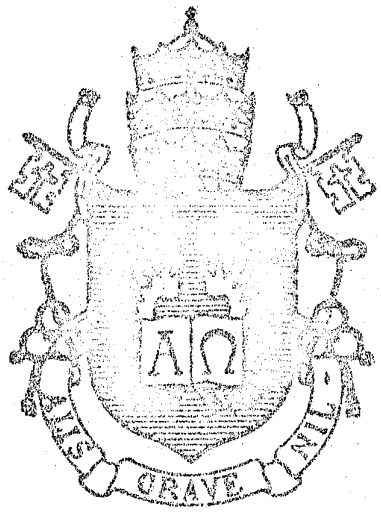

PUC

Série: Monografias em Ciência da Computação
Nº9/88

UM MONITOR PARA APOIO À PROGRAMAÇÃO EM LÓGICA

Jeferson Ferreira Soares
Paula Ypiranga dos Guaranys

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP 22453
RIO DE JANEIRO – BRASIL

PUC/RJ - Departamento de Informática

Série: Monografias em Ciência da Computação, nº9/88
Editor: Paulo A. S. Veloso Setembro de 1988

UM MONITOR PARA APOIO À PROGRAMAÇÃO EM LÓGICA*

Jeferson Ferreira Soares
Paula Ypiranga dos Guaranys

* Projeto parcialmente financiado pelo CNPq e pela SID Informática.
Apresentado pelo prof. Sérgio Eduardo Rodrigues de Carvalho.

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC/RJ-Depto. de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
BRASIL

RESUMO

O artigo descreve uma ferramenta de software para suporte à construção de programas em linguagem de programação em lógica usando-se regras de produção. Inclui-se funções de edição e monitoramento da construção de programas para garantir a completude da especificação e a compatibilidade entre os diversos componentes do programa.

PALAVRAS-CHAVES: ambientes para apoio ao desenvolvimento, programação em lógica, edição orientada por sintaxe, monitor inteligente

ABSTRACT

This paper describes a software tool that aids the construction of programs written in Rule Based Oriented programming languages. Edition and monitoring functions are included. The monitoring functions check the completeness and the matching of the various program components of an application.

KEYWORDS:

software development environments, rule based programming, logic programming, sintax oriented edition, intelligent monitoring.

SUMARIO

LISTA DE FIGURAS.

1. INTRODUÇÃO:

Este artigo aborda alguns aspectos do projeto de uma ferramenta de software utilizada para suporte à construção de programas escritos em uma linguagem de programação em lógica.

A idéia central é fornecer ao usuário um ambiente interativo onde há um monitoramento constante da construção de seu programa.

O capítulo 2 descreve o conjunto de operações de interface disponíveis no sistema e sua forma de utilização.

A estrutura para representação de programas e a estratégia de monitoramento escolhidas para o sistema são enfocadas no capítulo 3.

Discute-se brevemente, ao final do texto, o posicionamento deste trabalho no contexto de um projeto de pesquisa mais abrangente atualmente em curso no Departamento de Informática da PUC/RJ envolvendo alunos de graduação e pós-graduação.

2. O AMBIENTE

O sistema descrito neste trabalho oferece ao usuário um conjunto de funções de edição de programas orientadas para a manipulação da sintaxe de regras de produção do tipo cláusula de Horn. Há operações específicas para a edição de regras, cláusulas e asserções. Em qualquer dos casos pode-se fazer inclusões, retiradas e alterações de elementos da estrutura sintática que representa o programa. Estas operações são sempre acompanhadas por verificações automáticas de consistência a nível sintático.

O procedimento de uso é, em linhas gerais, o seguinte: o usuário interage com o sistema, através de diálogos e sistemas de "menus". A primeira tela exibida em uma sessão de uso pede a definição da regra que funcionará como raiz da estrutura hierárquica do programa a ser desenvolvido. O passo inicial é a descrição da regra em português, para fins de documentação, seguido de sua especificação na linguagem de programação em lógica adotada (Fig. 1).

A partir deste ponto todos os ciclos de operação iniciam-se com a escolha, por parte do usuário, da relação (regra ou fato) a ser editada. (Fig. 2).

As operações de edição utilizam duas janelas simultaneamente apresentadas: a primeira, situada à esquerda da tela, é usada para exibir as opções de operação e receber as entradas por parte do usuário; a segunda exibe o trecho de programa sendo editado naquele instante (Fig. 1). Assim o usuário pode acompanhar o resultado de seu trabalho.

