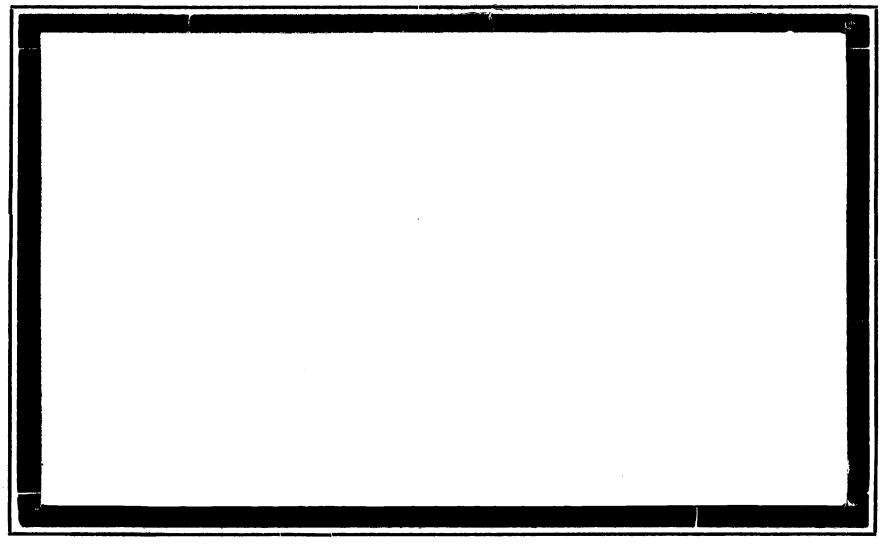
UNIVERSIDADE DE SÃO PAULO



RELATÓRIOS TÉCNICOS

Instituto de Ciências Matemáticas de São Carlos

Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2569

Uma Gramática de Interpretação de Perguntas Sim/Não em Linguagem Natural com Tratamento de Foco

CLAUDIO OSIRO

MARIA DAS GRAÇAS VOLPE NUNES

Nº 13

RELATÓRIOS TÉCNICOS DO ICMSC

São Carlos

Nov. / 93

Uma Gramática de Interpretação de Perguntas Sim/Não em Linguagem Natural com Tratamento de Foco

Cláudio Osiro

Maria das Graças Volpe Nunes

Universidade de São Paulo
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências da Computação e Estatística

Novembro 1993

Índice

• Introdução

Nas últimas décadas o barateamento dos meios computacionais e, por conseguinte, a popularização de seus usos, fizeram surgir a preocupação com a comunicação homem-computador. Tem-se procurado cada vez mais meios de se conseguir uma interface prática e acessível a todos. Essa busca fez surgir uma importante área de pesquisa avançada em computação, dentro da área de Inteligência Artificial, a de Processamento de Linguagem Natural.

Uma das metas das pesquisas da área, em cujo contexto insere-se a nossa, é a construção de um sistema de interface que processe Linguagem Natural, ou seja, um sistema que permita ao usuário leigo comunicar-se com o sistema através de um meio bem natural, a sua própria língua.

A aplicação fica por conta da utilização em interfaces diversas que, embora bastante favorecidas pelo crescimento das interfaces gráficas, necessitem de um meio de comunicação de fácil manuseio, tanto por leigos, quanto por especialistas não iniciados no contexto do programa. Um exemplo básico são as interfaces para Sistemas Baseados em Conhecimento - SBC .

O objetivo do nosso trabalho, em específico, é construir parte de um sistema de interpretação de perguntas em Linguagem Natural. Para tanto, tentaremos encontrar uma gramática que permita o entendimento das questões. Esse módulo de interpretação de Linguagem Natural fará parte de um sistema de geração de respostas cooperativas para SBCs que utilizem a Lógica de Primeira Ordem para a representação de conhecimento.

Visando este objetivo, tentaremos utilizar a estreita relação da Linguagem PROLOG com a Lógica de Primeira Ordem e o fato de PROLOG fornecer a facilidade de interpretar diretamente cláusulas em DCG (Definite Clause Grammar - Gramática de Cláusulas Definidas). Assim sendo, elegemos PROLOG e DGC como os meios de representação da Gramática das Perguntas em Linguagem Natural que pretendemos

interpretar.

No Capítulo 1 discutiremos a relação entre PROLOG e Lógica de Primeira Ordem e o processo de transformação de uma fórmula lógica em cláusula PROLOG.

O Capítulo 2 abrange a representação DCG; mostraremos como é feita a construção de uma gramática nesta representação.

O Capítulo 3 fornece uma visão geral de Ambientes Cooperativos para Sistemas Baseados em Conhecimento.

No quarto Capítulo discutiremos algumas construções básicas de frases que serão interpretadas pela interface que pretendemos implementar, bem como apresentaremos as regras gramaticais propostas para estas construções.

No Capítulo 5 discutiremos algumas maneiras de compor as construções básicas levantadas no capítulo 4 de modo a tratar relações mais complexas como quantificação universal, relações de implicação, etc...

No Apêndice A é fornecida uma lista de exemplos de perguntas tratadas em nosso programa.

No Apêndice B são discutidos alguns aspectos da implementação e sugestões para melhorias futuras.

No Apêndice C é fornecida um listagem completa do programa.

• Capítulo 1 - A Relação entre Lógica e PROLOG.

A Linguagem de Programação PROLOG foi criada por Alain Colmerauer por volta de 1970, como uma tentativa de desenvolver uma linguagem especifica para execução de tarefas em meio lógico, ou seja, construídas em termos lógicos. Vem daí o nome PROLOG, PROgramming in LOGic.

A representação em cláusulas PROLOG, como estudaremos a seguir, foi desenvolvida a partir da representação lógica de informações. Para entendermos este processo, estudaremos os passos para traduzir uma informação escrita na forma lógica para uma cláusula PROLOG [Clocksin-81].

1.1 Uma breve introdução ao Cálculo de Predicados.

O Cálculo de Predicados ou Lógica de Primeira Ordem pode ser descrito como um sistema formal apropriado à definição de teorias matemáticas. Um sistema formal consiste exatamente de uma linguagem formal e de uma abstração adequada para os princípios usados para decidir quando uma assertiva é consequência (lógica) de outras [CASANOVA-86].

Em Cálculo de Predicados representamos os objetos através de termos, e por meio da descrição de objetos, tentamos representar um "mundo". Um resultado importante disto é a incerteza da negação, ou seja, tomamos como verdadeiro uma Proposição que foi possível provar com o conhecimento disponível, caso contrário, ela é considerada falsa. É a chamada "hipótese do mundo fechado".

Proposição é uma sentença declarativa formada por um termo simples (um termo individualmente) ou por vários termos compostos por uso de conectivos (~,^,v,=>,<=>) ou por quantificadores (existencial ou universal). É recomendável a consulta de um livro de lógica aos que queiram se aprofundar ou tirar dúvidas deste ponto em específico.

1.2 Forma Clausal.

Na lógica existem diversas maneiras de se representar proposições. O fato é que a existência destas várias formas de representação pode se tornar problemática quando pensamos que teremos que inferir sobre elas. Sendo assim, torna-se desejável ter apenas uma representação padrão. Pensando nisto é que se propôs a ***Forma Clausal****, que é*

derivada do Cálculo de Predicados mas não tem a diversidade de representações possíveis.

Na Forma Clausal as proposições são representadas por cláusulas que, por sua vez, nada mais são que conjuntos de literais. Um literal é uma fórmula atômica (positivo) ou sua negação (negativo).

Entretanto será possível representar qualquer Proposição na forma clausal? A resposta é negativa. No entanto, a linguagem das de clausulas tem poder suficiente para o uso em raciocinadores automáticos. Além disso, possui algumas vantagens para formular várias regras de inferência [WOS-84]. A conversão de uma Proposição em Cálculo de Predicados para a Forma Clausal obedece os 6 estágios a seguir, onde vale a seguinte relação :

~ - não
 # - ou
& - e
-> - implica
all() - quantificador universal
exists() - quantificador existencial

Estágio 1 - Remoção de implicações.

Remove-se as ocorrências de implicações, -> , utilizando-se a seguinte equivalência:

(a -> b) equivale a (~a # b).

Estágio 2 - Mover negação para os termos atômicos.

Neste estágio remove-se as ocorrências de ~ (negação) sobre proposições não atômicas. Assim sendo, temos :

~(a & b) equivale a (~a # ~b).

~(a # b) equivale a (~a & ~b).

~exists(x,p) equivale a all(x, ~p).

~all(x,p) equivale a exists(x,~p).

Após o estágio 2 as negações são poderão aparecer aplicadas a proposições atômicas. Estas proposições, bem como suas negações, serão chamadas de literais e as trataremos como termos quaisquer pois seus significados são terão importância no final da

transformação.

Estágio 3 - Skolemização

Este estágio trata da remoção do quantificador existencial. Isso se faz através da introdução de constantes na expressão (Constantes de Skolen), estas constantes não representam indivíduos mas a existência destes. Veja os exemplos abaixo:

1) exists(X, fêmea(X) & mãe(X,eve)) passa a ser representado do seguinte modo: fêmea(g1) & mãe(g1,eve).

2) all(X,humano(X) -> exists(Y, mãe(X,Y))) passa a ser representado do seguinte modo: all(g2, mãe(X,g2).

Note que g1 e g2 não representam indivíduos, mas a existência destes.

Observações:

a) No exemplo 2 poderíamos ter:

all(X,humano(X) -> mãe(X,g2(X))).

Neste caso teríamos que g2 não é uma constante mas uma função que representa o valor da mãe de X.

b) Deve-se ter muito cuidado ao escolher o nome de uma constante de modo que ela não tenha sido escolhida anteriormente.

A skolemização é a responsável por a linguagem clausal ser menos expressiva que a Lógica de Primeira Ordem. No entanto, um importante resultado garante que se um conjunto de fórmulas em Lógica de Primeira Ordem não pode ser satisfeito então seu correspondente conjunto de clausulas também não o é . Esse resultado permite substituir Lógica de Primeira Ordem por Lógica Clausal nas provas por refutação, como é feito em PROLOG.

Estágio 4 - Deslocamento do quantificador universal para fora.

Nesta parte moveremos o quantificador universal para a parte externa da expressão, sem que isto afete o significado. Por exemplo:

all(X, homem(X) -> all(Y, mulher(Y) -> gosta(X,Y)))

se transforma em:

all(X, all(Y, homem(X) -> (mulher(Y) -> gosta(X,Y))))).

O deslocamento do quantificador universal para a parte externa da expressão constitui-se também uma tentativa de simplificação, pois assumindo que toda variável é quantificada universalmente (já que os quantificadores existenciais foram suprimidos), o quantificador em si não acrescenta nenhuma informação extra. Por exemplo:

all(X, vivo(X) # morto(X)) &
all(Y, gosta(mary, Y) # impuro(Y)).

É o mesmo que :

(vivo(X) # morto(X)) & (gosta(mary,Y) # impuro(Y)).

