, onde N é o número de ocorrências anteriores da regra :

$$c' = c + N / 10$$

quando o lado direito da igualdade for menor ou igual a 1; caso contrário, $c' = 1$. Observar que para $N = 0$ o valor de c' é igual a c e para qualquer outro valor de N se mantem dentro do intervalo [0,1].

Os resultados obtidos com a nova constante foram tabulados e se encontram na figura a seguir.

Valor da Constante c′	Planos e Número de Regras por Plano										Número de Planos Ótimos
	01	02	03	04	05	06	07	08	09	10	
0.0000	8	8	7	9	10	7	9	8	8	7	0
0.0005	8	8	7	9	10	7	9	8	8	7	0
0.0010	8	8	7	9	10	7	9	8	8	7	0
0.0050	8	8	6	11	9	9	11	8	10	8	0
0.0100	8	7	6	8	8	7	8	8	8	6	0
0.0500	6	6	8	6	6	8	6	6	8	6	0
0.1000	6	6	9	6	9	7	8	7	7	6	0
0.2500	6	5	6	5	6	5	6	5	6	5	5
0.5000	6	5	6	5	6	5	6	5	6	5	5
0.7500	6	5	5	6	5	6	5	6	5	6	5
0.9500	6	13	10	*	*	*	*	*	*	*	0
1.0000	8	8	*	*	*	*	*	*	*	*	0

Figura 7.2: Influência da constante c′ no modelo

Comparando estes resultados com os da Figura 7.1, observamos que atingiu-se o objetivo de coibir os disparos subsequentes de uma mesma regra, no caso de R1, porém aumentou-se a alternância entre R8 e R4 (R8 é fornecedora de informação para R4). O comprimento dos planos gerados chegou a ser

drasticamente reduzido para valores pequenos de c, porém apenas uma faixa estreita de valores - entre 0.25 e 0.75 - foi capaz de gerar planos ótimos.

O que justifica este comportamento é que, dependendo do valor de c arbitrado e da aplicação, certas regras precisam ser disparadas sucessivas vezes para que a ditribuição de potencialidades do sistema se modifique de maneira a atingir a distribuição que leva ao plano ótimo.

7.4 Conclusões e Sugestões

Apesar de, na nossa opinião, o modelo não exibir um "aprendizado baseado no refinamento de habilidade" como esperávamos que acontecesse, ele apresenta alguns pontos favoráveis.

As vantagens que o modelo apresenta sobre procedimentos estáticos (Sistemas Especialistas não adaptativos), são:

- Ser capaz de alcançar soluções com um menor custo computacional, através de planos compostos de um número menor de regras (eventualmente, o número ótimo).

- Evitar reorganizações em Bases de Conhecimento complexas, visando melhor desempenho nas consultas ao Sistema Especialista.

Estas características se verificaram mesmo com Bases de Conhecimento diferentes da usada na aplicação piloto, com as quais rodamos o Modelo.

Em relação ao fato do Modelo não exibir o refinamento de habilidade da maneira esperada, podemos evitar que esta impressão chegue ao usuário colocando um módulo crítico na saída do Motor de Inferência que verifique a cada plano gerado se este é ou não melhor (ou seja, possui um número de regras menor) que o anterior: se for igual ou melhor, o plano será passado ao usuário; caso contrário,

o Motor de Inferência considerará o plano como uma tentativa fracassada e irá buscar outro plano alternativo.

As desvantagens decorrentes do emprego do modelo são:

- A necessidade de se ajustar o valor de c para cada aplicação diferente, para que se obtenha planos eficientes.

- O esquema mais complexo de seleção de regras pode significar um custo computacional adicional, especialmente com aplicações que gerem conjuntos de discórdia grandes.

- Atualizações na Base de Conhecimento implicam em ter que regerar os planos afetados pela mudança.

Todo o nosso estudo foi conduzido levando-se em conta um sistema voltado para a produção - com motor de inferência com encadeamento progressivo - pode-se, porém, mediante algumas modificações, tranformar o modelo num sistema voltado ao consumo e utilizar-se o encadeamento regressivo, que tem um comportamento mais eficiente em sistemas que buscam respostas únicas.

Outros enfoques, além destes que usam regras de produção, foram sugeridos para o modelo [07], por exemplo a representação em rede neuronal, e considerados viáveis.

A idéia central do modelo, que é o aprendizado através do refinamento de habilidade, tem um valor intrínseco e, ao nosso ver, deveria ser melhor explorada. Sugerimos que, seguindo nesta linha de pesquisa, se busque alternativa às oscilações de comprimentos de planos e se torne mais explícito o refinamento de habilidade a que o modelo se propunha.

Anexo A. Programa Principal

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%  COPPE/UFRJ  : Trabalho de Tese de Mestrado                               %
%  ----------    ---------------------------                                %
%                                                                           %
%  Assunto   :  Programa Principal para acesso 'a Base de Conhecimento     %
%                                                                           %
%               utilizando o Motor de Inferencia Forward Chaining .         %
%                                                                           %
%                                                                           %
%  Aluno     :  Jayme Bentes .                                              %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

roda  :-
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    Define um world para dados                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
            create_world( mifcd ),
            data_world( World , mifcd ),
%
           limpa_memoria,
           prepara_ambiente,
           atribui_saida,
           exibe_constante_c,
           roda(1).


roda(N) :- N =< 1    ,                        % Numero de Planos Gerados
           limpa_memoria,
           abre_janela(inferencia),
           time(time(0,0,0,0)),
           chama_provador,
           grava_bd,
           mostra_resultado(N),
           indaga_como,
           refina_plano( s ),
           N1 is N + 1,
           !,
           roda( N1 ).

roda(N).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Chamada ao Nucleo do Sistema Especialista Forward Chaining                   %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

chama_provador :- forward.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Limpa relacoes temporarias da memoria - Base Algoritmica                     %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


limpa_memoria :- eraseall( resp(_,_)    ),
                 eraseall( provado(_)   ),
                 eraseall( plano(_)     ), recordz(    plano(_),plano( [] ),_),
                 eraseall( ambiente(_)  ), recordz( ambiente(_),ambiente(0),_),
                 eraseall( fornecedor(_,_) ),
                 eraseall( linha(_)     ),
                 eraseall( lista_addr(_)),
                 expunge,
                 recorda(  lista_addr(_), lista_addr([]), _  ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Prepara ambiente para chamada do Motor de Inferencia                         %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

prepara_ambiente    :- cls, nl,
                       indaga_resposta,
                       indaga_constante,
                       indaga_refinamento,
                       indaga_saida,
                       indaga_depuracao.


indaga_resposta     :- write('Qual a resposta desejada ? '),
                       read_line(0,G1),
                       nl,nl,string_term(G1,G),
                       abolish( goal/1 ), assert( goal(G) ).
```

```
indaga_constante     :- write('Qual a Constante de Realocacao de Capital ? '),
                        read_line(0,C1),
                        nl,nl,string_term(C1,C),
                        abolish( constante/1 ), assert( constante(C) ).

indaga_refinamento   :- write('Refinamento de Plano ( s / n ) ? '),
                        read_line(0,P1),
                        nl,nl,string_term(P1,P),
                        refina_plano(P).

indaga_saida         :- write('Saida em impressora  ( s / n ) ? '),
                        read_line(0,S1),
                        nl,nl,string_term(S1,S),
                        determina_saida(S).

indaga_depuracao     :- write('Depuracao ativada    ( s / n ) ? '),
                        read_line(0,D1),
                        nl,nl,string_term(D1,D),
                        determina_depuracao(D).

refina_plano( s )    :- eraseall( ciclo(_)           ),
                        eraseall( atrib(_,_,_,_,_)   ),
                        recordz( ciclo(_),ciclo(1),_ ),
                        leia_bd,
                        !.

refina_plano( _ )    :- eraseall( ciclo(_) ),
                        recordz( ciclo(_),ciclo(0),_).

determina_saida( s ):- eraseall( saida(_) ),
                       recordz( saida(_),saida(impressora),_ ),
                       !.

determina_saida( _ ):- eraseall( saida(_) ),
                       recordz( saida(_),saida(video),_ ).

determina_depuracao( s ) :- eraseall( depuracao(_) ),
                            recordz( depuracao(_),depuracao(on),_ ),
                            ! .

determina_depuracao( D ) :- eraseall( depuracao(_) ),
                            recordz( depuracao(_),depuracao( D ),_ ).


atribui_saida :- recorded(saida(_),saida(impressora),_),
                 open( H , lpt1 , w ),
                 eraseall( dispositivo(_) ),
```

```
              recordz( dispositivo(_),dispositivo( H ),_ ),
              !.

atribui_saida :- eraseall( dispositivo(_) ),
              recordz( dispositivo(_),dispositivo( 0 ),_ ).


exibe_constante_c :- recorded(dispositivo(_),dispositivo(H),_),
              constante(C),
              limpa_tela(H),
              write(H,'O valor estipulado para a constante C foi '),
              write(H,C),
              nl(H),nl(H).

limpa_tela(0) :- cls.

limpa_tela(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Mostra resultado obtido pelo Sistema Especialista                           %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mostra_resultado( N ) :-
            goal(X),
            recorded(provado(_), provado(X), _),
            recorded(dispositivo(_),dispositivo(H),_),
            X =.. [Pred,Arg],
            recorded(plano(_), plano(Pi), _),
            inverte_lista( Pi,[Y|P] ),  limpa_tela(H) ,
            write(H,'Plano '),write(H, N ),write(H,' : '),nl(H),
            write(H,'          ['),write(H,Y),
            imprime_plano(P,15),write(H,']'),
            nl(H),nl(H),
            write(H,' O resultado obtido para  ' ),
            write(H,Pred),
            write(H,' foi : '),
            tab(H,4),write(H,Arg),
            nl(H),nl(H),
            write(H,' Esta consulta levou '),
            time(time(Hs,M,S,D)),
            write(H,M),write(H,' minutos e '),
            write(H,S),write(H,'.'),
            write(H,D),write(H,' segundos.'),
            nl(H),nl(H).
```

```
imprime_plano([],_) :- !.

imprime_plano(P,K)  :- num_elementos(P,N),
                       N =< K,
                       !,
                       imprime_linha(P,_,K).

imprime_plano(P,K)  :- imprime_linha(P,P1,K),
                       recorded(dispositivo(_),dispositivo(H),_),
                       nl(H),write(H,'           '),
                       !,
                       imprime_plano(P1,16).

imprime_linha([],[],_) :- !.

imprime_linha(P,P,0)   :- !.

imprime_linha([X|C],C1,N) :-
                          recorded(dispositivo(_),dispositivo(H),_),
                          write(H,','),write(H,X),
                          N1 is N - 1,
                          !,
                          imprime_linha(C,C1,N1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Procedimentos de Leitura e Gravacao da Base Algoritmica                   %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

grava_bd :- open_base_write,
            grava_atrib,
            close_bd.

grava_atrib :- regra( Id , _ , _ , _ ),
               [! recorded(atrib(_,_,_,_,_),atrib(Id,L,S,St,Pt),Ref),
                  recorded( handle(_), handle( H ), _ ),
                  write( H , Id ), nl( H ),
                  write( H , L  ), nl( H ),
                  write( H , S  ), nl( H ),
                  write( H , St ), nl( H ),
                  write( H , Pt ), nl( H )  !],
               fail.
grava_atrib.
```

```
leia_bd :- open_base_read,
           le_atrib,
           close_bd.

le_atrib      :- recorded( handle(_), handle( H ), _ ),
                 le_reg( H ).

le_reg( H )   :-  read_line( H , Id ),
                  read_line( H , L1  ), string_term( L1, L ),
                  read_line( H , S1  ), string_term( S1, S ),
                  read_line( H , St1 ), string_term( St1, St ),
                  read_line( H , Pt1 ), string_term( Pt1, Pt ),
                  recordz(atrib(_,_,_,_,_),
                          atrib( Id, L, S, St, Pt ), _ ),
%                 write(  atrib( Id, L, S, St, Pt ) ), nl,
                  ! ,
                  le_reg( H ).