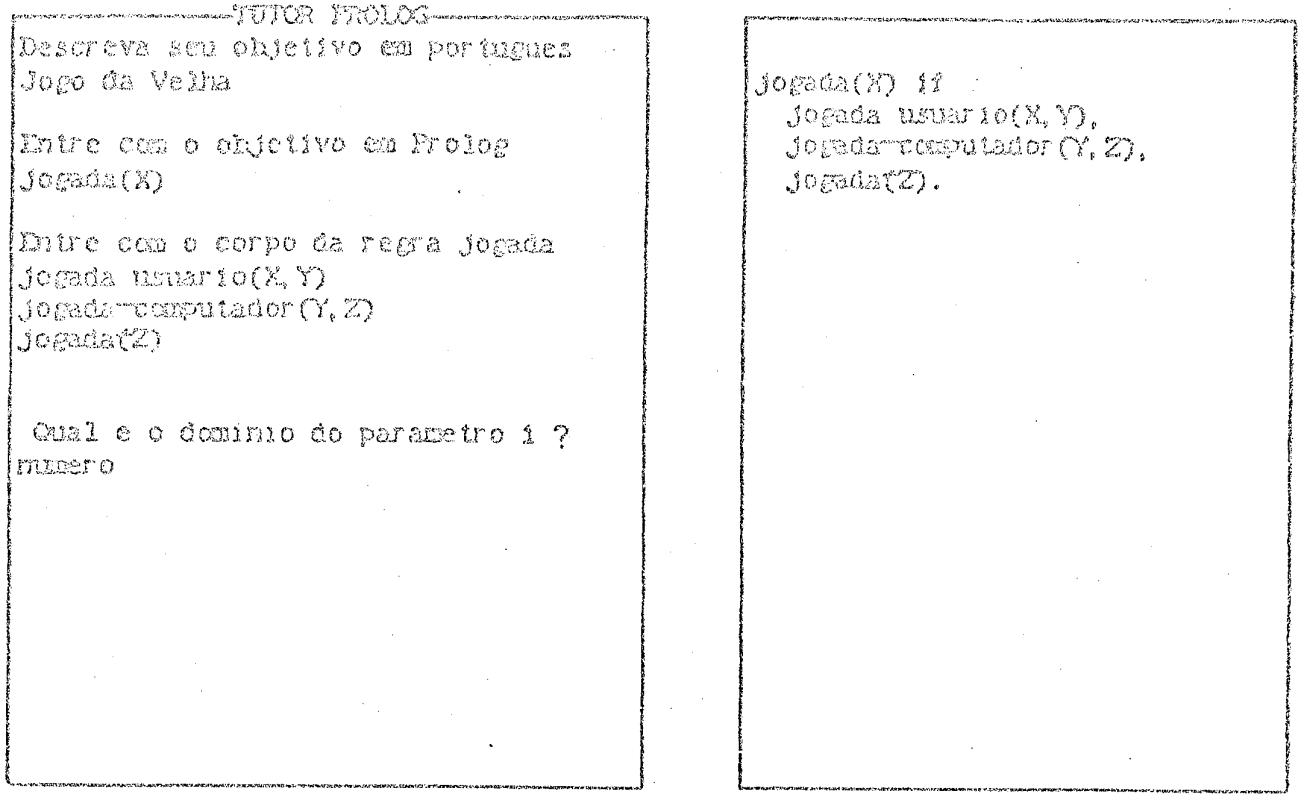

FIGURA 1

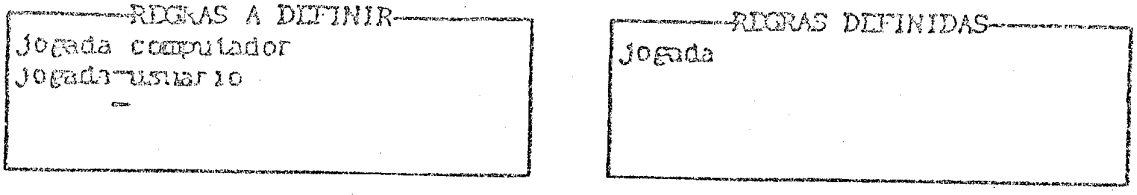

FIGURA 2

2.1 Operações a nível de regras:

Para relações já mencionadas em algum ponto do programa, mas não definidas até aquele momento, estão disponíveis as seguintes funções de edição mostradas na tela da figura 3 e comentadas a seguir.