Ambas as expressões dizem que qualquer que seja X ou ele está vivo ou ele está morto, e qualquer que seja Y ou mary gosta de Y ou Y é impuro.

Estágio 5 - Distributiva de & sobre #.

Na fórmula que temos agora não deve haver quantificadores explícitos e nem outros conectivos além de & e # (exceto ~). Tendo isto, devemos passar a expressão para a forma conjuntiva, onde não aparecem conjunções internas às disjunções. Para tanto nos utilizamos das seguintes equivalências:

(a & b) # c equivale a (a # b) & (b # c)

a # (b & c) equivale a (a # b) & (a # c)

Estágio 6 - Agrupar cláusulas.

Neste estágio devemos ter apenas literais, ou seja as cláusulas atômicas, ou composições de literais ligados por conjunções e disjunções, as quais devemos agrupá-las de modo a simplifica-las ao máximo.

1o. uma composição de & :

(a & b) & (c & (d & e)) é equivalente a (a&b&c&d&e).

o. uma composição de # :

(a # b) # (c # (d # e)) é equivalente a (a#b#c#d#e).

É importante lembrar que em ambos os casos a ordem dos literais não importa. Portanto, temos que a expressão pode ser representada por uma lista, com conjunção implícita de listas unidas por disjunções.

Exemplo:

Vejamos a aplicação dos estágios listados sobre a expressão
all(X,all(Y,pessoa(Y) -> respeita(Y,X)) -> rei(X)).

Estágio 1 - remover implicações.
all(X,~(all(Y,~pessoa(Y) # respeita(Y,X))) # rei(X)).

Estágio 2 - mover negação para os termos atômicos.
all(X,exists(Y,pessoa(Y) & ~respeita(Y,X)) # rei(X)).

Estágio 3 - skolemização.
all(X,(pessoa(f1(X)) & ~respeita(f1(X),X)) # rei(X)).

Estágio 4 - eliminação dos quantificadores universais.
(pessoa(f1(X)) & (~respeita(f1(X),X)) # rei(X)).

Estágio 5 - distributiva de & sobre #.
(pessoa(f1(X))#(rei(X))&(~respeita(f1(X),X)#rei(X)).

Estágio 6 - agrupar cláusulas.
Temos portanto duas listas, cada qual com dois literais, o primeiro com:
pessoa(f1(X))#rei(X),
a segunda com :
~respeita(f1(X),X)#rei(X).
E interligando ambas nós temos uma conjunção.

1.3 Forma Clausal.

Agora que definimos a Forma Clausal é preciso ter uma notação para representa-la. Parece ser mais natural escrever uma cláusula depois da outra, lembrando sempre que a ordem não importa, e sabendo também que cada cláusula é composta por literais ou suas negações. A forma selecionada agrupa primeiro os literais não negados separadas por ";" e depois os negados separados por "," e entre eles, dividindo-os, ":-". Assim sendo, teríamos:

(a # b # ...) # (~k # ~l # ...)

que equivale a:

(a # b # ...) # ~(k # l # ...)

que equivale a:

(k & l & ...) -> (a # b # ...).

Assim sendo, se escrevermos & como "," e # como ";" teremos a cláusula como sendo :

a;b;... :- k,l,... .

Exemplo:

aplicando o que acabamos de definir sobre :

(pessoa(adam)&pessoa(eva))&((pessoa(X)#~mae(X,Y))#~pessoa(Y))

temos:

pessoa(adam).

pessoa(eva).

pessoa(X):-mae(Y,X),pessoa(Y).

Ao adotarmos esta forma, temos uma representação que é bastante próxima ao PROLOG. A diferença está no fato de termos diversos literais não negados agrupados do lado esquerdo, o que não é permitido em PROLOG, uma vez que para permitir a utilização do mecanismo de prova por Resolução, PROLOG se restringe a um subconjunto das Formas Clausais, as Cláusulas de Horn.

1.4 Cláusulas de Horn.

As Cláusulas de Horn são cláusulas que tem como característica a presença no lado esquerdo de apenas um ou nenhum literal positivo.

As Cláusulas de Horn são essenciais para a prova automática de teoremas, utilizando-se o método da Resolução (discutido no próximo item). Vejamos porque. Existem duas subclasses de cláusulas de Horn :

Com um literal negativo (ditos "com cabeça");
Sem literais negativos (ditos "sem cabeça").

Em nossa base de dados PROLOG, admitiremos somente as cláusulas com cabeça, exceto por uma. A cláusula que desejamos provar é a única cláusula sem cabeça admitida. É fácil, então, entender o porquê das Cláusulas de Horn: basta vermos que se combinarmos cláusulas com cabeça com a cláusula sem cabeça, são teremos uma cláusula vazia como resultado final, se a cláusula sem cabeça representar uma sentença correta que foi negada.

Por exemplo:

:- animal(onça).
:- vive(onça).

como resultado teremos :
:- animal(onça),vive(onça).

que é justamente o desejado para que possamos provar uma cláusula do tipo " :- animal(X),vive(X)", onde X é instanciado como sendo "onça".

1.5 Resolução e prova de teoremas.

Uma das principais aplicações de PROLOG é na prova de teoremas. Isto é conseguido explorando-se a relação entre PROLOG e a lógica, uma vez que PROLOG usa a Lógica de Primeira Ordem para atribuir um significado preciso a suas cláusulas.

O resultado que nos permite utilizar a relação do PROLOG com o Cálculo de Predicados foi descoberto por J. Alan Robinson. O resultado nos diz que o sistema de resolução tem apenas uma regra de inferência, chamada de Regra de Resolução e que gera uma nova cláusula a partir de duas outras.

A Resolução é designada para trabalhar com sentenças na forma clausal. Dado um conjunto S de cláusulas e uma cláusula C, uma dedução de C a partir de S neste sistema formal consiste de uma seqüência de cláusulas terminando em C e gerada a parti da aplicação repetida da regra de resolução. Uma refutação a partir de S é uma dedução da cláusula vazia a partir de S. A regra de Resolução é definida de tal forma que S não é satisfeito se e somente se existe uma refutação a partir de S [CASANOVA-86].

Por exemplo, temos:

triste(cris);bravo(cris):-dia-de-serviço(hoje),chuvoso(hoje)

antipatico(cris):- bravo(cris),cansado(cris)
segue-se portanto:

triste(cris);antipatico(cris):- dia-de-serviço(hoje),
chuvoso(hoje),
cansado(cris).

Na primeira cláusula dizemos que se hoje é dia de serviço e está chuvoso, então cris está triste ou bravo. Na segunda cláusula dizemos que se cris está bravo e cansado, então cris está antipático. A partir de ambos concluímos que se hoje é dia de serviço, está chuvoso e cris está cansado, então cris está triste ou está antipático.

Usando o procedimento acima eliminamos uma redundância, mas as coisas não são tão simples quando temos variáveis envolvidas, uma vez que as fórmulas atômicas não são necessariamente idênticas. Assim sendo, é necessário saber combina-las de modo a remover a redundância. A operação envolve instanciação de variáveis para se encontrar a combinação de cláusulas. Em geral, em cláusulas PROLOG, o que se faz primeiro é a combinação, seguida da operação de instanciação e remoção de duplicatas. Por exemplo:

pessoa(f1(X)); rei(X):-. (1)

rei(Y):- respeita(f1(Y),Y). (2)

respeita(Z,artur):- pessoa(Z). (3)

O que temos aqui é que as duas primeiras cláusulas propõem que se todas as pessoas respeitam uma pessoa então esta pessoa é rei. A terceira cláusula propõe que todas as pessoas respeitam Artur. Feita a combinação de (2) e (3) temos:

rei(artur):- pessoa(f1(artur)). (4)

Obtivemos isto a partir da instanciação de Y em (2) com Artur em (3), e de Z em (3) com f1(Y) em (2). Agora combinando (4) com (1) temos :

rei(artur);rei(artur):-.

Esta cláusula diz o mesmo que as duas primeiras : Artur é rei.

Este processo é chamado de "unificação".

Infelizmente o processo não é aplicável diretamente pois não haveria como

garantir que se chegaria é cláusula que se quer provar.

Para poder-se usar a Resolução na prova de teoremas, utilizamos então a propriedade da Refutação Completa, que garante que se as cláusulas forem inconsistentes então a Resolução levará à cláusula vazia.

Assim sendo, se tomarmos um conjunto inicial consistente de proposições, a Resolução levará a uma cláusula vazia quando a nova cláusula a qual se quer testar é inconsistente, ou seja, ela gera uma contradição com relação ao conjunto inicial.

Utilizando o fato acima concluímos que podemos provar uma Proposição usando o artifício de incluir no conjunto sobre o qual vamos inferir, uma cláusula que é a negação daquela que desejamos provar. Se obtivermos como resultado da resolução a cláusula o7 3 vazia então está provado que a cláusula inicial é válida, caso contrário, haverá uma inconsistência na nossa base de cláusulas.

Vejamos o exemplo anterior :

Introduzimos na base a cláusula " :- rei(artur) ", que corresponde à negação de " rei(artur) ", teremos então :

rei(artur), rei(artur):-. (do exemplo anterior)

:- rei(artur). (nossa cláusula objetivo)

Aplicando resolução obteremos como resultado,

:-.

o que já era esperado.

A estratégia que combina Resolução + Refutação é conhecida como "Prova por Refutação".

1.6 PROLOG.

Como vimos, PROLOG utiliza-se do Cálculo de Predicados, da Resolução e da representação na forma de cláusulas de Horn para a resolução e prova de teoremas. A cláusula objetivo é dada na forma :

?- a1, a2, a3,... .

que será transformada em

:- a1, a2, a3,... .

Após isso, aplica-se o método de resolução sobre as cláusulas de Horn. Esse método usa a Resolução por Entrada Linear, ou seja, toma-se a cláusula objetivo e combina-se com a primeira cláusula da base; toma-se a cláusula resultante e combina-se com a segunda da base, e assim por diante até termos a resposta positiva ou esgotarmos a base. É por isso que a ordem das cláusulas em um programa PROLOG é importante.

É relevante lembrar que quando temos na base regras que necessitam instanciar variáveis para terem comprovadas as suas validades, ou seja, regras compostas por referências a outras, PROLOG irá executar a prova também dessas, com a instanciação proposta para as variáveis na cláusula objetivo. Este fato é explorado para a construção de programas em PROLOG.

PROLOG também provê funções que facilitam a construção de programas, entre elas é importante citar o "cut" (!), asserta, retract, gensym e findall. Estas funções e outras mais podem ser facilmente encontradas em qualquer livro sobre a linguagem e o conhecimento sobre o seu funcionamento é, no mínimo, recomendável a quem quiser programar em PROLOG.

Como vimos nestes tópicos, PROLOG é baseada na idéia de prova de teoremas. Como consequência, temos uma linguagem bastante diferente das mais tradicionais como FORTRAN ou PASCAL. Em PROLOG, ao contrário daquelas, a preocupação ao programar é em como definir o problema, enquanto que FORTRAN e outras se preocupam em como resolvê-lo. É esta basicamente a grande diferença do PROLOG e a grande motivação para a sua aplicação.