le_reg( _ ).

open_base_write :- create(H, 'basea.idb' ),
                   eraseall(handle(_)),
                   recordz( handle(_), handle(H), _ ).

open_base_read  :- open(H, 'basea.idb' , r ),
                   ! ,
                   eraseall(handle(_)),
                   recordz( handle(_), handle(H), _ ).

open_base_read  :- write('Arquivo "basea.idb" nao encontrado !' ),
                        nl, nl.

close_bd :- recorded( handle(_), handle( H ), _ ),
            close(H).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Procedimentos auxiliares do Programa Principal                              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

indaga_como :- nl,nl,
        write($ ==>  Quer saber como foi obtida esta resposta (sim/nao) ?$),
        read_line(0,Resp),
testa_resp_como(Resp).

testa_resp_como($sim$) :- goal(X),
```

```
                    recorded(provado(_), provado(X), _ ),
                    explica_como(X),
                    !.

testa_resp_como(_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%  Converte string de letras minusculas para maiusculas                     %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

converte_maiusc(Text_Min,Text_Mai):- verifica_nao_int(Text_Min),
                                     !,
                                     atom_string(Text_Min1,Text_Min),
                                     name(Text_Min1,LMin),
                                     cv_maiusc(LMin,[],LMai),
                                     name(Text_Mai1,LMai),
                                     atom_string(Text_Mai1,Text_Mai).
converte_maiusc(Text_int,Text_int).

verifica_nao_int(T) :- int_text(Int,T),
                       !,
                       fail.
verifica_nao_int(T).

cv_maiusc([],Linter,LMai)       :- !,
                                   inverte_lista(Linter,LMai).

cv_maiusc([X|LMin],Linter,LMai) :- X >= 97,
                                   X =< 122,
                                   !,
                                   X1 is  X - 32,
                                   cv_maiusc(LMin,[X1|Linter],LMai).

cv_maiusc([X|LMin],Linter,LMai) :- ( X < 97; X > 122 ),
                                   !,
                                   cv_maiusc(LMin,[X|Linter],LMai).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%  Operacoes com listas ...                                                 %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inverte_lista(L1,L2) :- inverte2(L1,[],L2).
```

```
inverte2([],[],[]):- !.

inverte2([],Lf,Lf):- !.

inverte2([X|L1],L,Lf):- inverte2(L1,[X|L],Lf).

num_elementos([],0) :- !.

num_elementos([H|C],N) :- num_elementos(C,N1),
                          N is N1 + 1 .

n_esimo( 1, X, [X|_] ) .

n_esimo( N, X, [_|Y] ) :- n_esimo(M, X, Y),
                          N is M + 1 .

concatenar( [], Lista, Lista ) :- !.

concatenar( [Elem|L1], L2, [Elem|L3] ) :- concatenar( L1, L2, L3 ).


retire_rep( [], [] ) :- !.

retire_rep( [ X|L ], [ X|L1 ] ) :- retire_oc( X, L, L2 ),
                                   ! ,
                                   retire_rep( L2, L1  ).

retire_oc( _ , [], [] )     :- !.

retire_oc( X, [X|U], L )    :- ! ,
                               retire_oc( X, U, L ).

retire_oc( X, [Y|U], [Y|Z] ):- X \== Y,
                               !,
                               retire_oc( X, U, Z ).

contar_oc( _ , [] , 0 ) :- !.

contar_oc( X, [X|U], N) :- contar_oc( X, U, N1), N is N1 + 1.

contar_oc( X, [Y|U], N) :- X \== Y, contar_oc( X, U, N ).

concatena(X,Y) :- concl(X,$$,Y),
                  !.
```

```
concl( [] , Z , Z ) :- !.

concl( [ X ] , W , Z ) :- string_term( SX, X  ),
                          concat( W , SX , Z  ),
                          !.

concl( [ (X) | Y ] , W , Z ) :- string_term( SX1, X ),
                                concat( $($, SX1, SX2 ),
                                concat( SX2, $)$, SX  ),
                                concat( W , SX , W1 ),
                                concat( W1, $,$, W2 ),
                                !,
                                concl( Y , W2 , Z ).

concl( [ X | Y ] , W , Z ) :- string_term( SX, X  ),
                              concat( W , SX , W1 ),
                              concat( W1, $,$, W2 ),
                              !,
                              concl( Y , W2 , Z ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  COPPE/UFRJ : Trabalho de Tese de Mestrado                                 %
%  ----------   ----------------------------                                 %
%                                                                            %
%  Assunto   :  Simulacao das Operacoes de Acesso 'as Janelas                %
%                                                                            %
%               da  Interface  de  Comunicacao com o Usuario.                %
%                                                                            %
%                                                                            %
%  Aluno     :  Jayme Bentes                                                 %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Rotinas de abertura das Janelas                                            %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

abre_janela(inferencia) :-
 cls,
 box( 0, 0, 4, 77),
 tmove( 0, 4),
 write($ Pergunta $),
 box( 7, 0, 9, 77),
```

```
 tmove( 7, 4),
 write($ Resposta $),
 box( 11, 0, 15, 77),
 tmove(11, 4),
 write($ Critica / Prova $),
 abre_janela(mensagem).

 abre_janela(mensagem):-
             box( 18, 0, 20, 77),
             tmove( 18, 4),
             write($ Mensagem $).

 abre_janela(porque) :-
             box( 1, 4, 20, 77),
             tmove( 1, 8),
             write($ Explicacao por_que $),
             limpa_janela( 1, 4, 20, 77).

 abre_janela(como) :-
             box( 1, 4, 20, 77),
             tmove( 1, 8),
             write($ Explicacao  como $),
             limpa_janela( 1, 4, 20, 77).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Rotinas de escrita nas Janelas                                               %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 escreve_janela(_,[]) :- !.

 escreve_janela(Area,Texto) :-
             ( Area = como ; Area = porque ),
             !,
             abre_janela(Area),
             tmove(2,6),
             escreve_janela2(Area,Texto).

 escreve_janela(Area,[Linha|Texto]) :-
             cursor(Area,X),
             tmove(X,3),
             wc(72,  ),
             write(Linha),
             !,
             escreve_janela(Area,Texto).
```

```
escreve_janela2(Area,[]) :- !.

escreve_janela2(Area,[Linha|Texto]) :-
            escreve_janela3(Area,X,Linha),
            !,
            escreve_janela2(Area,Texto).

escreve_janela3(Area,X,Linha):-
            string_length(Linha,L),
            L =< 72,
            !,
            escreve_janela4(Area,X,Linha).

escreve_janela3(Area,X,Linha):-
            string_length(Linha,L),
            procura_ultimo_branco(Linha),
            ultimo(N),
            substring(Linha,0,N,Linha1),
            escreve_janela4(Area,X,Linha1),
            K1 is N + 1, K2 is L - K1,
            substring(Linha,K1,K2,Linha2),
            escreve_janela4(Area,X1,Linha2).

escreve_janela4(Area,X,Linha):-
            cursor(Area,X),
            tmove(X,6),
            write(Linha).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                   %
%  Rotinas de leitura das Janelas                                                   %
%                                                                                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

leia_janela(Modo,Resp) :-
            ( Modo = cont1 ; Modo = cont2 ),
            !,
            abre_janela(mensagem),
            tmove(19,1),
            write($    - Deseja continuar (sim/nao) ? sim $),
            tmove(19,35),testa_resp(Resp),
            teste_fecha_janela(Modo,Resp).

testa_resp(Resp) :-
            read_line(0,RespC),
            string_term(RespC,Resp1),
            tmove(19,35),wc(39,  ),
```

```
            tmove(19,35),
            valida_resp( Resp1, Resp ),
            !.

testa_resp(Resp) :- !,
                    testa_resp(Resp1).

valida_resp( Resp1, sim   ) :-  Resp1 == end_of_file , !.

valida_resp( Resp1, Resp1 ) :-  Resp1 == sim ; Resp1 == nao .

 leia_janela(pausa,_) :-
            !,
            abre_janela(mensagem),
            tmove(19,3),
            write($ - Tecle enter para prosseguir $),
            put(7),
            flush,
            keyb(A,S),
            tmove(19,3),  wc(72,  ),
            tmove(19,3).

 leia_janela(_,Resposta) :-
            tmove(8,3),  wc(72,  ),
            read_line(0,RespC),
            string_term(RespC,Resposta),
            tmove(13,3), wc(72,  ),
            tmove(19,3), wc(72,  ),
            tmove(8,3).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%  Rotinas de teste de intercambio das janelas                             %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 teste_fecha_janela(Modo,sim)   :- !.

 teste_fecha_janela(cont1,nao)   :- abre_janela(inferencia),!.  % por_que

 teste_fecha_janela(cont2,nao)   :- cls,!.                      % como

 verifica_janela :- recorded(  depuracao(_),  depuracao(on), _ ),
                    recorded(dispositivo(_), dispositivo(0), _ ),
                    abre_janela(inferencia).

 verifica_janela.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Rotinas de tracado  da  moldura  de  uma  janela  generica                  %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 box( Y, X, Y1, X1 ) :-
            C is X1 - X ,
            tmove( Y, X ),
            wc( C, 196 ),
            tmove( Y, X1 ),
            wc( 1, 191 ),
            C1 is Y1 - Y ,
            Y2 is Y + 1 ,
            vert( X1, Y2, Y1 ),
            tmove( Y, X ),
            wc( 1, 218 ),
            tmove( Y2, X ),
            vert( X, Y2, Y1 ),
            tmove( Y1, X ),
            wc( 1, 192 ),
            X2 is X + 1 ,
            tmove( Y1, X2 ),
            wc( C, 196 ),
            tmove( Y1, X1 ),
            wc( 1, 217 ),!.

 vert( X, Y, Y1 ) :-
            ctr_set(0,Y),
            repeat,
            ctr_inc(0,Z),
            tmove(Z,X),
            wc(1,179),
            Z == Y1.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Rotina  de  limpeza  de  uma  janela  generica                              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 limpa_janela( X, Y, X1, Y1 ) :-
            DY is Y1 - Y - 1,       % espaco util entre colunas
            FX is X1 - 1,            % ultima linha util
            IY is Y + 1,             % coluna util inicial
```

```
        IX is X + 1,                % linha  util inicial
        ctr_set(0,IX),              % contador = linha inicial
        repeat,         % repetir ...
        ctr_inc(0,XN),              %  incrementa numero da linha
        tmove(XN,IY),               %  move cursor
        wc(DY,  ),                  %  limpa a linha
        XN == FX .                  %  testa condicao de parada

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%  Rotinas de apoio as operacoes com as pseudo-janelas                      %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 cursor(perg,2).

 cursor(crit,13).

 cursor(msg,19).

 cursor(porque,X) :- tget(R,C), X is R + 1 ,X < 17 , !.

 cursor(porque,2) :- leia_janela(pausa,_),
                     abre_janela(porque).

 cursor(como,X)   :- tget(R,C), X is R + 1, X < 17 , !.

 cursor(como,2)   :- leia_janela(pausa,_),
                     abre_janela(como).

 procura_ultimo_branco(Linha) :-
         string_search($ $,Linha,Nb),
         grava_ultimo(Nb),
         Nb > 71,!.

 procura_ultimo_branco(Linha).

 ultimo(0).

 grava_ultimo(Nb) :- Nb < 71,
                     abolish( ultimo/1   ),
                     assert(  ultimo(Nb) ),!.

 grava_ultimo(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
```

```
%  Fim do Programa  Principal                                  marco de 1992  %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Anexo B. Motor de Inferência Forward Chaining