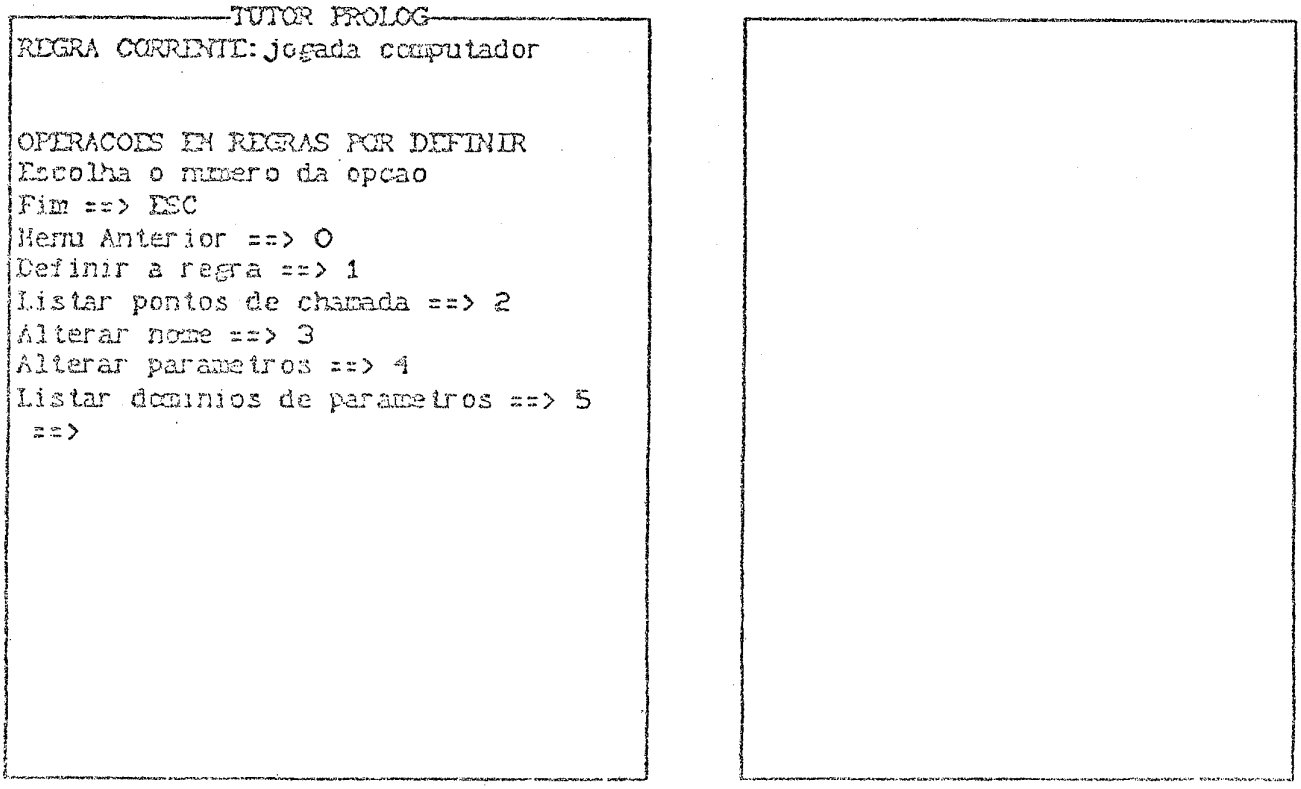

FIGURA 3

. definir a regra

O sistema apresenta um diálogo através do qual o usuário entra com a descrição da primeira cláusula de uma regra.

. listar pontos de chamada

São listadas todas as cláusulas de regras do programa nas

quais apareçam assertivas implementando chamadas a uma determinada regra.

. alterar nome

Dispara um diálogo com o qual o usuário altera o nome de uma regra qualquer. Automaticamente é alterado o nome da regra em sua definição e em todos os pontos de ocorrência da mesma no grafo representativo do programa.

. alterar parâmetros

Com esta função pode-se alterar a aridade e/ou o domínio dos parâmetros de uma regra.

. listar domínios de parâmetros

São exibidas informações sobre a aridade e os domínios dos parâmetros de uma regra.

Para as regras já definidas há ainda duas operações possíveis adicionais (Fig. 4):

. listar a regra

Exibe a descrição de todas as cláusulas que compõem a regra listada.

. operar cláusula

Dispara o menu de operações a nível de cláusula.

Obs: As demais operações apresentadas na figura 4 são também aplicáveis a regras não definidas e por isso já foram mencionadas anteriormente.