• Capítulo 2 - Gramática de Cláusulas Definidas(DCG).

2.1 Representação de GLC's.

Sentenças em qualquer idioma são muito mais que simples sequências arbitrárias de palavras, e as respostas a questões tais como: (a) Como construir uma sentença em um determinado idioma? (b) Como reconhecer se uma sentença pertence ao idioma ? Devemos levar em conta a gramática da linguagem.

Gramática é o conjunto de regras que especificam como são as sentenças de uma determinada linguagem. Se tivermos em mãos a gramática de uma linguagem, seremos capazes de construir e reconhecer sentenças naquela linguagem.

Iniciaremos o estudo com as gramáticas livres de contexto, e além disso, não nos preocuparemos com o significado das sentenças; procuraremos verificar apenas a corretude gramatical.

Baseado nisto, vejamos um exemplo de gramática muito simples:

sentença --> s_nominal, s_verbal.

s_nominal --> artigo, substantivo.

s_verbal --> verbo, s_nominal.

artigo --> [o].

artigo --> [a].

substantivo --> [homem].

substantivo --> [maçã].

verbo --> [come].

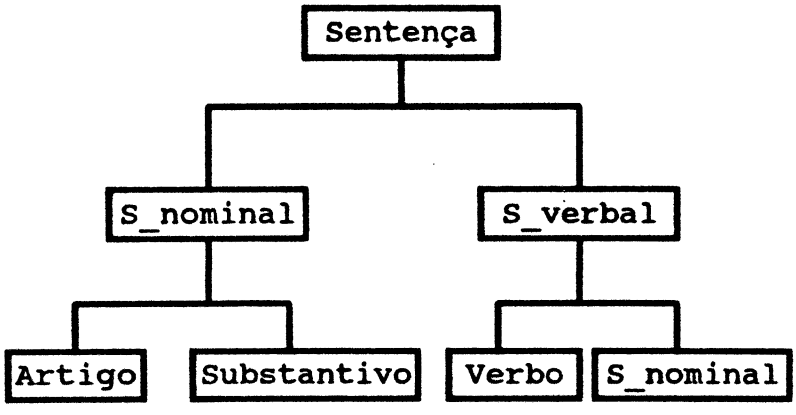
Em nossa gramática definimos que uma sentença é formada por duas partes, o sintagma nominal seguido de um sintagma verbal. Mas o que são um sintagma nominal e um sintagma verbal?

Os sintagmas verbal e nominal são categorias de nossa gramática, sendo que o sintagma nominal é formado por um artigo seguido de um substantivo e o sintagma verbal é formado por um verbo seguido de um outro sintagma nominal.

E quanto aos artigos, substantivos e verbos? são palavras pertencentes a classes de nossa gramática cuja composição definimos. Concluindo isso, temos como montar e reconhecer as sentenças, verificando se a organização das palavras obedece a ordem indicada.

Note que a nossa gramática reconhece tanto a sentença "o homem come a maçã " quanto " a maçã come o homem ".

É mais fácil visualizar a sentença através de sua representação na forma de árvore, a chamada árvore de análise. Vejamos o exemplo com a nossa gramática:



Definido o que é uma gramática, devemos nos preocupar em como representar o problema da análise gramatical em PROLOG. Como iremos ler uma sentença e dizer se ela obedece a nossa gramática ou não ?

Ignorando a forma superficial, podemos supor a entrada da sentença na forma de uma lista, e portanto teríamos na entrada algo na seguinte forma :

?- sentença([o,homem,come,a,maçã]).

Organizando a nossa gramática de forma a resolver esta consulta com um programa PROLOG, chegaríamos nas seguintes regras:

sentença(X):- append(Y,Z,X),s_nominal(Y),s_verbal(Z).

s_nominal(X):- append(Y,Z,X),artigo(Y),substantivo(Z).

s_verbal(X):- append(Y,Z,X),verbo(Y),s_nominal(Z).

artigo([o]).

artigo([a]).

substantivo([homem]).

substantivo([maçã]).

verbo([come]).

Note que a cláusula "append" divide a lista X de entrada em duas outras, Y e Z. Com isso nosso programa está completo e é capaz de reconhecer as sentenças válidas em nossa gramática. Mas note que, embora correto, este programa é ineficiente pois depende diretamente das instanciações feitas por "append". Por exemplo, para a nossa gramática de entrada, são possíveis as seguintes instanciações :

Y=[], Z=[o,homem,come,a,maçã]
Y=[o], Z=[homem,come,a,maçã]
Y=[o,homem], Z=[come,a,maçã]
Y=[o,homem,come], Z=[a,maçã]
Y=[o,homem,come,a], Z=[maçã]
Y=[o,homem,come,a,maçã], Z=[]

Observe que as definições dos sintagmas, analogamente, também estão sujeitas aos mesmos problemas de instanciação.

Percebe-se, portanto, o ponto em que se encontra a deficiência. Logo é necessário encontrar um meio mais eficiente de dividir a lista de entrada. A solução sugerida encontra-se abaixo:

sentença(X):- s_nominal(X,Y), s_verbal(Y,Z).

s_nominal(X,Y):- artigo(X,Z), substantivo(Z,Y).

s_verbal(X,Y):- verbo(X,Z), s_nominal(Z,Y).

artigo([o|X],X).

artigo([a|X],X).

substantivo([homem|X],X).

substantivo([maçã |X],X).

verbo([come|X],X).

A solução sugerida promove uma divisão muito mais eficiente da lista, ao mesmo tempo que testa simultaneamente a sentença. Note que se um determinado elemento na seqüência não se encaixa em nossa definição, a sentença é rejeitada de imediato enquanto que anteriormente teríamos que esperar até a última instanciação.

2.2 Notação DCG - Definite Clause Grammar.

Visando facilitar a descrição de regras gramaticais, foi desenvolvida uma notação especial para o PROLOG baseada na conhecida notação BNF: a notação DCG (Gramática de Clausulas Definidas.

A notação DCG é mais concisa que a representação em PROLOG pois dispensa a apresentação de vários argumentos. Por exemplo, em DCG as regras definidas anteriormente têm a seguinte forma:

sentença --> s_nominal, s_verbal.

s_nominal --> artigo, substantivo.

s_verbal --> verbo, s_nominal.

artigo --> [o].

artigo-> [a].

substantivo --> [homem].

substantivo --> [maçã].

verbo --> [come].

Note que o funtor '-->' tem o papel de dizer que o termo à esquerda é composto pelos termos à direita. Por exemplo, 'sentença' é composta de 's_nominal' e 's_verbal'.

A notação DCG é traduzida automaticamente para PROLOG para ser executada. Nesta passagem o tradutor se encarrega de colocar parâmetros que tornam a DCG idêntica à representação PROLOG. Quando à direita do funtor temos não-terminais, são acrescentados dois parâmetros a cada um dos termos. Por exemplo, a regra :

sentença --> s_nominal, s_verbal.
É traduzida como :

sentença(S0,S):- s_nominal(S0,S1), s_verbal(S1,S).

Quando à direita temos um termo terminal, são acrescentados dois parâmetros ao termo da esquerda, sendo o primeiro uma lista com o termo terminal na cabeça da lista e o segundo parâmetro sendo a cauda da lista do primeiro parâmetro. Por exemplo : o7 3

artigo --> [o].

É traduzido como :

artigo([o|S],S).

Percebe-se que as regras em DCG, ao serem traduzidas para PROLOG, são idênticas às representações originais, com a vantagem de que não temos que nos preocupar com a colocação de variáveis para a divisão da lista de entrada antes dos testes. Temos, portanto, que a representação adequada para descrever regras gramaticais em PROLOG é a DCG.

2.3 Adição de argumentos extras

Embora já tenhamos meios de representar gramáticas, percebe-se ainda que esta representação é bastante restrita. Note, por exemplo, como representar as sentenças:

" O homem com e a maçã ."

" Os homens comem a maçã ."

Se nos limitarmos ao que temos até agora, possivelmente o que faremos é duplicar as regras que temos para a forma no singular, de modo a representar também a forma no plural. Mudaríamos, portanto, a base para,

sentença --> sentença_singular.

sentença --> sentença_plural.

sentença_singular --> s_nominal_singular, s_verbal_singular.

sentença-plural --> s_nominal_plural, s_verbal_plural.

s_nominal_singular --> artigo_singular,substantivo_singular.

s_nominal_plural --> artigo_plural, substantivo_plural.

s_verbal_singular --> verbo_singular, s_nominal.

s_verbal_plural --> verbo_plural, s_nominal.

s_nominal --> s_nominal_singular.

s_nominal --> s_nominal_plural.

artigo_singular --> [o].

artigo_singular --> [a].

artigo_plural --> [os].

substantivo_singular --> [homem].

substantivo_singular --> [maçã].

substantivo_plural --> [homens].

verbo_singular --> [come].

verbo_plural --> [maçã].

Observe que embora a representação de fato funcione, ela é bastante volumosa e não muito elegante, principalmente se notarmos que as formas no singular e no plural têm estruturas idênticas, é exceção dos terminais. Foi pensando nisto que se resolveu adotar argumentos para se eliminar redundâncias em representações deste tipo. Vejamos neste caso o que ocorre ao adicionarmos um argumento para representar número. Teremos as seguintes regras :

sentença --> sentença(X).

sentença(X) --> s_nominal(X), s_verbal(X).

s_nominal(X) --> artigo(X), substantivo(X).

s_verbal(X) --> verbo(X), s_nominal(Y).

artigo(singular) --> [o].

artigo(singular) --> [a].

artigo(plural) --> [os].

substantivo(singular) --> [homem].

substantivo(singular) --> [maçã].

substantivo(plural) --> [homens].

verbo(singular) --> [come].

verbo(plural) --> [comem].

A adição de um novo argumento permite a melhor representação dos dados, como no caso exemplificado, com a pluralidade. Uma outra utilidade para os argumentos é construir a árvore de representação de análise. Para tanto, adicionamos um novo argumento que será instanciado e que nos retornará a representação da árvore de análise.

Exemplo:

As cláusulas abaixo formam a versão modificada para gerar a representação da árvore de derivação de nosso exemplo anterior.

sent(X,sentença(SN,SV)) --> s_nom(X,SN), s_verb(X,SV).

s_nom(X,s_nominal(A,S)) --> art(X,A), subst(X,S).

s_verb(X,s_verbal(V,SN)) --> verb(X,V), s_nom(Y,SN).

art(singular,artigo(o)) --> [o].

art(singular,artigo(a)) --> [a].

art(plural,artigo(os)) --> [os].

subst(singular,substantivo(homem)) --> [homem].

subst(singular,substantivo(maçã)) --> [macã].

subst(plural,substantivo(homens)) --> [homens].

verb(singular,verbo(come)) --> [come].

verb(plural,verbo(comem)) --> [comem].