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  COPPE-UFRJ: Trabalho de Tese de Mestrado em IA                               %
%  ----------- ----------------------------------                               %
%                                                                               %
%  Assunto   :  Motor de Inferencia Forward Chaining                            %
%                                                                               %
%                                                                               %
%  Aluno     :  Jayme Bentes                                                    %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                                                                               %
%  Programa Principal com Resolucao de Conflitos e Aprendizado                  %
%                                                                               %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


forward :- verifica_goal,              % < tarefa terminada ? >
           paga_ultima_regra,          % paga regra que completou a tarefa
           paga_plano.                 % paga regras que pertencem ao plano


forward :- inicia_ciclo,               % comeca o ciclo de reconhecimento-acao
           testa_con,                  % testa se foi gerado conjunto discordia
           probabilidade_priori,       % calcula prob. a priori das regras
           gera_subcon,                % gera subconjuntos por criterios sel.
           testa_uniao,                % testa se uniao igual conj discordia
           estrategia_selecao,         % aplica estrategia de selecao
           potencialidade,             % determina potencialidade cada regra
           potencialidade_esperada,    % idem, potencialidade esperada
           seleciona_regra,            % seleciona regra max potenc. esperada
           modifica_forca,             % regra selecionada paga sistema
           dispara_regra,              % dispara a regra selecionada
           paga_regras,                % paga regras que contribuiram
           novo_ciclo,                 % identifica novo ciclo
           !,                          % ------------------------------------
           forward.                    % chama MIFC recursivamente
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%                                                                             %
%                         S U B P R O G R A M A S                             %
%                                                                             %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  inicia_ciclo: verifica, face aos descritores de estado do problema, que   %
%                regras sao "disparaveis" e gera o conjunto de discordia     %
%                CON(t). Se for o ciclo inicial, normaliza os valores dos    %
%                atributos de cada regra, relativos a cada um dos criterios %
%                de selecao (P, C, U, O).                                    %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

inicia_ciclo :- recorded(ciclo(_),ciclo(0),Ref),
                eraseall( no_regras(_)  ),
                zera_contadores(4),
                normaliza_atributos,
                replace( Ref, ciclo(1) ),
                !,
                inicia_ciclo.

inicia_ciclo :- limpa_memorial,
                inicializa_bd,
                pesquisa_regras.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  normaliza_atributos: normaliza os valores de P, C, U e O de cada regra    %
%                       presente na Base de Conhecimento.                    %
```

```
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

normaliza_atributos :- ctr_set(5,0),
                       eraseall(atrib(_,_,_,_,_)),
                       normaliza2,
                       ctr_is(5,M),
                       recordz(no_regras(_),no_regras(M),_),
                       normaliza3,
                       imprime_atributos.


zera_contadores(N) :- ctr_set( 0, 1 ),              % zera contadores de 1 a N
                      eraseall( contador(_,_) ), expunge,
                      repeat,
                      ctr_inc( 0, X ),
                      cont_set0( X, 0 ),
                      X == N.

normaliza2 :- regra(Id, [P, C, U, O], X, Y),
              [! cont_is(1, SP), NSP is SP + P,         cont_set(1, NSP),
                 cont_is(2, SC), NSC is SC + (100 - C), cont_set(2, NSC),
                 cont_is(3, SU), NSU is SU + (1 / U),   cont_set(3, NSU),
                 cont_is(4, SO), NSO is SO + O,         cont_set(4, NSO),
                 ctr_inc(5,N)                                          !],
               fail.

normaliza2.


normaliza3 :- ctr_is(5,M),
              ctr_set( 0 , 1 ),
              repeat,
              regra(Id, [P, C, U, O], X, Y),
              [! ctr_inc( 0 , N ),
                 cont_is(1, SP), NP1 is ( P / SP )   * 100, NP is round(NP1,4),
                 cont_is(2, SC), NC1 is (100 - C)/SC * 100, NC is round(NC1,4),
                 cont_is(3, SU), NU1 is (1 / U) / SU * 100, NU is round(NU1,4),
                 cont_is(4, SO), NO1 is (M -O +1)/SO * 100, NO is round(NO1,4),
                 Snor is NP + NC + NU + NO,
                 recordz(atrib(_,_,_,_,_),
                         atrib(Id, [NP, NC, NU, NO], Snor, 100, 0),_) !],
               N == M.

% normaliza3.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
imprime_atributos :- recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo(H),_),
                     limpa_tela(H),
                     write(H,'Atributos das Regras, normalizados : '),
                     nl(H),
                     write(H,'Regra    P        C        U        O       Soma')
                     nl(H), %   xx   xx.xxxx  xx.xxxx  xx.xxxx  xx.xxxx  xxx.xx
                     imprime1(H),
                     testa_pausa(H).

imprime_atributos.

imprime1(H) :- recorded(no_regras(_),no_regras(M),_),
               ctr_set( 0 , 1 ),
               repeat,
               recorded(atrib( _ , _ , _ , _ , _ ),
                        atrib( Id, [NP, NC, NU, NO], Snor, X , Y ),_),
            [! ctr_inc( 0 , N ),
               tab(H,2)   ,write(H,Id),
               tab(H,3)   ,write(H,NP),
               tab(H,5)   ,write(H,NC),
               tab(H,2)   ,write(H,NU),
               tab(H,2)   ,write(H,NO),
               tab(H,2)   ,write(H,Snor),
               nl(H)              !],
             N == M .

% imprime1(_).

testa_pausa(0) :- read_line(0,R).

testa_pausa(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  pesquisa_regras: este procedimento verifica todas as regras que sao          %
%                   disparaveis na Base de Conhecimento, frente ao estado       %
%                   atual dos descritores de estado do sistema.                 %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pesquisa_regras :- pesquisa_regras2,
                   inverte_con,
                   imprime_con.
```

```
pesquisa_regras2 :- regra(Id,X,Y,Z),
                    [! disparavel(regra(Id,X,Y,Z)),
                       recorded( con(_), con(C), Ref),
                       replace( Ref, con([Id|C])) !] ,
                    fail.

pesquisa_regras2 .

disparavel(regra(Id,_,_,[])) :- !.

disparavel(regra(Id,_,_,[X|Y])) :- ( fato(X);
                                     call(X);
                                     recorded(provado(_),provado(X),_);
                                     perguntavel(X)                   ),
                                     ! ,
                                     disparavel(regra(Id,_,_,Y)).

disparavel(regra(Id,_,_,(A;B))) :- ( disparavel(regra(Id,_,_,A)) ;
                                    disparavel(regra(Id,_,_,B))  ),
                                    !.

disparavel(regra(Id,_,_,(A,B))) :- disparavel(regra(Id,_,_,A)),
                                   disparavel(regra(Id,_,_,B)).




inverte_con :- recorded( con(_), con(C), Ref ),
               inverte_lista(C,IC),
               replace( Ref, con( IC ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_con :- recorded(depuracao(_),depuracao(on),_),
               recorded(dispositivo(_),dispositivo(H),_),
               nl(H), nl(H),
               write(H,'Conjunto de Discordia gerado no ciclo '),
               recorded(ciclo(_),ciclo( T ),_),
               write(H, T ), write(H,' : '),
               recorded(con(_),con( Discordia ),_),
               write(H, Discordia ),
               nl(H), nl(H),
               testa_pausa(H).

imprime_con.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  testa_con: verifica se o conjunto de discordia gerado e' nao vazio.         %
%             Se CON(t) for vazio, o procedimento para, pois nao existe         %
%             nenhuma regra possivel de ser disparada face aos descritores      %
%             de estado do problema.                                            %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

testa_con :- recorded(con(_), con( [] ), _ ),
             recorded(didpositivo(_), dispositivo( H ), _ ),
             write(H,'Nenhuma regra foi selecionada.'),
             break.

testa_con.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Calculo da probabilidade "a priori" de todas as regras pertencentes          %
%  ao conjunto de regras RS(t) presentes na Base de Conhecimento.               %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

probabilidade_priori :- zera_contadores(1),
                        probabilidade_priori2,
                        probabilidade_priori3,
                        imprime_priori.


probabilidade_priori2 :- recorded(no_regras(_),no_regras(M),_),
                         ctr_set( 0 , 1 ),
                         repeat,
                         recorded( atrib(_,_,_,_,_),
                                   atrib(Id,X,Y,St,Pt), _ ),
                         [! ctr_inc( 0 , N ),
                            cont_is(1,SS), NSS is SS + St,
                            cont_set(1,NSS)              !],
                         N == M.

probabilidade_priori2.

probabilidade_priori3 :- recorded(no_regras(_),no_regras(M),_),
                         ctr_set( 0 , 1 ),
                         repeat,
                         regra( Id, _ , _ , _ ),
                         [! ctr_inc( 0 , N ),
                            recorded( atrib(_,_,_,_,_),
```

```
                                    atrib(Id, X, Y, St, Pt), Ref ) ,
                            cont_is(1,SS), N1 is St / SS , NPt is round(N1,4),
                            replace( Ref, atrib( Id, X, Y, St, NPt ) )   !],
                        N == M.

% probabilidade_priori3 .

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_priori :-    recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo( H ),_),
                     limpa_tela(H),
                     write(H,'Probabilidade a priori das regras : '),
                     nl(H),
                     write(H,'Regra     St       Pt   '),
                     nl(H), %   xxx   xx.xxxx  xx.xxxx
                     imprime2(H),
                     testa_pausa(H).

imprime_priori.


imprime2(H) :- recorded(no_regras(_),no_regras(M),_),
               ctr_set( 0 , 1 ),
               repeat,
               recorded(atrib(_,_,_,_,_),atrib(Id, X, Y, St, Pt ), _ ),
              [! ctr_inc( 0 , N ),
                 tab(H,2)   ,write(H,Id),
                 tab(H,3)   ,write(H,St),
                 tab(H,2)   ,write(H,Pt),
                 nl(H)              !],
               N == M .
% imprime2(_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Geracao dos subconjuntos de regras RULE(t,k) a partir do conjunto de        %
%  discordia, pela aplicacao dos criterios de selecao segundo as faixas        %
%  especificadas.                                                              %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cr( 1, P ) :- P >= 70.

cr( 2, C ) :- C =< 20.

cr( 3, U ) :- U =< 2 .
```

```
cr( 4, 0 ) :- 0 =< 5 .


gera_subcon :- recorded(con(_), con(X), _),
               gera_subcon2(X),
               imprime_subcon.

gera_subcon2( [] ) :- !.

gera_subcon2( [ Id | Cauda ] ) :- regra( Id, [ P, C, U, O ], X, Y ),
                                  gera_rule( 1, Id, P ),
                                  gera_rule( 2, Id, C ),
                                  gera_rule( 3, Id, U ),
                                  gera_rule( 4, Id, O ),
                                  !,
                                  gera_subcon2( Cauda ).

gera_rule( N, Id, X ) :- cr( N, X ),
                         !,
                         recorded( rule(_,_), rule( N, Lista ) , Ref ),
                         replace( Ref, rule( N, [ Id | Lista ] ) ).

gera_rule(_,_,_).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_subcon  :-  recorded(depuracao(_),depuracao(on),_),
                    recorded(dispositivo(_),dispositivo(H),_),
                    nl(H), nl(H),
                    write(H,'Subconjuntos gerados a partir de CON(t) :'),
                    nl(H),
                    write(H,'  k    Regras '),
                    nl(H),  %  k    [ Ra, Rb, ... ,Rn ]
                    imprime3(H),
                    testa_pausa(H).

imprime_subcon.

imprime3(H) :- ctr_set( 0, 1 ),
           repeat,
           ctr_inc( 0, N ),
           recorded(rule(_,_), rule( N, Lista ), _ ),
           tab(H,2),write(H,N),
           tab(H,4),write(H,Lista),
           nl(H),
```

```
        N == 5.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%  Testa se a uniao dos subconjuntos RULE(t,k) gerados iguala o conjunto    %
%  de discordia. Para tanto, e' feita a uniao de todos os RULE(t,k) que     %
%  foram gerados e eliminados os elementos repetidos, antes de efetuar a    %
%  subtracao deste conjunto uniao do conjunto de discordia. Se houver       %
%  resto, e' gerado um RULE(t,k+1) com estas regras.                        %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

testa_uniao :- concatena_rules( Rules ),
               recorded(con(_),con( Discordia ), _ ),
               subtrai_rules( Rules, Discordia, Sobra),
               gera_rule5( Sobra ).