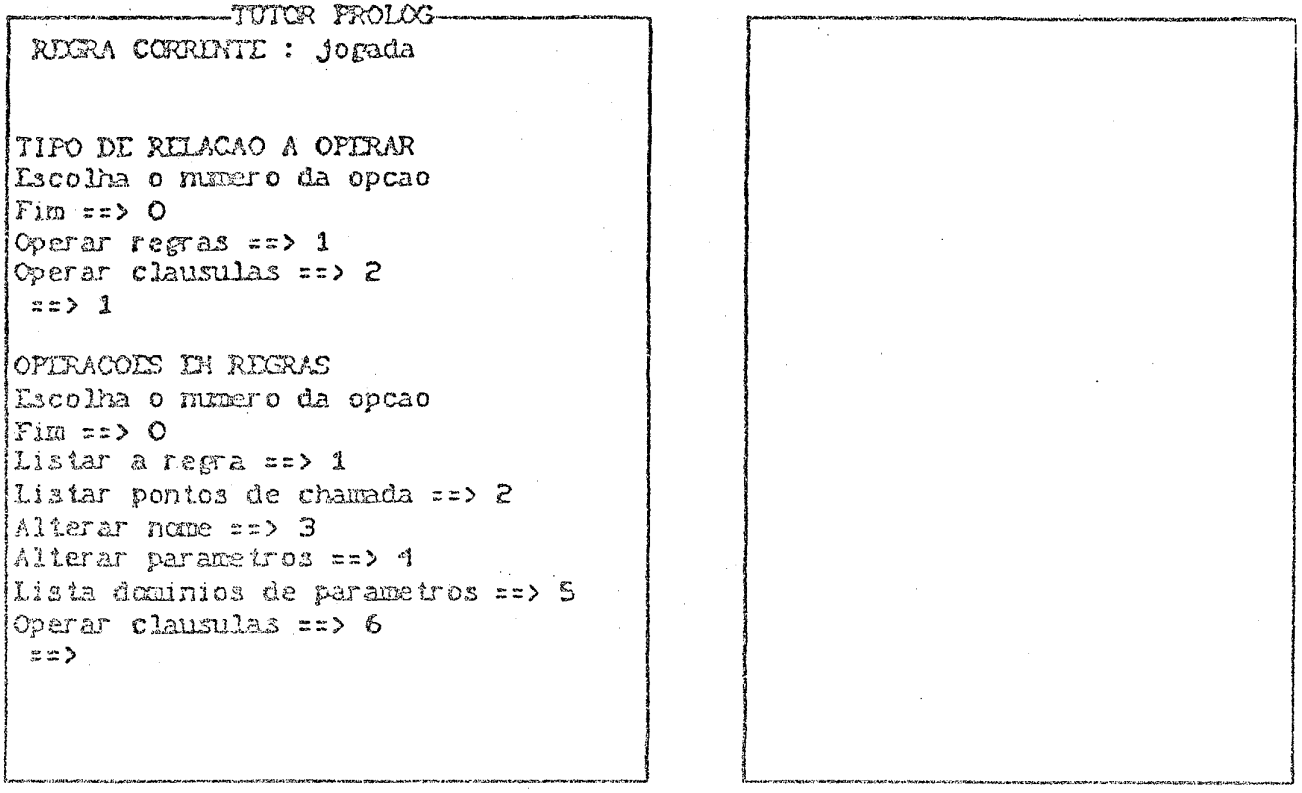

FIGURA 4

2.2 Operações a nível de cláusula:

A figura 5 ilustra as opções de operação a nível de cláusula a seguir enumeradas:

. incluir cláusula

Usa-se esta função para editar uma nova cláusula de uma regra.

. alterar cláusula

Podem-se, com esta função, ativar as operações que

manipulem a edição de assertivas que compõem as cláusulas.