Veja que os argumentos sentença, s_nominal, s_verbal, sofrem instanciação de modo a gerar uma árvore do tipo,

```
sentença(
  s_nominal(
        artigo(o),
      substantivo(homem)
      ),
  s_verbal(
      verbo(come),
      s_nominal(
                  artigo(a),
            substantivo(maçã )
          )
        )
    )
```

que é a representação do percurso feito para aceitar-se a cadeia "o homem comeu a maçã ". A árvore de análise é uma ferramenta muito útil para se descobrir como se comporta o analisador da linguagem que construímos.

Analogamente podemos estender a utilização de argumentos para representar outros dados que venham nos interessar, tais como conjugação de verbos, tempos, ... Estamos vendo assim a grande importância dos argumentos na representação de gramáticas.

2.4 Adição de regras extras.

Ainda enfocando a melhor representação das regras, principalmente as que têm estruturas semelhantes, vejamos agora a utilização de representação de dados feitas em PROLOG na descrição DCG. É possível compor regras PROLOG em representações DCG, através da adição dessas regras inclusas dentro de chaves "{_ }". Uma aplicação dessa adição de regras é a representação de estruturas atômicas de nossa gramática, por

exemplo substantivos. Observa-se que a estrutura é repetida uma vez para cada palavra que se insere nesta classe. Para modificar isto podemos alterar a regra para ficar do seguinte modo:

```
subst(X,substantivo(N)) --> [N], {nome(N,X)}.
```

onde "nome(N,X)" é uma regra PROLOG que remeteria a um banco de dados de palavras do tipo:

```
nome(homem,singular).

nome(maçã ,singular).
```

Deste modo evitamos repetir as estruturas que compõem a nossa gramática, e ao mesmo tempo facilitamos a representação de dados que poderá ser feita em cláusulas PROLOG.

Um outro exemplo de utilização que facilita a nossa representação de dados é a generalização da regra de construção de palavras no plural, que diz, que geralmente palavras no plural são formadas por palavras no singular acrescidas de "s" no final. Sabendo disto podemos modificar novamente nossa regra de modo a termos:

```
subst(plural,substantivo(N)) -->[N],
                    { nome(singular,K),
                      append(K,"s",N) }.
```

É lógico que esta regra não é válida para todas as palavras, e nós ainda precisaríamos definir que a regra só é válida para a representação de palavras "regulares", o que implicaria na utilização de mais um argumento em nossa base de dados. De qualquer modo, no entanto, obteríamos uma redução substancial no tamanho de nossa base de dados.

2.5 Formalização.

A melhor descrição formal da representação DCG pode ser dada via o próprio formalismo DCG, e ficaria do seguinte modo :

```
regra_gramatical --> regra_cabeça, [-->], regra_corpo.

regra_cabeça --> terminal.
```

regra_cabeça --> não_terminal, [,], terminal.

regra_corpo --> regra_corpo, [,], regra_corpo.

regra_corpo --> regra_corpo, [;], regra_corpo.

regra_corpo --> item_corpo.

item_corpo --> [!].

item_corpo --> [{], cláusula_prolog, [}].

item_corpo --> não_terminal.

item_corpo --> terminal.

Um "terminal" é uma palavra ou símbolo, que pode fazer parte da seqüência de entrada, deve estar sempre entre colchetes "[ ]".

Um "não-terminal" é uma regras ou função que leva a outros não-terminais ou a um terminal.

Temos assim definida a nossa representação.

• Capítulo 3 - Ambientes Cooperativos para Sistemas Baseados em

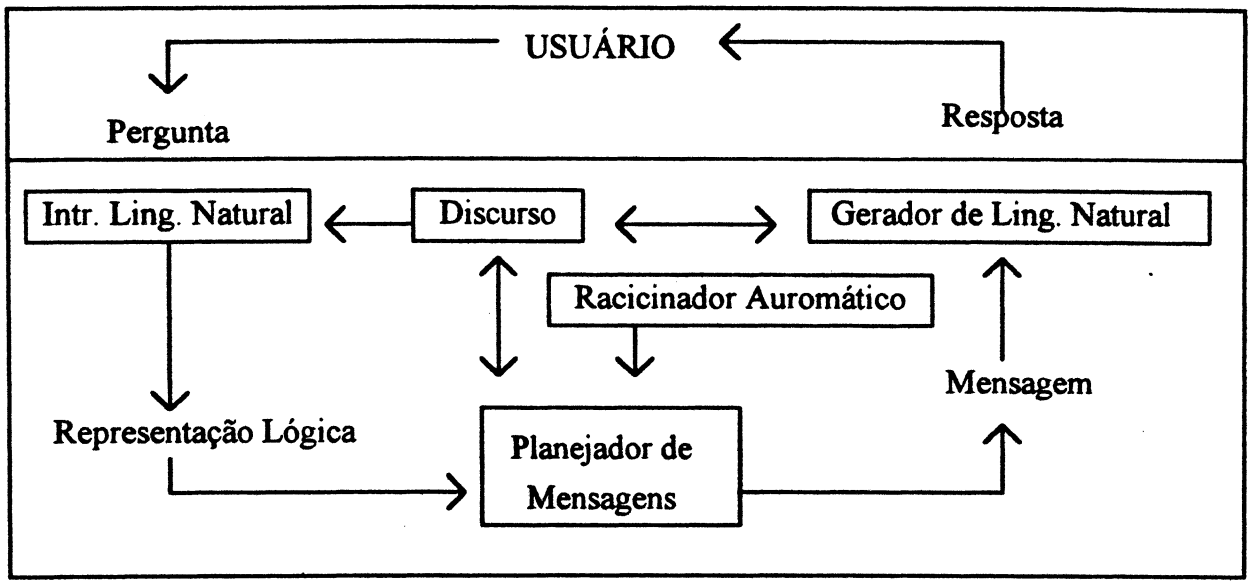
Conhecimento.

Em nosso trabalho enfocamos mais especificamente Sistemas que utilizam a lógica como forma de raciocínio. Discutiremos principalmente sobre sistemas que utilizam esta forma de raciocínio para processar Linguagem Natural, objetivando a geração de respostas cooperativas e a explicação do raciocínio.

Entenda-se por geração de respostas cooperativas a geração de respostas que, além de responder é pergunta explícita do usuário, também fornece informações relativas a questões implícitas na pergunta. Por exemplo, correção de suposições incorretas implícitas na pergunta, como ao perguntar: "com quem Maria é casada?", quando Maria não é casada. Outro exemplo é a apresentação da resposta correta quando ocorre a negação da pergunta original, como quando perguntamos: "Maria é casada com João?", se a resposta for negativa, tenta-se informar com quem Maria é casada.

A geração da explicação do raciocínio é uma característica essencial para que possamos classificar um sistema como inteligente. Consiste basicamente na capacidade de o sistema explicar como ele chegou é resposta dada à questão proposta. O sistema deve explicar o raciocínio usado e, sempre que for o caso, as suposições que levaram à resposta final.

Para cumprir os objetivos de geração de respostas cooperativas e explicação do raciocínio, utilizaremo-nos do ambiente proposto em [NUNES-89]. Este trabalho fornece-nos a descrição de como funcionaria um hipotético sistema com o ambiente cooperativo.



A geração de respostas cooperativas e de explicação de raciocínio ficaria por conta de um planejador de mensagens que, para tanto, utilizaria uma árvore de derivações geradas por um Raciocinador Automático de uso geral (Provador de Teoremas em Lógica de Primeira Ordem).

O Raciocinador Automático geraria a resposta (afirmativa caso tenha conseguido provar a proposição, negativa caso contrário) e árvore de derivações a partir da análise da pergunta traduzida em Linguagem Lógica de Primeira Ordem, sob a visão do foco da pergunta. A obtenção da representação e do foco ficaria por conta de um interpretador de linguagem natural que é justamente o objetivo de nossa pesquisa.

3.1 Um Sistema com Ambiente Cooperativo.

Não discutiremos aqui detalhadamente esse sistema proposto, mas é importante que se entenda a idéia dos vários mecanismos para que tenhamos uma visão da importância e dos objetivos da pesquisa.

A geração de respostas cooperativas, como já foi dito, utiliza-se do Raciocinador Automático para inferir informações que devem fazer parte da resposta. Para tanto, o sistema deve ser capaz de julgar e selecionar apenas informações consideradas relevantes, para que o usuário não obtenha informações impróprias, ou irrelevantes.

A apresentação das respostas deve ser feita de maneira natural, ou seja, sempre que possível em linguagem natural e levando em conta as características do sistema.

É da árvore de derivações que obtemos as informações necessárias para a geração da resposta cooperativa. Pois é analisando as informações da árvore através do foco que julgamos quais são as informações relevantes e quais não são. A partir daí as informações relevantes são encaixadas em um plano/esquema de texto de forma a gerar a resposta cooperativa, que, espera-se, seja um texto bastante explicativo.

3.2 Foco.

O Foco de uma pergunta é o centro principal das atenções sobre a pergunta, ou seja, é o ponto de maior interesse dentre os objetos envolvidos. O foco indica qual o elemento da pergunta sobre o qual o questionador tem mais dúvida. Por exemplo, nas

perguntas :

1) É João que foi assassinado?
2) Mário não é casado com Maria?
3) É Ana que matou João?

O foco da primeira pergunta é o paciente da ação "assassinar", que pode ter sido João. Na segunda, o foco é a relação que existe entre Mário e Maria. E na última, o foco é o agente que, no caso, pode ter sido Ana.

Os candidatos naturais a foco de uma pergunta são trás:

Agente;

ação ou Relacionamento;

Paciente.

Quando o paciente ou o agente é o foco da pergunta, dizemos que ele recai sobre um argumento da representação formal. No exemplo

"Maria é casada com João ?"

caso consideremos "Maria" ou "João" como o elemento mais importante da pergunta, o foco é dito ser o argumento na representação "casados(maria,joao)". Caso a tentativa de se provar esta clausula falhe, então tentaremos instanciar "casados(X,joao)" se João for o foco, ou "casados(maria,X)" caso Maria seja o foco. Isso significa procurar um outro valor para o foco negado. Como resposta, na segunda hipótese, poderíamos obter "Não, Maria é casada com José".

Quando o foco recai sobre a relação ou a ação, o foco na representação passa a ser "casados", ou seja, no nosso exemplo tentaríamos encontrar "X(maria,joao)", no qual X é um predicado alternativo a "casados". Como sabemos não ser possível realizar esta instanciação na LPO se faz necessário encontrar um outro modo de encontrar o valor alternativo.

A sugestão apresentada é de se tentar dividir os predicados em classes de equivalência semântica e exclusividade, dentro das quais se tentará encontrar o valor alternativo para X, por isso a necessidade da equivalência e da exclusividade.

A equivalência garante que a alternativa encontrada tem relação com a proposta inicial. Por exemplo: não responderemos "João e Maria viajaram", já que "viajar" e "casar" não são equivalentes semanticamente.