concatena_rules( Rules ) :- recorded(rule(_,_),rule( 1, L1 ),_),
                            recorded(rule(_,_),rule( 2, L2 ),_),
                            recorded(rule(_,_),rule( 3, L3 ),_),
                            recorded(rule(_,_),rule( 4, L4 ),_),
                            concatenar( L1, L2, LX ),
                            concatenar( L3, L4, LY ),
                            concatenar( LX, LY, LZ ),
                            retire_rep( LZ, Rules  ).

subtrai_rules( [], _, Sobra ) :- !.

subtrai_rules( [ X | Y ], L, Z ) :- retire_oc( X, L, Z1 ),
                                    ! ,
                                    subtrai_rules( Y , Z1, Z ).

gera_rule5( [] ) :- !.

gera_rule5( Sobra ) :- recorded( rule(_,_), rule( 5, _ ), Ref ),
                       replace( Ref, rule( 5, Sobra ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%  Aplica a estrategia de selecao padrao ou a estrategia default            %
%  conforme o caso.                                                         %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

estrategia_selecao :- recorded( rule(_,_), rule( 5 , [] ), _ ),
```

```
                    ! ,
                    probabilidade_posteriori,
                    probabilidade_selecao.

estrategia_selecao :- prob_selecao2,
                    imprime_psel.

prob_selecao2 :- recorded(con(_),con( L ),_),
                 prob_sel2( L ).

prob_sel2( [] ) :- !.

prob_sel2( [ Id | Cauda ] ) :- recorded(atrib(_,_,_,_,_),
                                        atrib(Id, Lista, Sigma, St, Pt ),_),
                               recordz( pregra(_,_),pregra( Id, Pt ),_ ),
                               ! ,
                               prob_sel2( Cauda ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Calcula a probabilidade "a posteriori" de cada subconjunto de regras         %
%  RULE(t,k)  gerado, condicionada ao conjunto de discordia CON(t) e a          %
%  cada criterio de selecao CR(k).                                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

probabilidade_posteriori :- probabilidade_posteriori2,
                            probabilidade_posteriori3,
                            probabilidade_posteriori4,
                            probabilidade_posteriori5,
                            imprime_posteriori.

probabilidade_posteriori2 :- zera_contadores(10),
                             ctr_set(0,1),
                             repeat,
                             ctr_inc( 0, Cont),
                             recorded(rule(_,_),rule( Cont, Lista ),_),
                             [! prob_post( Cont, Lista ),
                                prob_post2( Cont )    !],
                             Cont == 4.

prob_post( N, [] ) :- !.

prob_post( N, [ Id | Cauda ] ) :-
          recorded(atrib(_,_,_,_,_),atrib(Id, Lista1, Sigma, St, Pt ),_),
          n_esimo( N, X, Lista1 ),
          N5 is N + 5,
```

```
        cont_is( N, SN ), NSN is SN + X ,    cont_set( N , NSN ),
        cont_is( N5, SD), NSD is SD + Sigma, cont_set( N5, NSD ),
        !,
        prob_post( N, Cauda ).

prob_post2( N ) :- cont_is( N, NUM ),
                   N5 is N + 5,
                   cont_is( N5, DEN ),
                   DEN \== 0 ,
                   ! ,
                   P1 is NUM / DEN ,
                   P is round( P1,4 ),
                   recordz( pcrule(_,_),pcrule( N, P ),_ ).

prob_post2( N ) :- recordz( pcrule(_,_),pcrule( N, 0 ),_ ).


probabilidade_posteriori3 :- zera_contadores(10),
                             ctr_set(0,1),
                             recorded(con(_),con( L ),_),
                             repeat,
                             ctr_inc( 0, Cont),
                             [! prob_post( Cont, L  ),
                                prob_post3( Cont ) !],
                             Cont == 4.

prob_post3( N ) :- cont_is( N, NUM ),
                   N5 is N + 5,
                   cont_is( N5, DEN ),
                   DEN \== 0 ,
                   ! ,
                   P1 is NUM / DEN ,
                   P is round( P1,4 ),
                   recordz( pcrcon(_,_),pcrcon( N, P ),_ ).

prob_post3( N ) :- recordz( pcrcon(_,_),pcrcon( N, 0 ),_ ).


probabilidade_posteriori4 :- zera_contadores(6),
                             recorded(con(_),con( L ),_),
                             prob_post4( L ),
                             ctr_set(0,1),
                             repeat,
                             ctr_inc( 0, Cont),
                             recorded(rule(_,_),rule( Cont, Lista ),_),
                             [! prob_post5( Cont, Lista ),
                                prob_post6( Cont )     !],
```

```
                              Cont == 4.

prob_post4( [] ) :- !.

prob_post4( [ Id | Cauda ] ) :- recorded(atrib(_,_,_,_,_),
                                        atrib(Id, Listal, Sigma, St, Pt ),_),
                                cont_is( 6, SD ), NSD is SD + Pt,
                                cont_set( 6, NSD ),
                                !,
                                prob_post4( Cauda ).

prob_post5( N, [] ) :- !.

prob_post5( N, [ Id | Cauda ] ) :- recorded(atrib(_,_,_,_,_),
                                          atrib(Id, Listal, Sigma, St, Pt ),_)
                                   cont_is( N, SN ), NSN is SN + Pt,
                                   cont_set( N, NSN ),
                                   ! ,
                                   prob_post5( N, Cauda ).


prob_post6( N ) :- cont_is( N, NUM ),
                   cont_is( 6, DEN ),
                   DEN \== 0 ,
                   ! ,
                   P1 is NUM / DEN ,
                   P is round( P1,4 ),
                   recordz( prule(_,_),prule( N, P ),_ ).

prob_post6( N ) :- recordz( prule(_,_),prule( N, 0 ),_ ).


probabilidade_posteriori5 :- ctr_set(0,1),
                             repeat,
                             ctr_inc( 0, Cont),
                          [! prob_post7( Cont ) !],
                             Cont == 4.

prob_post7( N ) :- recorded(pcrule(_,_),pcrule( N, X1 ),_),
                   recorded(pcrcon(_,_),pcrcon( N, X2 ),_),
                   recorded( prule(_,_), prule( N, X3 ),_),
                   X2 \== 0,
                   ! ,
                   X4 is ( X1 / X2 ) * X3 , X is round( X4,4 ),
                   recordz( probpos(_,_),probpos( N, X ),_).

prob_post7( N ) :- recordz( probpos(_,_),probpos( N, 0 ),_).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_posteriori:- recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo(H),_),
                     limpa_tela(H),
                     write(H,'Probabilidades de CR(k)  RULE( t, k )'),
                     nl(H),
                     write(H,'  k      P( CR(k)  RULE( t, k ) ) '),
                     nl(H),  %   x     x.xxxx
                     imprime4(H),
                     testa_pausa(H),
                     write(H,'Probabilidades de CR(k)  CON(t)'),
                     nl(H),
                     write(H,'  k      P( CR(k)  CON(t) ) '),
                     nl(H),  %   x     x.xxxx
                     imprime5(H),
                     testa_pausa(H),
                     write(H,'Probabilidades de RULE(t,k) / P( CON(t) )'),
                     nl(H),
                     write(H,'  k    P( RULE(t,k) ) / P( CON(t) ) '),
                     nl(H),  %   x     x.xxxx
                     imprime6(H),
                     testa_pausa(H),
                     write(H,'Probabilidades de RULE(t,k)  CON(t), CR(k)'),
                     nl(H),
                     write(H,'  k    P( RULE(t,k)  CON(t), CR(k) ) '),
                     nl(H),  %   x     x.xxxx
                     imprime7(H),
                     testa_pausa(H).

imprime_posteriori.

imprime4(H) :- ctr_set( 0, 1 ),
               repeat,
               ctr_inc( 0, N ),
               recorded(pcrule(_,_),pcrule( N, Valor ),_),
               tab(H,2),write(H,N),
               tab(H,4),write(H,Valor),
               nl(H),
               N == 4.

imprime5(H) :- ctr_set( 0, 1 ),
               repeat,
               ctr_inc( 0, N ),
               recorded(pcrcon(_,_),pcrcon( N, Valor ),_),
```

```
            tab(H,2),write(H,N),
            tab(H,4),write(H,Valor),
            nl(H),
            N == 4.

imprime6(H) :- ctr_set( 0, 1 ),
            repeat,
            ctr_inc( 0, N ),
            recorded( prule(_,_), prule( N, Valor ),_),
            tab(H,2),write(H,N),
            tab(H,4),write(H,Valor),
            nl(H),
            N == 4.

imprime7(H) :- ctr_set( 0, 1 ),
            repeat,
            ctr_inc( 0, N ),
            recorded(probpos(_,_),probpos( N, Valor ),_),
            tab(H,2),write(H,N),
            tab(H,4),write(H,Valor),
            nl(H),
            N == 4.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%  Calcula a probabilidade de cada regra pertencente ao conjunto de       %
%  discordia CON(t) ser a regra selecionada.                               %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

probabilidade_selecao :- prob_selecao,
                         imprime_psel.

prob_selecao :- recorded(con(_),con( L ),_),
        ctr_set( 0, 1 ),
        repeat,
        ctr_inc( 0, Cont ),
      [! prob_sel1( Cont, L ) !],
        Cont == 4.

prob_sel1( N, [] ) :- !.

prob_sel1( N, [ Id | Cauda ] ) :- recorded(atrib(_,_,_,_,_),
                                          atrib(Id, Lista, Sigma, St, Pt ),_),
                                  n_esimo( N, X1, Lista ),
                                  X2 is X1 / Sigma,
                                  recorded(probpos(_,_),probpos( N, X3 ),_),
```

```
                                X3 \== 0,
                                X4 is ( X2 * Pt ) / X3,
                                recorded( pregra(_,_), pregra( Id, X5 ), Ref),
                                X6 is X4 + X5, X is round( X6,4 ),
                                replace( Ref, pregra( Id, X ) ),
                                ! ,
                                prob_sell( N, Cauda ).

prob_sell( N, [ Id | Cauda ] ) :- prob_sell( N, Cauda ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_psel      :- recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo(H),_),
                     nl(H),
                     write(H,'Probabilidade de Selecao de cada regra:'),
                     nl(H), nl(H),
                     write(H,'Regra  Probabilidade '),
                     nl(H),  %   xx     x.xxxx
                     recorded(con(_),con( L ),_),
                     imprime8(H,L),
                     testa_pausa(H).

imprime_psel.

imprime8( _ , [] ) :- !.

imprime8( H , [Id|Y] ) :- recorded( pregra(_,_), pregra( Id, Valor ), _ ),
                          tab(H,2),write(H, Id ),
                          tab(H,4),write(H, Valor ),
                          nl(H),
                          !,
                          imprime8( H , Y ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Calcula a potencialidade de cada regra pertencente ao conjunto de           %
%  discordia CON(t) ser a regra selecionada.                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

potencialidade :- recorded(con(_),con( L ),_),
                  potenl( L ),
                  imprime_poten.

potenl( [] ) :- !.
```

```
potenl( [ Id | Cauda ] ) :- recorded(atrib(_,_,_,_,_),
                                 atrib(Id, [ P, C, U, O ], Sigma, St, Pt ),_),
                            Soma2 is P*P + C*C + U*U + O*O + St * St  ,
                            Pl is sqrt( Soma2 ), Pot is round( Pl,4 ),
                            recordz( potencial(_,_),potencial( Id, Pot ),_ ),
                            ! ,
                            potenl( Cauda ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_poten     :- recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo( H ),_),
                     nl(H),
                     write(H,'Potencialidade de Selecao de cada regra:'),
                     nl(H), nl(H),
                     write(H,'Regra  Potencialidade'),
                     nl(H),  %   xx     x.xxxx
                     imprime9( H ),
                     testa_pausa(H).

imprime_poten.

imprime9( H ) :- recorded(potencial(_,_), potencial( Id, Valor ), _),
                 [! tab(H,2),write(H, Id ),
                    tab(H,4),write(H, Valor ),
                    nl(H)                    !],
                    fail.