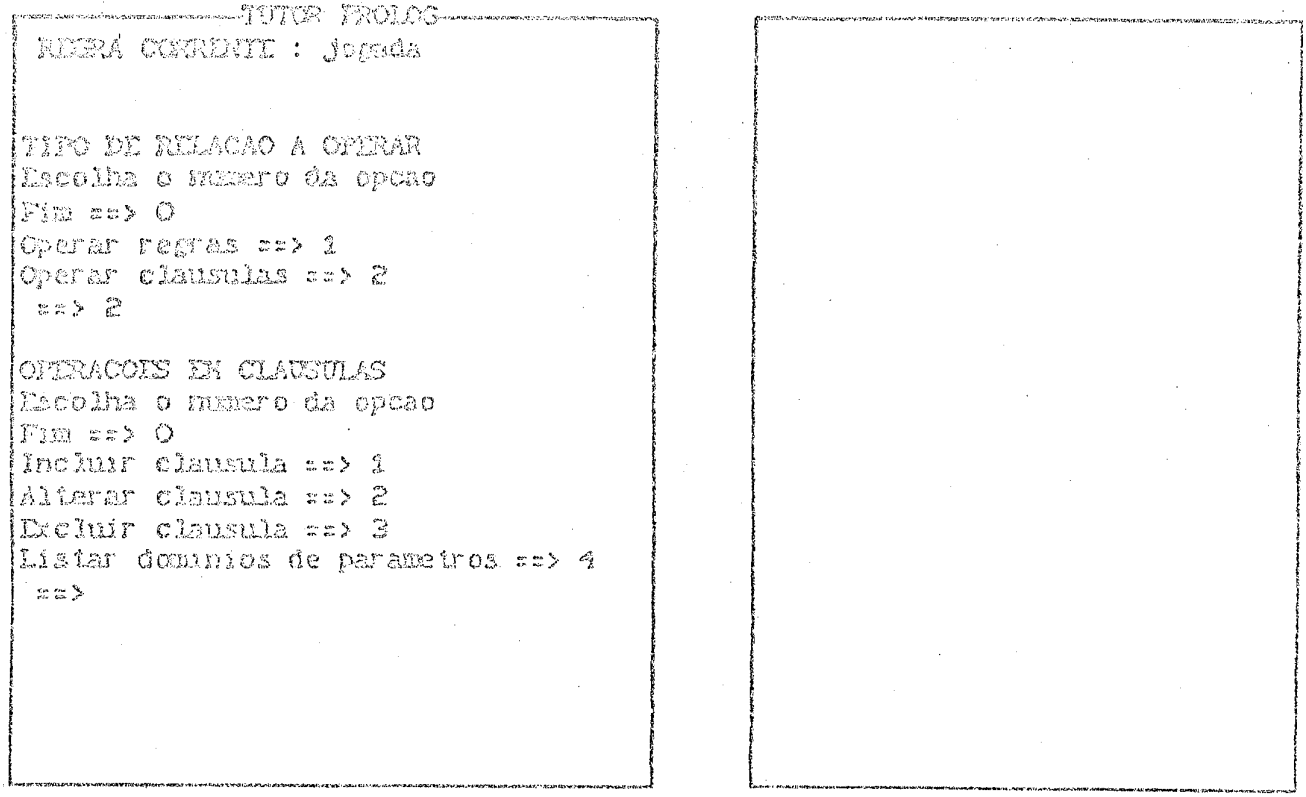

FIGURA 5

. excluir cláusula

Com esta função processa-se a exclusão de uma cláusula de uma regra.

. listar domínios de parâmetros

Para cada nome de parâmetro existente na descrição de uma cláusula são indicados os pontos da cláusula (cabeça da cláusula ou posição de assertiva) em que aquele nome ocorre e o domínio associado àquele

parâmetro em cada ocorrência (Fig. 6).

CLAUSULA ANALISADA:

```
(1) jogada(X) if
[1]       jogada_usuario(X,Y),
[2]       jogada_computador(Y,Z),
[3]       jogada(Z).
```

TABELA DE DOMINIOS DOS PARAMETROS
REGRA: jogada
CLAUSULA: 1

NOME	DOMINIOS	ASSER
X	num	header
		1
Y	indef	1
		2
Z	num	2
		3

Digite qualquer tecla para continuar.
==>

FIGURA 6

2.3 Operações a nível de assertiva:

Alterações no corpo de uma cláusula são feitas através das operações apresentadas na figura 7, que mostra as seguintes opções de funções de edição a nível de assertivas:

. incluir assertiva

Usada para editar a inclusão de uma assertiva no corpo de uma cláusula.

. excluir assertiva

Processa-se com esta função a exclusão de uma assertiva do corpo de uma cláusula.

Obs: Todas as operações de edição, notadamente as que implicam em inclusão, exclusão e alterações sobre elementos componentes de um programa, não apoiadas por tratamento específico de consistência. Com isto verifica-se a possibilidade, ou não, de se executar cada operação no momento de seu disparo e avalia-se o resultado desta execução quando ela acontece.