A exclusividade evita que respondamos algo que seja inconsistente, por exemplo:

João e Maria serem amigos não impede que eles sejam casados.

Embora esta solução pareça ser eficiente, este ainda é um problema a ser pesquisado.

Nesta altura já deve-se ter entendido a importância da determinação do foco, mas vale a pena ressaltar de novo. É através do foco que conseguimos estabelecer qual a informação principal a se corrigir quando ocorre a negação da proposição da pergunta.

3.3 Classes de Perguntas.

Objetivando o melhor estudo das perguntas para podermos trata-las, dividimos as perguntas em trás classes distintas:

Tipo Qu_;

Tipo Alternativas;

Tipo Sim/Não.

As perguntas do tipo Qu_ são facilmente reconhecidas; são geralmente marcadas pela presença dos pronomes interrogativos, como Quem, Qual, Quando, ... Essas perguntas têm o foco explícito, e pedem respostas objetivas, geralmente quantificadas existencialmente. O foco é dado pelo sujeito a saber, ou mais especificamente, pelo valor que pode ser instanciado como agente ou paciente. Por exemplo:

"Quem é o irmão de Ana ?"

Esta pergunta pede claramente o valor que pode ser instanciado como o irmão de Ana, logo o foco é "irmão de Ana".

As perguntas do tipo Alternativas são caracterizadas pela presença de diversas opções, dentre as quais espera-se esteja a(s) resposta(s) correta(s). Nesta classe de perguntas teremos não um, mas vários focos, um em cada alternativa. Por exemplo:

"É João ou José que ama Maria ?"

Note que aqui queremos saber quem ama Maria, portanto temos dois focos, João e

José.

As perguntas do tipo Sim/Não pedem uma resposta Sim ou Não. O foco da pergunta não tem uma posição ou número definido, sendo portanto de mais difícil determinação. Por exemplo:

"João ama Maria ?"

Perceba que aqui não temos claramente qual é o centro de dúvidas do usuário. Pode ser quem João ama, ou pode ser quem ama Maria, ou pode ser ainda o que João sente por Maria. Veja que portanto é muito mais complexo a determinação do foco.

Em nosso projeto preocuparemo-nos principalmente com as perguntas do tipo Sim/Não. Mais especificamente, tentaremos formular uma gramática que englobe várias construções gramaticais de perguntas do tipo Sim/Não e, através dela, traduzir a pergunta em LPO e determinar onde recai o foco.

Com tais informações é disposição, o módulo planejador de respostas cooperativas se encarregará de construir um plano de texto para ser realizado por uma gramática geradora de Linguagem Natural (no nosso caso a Língua Portuguesa do Brasil).

• Capítulo 4 - Estruturas de Frases para Perguntas do Tipo Sim/Não.

Passamos agora a discutir quais estruturas sintáticas terão as perguntas SIM/NÃO reconhecidas por nossa gramática.

As discussões neste capítulo têm como principais objetivos demonstrar como é possível delimitar o foco da pergunta, ou seja, o principal ponto de dúvida, e, além disso obter a fórmula lógica correspondente para o provador de teoremas (vide figura do capítulo 3), tudo isso a partir da estrutura da pergunta.

*O foco da pergunta será indicado sobre um dos componentes semânticos da sentença. Logo, são os elementos semânticos e não os sintáticos que deverão ser detectados durante o parsing. A opção se deve ao fato de ser desejável que na forma de representação final que escolhemos - **Lógica de Primeira Ordem** - não haver multiplicidade de representações em função das várias formas de representação de uma mesma informação. Por exemplo, ao falarmos:*

"João matou Luís" e "Luís foi morto por João"

em ambos os casos veiculamos a mesma informação, mas note que analisando os papéis sintáticos temos que no primeiro caso João é sujeito e, no segundo, o sujeito já é Luís. Logo não é interessante utilizarmos os papéis sintáticos para definir nossa representação final. Por outro lado, analisando os papéis semânticos, temos que eles não mudam em função da estrutura da sentença e mais ainda, podemos enfatizar ora um, ora outro, conforme mudamos a estrutura escolhida. No exemplo, João é o agente da ação em ambos os casos.

Fica claro, portanto, que, para indicar qual dos componentes semânticos constitui o foco, é preciso detectar, via gramática e dicionário com informações semânticas, qual componente da sentença corresponde a cada elemento semântico.

4.1 - Os Elementos Semânticos que Constituem a Pergunta.

A fim de facilitar a determinação do foco da pergunta, dividiremos esta em trás componentes semânticos principais :

Agente;

Paciente;

Ação ou Predicado.

*O **Agente** de uma sentença é o elemento da frase a quem se atribui uma ação ou característica. Por exemplo, na frase:*

"João viajou para Salvador ?"

o agente da frase é João, a quem se atribui a ação de viajar.

*O **Paciente** da sentença é o elemento da pergunta que sofre uma determinada ação, por exemplo em :*

"Carlos matou Maria ?"

O paciente da pergunta é Maria que sofre a ação de matar.

*A **Ação ou Predicado** de uma pergunta é a atividade que é executada pelo agente e/ou sofrida pelo paciente, ou a característica que é atribuída ao agente. É fácil de se perceber que estas funções são excludentes entre si numa mesma frase por desempenharem uma mesma função - **predicado da oração** - e por isso mesmo podem ser tratadas como uma mesma função semântica. Por exemplo, na frase:*

"Mário é casado com Ana ?"

o predicado da frase é "casado com Ana", pois constitui-se numa peculiaridade que é atribuída a Mário. já em :

"João casou com Ana ?"

"casar" é uma atividade desempenhada por João.

*É convenção entre os que trabalham com Lógica de Primeira Ordem fazer as seguintes correspondências: a **ação** ou o **predicado** da sentença se transformam no predicado lógico, enquanto que o **agente**, o **paciente**, o objeto da ação se transformam nos argumentos da fórmula lógica. Nos exemplos acima, teríamos: viajou(João,Salvador);*
casado(Mario,Ana);
matou(Carlos,Maria);
casou(João,Ana).

4.2 - Usando a Estrutura da Frase para Encontrar o Foco.

Tendo agora delimitado os componentes semânticos das frases, vamos tentar localizar o foco utilizando a estrutura da pergunta, ou seja, através da construção tentaremos descobrir qual é o ponto de maior dúvida quando de sua elaboração.

As estruturas mais comumente usadas para se construir perguntas do tipo Sim/Não são as seguintes :

<agente><ação><paciente> - "Mário odeia João ?"

<agente><verbo ser><predicado> - "Mário é bonito ?"

Nestas construções não temos claramente qual é o principal ponto de dúvida com relação à pergunta, ou seja, não temos como determinar com precisão qual é o foco da pergunta. Contudo, embora as estruturas das perguntas do tipo Sim/Não muitas vezes não nos permitem determinar com certeza qual é o foco da pergunta, existem várias construções que enfatizam claramente um dos elementos semânticos da pergunta e são estas construções que examinaremos mais atentamente.

4.2.1 - Foco no Agente.

Existem duas construções sintáticas que indicam claramente quando o foco da pergunta é o agente, são elas :

1) <verbo ser> <agente> <pronome relativo> <sentença c/ sujeito ante_posto>
Exemplos :

1.a) "Foi João quem viajou ?" F. Lógica - viajar(João)

1.b) "Foi o carro que bateu no muro ?" F. Lógica - bater(carro,muro)

1.c) "É José que odeia Maria ?" F. Lógica - odiar(José,Maria)

2) <agente> <verbo ser> <artigo definido> <predicado>

Exemplos:

2.a) "João é o culpado ?" F. Lógica - culpado(João)

2.b) "Maria é a esposa de João ?" F. Lógica - esposa(Maria,João)

Na primeira construção vemos que é a utilização do sujeito ante-posto que evidencia o agente, indicando que este é o principal centro de preocupação.

Note que, ao formular a pergunta, o usuário pensa que a informação dada pela sentença com o sujeito aposto é verdadeira (pressuposição) e é em cima desta informação que ele tem dúvida sobre quem executou ação ou possui a característica. Veja, por exemplo, na sentença 1.a, o usuário sabe que alguém viajou, mas não sabe quem e pergunta se foi João.

Já na segunda construção o artigo definido coloca em evidência o predicado, mas este destaque serve para indicar que o que desejamos saber na realidade é qual valor pode ser instanciado como tendo aquele predicado, pois queremos saber quem é "o culpado", "a esposa", "o ladrão", portanto, o agente. Novamente detectamos a pressuposição do usuário de que há alguém com aquela característica, e ele quer saber quem.

4.2.2 - Foco no Paciente.

Quando desejamos destacar o foco no paciente temos duas principais construções para tanto:

3) <verbo ser> <paciente> <pronome relativo> <sentença com objeto ante_posto>

Exemplos:

3.a) "É de João que Paulo é irmão ?" F. Lógica - irmão(João,Paulo)

3.b) "É de José que Maria gosta ?" F. Lógica - gostar(Maria,José)

3.c) "É José que Maria ama ?" F. Lógica - amar(Maria,José)

4) <verbo ser> <paciente> <pronome relativo> <sentença na passiva com sujeito ante_posto>

Exemplo:

4.a) "É José que é odiado por Maria ?" F. Lógica - odiar(Maria,José)

Note que, a rigor, o que coloca o paciente em evidência em ambos os casos é a anteposição da função desempenhada pelo paciente na frase. No primeiro caso, o objeto e no segundo o sujeito de uma oração na passiva. Observe ainda que, novamente a sentença com um componente aposto, neste caso o paciente, tem função de apresentar uma pressuposição feita pelo usuário. Aqui o usuário supõe que alguém sofre uma ação e deseja saber quem é essa pessoa. Por exemplo em 4.a, ele sabe que Maria odeia alguém e deseja saber quem.

4.2.3 - Foco na Ação ou no Predicado.

Tratamos juntas as construções que evidenciam o foco na Ação ou no Predicado por estas serem excludentes ou seja, não existe uma construção em que o foco pode estar no Predicado ou na Ação pois uma sentença simples pode representar apenas um dos dois. Além disso algumas construções que evidenciam o foco em um deles tem uma construção semelhante usada para evidenciar o foco em outro. Vejamos agora algumas formas de evidenciar o foco na Ação ou no Predicado.

5) <agente> <verbo negado> <paciente>

Exemplo:

5.a) "O gato não gosta do rato ?" F. Lógica - gostar(gato,rato)

6) <agente> <verbo ser negado> <predicado>

Exemplo:

6.a) "João não é casado com Ana ?" F. Lógica - casado(João,Ana)

7) <paciente> <verbo na passiva negado> <agente>

Exemplo:

7.a) "Maria não foi atropelada por João ?" F. Lógica - atropelar(João,Maria)

8) <agente> <verbo ser> <artigo indefinido> <predicado>

Exemplo:

8.a) "Marcos é um aluno ?" F. Lógica - aluno(Marcos)

Observe que nos trás primeiros casos a negação enfatiza a importância de uma ação ou de um predicado servindo portanto para destacar estes como foco. Percebe-se claramente que, a priori, o usuário pressupunha haver uma relação entre o agente e o paciente, mas agora há uma dúvida sobre esta relação e, portanto, ao perguntar, seu objetivo é a relação entre ambos.