imprime9( _ ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Calcula a potencialidade esperada de cada regra pertencente ao conjunto     %
%  de discordia CON(t) ser a regra selecionada.                                %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

potencialidade_esperada :- recorded(con(_),con( L ),_),
                           potesp( L ),
                           imprime_pesp.

potesp( [] ) :- !.

potesp( [ Id | Cauda ] ) :- recorded(potencial(_,_),potencial( Id, P ),_),
                            recorded(   pregra(_,_),   pregra( Id, X ),_),
                            Pl is P * X , P_esp is round( Pl,4 ),
                            recordz( pesp(_,_),pesp( Id, P_esp ),_ ),
```

```
                        ! ,
                        potesp( Cauda ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_pesp      :- recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo(H),_),
                     nl(H),
                     write(H,'Potencialidade Esperada de cada regra:'),
                     nl(H), nl(H),
                     write(H,'Regra  P. Esperada   '),
                     nl(H),  %   xx     x.xxxx
                     imprime10(H),
                     testa_pausa(H).

imprime_pesp.

imprime10(H) :- recorded(pesp(_,_), pesp( Id, Valor ), _ ),
             [! tab(H,2),write(H, Id ),
                tab(H,4),write(H, Valor ),
                nl(H)                    !],
                fail.

imprime10( _ ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                 %
%  Selecao da regra que deve ser disparada: aquela com maior potencialidade %
%  esperada dentre as regras pertencentes ao conjunto de discordia CON(t).  %
%                                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

seleciona_regra :- recorded(con(_),con( L ),_),
                   seleciona( L ),
                   imprime_sel.

seleciona( [] ) :- !.

seleciona( [ Id | Cauda ] ) :- recorded(pesp(_,_),pesp( Id, P ),_),
                               checa_max( Id, P ),
                               ! ,
                               seleciona( Cauda ).

checa_max( Id, P ) :- recorded( maxpot(_,_), maxpot( X, P1 ) , Ref ),
                      P > P1,
                      !,
                      replace( Ref, maxpot( Id, P ) ).
```

```
checa_max( _ , _ ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_sel       :- recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo(H),_),
                     nl(H),
                     write(H,'Regra Selecionada e sua potencialidade :'),
                     nl(H), nl(H),
                     write(H,'Regra  P. Esperada Max'),
                     nl(H),  %   xx     x.xxxx
                     recorded( maxpot(_,_), maxpot( Id, Valor ), _ ),
                     tab(H,2),write(H, Id ),
                     tab(H,4),write(H, Valor ),
                     nl(H),
                     testa_pausa(H).

imprime_sel.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Fim do ciclo de reconhecimento_acao: se o objetivo foi alcancado ( fim      %
%  da tarefa ), o sistema notifica .                                           %
%  Se o plano formado nao foi considerado satisfatorio, o processo recomeca    %
%  do ciclo 1.                                                                 %
%  Se a tarefa nao terminou, o processo continua num novo ciclo.               %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

modifica_forca :- recorded(   maxpot(_,_),    maxpot( Id, P ) , _ ),
                  recorded(potencial(_,_), potencial( Id, Pl ), _ ),
                  constante( C ),
                  Pag is C * Pl ,
                  imprime_pag( Pag ),
                  recorded( atrib(_,_,_,_,_),
                            atrib( Id, X, Y, St, Z ), Ref ),
                  N is St - Pag, NSt is round( N,4 ),
                  replace( Ref, atrib( Id, X, Y, NSt, Z ) ),
                  atualiza_plano( Id ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_pag( Pag ):- recorded(depuracao(_),depuracao(on),_),
                     recorded(dispositivo(_),dispositivo(H),_),
```

```
                    nl(H),
                    write(H,'Regra Selecionada e pagamento efetuado :'),
                    nl(H), nl(H),
                    write(H,'Regra  Pagamento '),
                    nl(H),  %   xx     xxxxxx
                    recorded( maxpot(_,_), maxpot( Id, _ ), _ ),
                    tab(H,2),write(H, Id ),
                    tab(H,4),write(H, Pag ),
                    nl(H),
                    testa_pausa(H).

imprime_pag( _ ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Inclui a regra selecionada no novo plano sendo gerado.                       %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

atualiza_plano( Id ) :- recorded( plano(_), plano( P ),Ref ),
                        num_elementos( P, N ),
                        N \== 0 ,
                        ! ,
                        replace( Ref, plano( [ Id | P ] ) ).

atualiza_plano( Id ) :- recorded( plano(_), plano( [] ), Ref ),
                        replace( Ref, plano( [ Id ] ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  A regra selecionada e' disparada pelo motor de inferencia.                   %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dispara_regra :- recorded(maxpot(_,_), maxpot( Id, P ), _ ),
                 regra( Id, _ , Acao, Requisitos ),
                 verifica_janela,
                 dispara( regra( Id, _, Acao, Requisitos) ),
                 atualiza_descritor( Id , Acao ).

dispara( regra( Id, _ , Acao, [] ) ) :- !.

dispara( regra( Id, _ , Acao, ( A ; B ) ) ) :-
                   ( dispara( regra( Id, _ , Acao, A ) ) ;
                     dispara( regra( Id, _ , Acao, B ) ) ),
                     !.
```

```
dispara( regra( Id, _ , Acao, ( A , B ) ) ) :-
                    dispara( regra( Id, _ , Acao, A ) ) ,
                    dispara( regra( Id, _ , Acao, B ) ) ,
                    !.

dispara( regra( Id, _ , Acao, [known( X )|Requisitos ] ) ):-
                    ! ,
                    dispara( regra( Id, _ , Acao, [ X|Requisitos ] ) ).

dispara( regra( Id, _ , Acao, [ X|Requisitos ] ) ):-
                    recorded(provado(_), provado( X ) , _ ),
                    lembre_fornecedor( X ),
                    ! ,
                    dispara( regra( Id, _ , Acao, Requisitos ) ).

dispara( regra( Id, _ , Acao, [ X|Requisitos ] ) ):-
                    ( fato(X) ; call(X) ),
                    ! ,
                    dispara( regra( Id, _ , Acao, Requisitos ) ).

dispara( regra( Id, _ , Acao, [ X|Requisitos ] ) ):-
                    perguntavel( X ) ,
                    resposta( sim, X ),
                    ! ,
                    dispara( regra( Id, _ , Acao, Requisitos ) ).

dispara( regra( Id, _ , Acao, [ X|Requisitos ] ) ):-
                    perguntavel( X ) ,
                    nao_foi_perguntada( X ),
                    !,
                    faz_pergunta( X, Meta_Saida, Id, Tipo ),
                    lembre_resposta( Meta_Saida, Tipo ),
                    Tipo = sim,
                    X = Meta_Saida,
                    dispara( regra( Id, _ , Acao, Requisitos ) ).

lembre_fornecedor( X ) :- recorded( fornecedor(_,_), fornecedor( X , Idx ),_),
                          recorded( rlinha(_), rlinha( Y ), Ref ),
                          replace( Ref, rlinha( [ Idx | Y ] ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Atualiza o descritor de estado do problema com a parte acao da regra         %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
atualiza_descritor( Id, [] ) :- !.

atualiza_descritor( Id, [X|Acao] ) :- not( recorded(provado(_),provado( X ),_)),
                                      lembre_provado( X ),
                                      recordz( fornecedor(_,_),
                                      fornecedor( X , Id ),_ ),
                                      ! ,
                                      atualiza_descritor( Id, Acao ).

atualiza_descritor( Id, [X|Acao] ) :- recorded(provado(_),provado( X ),_),
                                      !,
                                      atualiza_descritor( Id, Acao ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                 %
%  Paga as regras que atenderam aos requisitos da regra selecionada               %
%                                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

paga_regras :- recorded( rlinha(_), rlinha( L ), _ ),
               num_elementos( L, N ),
               N \== 0,
               !,
               paga_regrasl( L, N ),
               imprime_rlinha.

paga_regras :- paga_ambiente.      % Se rlinha e' vazio, paga ao ambiente

paga_regrasl( [], N ) :- !.

paga_regrasl( [ Id | Cauda ], N ) :- recorded( atrib(_,_,_,_,_),
                                              atrib( Id, X, Y, St, Z ), Ref ),
                                     constante( C ) ,
                                     recorded(maxpot(_,_), maxpot( R , _ ), _),
                                     recorded(potencial(_,_),potencial(R,Pl),_),
                                     Nl is St + ( C * Pl ) / N ,
                                     NSt is round( Nl,4 ),
                                     replace( Ref, atrib(Id, X, Y, NSt, Z ) ),
                                     ! ,
                                     paga_regrasl( Cauda, N ).

paga_ambiente :- constante( C ),
                 recorded(   maxpot(_,_),   maxpot( Id , _ ),_),
                 recorded(potencial(_,_),potencial( Id , P ),_),
                 Nl is C * P ,
                 recorded( ambiente(_), ambiente( Capital ), Ref ),
                 N is Capital + Nl ,
```

```
                 NC is round( N , 4 ),
                 replace( Ref, ambiente( NC ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_rlinha     :- recorded(depuracao(_),depuracao(on),_),
                      recorded(dispositivo(_),dispositivo(H),_),
                      nl(H),
                      write(H,'Conjunto de regras que contribuiram :'),
                      nl(H), nl(H),
                      write(H,'Ciclo  ¢ rlinha !      '),
                      nl(H),  %   nn     [R1, R2, ... ,Rn]
                      recorded( ciclo(_), ciclo(T),_),
                      recorded(rlinha(_),rlinha(L),_),
                      tab(H,2),write(H, T ),
                      tab(H,4),write(H, L ),
                      nl(H),
                      testa_pausa(H).

imprime_rlinha.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%  Verifica se a tarefa do sistema terminou.                               %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

verifica_goal :- goal( X ),
                 ( recorded(provado(_),provado( X ),_) ; fato( X ) ).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%  Efetua pagamentos finais do ambiente ao plano gerado                    %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

paga_ultima_regra :- constante( C ),
                     recorded(ambiente(_),ambiente( Capital ),_),
                     recorded(maxpot(_,_),  maxpot( Id , _ ) ,_),
                     recorded( atrib(_,_,_,_,_),
                               atrib( Id, X, Y, St, Z ), Ref ),
                     N is St + C * Capital,
                     imprime_capital(C,Capital),
                     NSt is round( N,4 ),
                     replace( Ref, atrib( Id, X, Y, NSt, Z )).
```

```
paga_plano :- constante( C ),
              recorded(ambiente(_),ambiente( Capital ),_),
              recorded(   plano(_),   plano( Pl )     ,_),
              retire_rep( Pl , P ),
              num_elementos( P , N ),
              V is Capital * ( 1 - C )/N,
              imprime_pag_plano( C, Capital, N ),
              paga_planol( V , P ).

paga_planol( V , [] ) :- !.

paga_planol( V , [ Id | Cauda ] ) :- recorded( atrib(_,_,_,_,_),
                                               atrib( Id, X, Y, St, Z ), Ref ),
                                     S is St + V ,
                                     NSt is round( S, 4 ),
                                     replace( Ref, atrib( Id, X, Y, NSt, Z )),
                                     !,
                                     paga_planol( V , Cauda ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_capital( C, Capital ):-
                    recorded(depuracao(_),depuracao(on),_),
                    recorded(dispositivo(_),dispositivo(H),_),
                    Pgrl is C * Capital,
                    Pgr is round(Pgrl,4),
                    nl(H),
                    write(H,'Capital final acumulado pelo ambiente :'),
                    nl(H), nl(H),
                    write(H,'Ciclo  Capital    Pag. regra final '),
                    nl(H),  %  nn    XXX.XXXX    XXX.XXXX
                    recorded( ciclo(_), ciclo(T),_),
                    tab(H,2),write(H, T ),
                    tab(H,4),write(H, Capital ),
                    tab(H,16),write(H, Pgr ),
                    nl(H),
                    testa_pausa(H).