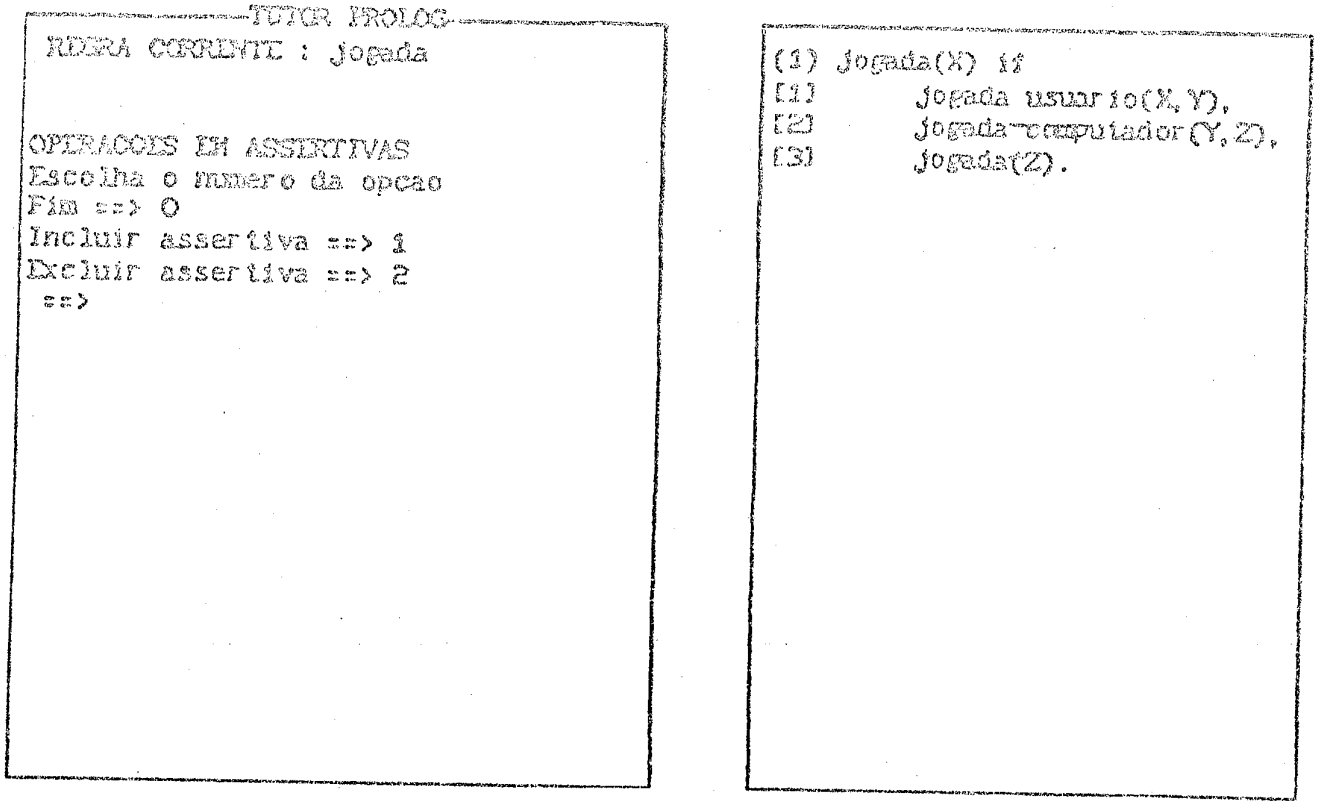

FIGURA 7

2.4 Operações a nível de arquivo:

A qualquer momento podem ser ativadas operações para manipulação de arquivos que contém programas gerados pelo usuário. Estes programas não precisam necessariamente ter sido construídos com o auxílio do sistema descrito neste arquivo.

As operações disponíveis são:

a) criar arquivo.

b) carregar arquivo do disco para edição.

c) salvar sem abandonar a sessão.

d) salvar abandonando a sessão.

Pode-se trabalhar apenas com arquivos contendo códigos fonte compatíveis com o padrão do TURBO - PROLOG 1.1. Dentro do padrão de código adotado um programa dirigido é dividido em quatro seções: definição de domínios, declaração da base de fatos, declarações das regras que compõem o programa e por último a definição das mesmas, isto é, o corpo do programa.

A partir das informações coletadas na interação com o usuário durante a definição do corpo do programa, o sistema descrito monta, automaticamente, as seções de definição e declarações liberando o usuário desta tarefa. Sobre estas, vale ressaltar algumas características interessantes:

a) Na definição de domínios, a declaração da base de fatos e as definições das regras são organizadas em ordem alfabética, com o objetivo de facilitar a localização de nomes pelo usuário. Este cuidado é bastante útil principalmente para as

definições das regras.

b) A declaração das regras é organizada (por meio de identação) refletindo sua hierarquia de chamadas. Esta organização permite que o usuário visualize a arquitetura do programa em termos desta hierarquia.

3. O MONITOR

3.1 A estrutura de representação:

É criada através das funções de edição do monitor, uma estrutura de representação interna que descreve a sintaxe de um programa e garante sua organização semântica de acordo com um modelo descrito por um grafo de dependências (Fig. 8). Neste grafo os arcos representam relacionamentos entre os elementos semânticos do programa, que são representados pelos nós.

De acordo com o modelo escolhido uma aplicação é definida por um conjunto de regras de produção e uma base de fatos.

Os relacionamentos mencionados na figura 8, definem os aspectos mais importantes que estabelecem a estrutura de construção de um programa segundo a abordagem deste trabalho. A leitura para a ocorrência dos mesmos é:

a) um programa é composto por um conjunto de relações.

b) uma regra de produção é composta por um conjunto de cláusulas disjuntas entre si.

c) uma cláusula é composta por uma sequência formada por uma conclusão (a cabeça da cláusula) e um conjunto de assertivas ligadas por conjunções.

d) uma relação pode ser uma regra de produção ou um fato.

e) uma assertiva pode ser um de três tipos:

- uma chamada de uma regra de produção definida pelo usuário, ou

```
e-composto-por ( Programa, "conjunto", [Relacao])
e-composto-por ( Regra, "disjuncao", [Clausula])
e-composto-por ( Clausula, "sequencia", [Conclusao, Corpo-clausula])
e-composto-por ( Corpo-clausula, "conjuncao", [Assertiva])

e-do-tipo ( Relacao, "regra")
e-do-tipo ( Relacao, "fato")
e-do-tipo ( Assertiva, "chamada")
e-do-tipo ( Assertiva, "instanciacao de fato")
e-do-tipo ( Assertiva, "embutida")

tem-dominio_parametros ( Relacao, "col-ord", [nome-parametros])

tem-nome-parametros ( Conclusao, "col-ord", [nome-parametros])
tem-nome-parametros ( Assertivas, "col-ord", [nome-parametros])
tem-nome-parametros ( Instancia-de-fato, "col-ord", [nome-parametros])

tem-valor-associado ( "col-ord", [nome-parametros], "col-ord", [dominio-parametros])

tem-aridade (relacao)
```

FIGURA C

. uma chamada de uma função embutida da linguagem de programação sendo utilizada, ou
. uma instanciação de um fato que consta da base da aplicação

f) cada relação, regra ou fato, tem associada a si uma coleção ordenada (col. ord.) de domínios de parâmetros.

g) cada conclusão de cláusula, assertiva e instanciação de tem associada a si uma coleção ordenada de nomes de parâmetros.

h) cada elemento de uma coleção ordenada de parâmetros tem associado a si um elemento na coleção ordenada de domínios correspondente. Esta associação é feita considerando-se a posição relativa de cada elemento em sua coleção ordenada.