No quarto caso, é o artigo indefinido que coloca o predicado em evidência e, ao contrário do que acontece quando o artigo é definido, não remete a importância ao agente pois neste caso fica claro que o queremos saber é se o agente possui uma característica e não qual é o agente que tem esta característica.

4.2.4 - Focos Determinados Heuristicamente.

Assumimos ainda duas construções como formas de se enfatizar o foco na ação e no predicado, embora não exista uma regra formal que motive isto. Elas seriam usadas quando não houvesse indicação forte como as mostradas anteriormente. Essas construções são:

9) <agente> <verbo ser> <predicado>

Exemplo:

9.a) "Maria é esposa de João ?" F. Lógica - esposa(Maria,João)

10) <paciente> <locução verbal> <part.apass> <agente>

Exemplo:

10.a) "A bola foi jogada por João ?" F. Lógica - jogar(João, bola)

11) <agente> <verbo> [<paciente>]

11.a) "João ama Maria ?" F. Lógica - amar(João,Maria)

11.b) "Pedro gosta de pêssego ?" F. Lógica - gostar(Pedro,pêssego)

11.c) "Maria viajou ?" F. Lógica - viajar(Maria)

Observe que nas trás construções não temos claro qual é o componente que constitui o foco, no entanto podemos assumir que o ambiente em que transcorre o diálogo entre o homem e o computador é cooperativo, e portanto, há também por parte da pessoa que está utilizando o sistema interesse em usar construções que facilitem a interpretação do foco.

Tendo em vista este fato, heuristicamente tomamos as construções como enfatizadoras do foco no predicado ou na ação devido a facilidade que o usuário perguntador teria de modificar a construção da sentença de modo a explicitar que o foco estaria em outro componente da sentença. Por exemplo, em 10.a se o foco fosse o paciente poderíamos dizer : "Foi a bola que foi jogada por João ?" (construção 4). Ou ainda caso o foco recaísse sobre o agente poderíamos dizer : "Foi João que jogou a bola ?" (construção 1).

Note que embora tenhamos também modos de enfatizar claramente o foco na ação ou no predicado estas construções muitas vezes não são naturais, e portanto de dificil utilização. Por exemplo, ao iniciar um dialogo com uma construção na forma:

"A bola não foi jogada por João ?"

parece-nos que o usuário pressupõe que já havia um dialogo anterior referindo-se à bola. No entanto, ambas as sentenças estão gramaticamente correta, e, pela construção 7 de nosso levantamento, ela realmente enfatiza o foco como sendo a relação.

4.3 - Conclusão.

Verificou-se que existem diversas maneiras de se construir uma pergunta do tipo Sim/Não e também diversas maneiras de se enfatizar o foco em um ou outro elemento semântico da frase. Neste trabalho, tratamos apenas um conjunto das construções possíveis, que poderá vir a ser ampliado no futuro.

Diante desta variedade de formas de se enfatizar o foco, e sabendo que algumas delas são baseadas em conhecimento heurístico, fica claro que, quando for construido o programa que tratará o problema de delimitar o foco, deve-se organizar a busca através das regras de modo que tente-se encaixar a pergunta primeiro nas formas mais utilizadas de construção com regras bem definidas e por último nas formas de construção que utilizam regras heurísticas.

Capítulo 5 - Tratando Algumas Construções Complexas.

Neste capítulo veremos como compor as construções estudadas no capítulo 4 de modo a construir sentenças mais complexas. O conjunto analisado anteriormente, embora consiga dar representação a um número bastante expressivo das perguntas que pode ocorrer ao usuário, muitas vezes força um trabalho excessivo para se obter a informação desejada, ou até mesmo é impossível obter essa informação. E então que temos a motivação para o estudo de construções mais complexas.

Não tentaremos, nem e intenção, cobrir todas as construções complexas, mas e nosso desejo deixar um conjunto bastante significativo delas pronto, de modo a facilitar um estudo a posteriori. Tendo isto em mente trataremos 5 tipos de construções :

Composição com quantificador universal;

Composição de sentenças com um termo em comum;

Composição de sentenças com um termo em comum em aposto;

Composição com relação de implicação;

Sentenças com um termo composto.

5.1 - Tratando Quantificador Universal.

O tratamento de quantificador universal foi feito em cima da detecção da

existência de um elemento da frase que representa toda uma classe ou categoria de indivíduos. Por exemplo, em:

" O homem pensa?"

o termo homem representa não uma pessoa mas a espécie humana.

O que tem que se fazer é modificar as estruturas de que já dispomos de modo a tratar os casos em que temos um termo representando uma categoria de indivíduos. Assim sendo, as novas construções são:

1) <verbo ser> <categoria> <pron.rel.> <sentença c/ agente ante_posto>

Ex. São os homens que amam Maria?
São os homens que são feios?
São os homens que são irmãos de José?

2) <artigo definido> <categoria> <verbo ser> <artigo definido> <predicado>

Ex. Os homens são os feios?
Os homens são os irmãos de José?

3) <verbo ser> <categoria> <pron.rel.> <sentença c/ agente na passiva ante_posto>

Ex. São os homens que são amados por Maria?

4) <verbo ser> <preposição> <categoria> <pron.rel.> <sentença c/ paciente ante_posto preposicionada>

Ex. É de carros que Maria gosta?

5) <verbo ser> <categoria> <pron.rel> <sentença com o paciente ante_posto>

Ex. São os homens que Maria ama?

6) <categoria> <verbo negado> [<paciente>]

Ex. Os homens não gostam de Maria?
Os homens não amam Maria?
Os homens viajaram?

7) <categoria> <verbo ser negado> <predicado>

Ex. Os homens não são feios?

Os homens não são irmãos de Maria?

8) <categoria> <locução verbal negada> <pron.apass> <agente>

Ex. Os homens não são amados por Maria?

9) <categoria> <verbo ser> <predicado>

Ex. Os homens são feios?
Os homens são irmãos de Maria?

10) <categoria> <locução verbal> <pron.apass.> <agente>

Ex. Os homens são amados por Maria?

11) <categoria> <verbo> [<paciente>]

Ex. Os homens amam Maria?
Os homens gostam de Maria?
Os homens viajaram?

12) <verbo ser> <agente> <pron.rel.> <sentença c/ agente ante_posto>

Ex. É Maria que ama os homens?
É Maria que gosta dos homens?

13) <agente> <verbo ser> <artigo definido> <predicado>

Ex. Maria é irmã dos homens?

14) <verbo ser> <paciente> <pron.rel> <sentença c/ paciente ante_posto na passiva>

Ex. É Maria que é amada pelos homens?

15) <verbo ser> <preposição> <paciente> <pron.rel.> <sentença c/ paciente ante_posto preposicionado>

Ex. É de Maria que os homens gostam?

16) <verbo ser> <paciente> <pron.rel.> <sentença c/ paciente ante_posto>

Ex. É Maria que os homens amam?

17) <agente> <verbo negado> <categoria>

Ex. Maria não ama os homens?
Maria não gosta dos homens?

18) <agente> <verbo ser negado> <predicado>

Ex. Maria não é irmã dos homens?

19) <paciente> <locução verbal negada> <pron.apass> <categoria>

Ex. Maria não é amada pelos homens?

20) <agente> <verbo ser> <predicado>

Ex. Maria é irmã dos homens?

21) <paciente> <locução verbal> <pron.apass.> <categoria>

Ex. Maria é amada pelos homens?

22) <agente> <verbo> <categoria>

Ex. Maria ama os homens?
Maria gosta dos homens?

Note que há um aproveitamento total das estruturas usadas no capítulo anterior. Isso se deve ao fato de que todas as pressuposições que haviam sido feitas para as construções simples continuam valendo para suas derivadas desta categoria, com a ressalva que agora se aplicam a toda uma categoria e não só a um indivíduo.

Ao gerarmos a fórmula lógica teremos uma relação de implicação, e isso se deve à existência do quantificador universal que passa a ser o termo antecendente de uma relação de implicação enquanto que a fórmula original que seria gerada, caso não houvesse o quantificador se torna o conseqüente da relação de implicação*.*

Sabendo disto é que podemos também determinar o foco. Por ser na realidade uma construção da fórmula de implicação o foco na fórmula lógica deve ser destacado agora sobre um dos termos da relação de implicação, ou seja, nas sentenças em que o termo representante da categoria esta em destaque, o foco passa a ser o termo antecedente da relação de implicação, caso contrário o foco é o termo conseqüente. Sendo que em algumas sentenças é possível ainda destacar o termo não quantificado universalmente como o foco da sentença, como nas construções 12 a 16 descritas acima.

** Observação: para efeito de esclarecimento, numa relação de implicação* ***X -> Y***

X é dito antecedente e Y é dito conseqüente.

5.2 Tratamento de Composição de Sentenças com um Termo em Comum.

O tratamento da composição de sentenças ligadas por um termo comum é feito via abstração dos dados do termo em comum e, em seguida, tratamento de cada sentença que forma a composição levando-se em conta os dados do termo em comum. É só quando do tratamento de cada sentença é que temos como saber a função desempenhada pelo termo em comum. Assim sendo, o que se faz é dividir a pergunta inicial em várias e tratar cada uma delas de modo independente realizando o reagrupamento de seus vários componentes na geração da fórmula lógica.

Tomemos uma construção para ilustrar a nossa idéia :

1) <termo> <lista sentenças>

Ex. João é feio , ama Maria e não gosta de José?
João é amado por Maria , ama Ana e não viajou?

Note que a sentença poderia ser desmembrada do seguinte modo:

<verbo ser> <termo em comum> <sentenças c/ um termo em comum> &
<verbo ser> <termo em comum> <sentença c/ um termo em comum> &
. . . &
<verbo ser> <termo em comum> <sentença c/ um termo em comum>.

O resultado desse desmembramento é equivalente à pergunta inicial e portanto pode substituir a entrada tornando mais fácil nossa tarefa de gerar nossa fórmula de saída.

A determinação do foco em cada componente é feita obedecendo as idéias formuladas no capitulo anterior, ou seja, o foco é indicado no predicado de cada uma das fórmulas.

5.3 - Tratando Composição de Sentenças com um Termo em Comum Ante-posto.

O tratamento de composição de sentenças de um termo em comum em ante-posto é idêntico ao tratamento de sentenças com um termo em comum. A separação é feita coma finalidade de facilitar a determinação do foco. Vejamos então, tomemos a construção:

1) <verbo ser> <termo em comum> <lista de sentenças>

Ex. É José que ama Maria , gosta de João, é feio e é amado por Ana?
É José que não ama Maria , é irmão de João e viajou?