imprime_capital( _,_ ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

imprime_pag_plano( C, Capital, N ):-
                    recorded(depuracao(_),depuracao(on),_),
                    recorded(dispositivo(_),dispositivo(H),_),
                    Pgrl is Capital * ( 1 - C )/N,
                    Pgr is round(Pgrl,4),
```

```
                    nl(H),
                    write(H,'Pagamento efetuado a cada regra no plano :'),
                    nl(H), nl(H),
                    write(H,'Ciclo  Pag. regras '),
                    nl(H),  %  nn    XXX.XXXX
                    recorded( ciclo(_), ciclo(T),_),
                    tab(H,2),write(H, T ),
                    tab(H,4),write(H, Pgr ),
                    nl(H),
                    testa_pausa(H).

imprime_pag_plano( _,_,_ ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Mostra plano gerado para avaliar se e' satisfatorio ou nao                 %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mostra_plano   :- recorded( plano(_),plano( Pi ), _ ),
                  inverte_lista( Pi, P ),  limpa_tela(H) ,
                  recorded(dispositivo(_),dispositivo(H),_),
                  nl(H), nl(H),
                  write(H,'Plano  : ' ),
                  write(H, P ), nl(H).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Atualiza ciclo de reconhecimento-acao : t = t + 1                          %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

novo_ciclo :- recorded( ciclo(_), ciclo( T ), Ref ),
              Tn is T + 1 ,
              replace( Ref, ciclo( Tn ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Procedimentos de manutencao da Base de Dados do Motor de Inferencia.       %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

limpa_memorial :- eraseall( con(_)        ),
                  eraseall( rule(_,_)     ),
                  eraseall( pcrule(_,_)   ),
                  eraseall( pcrcon(_,_)   ),
                  eraseall( prule(_,_)    ),
```

```
                 eraseall( probpos(_,_)  ),
                 eraseall( pregra(_,_)   ),
                 eraseall( potencial(_,_)),
                 eraseall( pesp(_,_)     ),
                 eraseall( maxpot(_,_)   ),
                 eraseall( rlinha(_)     ),
                 expunge.

inicializa_bd :- recordz( con(_),      con( [] )     , _ ),
                 recordz( rlinha(_),   rlinha( [] )  , _ ),
                 recordz( maxpot(_,_), maxpot( _ , 0 ), _ ),
                 inicializa_bdl,
                 ctr_set( 0, 1 ),
                 repeat,
                 ctr_inc( 0, Cont ),
                 recordz( rule(_,_), rule( Cont, [] ), _ ),
                 Cont == 5.

inicializa_bdl :- regra( Id, _ , _ , _ ),
                  [! recordz( pregra(_,_),pregra( Id , 0 ),_ ) !],
                  fail.

inicializa_bdl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                 %
%  Fim do Motor de Inferencia Forward Chaining                    marco de 1992   %
%                                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Anexo C. Rotinas Auxiliares

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  COPPE/UFRJ : Trabalho de Tese de Mestrado em IA                            %
%  ----------   ----------------------------------                            %
%                                                                              %
%  Assunto   :  Rotinas Auxiliares do Motor de Inferencia do S.E.             %
%                                                                              %
%                                                                              %
%  Aluno     :  Jayme Bentes                                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ja_provado(Meta) :-
                        recorded( provado(_), provado(Meta) , _ ).

sistema(Meta) :-
                        functor(Meta,NMeta,AMeta),
                        system(NMeta/AMeta).

lembre_provado(Meta) :-
                        recordz(provado(_),provado(Meta),_),
                        explica_pred(Meta,MetaExp),
                        concat($Provado que $,MetaExp,Expl),
                        escreve_janela(crit,[Expl]),
                        leia_janela(pausa,_).


base_conhecimento(Meta,C) :-
                        regra(Id, Atrib, Acao, Requisitos ),
                        pertence( Meta, Acao ),
                        conv_reqs_term( Id, C).

conv_reqs_term(Id,C) :- regra(Id, Atrib, Acao, ( ReqA ; ReqB ) ),
                        !,
                        concatena( ReqA, C1 ),
                        concatena( ReqB, C2 ),
                        concat( $($,C1,X1), concat(X1,$);$,CA),
                        concat( $($,C2,X2), concat(X2,$)$ ,CB),
                        concat( CA , CB , CS),
                        string_term( CS , C ).

conv_reqs_term(Id,C) :- regra(Id, Atrib, Acao, ( ReqA , ReqB ) ),
                        !,
                        concatena( ReqA, C1 ),
```

```
                        concatena( ReqB, C2 ),
                        concat( $($,C1,X1), concat(X1,$),$,CA),
                        concat( $($,C2,X2), concat(X2,$)$ ,CB),
                        concat( CA , CB , CS),
                        string_term( CS , C ).


conv_reqs_term(Id,C) :-   regra(Id, Atrib, Acao, Reqs ),
                        concatena( Reqs, C1  ),
                        string_term( C1 , C  ).

conv_acao_term(Id,C) :-   regra(Id, Atrib, Acao, Reqs ),
                        concatena( Acao, C1  ),
                        string_term( C1 , C  ).

resposta(nao_sei,Meta) :-
                        desinstancie(Meta,Meta1),
                        recorded( resp(_,_), resp(Meta1,nao_sei), _).

resposta(sim,Meta) :-
                        recorded( resp(_,_), resp(Meta,sim), _ ).

lembre_resposta(Meta,Tipo) :-
                        recordz(resp(_,_),resp(Meta,Tipo),_).

desinstancie(Meta,Meta1) :-
                        functor(Meta,NMeta,Ameta),
                        functor(Meta1,NMeta,Ameta).

nao_foi_perguntada(Meta) :-
                        desinstancie(Meta,Meta1),
                        not(recorded( resp(_,_), resp(Meta1,_), _ ) ).


cont_set0( N , Valor ) :-  recordz( contador(_,_),contador( N, Valor),_ ).

cont_set( N , Valor )  :-  recorded( contador(_,_), contador( N, _ ), Ref ),
                        replace( Ref, contador( N, Valor) ).

cont_is( N , Valor )   :-  recorded( contador(_,_), contador( N, Valor ), Ref ).


limpa_tela(0) :- cls.

limpa_tela(_).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  COPPE/UFRJ : Trabalho de Tese de Mestrado em IA                            %
%  ----------   ----------------------------------                            %
%                                                                             %
%  Assunto   :  Modulo Coletor de Dados do Sistema Especialista               %
%                                                                             %
%                                                                             %
%  ==> Adaptado para uso com a interface de comunicacao com o usuario         %
%                                                                             %
%  Aluno     :  Jayme Bentes                                                  %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

faz_pergunta(MEnt,MSai,Regra,Classe):-
        pergunta(MEnt,Perguntas,ValoresValidos),
        pergunte(MEnt,Perguntas,Regra,ValoresValidos,MSai,Classe).

pergunte(MEnt,Perguntas,Regra,ValoresValidos,MSai,Classe):-
        MEnt=..[RelMet Args],
        escreve_perguntas(Perguntas),
        leia_respostas(Args,Regra,Perguntas,Respostas),
        verifica_respostas(Respostas,ValoresValidos,RespostasCorretas,Classe),
        verifica_var_classe(Classe),
        MSai=..[RelMet RespostasCorretas].

escreve_perguntas(Perguntas):-
        Perguntas=..[_|Progs],
        execute(Progs).

execute([]) :- !.

execute([P|Ps]):-
       call(P),
       !,
       execute(Ps).

leia_respostas([],Regra,Perguntas,NResposta):-
       !,
       leia_janela(inicio,Resposta),
       recorded(plano(_),plano( Planol ), _ ),
       retire_rep( Planol, Plano ),
       transfira( [Regr Plano], Resposta, Perguntas, NResposta).

leia_respostas(Args,Regra,Perguntas,Respostas) :-
       faz_leitura(Args,Regra,Perguntas,Respostas).
```

```
faz_leitura([],_,_,[]) :- !.

faz_leitura( Arg1|Args],Regra,Perguntas,[NRespost Respostas]) :-
       leia_janela(inicio,Resposta),
       recorded(plano(_),plano( Planol ), _ ),
       retira_rep( Planol, Plano ),
       transfira( [Regr Plano], Resposta, Perguntas, NResposta).
       !,
       faz_leitura(Args,Regra,Perguntas,Respostas).

verifica_respostas([],_,[],_):-
       !.

verifica_respostas(Resposta,ValoresValidos,[],Classe):-
       atomic(Resposta),
       !,
       resposta_correta(Resposta,ValoresValidos,Classe).

verifica_respostas([Respost Respostas],[ValorValid|ValoresValidos],
                   [Respost RespCorretas],Classe):-
       verifica_boa_resposta(Resposta,ValorValido,Classe),
       !,
       verifica_respostas(Respostas,ValoresValidos,RespCorretas,Classe).

verifica_respostas([Respost Respostas],ValoresValidos,
                    RespostasCorretas,Classe):-
       leia_janela(inicio,NovaResposta),
       !,
       verifica_respostas([NovaRespost Respostas],
                          ValoresValidos,
                          RespostasCorretas,Classe).

resposta_correta(Resposta,ValoresValidos,Classe):-
       (pertence(Resposta,ValoresValidos);
        Resposta = nao_sei),
       !,
       verifica_classe(Resposta,Classe).

resposta_correta(Resposta,ValoresValidos,Classe):-
       escreve_valores_validos(ValoresValidos),
       leia_janela(inicio,NovaResposta),
       !,
       resposta_correta(NovaResposta,ValoresValidos,Classe).

verifica_boa_resposta(nao_sei,_,nao_sei):-
       !.
```

```
verifica_boa_resposta(Resposta,ValorValido,Classe):-
        Critica_resposta=..[ValorValido,Resposta],
        Critica_resposta.


transfira([Regr Plano],por_que,Perguntas,NResposta):-
        !,
        conv_acao_term( Regra, Meta ),
        conv_reqs_term( Regra, CorpoMeta ),
        explica_porque((Meta,CorpoMeta)),
        leia_janela(cont1,Resp),
        verifica_resposta2(Resp,Plano,NResp,Perguntas,NResposta).

transfira([],por_que,Perguntas,NResposta) :-
        !,
        escreve_janela(porque,
                              [$   Nao h  mais explicacoes disponiveis ... $]),
        leia_janela(pausa,Resp),
        verifica_resposta2(Resp,[],NResp,Perguntas,NResposta).


transfira(Regras,Resposta,_,Resposta).

verifica_resposta2(sim,Plano,NResp,Perguntas,NResposta) :-
        transfira(Plano,por_que,Perguntas,NResposta),!.

verifica_resposta2(nao,Plano,NResp,Perguntas,NResposta) :-
        escreve_perguntas(Perguntas),
        leia_janela(inicio,Resp),
        transfira(Plano,Resp,Perguntas,NResposta).

escreve_valores_validos(ValValidos):-
        monta_lista(ValValidos,Lval),
        concatena_args([ $Os valores validos sao : ( $ | Lval ], $$, Lvalx ),
        fecha_parenteses( Lvalx, Lvaly ),
        escreve_janela(crit,[ Lvaly ]),
        escreve_janela(msg,[ $ Por favor, responda novamente $]).

monta_lista([],Lval) :- !.

monta_lista([X|Y],[X,$ $|_]):- monta_lista(Y).


verifica_classe(nao,nao):-
        !.
verifica_classe(sim,sim):-
```

```
        !.
verifica_classe(nao_sei,nao_sei).


verifica_var_classe(Classe):-
        not var(Classe), !.

verifica_var_classe(sim).

pertence(X,[X|Z]):-
        !.
pertence(X,[ Z]):- !,
        pertence(X,Z).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  COPPE/UFRJ : Trabalho de Tese de Mestrado em IA                             %
%  ----------   ----------------------------------                             %
%                                                                              %
%  Assunto  :  Modulo Justificador das conclusoes do S.E. .                    %
%              Responde  as indagacoes do tipo "por_que" e                     %
%              do tipo " como  " do usuario do sistema.                        %
%                                                                              %
%  ==> Adaptado para uso com a interface de comunicacao com o usuario          %
%                                                                              %
%  Aluno    :  Jayme Bentes                                                    %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Rotinas de explicacao do motivo das perguntas ( explica "por_que" )          %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

explica_porque((Meta,CorpoMeta)):-
     monta_tela(Meta,CorpoMeta),
     monta_texto(porque).