Obs: Construções sintáticas válidas na linguagem de programação adotada podem ser usadas como parâmetros de assertivas do tipo embutido.

i) relações tem aridade fixa.

3.2 Funções de monitoramento:

Com o auxílio das informações obtidas a partir da representação acima mencionada é possível fazer-se um monitoramento da construção do programa através das seguintes funções principais:

3.2.1 - Verificação da completude da especificação em função da eventual existência, ao final de uma sessão de edição, de relações que sejam mencionadas no programa e para as quais não

haja uma definição explícita.

Impede-se também que durante a construção, sejam definidas relações cujo uso não esteja mencionado anteriormente no programa (Fig. 2). Importante notar que esta restrição força o usuário a seguir uma estratégia descendente (TOPDOWN) para a construção de seu programa. Questiona-se, naturalmente a conveniência desta ser a única estratégia de construção suportada por este versão do sistema. Realmente uma continuação prevista para este trabalho é exatamente o apoio a mais de uma estratégia de construção. De qualquer forma o suporte a alguma estratégia já é um passo útil na tentativa de sugerir uma disciplina na construção de programas.

Para aplicações de programação em lógica a estratégia descendente tem-se mostrado, particularmente, bastante adequada.

3.2.2 - Verificação da compatibilidade entre as definições e as instanciações (chamadas) das relações do programa em termos de sua aridade e dos domínios de seus atributos. A aridade é verificada toda vez que a relação, regra ou fato, é mencionada (instanciada)

Instanciações de relações que apresentem incompatibilidade de aridade com suas respectivas definições, são detectadas em tempo de edição. A entrada é ignorada, o usuário recebe a explicação correspondente à ocorrência e é convidado a substituí-la.

A verificação da compatibilidade do domínio dos atributos (parâmetros) das relações é feita, também em tempo de edição, segundo o seguinte procedimento:

- o usuário define uma cláusula de uma regra,

- para cada nome de parâmetro mencionado na cláusula verifica-se qual é o domínio a ele associado em cada uma de suas ocorrências.

- quando o nome de parâmetro aparece com o domínio indefinido ele herda automaticamente o domínio de alguma outra ocorrência sua dentro da cláusula, se houver. A origem do domínio herdado segue a seguinte prioridade:
  - ocorrência na cabeça (conclusão) da cláusula
  - ocorrência em assertiva que seja uma chamada de alguma regra anteriormente definida
  - ocorrência em assertiva que seja chamada de regra não definida e que tenha o domínio daquele parâmetro definido através de mecanismo de herança.

- após o primeiro passo da herança o usuário é convidado a definir os domínios dos parâmetros mencionados na cabeça da cláusula e que tenham permanecido com domínio indefinido. Os que já tenham, neste momento, valor de domínio definido tem este valor explicitamente confirmado, ou alterado, pelo usuário através de diálogo com o sistema.

- automaticamente as ocorrências destes nomes de parâmetros, associadas a domínios indefinidos, nas assertivas da cláusula herdam o domínio arbitrado pelo usuário no passo anterior.

a cláusula é avaliada e se há nomes de parâmetros iguais com domínios diferentes a mesma recebe uma marca. Esta marca serve para chamar a atenção do usuário para a

incompatibilidade detectada e permanece ativa enquanto esta perdurar.

3.2.3 - detecção e indicação automática de trechos do programa que estejam completamente definidos e portanto, prontos para serem testados (Fig. 9).