A sentença poderia ser desmembrada do seguinte modo:

<verbo ser> <termo em comum> <sentença c/ um termo em ante_posto> &
<verbo ser> <termo em comum> <sentença c/ um termo em ante_posto> &
. . . &
<verbo ser> <termo em comum> <sentença c/ um termo em ante_posto>.

O resultado desse desmembramento é equivalente a entrada, assim como no caso anterior, e também como no caso anterior precisamos determinar o foco parte por parte da sentença. É a partir desta idéia que se pode ver que o foco da pergunta é o termo em ante-posto e, portanto, caso resolvamos analisar cada parte de modo independente, devemos determinar para cada uma delas o foco como sendo a função desempenhada pelo termo em aposição.

É fácil ver que a elaboração de construções que tratam de composições de disjunção sofrem um processo análogo.

5.4 Tratamento de Relações de Implicação.

O tratamento de relações de implicação feita para o caso mais simples, em que o foco recai no antecedente, foi feito via detecção da existência do advérbio de causa ligando duas sub-sentenças dentro da pergunta original. Por exemplo, na pergunta:

*"Maria é gorda **porque** come muito?"*

temos duas sentenças "Maria é gorda" e "Maria come muito" ligadas por um adverbio de causa, que promove a relação de implicação entre as duas sentenças, pois o que estamos dizendo é :

come muito(Maria) -> gorda(Maria)

Assim sendo temos duas construções representativas para este caso:

1) <termo> <sentença c/ elemento ante_posto> <adv. de causa>
<sentença c/ elemento ante_posto>

Ex. José gosta de Maria porque ama Maria?
José é amado por Maria porque é bonito?
José é irmão de Ana porque não é amado por Maria?

José é amado por Maria porque é amado por João?

2) <termo> <sentença c/ elemento ante_posto> <adv. de causa> <termo> <sentença c/ elemento ante_posto>

Ex. José gosta de Maria porque Maria ama José?
José é amado por Maria porque José é bonito?
José é feio porque João não é amado por Maria?
José é amado por Maria porque Ana é amada por João?

Ambas as construções acima diferem apenas no tratamento da elipse do termo na sentença antecedente, mas em ambos os casos tratamos uma única pressuposição, o usuário assume que o conseqüente é verdade e quer saber se isso se deve ao fato do antecedente também ser verdade. Assim sendo, o foco da sentença, como já dissemos acima, fica indicado como sendo no antecedente, isso se dá porque quando o foco recai sobre o antecedente, temos um pedido de confirmação da explicação, ou seja o usuário pergunta se caso o conseqüente é verdade é devido ao antecedente o ser, como é nosso caso.

5.5 Tratamento de Sentenças com um Termo Composto.

O tratamento de sentenças com um termo composto é bastante simples. O que se faz é basicamente estender as construções do capítulo 4 para que passem a aceitar construções nas quais o paciente ou o agente sejam na verdade uma lista de pacientes ou agentes. Assim sendo as construções passam a algo do seguinte modo:

1) <verbo ser> <composição de termo> <pron.rel.> <sentença c/ agente ante_posto>

Ex. São João , José e Maria que amam Ana?
São João , José e Maria que gostam de Ana?
São João , José e Maria que viajaram?
São João , José e Maria que são feios?
São João , José e Maria que são irmão de Ana?

2) <verbo ser> <composição de termo> <pron.rel> <sentença c/ agente ante_posto na passiva>

Ex. São João e José que são amados por Ana?

3) <verbo ser> <composição de termo> <pron.rel.> <sentença c/ paciente ante_posto>

Ex. São João e José que Maria ama?

4) <verbo ser> <preposição> <composição de termo> <pron.rel.>
<sentença c/ paciente ante_posto Preposicionado>

Ex. É de João e José que Maria gosta?

Observe que estas quatro construções são claramente extensões de construções do capítulo 4, analogamente poderia-se gerar extensões para cada umas das construções do capítulo anterior. O que se faz é basicamente gerar um novo termo para tratar paciente e agente que aceite não somente um, mas toda uma lista de agentes ou pacientes. A determinação do foco continua a obedecer as mesmas hipóteses levantada para cada construção no capítulo 4, o que difere agora é que teremos uma conjunção de fórmulas lógicas (uma fórmula para cada elemento da composição de termos) e o foco deverá ser indicado em cada uma das fórmulas da conjunção. Por exemplo, para a pergunta :

"João e José amam Maria?"

teremos a seguinte fórmula :

amar(João,Maria) ^ amar(José,Maria)

Em ambas as fórmulas devemos indicar que o foco está no predicado; esta informação foi obtida da observação das hipóteses levantadas no capítulo anterior.

5.6 - Conclusão.

Como se pode observar, e como dissemos no inicio do capítulo não cobrimos, e nem pretendíamos, todas as construções complexas, mas esperamos ter deixado um bom número de exemplos de exemplos de referência para quem pretenda se aventurar nesta tarefa.

Apêndice - A

Exemplos de Perguntas Tratadas pela Gramática.

Construções para o tratamento de quantificador universal

São os homens que amam Maria?
São os homens que são feios?
São os homens que são irmãos de José?
Os homens são os feios?
Os homens são os irmãos de José?
São os homens que são amados por Maria?
É de carros que Maria gosta?
São os homens que Maria ama?
Os homens não gostam de Maria?
Os homens não amam Maria?
Os homens viajaram?
Os homens não são feios?
Os homens não são irmãos de Maria?
Os homens não são amados por Maria?

Os homens são feios?
Os homens são irmãos de Maria?
Os homens são amados por Maria?
Os homens amam Maria?
Os homens gostam de Maria?
Os homens viajaram?
Maria é irmã dos homens?
É Maria que é amada pelos homens?
É de Maria que os homens gostam?
É Maria que os homens amam?
Maria não ama os homens?
Maria não gosta dos homens?
Maria não é irmã dos homens?
Maria não é amada pelos homens?
Maria é irmã dos homens?
Maria é amada pelos homens?
Maria ama os homens?
Maria gosta dos homens?

Construções para o tratamento de sentenças elementares

É João que ama Maria?
É João que gosta de Maria?
É João que viajou?
É João que é feio?
É João que é irmão de José?
João é o feio?
João é o irmão de José?
É João que Maria ama?
É de João que Maria gosta?
É João que é amado por Maria?
Maria não ama José?
Maria não gosta de José?
Maria não viajou?
Maria não é feia?
Maria não é irmã de José?
Maria não é amada por José?
Maria é uma feia?
Maria é uma irmã de José?
Maria é feia?
Maria é irmã de José?
Maria é amada por José?

José ama Maria?
José gosta de Maria?
José viajou?

Construção para o tratamento de composição de sentenças com um termo em comum em Ante-posto.

É José que ama Maria , gosta de João , e feio e é amado por Ana?
É José que não ama Maria , é irmão de João e viajou?

Construções para o tratamento de relações de implicação.

José gosta de Maria porque ama Maria?
José é amado por Maria porque é bonito?
José é irmão de Ana porque não é amado por Maria?
José é amado por Maria porque é amado por João?
José gosta de Maria porque Maria ama José?
José é amado por Maria porque José é bonito?
José é feio porque João não é amado por Maria?
José é amado por Maria porque Ana é amada por João?

Construção para o tratamento de composição de sentenças com um termo em comum

João é feio , ama Maria e não gosta de José?
João é amado por Maria , ama Ana e não viajou?

Construções para o tratamento de sentenças com um termo composto

São João , José e Maria que amam Ana?
São João , José e Maria que gostam de Ana?
São João , José e Maria que viajaram?
São João , José e Maria que são feios?
São João , José e Maria que são irmão de Ana?

São João e José que são amados por Ana?
São João e José que Maria ama?
É de João e José que Maria gosta?

Apêndice B

Conclusão.

A implementação das estruturas sugeridas neste estudo foi realizada em um programa (ver apêndice C) no qual fica comprovada a viabilidade das mesmas.

Durante a implementação foram tomadas algumas decisões que passamos a detalhar agora :

1 - Dividir diversas construções em duas ou mais cláusulas como no caso das construções que tratam variações de objetos. Esta medida foi adotada para fins de melhor entendimento e, posteriormente, facilidade de manutenção.

2 - Posicionar as cláusulas que tratam construções com quantificador universal antes das que tratam construções simples e somente depois as que tratam as demais composições de sentenças. Isto deve-se principalmente a dois fatores, necessidade de

desempenho e o fato de que em Prolog a seqüência de apresentação das cláusulas ter diferença quanto ao resultado.

3 - Manter as cláusulas agrupadas por caso tratado mesmo ao custo do desempenho (tempo) de processamento. Esta decisão foi tomada em função da facilidade, mais tarde, em manter e entender o programa.

4 - Guardar na base de dados apenas uma representação para cada palavra (masculino,singular no caso de substantivos e forma no infitivo no caso de verbos). Esta decisão foi tomada visando facilitar uma possível aplicação prática futura do programa, pois deste modo torna-se mais fácil a inserção de novos componentes à base de dados uma vez que há uma forma padrão de entrada e, ao mesmo tempo, o arquivo de dados é muito mais enxuto.

5 - Manter os substantivos em árvore btree. Esta é outra decisão tomada pensando-se no futuro pois ao manter os dados guardados em btree torna o tempo de acesso muito menor, o que pode ser um fator crítico quando tivermos bases maiores.

6 - Guardar os radicais das palavras para os tratamentos de conversão em forma canônica (forma guardada na base de dados) em btree e buscando-os em cláusulas que exploram muito mais a recursão que o backtracking. Esta decisão foi tomada em função de minimizar o tempo de processamento, uma vez que verificou-se que uma das funções que mais dispendiam tempo era o tratamento de palavras devido ao número de backtracking para cada palavra.

Tomadas estas decisões e pondo-as em prática, chegou-se à versão atual do programa. Há no entanto diversas sugestões que posso fazer para melhorá-lo e que poderão ser implementadas ou não a critério de quem no futuro vier a utilizá-lo :

1 - Posicionar todos os termos terminais em árvores Btree;

2 - Tratar a componente temporal das sentenças;

3 - Rearranjar as cláusulas usando-se teoria de compiladores de modo a tornar as construções deterministicas

4 - Tratar as variações de inflexão dos verbos.

O objetivo destas medidas é diminuir as limitações do programa tanto quanto a representatividade das informações quanto a desempenho no tempo de processamento.