monta_tela(Meta,CMeta):-
     monta_linha($se for provado$),
     nova_linha,
     monta_linha($   $),
     escreve(CMeta),
     nova_linha,
     monta_linha($entao pode-se concluir$),
     nova_linha,
```

```
     monta_linha($    $),
     escreve(Meta).

escreve((A,B)):-
     !,
     monta_linha($( $),
     escreve(A),
     nova_linha,
     monta_linha($e$),
     nova_linha,
     monta_linha($    $),
     escreve(B),
     monta_linha($ )$).

escreve((A;B)):-
     !,
     monta_linha($( $),
     escreve(A),
     nova_linha,
     monta_linha($ou$),
     nova_linha,
     monta_linha($    $),
     escreve(B),
     monta_linha($ )$).

escreve(A):-
    monta_linha(A).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Rotinas de explicacao dos resultados provados ( explica "como" )            %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

explica_como(Meta):-
    [! constroi_arvore(Meta,ArvoreMeta) !],
    interpreta_arvore(ArvoreMeta),
    monta_texto(como),
    leia_janela(pausa,_).

explica_como(Meta).

constroi_arvore(true,true).

constroi_arvore((A,B),(ArvoreA,ArvoreB)):-
    constroi_arvore(A,ArvoreA),
    constroi_arvore(B,ArvoreB).
```

```
constroi_arvore((A;B),(ArvoreA;ArvoreB)):-
    constroi_arvore(A,ArvoreA);
    constroi_arvore(B,ArvoreB).

constroi_arvore(not A,nao(A)):-
    not(constroi_arvore(A,ArvoreA)).

constroi_arvore(known(A),f(A)):-
    fato( A ),
    !.


constroi_arvore( known(A), (A:-ArvoreA) ):-
    ja_provado(A),
    !,
    base_conhecimento(A,CorpoA),
    constroi_arvore( CorpoA, ArvoreA ).

constroi_arvore(A,p(A)):-
    primitiva(A),
    !,
    call(A).

constroi_arvore(A,s(A)):-
    sistema(A),!,
    call(A).

constroi_arvore(A,f(A)):-
    fato(A),
    !.

constroi_arvore(A,f(A)):-
    foi_perguntado(A,sim),
    !.

constroi_arvore(desconhecido(A),desconhecido(A)):-
    foi_perguntado(A,nao_sei),
    !.

constroi_arvore(A,(A:-ArvoreA)):-
    ja_provado(A),!,
    base_conhecimento(A,CorpoA),
    constroi_arvore(CorpoA,ArvoreA).

foi_perguntado(Meta,R):-
```

```
    recorded(resp(_,_), resp(Meta,R), _ ).

interpreta_arvore((A,B)) :-
   interpreta_arvore(A),
   nova_linha,
   monta_linha($e$),
   nova_linha,
   interpreta_arvore(B).

interpreta_arvore((A;B)) :-
   var(A),
   interpreta_arvore(B).

interpreta_arvore((A;B)) :-
   var(B),
   interpreta_arvore(A).

interpreta_arvore(p(A)) :-
   escreve_primitiva(A).

interpreta_arvore(s(A)) :-
   escreve_sistema(A).

interpreta_arvore(f(A)) :-
   escreve_fato(A).

interpreta_arvore(desconhecido(A)) :-
   escreve_desconhecido(A).

interpreta_arvore(nao(A)) :-
   escreve_nao(A).

interpreta_arvore(decisao(A)) :-
   escreve_decisao_principal(decisao(A)).

interpreta_arvore((A :- B)) :-
   escreve_regra(( A:- B)),
   monta_texto(como),
   !,
   mensagem_pergunta(B).

escreve_primitiva(A) :-monta_linha($   $),
   monta_linha(A),
   monta_linha($   uma primitiva $).

escreve_fato(A) :- monta_linha($   $),
   monta_linha(A),
```

```
   monta_linha($   um fato $).

escreve_sistema(A) :- monta_linha($   $),
   monta_linha(A),
   monta_linha($    uma clausula do sistema $).

escreve_desconhecido(A) :-monta_linha($   $),
   monta_linha(A),
   monta_linha($ foi respondido com "nao_sei"  $).

escreve_nao(A) :-monta_linha($   $),
   monta_linha(A),
   monta_linha($ foi provado como falso $).

escreve_regra((A:-B)):-
      monta_linha($considerando verdade $),
      nova_linha,
      escreve_condicoes_regra(B),
      nova_linha,
      monta_linha($conclui-se que$),
      nova_linha,
      monta_linha($   $),
      monta_linha(A).

escreve_condicoes_regra((A,B)):-
      escreve_condicoes_regra(A),
      nova_linha,
      monta_linha($e$),
      nova_linha,
      escreve_condicoes_regra(B).

escreve_condicoes_regra((A;B)):-
      var(A),
      escreve_condicoes_regra(B).

escreve_condicoes_regra((A;B)):-
      var(B),
      escreve_condicoes_regra(A).

escreve_condicoes_regra((A:-B)):-
      !,
      monta_linha($   $),
      monta_linha(A).

escreve_condicoes_regra(Meta):-
      Meta=..[_| A1],
      monta_linha($   $),
```

```
        monta_linha(A).

mensagem_pergunta(B):-
        leia_janela(cont2,Resp),
        verifica_respostal(Resp,B).

mensagem_pergunta([]):-
        leia_janela(pausa,_).

verifica_respostal(sim,B) :- !,
                              interpreta_arvore(B).

verifica_respostal(nao,_) :- !,
                              fail.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Rotinas auxiliares 'a montagem dos textos explicativos para as janelas     %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

monta_linha(X) :-string(X),
                 recorded(linha(_),linha(Linha),_),
                 !,
                 concat(Linha,X,Linhal),
                 deleta_linha(Linha),
                 recorda(linha(_),linha(Linhal),_).

monta_linha(X) :-string(X),
                 !,
%                concat($$,X,Linhal),
                 recorda(linha(_),linha( X ),_).

monta_linha(X) :- explica_pred(X,X1),
                  !,
                  monta_linha(X1).

nova_linha      :- recorda(linha(_),linha($$),_).

deleta_linha(X) :- recorded( linha(_),linha(X), Ref),
                   erase( Ref ),
                   !.
deleta_linha(_).

monta_texto(X) :- not( recorded(linha(_),linha(Linha),_) ),
                  !.
```

```
monta_texto(X) :-
              findall( Linha,
                       recorded(linha(_),linha(Linha),_),
                       TextoI),
              inverte_lista(TextoI,Texto1),
              limpa_texto_bc(TextoI),
              prim_maiusc(Texto1,Texto),
              escreve_janela(X,Texto).

limpa_texto_bc([]) :- !.

limpa_texto_bc([Linh Texto]) :-
              deleta_linha(Linha),
              !,
              limpa_texto_bc(Texto).

prim_maiusc([C1|Cauda],[C|Cauda]) :-
              string_length(C1,L),
              prim_str_nao_branco(C1,L,0,Ind),
              substring(C1,Ind,1,C2),
              converte_maiusc(C2,C0),
              substring(C1,0,Ind,C3),
              L1 is L - Ind - 1,
              Ind1 is Ind + 1,
              substring(C1,Ind1,L1,C4),
              concat([C3,C0,C4],C).

prim_str_nao_branco(C,0,Ind,0) :- !.

prim_str_nao_branco(C,Tam,Pos,Ind) :-
              substring(C,Pos,1,C1),
              C1 == $ $,
              Tam1 is Tam - 1,
              Pos1 is Pos + 1,
              prim_str_nao_branco(C,Tam1,Pos1,Ind).

prim_str_nao_branco(C,Tam,Pos,Pos) :-
              substring(C,Pos,1,C1),
              C1 \== $ $,
              !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%  COPPE/UFRJ : Trabalho de Tese de Mestrado em IA                         %
%  ----------   ---------------------------------                          %
%                                                                          %
%  Assunto  :  Modulo Tradutor das relacoes na BC do S.E. .                %
```

```
%              Traduz os predicados "explicaveis" e troca os                %
%              enderecos das variaveis por nomes simbolicos.                %
%                                                                           %
%  ==> Adendo ao Nucleo de Sistema Especialista ( UFSC / ILTC )             %
%                                                                           %
%  Aluno     :  Jayme Bentes                                                %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%  Rotinas para substituir predicados pelas explicacoes respectivas         %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

explica_pred(P,Z) :-                    % P   um termo
     P =.. [ Pred ],
     !,
     explica(Pred,Z).


explica_pred(desconhecido(P),Z) :-      % P   "desconhecido"
       !,
       explica_pred(P,Z1),
       concat($desconhecido $,Z1,Z).

explica_pred(known(P),Z) :-             % P   "conhecido"
       !,
       explica_pred(P,Z1),
       concat($conhecido $,Z1,Z).

explica_pred(P,Z) :-                    % P   um predicado unario,
     P =.. [ Pred, Arg ],               %       nao instanciado.
     var(Arg),
     !,
     explica(Pred,X1),
     concat(X1,$   igual a $,X2),
     renomeia_var(Arg,Var),
     concat(X2,Var,Z).

explica_pred(P,Z) :-                    % P   um predicado unario,
     P =.. [ Pred, Arg ],               %       instanciado
     !,
     explica(Pred,X1),
     concat(X1,$   igual a $,X2),
     explica(Arg,Val),
     concat(X2,Val,Z).
```

```
explica_pred(P,Z) :-                     % P   um predicado binario
     P =.. [ Op, Arg1, Arg2 ],
     !,
     explica( Op, Opx ),
     explica_args(  Arg1,Arg2], [], Caudax ),
     inverte_lista( Caudax,  Argx,Argy] ),
     concat( $ $, Opx, Opy ),
     concat(Argx,Opy,XX),
     concat(XX,Argy,Z).

explica_pred(P,Z) :-                     % P   um predicado n-ario
       P =.. [ Op | Cauda ],
       !,
       explica( Op, Opx ),
       explica_args(  Cauda, [], Caudax ),
       inverte_lista( Caudax, Cauday ),
       concat( Opx, $( $, Opy ),
       concatena_args( [ Opy | Cauday ], $$ ,XX ),
       fecha_parenteses( XX, Z ).

explica_pred(P,Z) :- explica( P, Z ).



explica_args( [], Lista1, Lista1 ):-!.


explica_args( [ X | C ], [], Lista2 ) :-
       string_P( X, Y),
       !,
       explica_args( C , [ Y ], Lista2 ).


explica_args( [ X | C ], Lista1, Lista2 ) :-
       string_P( X, Y),
       !,
       explica_args( C , [ Y | Lista1 ], Lista2 ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Rotinas auxiliares 'a substituicao dos predicados                         %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

string_P( X, Y )  :- explica_pred( X, Y ),
                     not contem_var_P( Y),!.


string_P( X, XN ) :- explica_pred( X, Y ),
```

```
                    substitui_var( Y, XZ ),
                    !,
                    string_P( XZ, XN ).

contem_var_P( X ) :- string_search( $_$, X, N1 ).

substitui_var( X, XN ) :-
                    string_search( $_$, X, N1 ),
                    string_length( X, L ),
                    substring( X, N1, 5, X1 ),
                    renomeia_var( X1, X2 ),
                    substring( X, 0, N1, X3 ),
                    N2 is N1 + 5, L3 is L - N2,
                    substring( X, N2, L3, X4 ),
                    concat( X3, X2, X5 ),
                    concat( X5, X4, XN ).

explica(Predicado,Explicacao) :- explicavel(Predicado,Explicacao),!.

explica(Predicado,Explicacao) :- string_term(Explicacao,Predicado).

concatena_args( [], S1, S1 ) :-!.