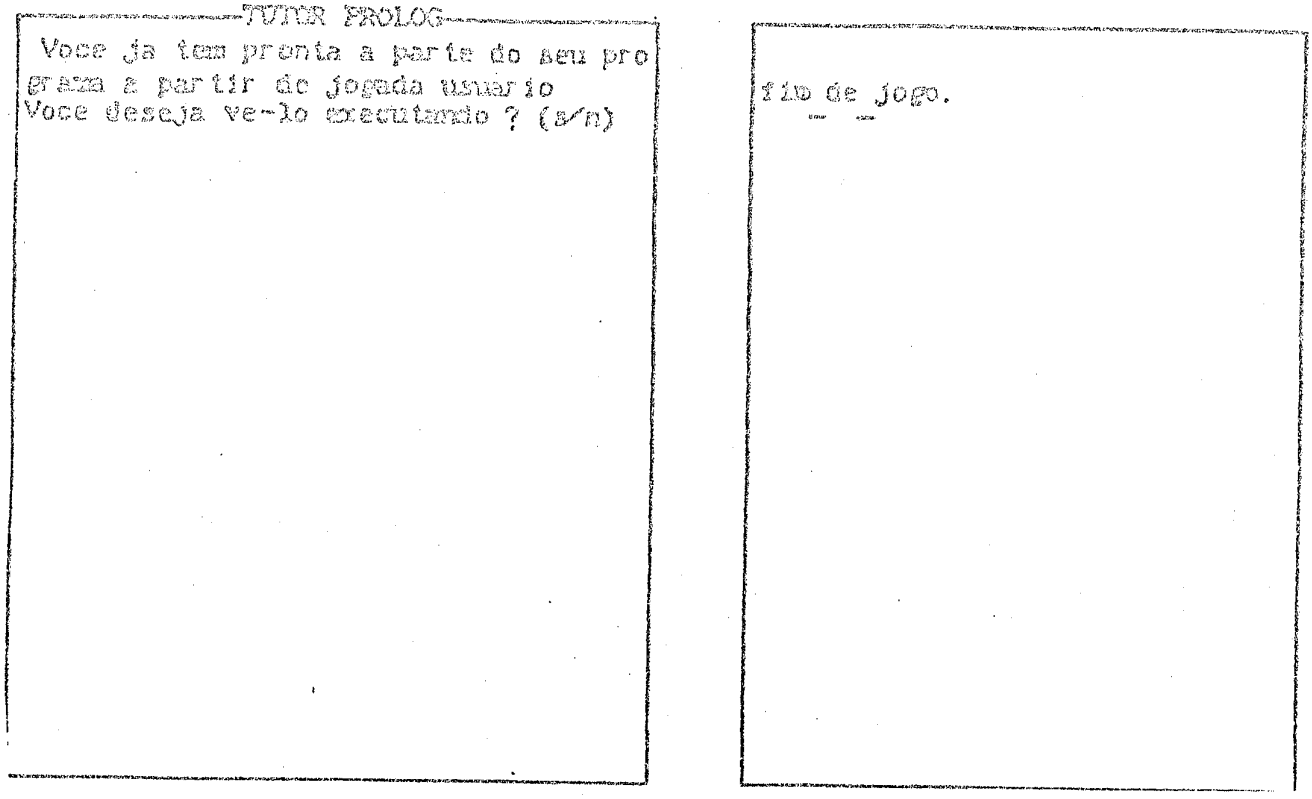

FIGURA 9

Uma regra é considerada pronta para ser testada quando todas as assertivas de todas as suas cláusulas referem-se a:

. instanciações de fatos que constem da base da aplicação ou

. funções embutidas da linguagem de programação utilizada ou

. chamadas de regras construídas pelo usuário e consideradas

prontas para serem testadas segundo este mesmo critério.

Assertivas que determinam chamadas recursivas, diretas ou indiretas, são detectadas e submetidas a tratamento especial. Os caminhos do grafo gerados a partir das mesmas não são percorridos durante a detecção dos trechos de programa eventualmente prontos para execução.

Obs: Há ainda o caso de cláusulas que não tem assertivas e que são consideradas prontas para teste.

3.2.4 - Controle de integridade física do banco de fatos do programa sendo desenvolvido. Não é permitida a edição de assertivas que retirem fatos da base de aplicação que não tenham sido previamente inseridos nesta base por outra assertiva do programa em desenvolvimento.

4. PERSPECTIVA DE INTEGRAÇÃO DESTE TRABALHO

O monitor para montagem de programas acima descrito é, na verdade um componente de um ambiente mais abrangente que visa dar suporte não apenas à implementação e manutenção, mas também ao projeto de programas construídos com técnicas de programação em lógica.

Este tutor é composto por dois grupos básicos de funções: um para a apresentação de tópicos conceituais sobre programação em lógica, e um segundo grupo para a apresentação de exercícios práticos com o acompanhamento de sua solução. Este último grupo é usado também para assistir, de forma interativa, o projeto de programas feitos pelo usuário.

REFERENCIAS BIBLIOGRAFICAS:

. Borland International Inc., Turbo Prolog Owner's Handbook, 1st. Edition, Scotts Valley, 1986, 196.

. C.J.Hogger, Introduction to Logic Programming, 1st.Edition, London, Academic Press Inc. L, 1984, 271.

. D.Sleeman, Intelligent Tutoring Systems, 1st. Edition, London, Academic Press Inc., 1982, 345.

. M. van Caneghem, D.H.D.Warren, Logic Programming and its Aplications, 1st. Edition, New Jersey, Ablex Publishing Corporation, 1986, 318.

. T.Conlon, Start Problem-Solving with Prolog, 1st. Edition, Wokingham, Addison-Wesley Publishers, 1985, 182.