concatena_args( [ X | C ], $$ ,S2 ) :-
       !,
       concatena_args( C, X, S2 ).

concatena_args( [ X | C ], S, S2 ) :-
       concat( X, $, $, X1 ),
       concat( S, X1, Y ),
       !,
       concatena_args( C, Y, S2 ).

fecha_parenteses( XX, Z ) :-
       string_length( XX, L ),
       L1 is L - 2 ,
       substring( XX, 0, L1, Y ),
       concat( Y, $ )$, Z ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Rotinas para converter enderecos em nomes de Variaveis (X1,...,Xn)          %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

renomeia_var(Addr,Var) :- recorded(lista_addr(_),lista_addr([]),_),
                          !,
```

```
                        string_term(Addrc,Addr),
                        insere4(Addrc,Var,[]).

renomeia_var(Addr,Var) :- recorded(lista_addr(_),lista_addr(Lista_Addr),_),
                        nonvar(Lista_Addr),
                        string_term(Addrc,Addr),
                        pertence3(Addrc,Var,Lista_Addr),!.

renomeia_var(Addr,Var) :- recorded(lista_addr(_),lista_addr(Lista_Addr),_),
                        nonvar(Lista_Addr),
                        string_term(Addrc,Addr),
                        insere4(Addrc,Var,Lista_Addr).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                 %
%  Rotinas auxiliares 'a substituicao de enderecos por nomes de variaveis         %
%                                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pertence3(Addr,Var,Lista_Addr):-
                        n_esimo(N,Addr,Lista_Addr),
                        num_elementos(Lista_Addr,N1),
                        N2 is N1 - N + 1,
                        string_term(SN,N2),
                        concat($ X$,SN,SNV),
                        concat(SNV,$ $,Var),!.

insere4(Addr,$ X1 $,[]) :- !,
                        recorded( lista_addr(_), X, Ref),
                        replace( Ref,lista_addr( Addr])).

insere4(Addr,Var,Lista) :- num_elementos(Lista,N),
                        N1 is N + 1 ,
                        string_term(SN,N1),
                        concat($ X$,SN,SNV),
                        concat(SNV,$ $,Var),
                        recorded( lista_addr(_), X, Ref),
                        replace( Ref, lista_addr( Addr|Lista])).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                 %
%  Fim dos Modulos Auxiliares do SE                              abril de 1991    %
%                                                                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Anexo D. Base de Conhecimentos - Exemplo

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  COPPE-UFRJ: Trabalho de Tese de Mestrado em IA                             %
%  ----------- ----------------------------------                             %
%                                                                             %
%  Assunto   :  Base de Conhecimento do Problema Exemplo                      %
%                                                                             %
%                                                                             %
%  Aluno     :  Jayme Bentes                                                  %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Fatos  pr -estabelecidos                                                   %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fato(rep($Toby$)).
fato(prod(romance)).
fato(qtr(3)).
fato(sales(24000)).
fato(oldtitles(100)).
fato(newtitles(10)).
fato(localads(1200)).
fato(quota(20000)).
fato(growth(0.01)).
fato(unemployment(0.086)).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Regra numero 01                                                            %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
regra($R01$,[70,20,2,1],[base(X)],
                    [sales(Y),quota(Z),Y>1.15*Z,X is Z+(Y-1.15*Z)]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                             %
%  Regra numero 02                                                            %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
regra($R02$,[60,10,2,2],[base(Z)],
                         [sales(Y),quota(Z),Y =< 1.15 * Z ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 03                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R03$,[80,8,2,3],[efactor(X)],
                        [economy(good),known(growth(X))]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 04                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R04$,[60,20,3,4],[efactor(X1),lafactor(X2)],
                         [economy(fair),known(localads(Z)),known(growth(Y)),
                          X1 is Y / 3 , X2 is Z / 120000 ]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 05                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R05$,[55,25,3,5],[efactor(X)],
                         [economy(poor),known(unemployment(Z)),known(growth(Y)),
                          W1 is 0.085 - Z , W is round(W1,3),  min( Y, W, X) ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 06                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R06$,[85,8,2,6],[economy(good)],
                        [growth(X),X >= 0.04, unemployment(Y), Y < 0.076 ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 07                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
regra($R07$,[80,12,3,7],[economy(good)],
                      [growth(X), X >= 0.02, X < 0.04, unemployment(Y),
                       Y < 0.055 ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 08                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R08$,[75,16,4,8],[economy(fair)],
                      [growth(X), X >= 0.02, X < 0.04,
                       unemployment(Y), Y >= 0.055, Y < 0.082 ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 09                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R09$,[80,8,2,9],[economy(poor)],
                     [ growth(X), X < 0.02,
                       unemployment(Y), Y >= 0.082 ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 10                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R10$,[60,12,2,10],[lafactor(X)],
                        [ economy(good), localads(Y), Y > 2000,
                          X is Y / 100000 ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
%  Regra numero 11                                                           %
%                                                                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R11$,[50,10,2,11],[lafactor(-0.015)],
                        [ economy(poor), localads(X), X < 1500 ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                            %
```

```
%  Regra numero 12                                                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R12$,[40,15,2,12],[ lafactor(0) ],
                         ([ economy(poor), localads(X), X >= 1500 ];
                          [ economy(good), localads(Y), Y < 2000 ] ) ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Regra numero 13                                                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R13$,[35,30,1,13],[ pfactor(X), strong(true), weak(false) ],
                          [ prod(Y), ( Y = computer; Y = romance; Y = scifi ),
                            newtitles(X1), oldtitles(X2),
                            X is ( X1 + X2 ) / X2 - 1 ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Regra numero 14                                                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R14$,[35,30,1,14],[ pfactor(X), strong(false), weak(false) ],
                [ prod(Y),
                  ( Y = reference; Y = biography; Y = psychology; Y = sports ),
                  newtitles(X1), oldtitles(X2),
                  X is 0.75 * ((X1+X2)/X2-1) ]).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Regra numero 15                                                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R15$,[30,40,5,15],[ newquota(X) ],
                         [ strong(true), known(base(X1)), known(efactor(X2)),
                           known(pfactor(X3)), known(lafactor(X4)),lerise(Rise),
                           X is X1 * ( 1 + X2 + X3 + X4 + Rise/100 ) ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%  Regra numero 16                                                              %
```

```
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R16$,[30,40,5,16],[ newquota(X) ],
                         [ weak(true), known(base(X1)), known(efactor(X2)),
                           known(pfactor(X3)), known(lafactor(X4)),lefall(Fall),
                           X is X1 * ( 1 + X2 + X3 + X4 - Fall/100 ) ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Regra numero 17                                                             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R17$,[35,25,6,17],[ newquota(X) ],
                         [ weak(false) , strong(false) ,
                           known(base(X1)), known(efactor(X2)),
                           known(pfactor(X3)), known(lafactor(X4)),
                           X is X1 * ( 1 + X2 + X3 + X4 ) ]  ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Regra numero 18                                                             %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

regra($R18$,[40,45,1,18],[ pfactor(X), weak(true), strong(false) ],
                       [ prod(Y),
                         Y \= computer, Y \= romance, Y \= scifi, Y \= reference,
                         Y \= biography, Y \= psychology, Y \= sports ,
                         newtitles(X1), oldtitles(X2),
                         X is 0.45 * ((X1+X2)/X2-1) ]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  Relacoes primitivas da BC                                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

primitiva(known(X)).

primitiva(min(X,Y,Z)).

known( X ) :- recorded(provado(_),provado( X ),_) ;
              fato( X ).

min(X,Y,X) :- X =< Y, !.
```

```
min(X,Y,Y) :- X > Y .

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                    %
%  Relacoes perguntaveis da BC                                       %
%                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

perguntavel(lerise(X)).

perguntavel(lefall(X)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                    %
%  Relacoes explicaveis  da BC                                       %
%                                                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

explicavel(>=,$maior ou igual a $).

explicavel(=<,$menor ou igual a $).

explicavel(>,$maior que $).

explicavel(<,$menor que $).

explicavel(=,$  igual a $).

explicavel(is,$  igual a $).

explicavel(true,$verdade $).

explicavel(false,$falso $).

explicavel(round,$arredondado em $).

explicavel(min,$minimo$).

explicavel(sales,$as vendas do produto $).

explicavel(quota,$a cota de venda do vendedor $).

explicavel(base,$a quantidade base para a nova cota $).

explicavel(economy,$a perspectiva economica local $).
```

```
explicavel(good,$boa $).

explicavel(fair,$regular $).

explicavel(poor,$ruim $).

explicavel(prod,$o produto $).

explicavel(oldtitles,$o numero de titulos antigos nesta linha $).

explicavel(newtitles,$o numero de titulos novos nesta linha $).

explicavel(growth,$a taxa de crescimento economico $).

explicavel(unemployment,$a taxa de desemprego $).

explicavel(rep,$o representante $).

explicavel(qtr,$o trimestre $).

explicavel(lafactor,$o fator de propaganda local $).

explicavel(localads,$a quantia gasta em propaganda local $).

explicavel(pfactor,$o fator produto $).

explicavel(strong,$linha de produto forte $).

explicavel(weak,$linha de produto fraca $).

explicavel(efactor,$o fator economico $).

explicavel(newquota,$a nova cota de vendas $).

explicavel(lerise,$o crescimento esperado das vendas $).

explicavel(lefall,$a queda esperada das vendas $).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%  COPPE-UFRJ: Trabalho de Tese de Mestrado em IA                              %
%  ---------- ----------------------------------                               %
%                                                                              %
%  Assunto   :  Base de Conhecimento do Problema Exemplo ( Perguntas )         %
%                                                                              %
```

```
%                                                                              %
%  Aluno     :  Jayme Bentes                                                   %
%                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

pergunta(lerise(X), pg(pg_lerise), [crit_lerise]).

pergunta(lefall(X), pg(pg_lefall), [crit_lefall]).

pg_lerise :- fato(prod(Y)),

             concat($Qual o % crescimento estimado vendas do produto $,Y,X),

             escreve_janela(perg,[X]).

pg_lefall :- fato(prod(Y)),

             concat($Qual o % decrescimo estimado vendas do produto $,Y,X),

             escreve_janela(perg,[X]).

crit_lerise(X) :- number(X),

                  X >= 0, X =< 100.

crit_lerise(_) :- escreve_janela(crit,

                  [$Responda com valores numericos entre 0 e 100$]),

                  favor_nova_resposta,

                  fail.

crit_lefall(X) :- number(X),

                  X >= 0, X =< 100.

crit_lefall(_) :- escreve_janela(crit,

                  [$Responda com valores numericos entre 0 e 100$]),

                  favor_nova_resposta,

                  fail.

favor_nova_resposta :-
```

```
                escreve_janela(msg,

                [$- Por favor responda novamente$]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                                      %
%  Fim da Base de Conhecimento                                    abril de 1991        %
%                                                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Bibliografia

[01] Naylor, Chris; - *"Build your own PC Expert System"*; Halsted Press; 1987

[02] Waterman, D A; - *"A Guide to Expert Systems"*; Addison-Wesley; 1986

[03] Cohn, Louis F; - *"Uma visão geral dos SE e sua utilização na Engenharia de Transportes nos EUA"*; artigo; Universidade de Louisville; 1988

[04] Rodrigues, S R; Monard, M C; - *"Uso de Meta-Interpretadores no Desenvolvimento de Núcleos de Sistemas Especialistas"*; ICMS-USP e ILTC; 1990

[05] Forsyth, Richard; - *"Expert Systems - Principles and Case Studies"*; Chapman and Hall Computing; in LEARNING Cap 10-13; 1984

[06] Smith, Stephen F; - *"Adaptative Learning Systems"*; in Expert Systems - Principles and Case Studies; Cap 11; 1984

[07] Deng, Pi-Sheng et alli; - *"A Skill Refinement Learning Model for Rule-Based Expert Systems"*; IEEE Expert; April 1990; pp 15-28

[08] *"The Arity/Prolog Language Reference Manual"*; Arity Corporation; 1988