Apêndice - C

Listagem dos Programas

```
/* Listagem do programa de geração das telas de apresentação */

janela_entrada:-
  define_window(janela1,_,(0,0),(24,79),(30,-7)),
  define_window(janela2,_,(3,3),(21,76),(30,-7)),
  define_window(janela3,_,(6,6),(18,73),(30,-7)),
  current_window(X,janela1),
  cls,
  current_window(janela1,janela2),
  cls,
  current_window(janela2,janela3),
  cls,
```

```
  tmove(3,18),wa(27,10),write(' Universidade de Sao Paulo'),
  tmove(5,8),wa(49,10),write(' Instituto de Ciencias
                                        Matematicas de Sao Carlos'),
  tmove(7,6),wa(54,10),write(' Departamento de Ciencias da
                                          Computacao e Estatistica'),
  keyb(_,_),
  current_window(janela3,janela1),
  tmove(2,12),wa(44,19),
  write('UMA GRAMATICA PARA PERGUNTAS DO TIPO
  SIM/NAO'),
  tmove(4,21),wa(23,19),
  write('COM TRATAMENTO DE FOCO'),
  tmove(8,6),write('Autor : '),
  wa(13,23),write('CLAUDIO OSIRO'),
  tmove(10,6),write('Orientadora : '),
  wa(37,23),write('Profa. Dra. Maria das Gracas V. Nunes'),
  tmove(14,6),write('Este programa e parte integrante do
                                      Relatorio final da Bolsa de'),
  tmove(16,3),write('de Iniciacao Cientifica de mesmo titulo e
                                    contem a implementacao das'),
  tmove(18,3),write('construcoes para tratamento de
                                  perguntas proposto no mesmo.'),
  tmove(21,10),write('Tecle algo para continuar...'),
  keyb(_,_),
  delete_window(_).


/* Listagem da cláusula geradora da mensagem de saida do programa */

janela_saida:-
  define_window(jan_s1,_,(9,6),(13,60),(30,-7)),
  current_window(X,jan_s1),
  cls,
  tmove(1,10),wa(33,10),write(' Programa encerrado corretamente.'),
  keyb(_,_),
  delete_window(_).

/* Listagem do programa gerador da tela principal e do menu de controle */

janela1:-
  define_window(jal1,'Modulo de Geração de
                                  Representação Logica',_,_,(30,-7)),
  define_window(jal2,'Leitura de Pergunta',(2,5),(9,60),(14,
                                                        -26)),
  define_window(jal3,'Resposta',(11,5),(22,70),(14,-26)),
```

```
  current_window(_,jal1),
  wa(78,23),
  tmove(22,64),wa(11,15),write($ ICMSC-USP $).

apaga:-
  delete_window(_).

begin_menu(teste,78,colors((48,95),(127,15),(123,123),
                                                    (116,79))).
  item($~Ativar$,ativar).
  item($~Termina$,fim).
end_menu(teste).

menu_teste(Valor):-
 send_menu_msg(draw(teste,(0,0)),true),
 send_menu_msg(activate(teste,(0,0)),Valor).

x:-
 janela1,
 corpo.

corpo:-
 menu_teste(A),
 continua(A).

continua(ativar):-
  entrada(Lista),
  create_popup(_,(10,35),(13,65),(27,114)),
  sentenca(Foco,Mens,Lista,[]),
  exit_popup,
  current_window(jal2,jal3),
  tmove(1,2),wa(16,15),
  write('Formula Logica : '),
  saida(3,2,Foco,Mens),
  tmove(8,2),wa(34,15),write('Pressione uma tecla para
                                                  continuar '),
   keyb(_,_),
  cls,
  current_window(jal3,jal2),
  cls,
  current_window(jal2,jal1),
  corpo.

continua(_):-
  apaga.
```

```
entrada(Lista):-
  current_window(jal1,jal2),
  tmove(2,2),wa(26,15),
  write('Entre a pergunta desejada: '),
  tmove(4,2),leitor(Lista).


/* Cláusulas geradoras das mensagens de resposta do programa */

saida(X,Y,Foco,Mens):-
   atomic(Foco),
   atomic(Mens),
   tmove(X,Y),write(Mens),
   X1 is X+1,
   tmove(X1,Y),write(Foco).

saida(X,Y,Foco,[Mens|Cauda]):-
   atomic(Foco),
   tmove(X,Y),write(Mens),
   X1 is X+1,
   saida2(X1,Y,Foco,Cauda).

saida(X,Y,[Foco|Cauda1],[Mens|Cauda2]):-
   tmove(X,Y),write(Mens),
   write(' - '),write(Foco),
   X1 is X+1,
   saida3(X1,Y,Cauda1,Cauda2).

saida2(X,Y,Foco,[]):-
   tmove(X,Y),write(Foco).

saida2(X,Y,Foco,[Mens|Cauda]):-
   tmove(X,Y),write('and '),write(Mens),
   X1 is X+1,
   saida2(X1,Y,Foco,Cauda).

saida3(X,Y,[],[]).

saida3(X,Y,[Foco|Cauda1],[Mens|Cauda2]):-
   tmove(X,Y),write('and '),write(Mens),
   write(' - '),write(Foco),
   X1 is X+1,
   saida3(X1,Y,Cauda1,Cauda2).
```

```
ret(X):-
  tmove(1,2),
  write('Pesquisando Clausula      '),
  tmove(1,23),
  write(X).

mensagem_erro1:-
  create_popup('ERRO',(18,15),(24,65),(47,135)),
  tmove(1,1),write('    Atenção esta sentença nao tem
                                                    sentido'),
  tmove(2,1),write(' o Agente nao concorda com o tipo de
                                                    Relação '),
  tmove(4,6),write('Pressione uma tecla para continuar...'),
  keyb(_,_),exit_popup.

mensagem_erro2:-
  create_popup('ERRO',(18,15),(24,65),(47,135)),
  tmove(1,1),write('    Atenção esta sentença nao tem
                                                    sentido'),
  tmove(2,1),write(' o Paciente nao concorda com o tipo de
                                                    Relação '),
  tmove(4,6),write('Pressione uma tecla para continuar...'),
  keyb(_,_),exit_popup.


/* Cláusulas de tratamento da frase de entrada do programa */
leitor(Invertida):-
    read_line(0,Frase),
    elimina(Frase,NF),
    divide(NF,[],Lista),
    inverte(Lista,[],Invertida).

elimina(F,NF):-
    elimina_maiusculas(F,F1),
    elimina_int(F1,F2),
    elimina_virg(F2,NF).

elimina_maiusculas(F,NF):-
    atom_string(A,F),
    name(A,Lista),
    reduz(Lista,L1),
    name(B,L1),
    atom_string(B,NF).

reduz([],[]).
```

```
reduz([X|Cauda],[Y|Cauda2]):-
    X >=65,
    X =<90,
    Y is X+32,
    reduz(Cauda,Cauda2).
reduz([X|Cauda],[X|Cauda2]):-
    reduz(Cauda,Cauda2).

elimina_int(F,NF):-
    string_search($?$,F,P),
    substring(F,0,P,NF).
elimina_int(F,F).

elimina_virg(F,NF):-
    string_search($,$,F,P),
    substring(F,0,P,F1),
    Pos is P+1,
    string_length(F,T),
    Tam is T-Pos,
    substring(F,Pos,Tam,F2),
    concat([F1,' e ',F2],F3),
    elimina_virg(F3,NF).
elimina_virg(F,F).


inverte([],Invert,Invert).
inverte([X|Cauda],Y,Invert):-
   string_term(X,X1),
   inverte(Cauda,[X1|Y],Invert).

divide(Frase,X,Cauda):-
    string_search($ $,Frase,Pos),
    string_length(Frase,Tam),
    Pos>0,
    Tam2 is Tam-Pos-1,
    Pos2 is Pos + 1,
    substring(Frase,0,Pos,Palavra),
    substring(Frase,Pos2,Tam2,Resto),
    divide(Resto,[Palavra|X],Cauda),!.

divide(Frase,X,[Frase|X]):-
    string_length(Frase,Tam),
    Tam > 0.

divide(X,Y,Y).
```

```
/* Cláusulas de tratamento dos substantivos das sentenças */

substantivo(X,C,[X|Cauda],Cauda):-
      !,substant1(X,C).

substant1(X,C):-
       retrieveb(substantivos,X,(C,Gen,Num)).

substant1(X,C):-
         trata_palavra(X,Y),
       retrieveb(substantivos,Y,(C,Gen,Num)).

recupera_substantivos:-
  findall((X,Y,Z,W),substant2(X,Y,Z,W),Lista),
  cria_arvore(Lista).

cria_arvore([]).
cria_arvore([(Radical,Classe,Genero,Numero)|R]):-
  recordb(substantivos,Radical,(Classe,Genero,Numero)),
  cria_arvore(R).


substant2(maria,4,f,s).
substant2(jose,4,m,s).
substant2(joao,4,m,s).
substant2(carro,2,m,s).
substant2(trem,2,m,s).
substant2(casa,3,f,s).
substant2(homem,5,m,s).
substant2(mulher,5,f,s).
substant2(objeto,5,m,s).
substant2(mesa,3,f,s).


/* Cláusulas para tratamento de palavras,
 obtenção das formas caônicas */

trata_palavra(Pal_a,F_Canonica) :-
      atom_string(Pal_a,Pal_s),
      string_length(Pal_s,Tam),
      singularize(Pal_s,Canonica,Tam,1),
      atom_string(F_Canonica,Canonica).
                                        % a palavra esta no plural
```

```
trata_palavra(Pal_a,F_Canonica) :-
        atom_string(Pal_a,Pal_s),
        string_length(Pal_s,Tam),
        masculinize(Pal_s,Canonica,Tam,1),
        atom_string(F_Canonica,Canonica).
                                        % a palavra esta no feminino

trata_palavra(Pal_a,F_Canonica) :-
        atom_string(Pal_a,Pal_s),
        string_length(Pal_s,Tam),
        singularize(Pal_s,Singular,Tam,1),
                                        % a palavra esta no plural
        string_length(Singular,Tam2),
        masculinize(Singular,F_Canonica,Tam2,1),
        atom_string(F_Canonica,Canonica).
                                                % feminino

    % faz o tratamento de uma palavra,
    % voltando seu singular
singularize(Palavra,Canonica,Tam,Ext) :-
        Pos is Tam-Ext,
        substring(Palavra,Pos,Ext,Pal),
        retrieveb(singular,Pal,Extensao),
        substring(Palavra,0,Pos,Radical),
        concat(Radical,Extensao,Canonica).

singularize(Palavra,Canonica,Tam,Ext) :-
        Ext2 is Ext+1,!,
        Ext2 =< Tam,
        singularize(Palavra,Canonica,Tam,Ext2).

   % faz o tratamento de uma palavra,
   % voltando seu masculino
masculinize(Palavra,Canonica,Tam,Ext) :-
        Pos is Tam-Ext,
        substring(Palavra,Pos,Ext,Pal),
        retrieveb(masculino,Pal,Extensao),
        substring(Palavra,0,Pos,Radical),
        concat(Radical,Extensao,Canonica).

masculinize(Palavra,Canonica,Tam,Ext) :-
        Ext2 is Ext+1,
        Ext2 =< Tam,!,
        masculinize(Palavra,Canonica,Tam,Ext2).
```

```
recupera_singular:-
  findall((X,Y),plural_singular(X,Y),Lista),
  cria_singular(Lista).

cria_singular([]).
cria_singular([(Plural,Singular)|R]):-
  recordb(singular,Plural,Singular),
  cria_singular(R).

recupera_masculino:-
  findall((X,Y),feminino_masculino(X,Y),Lista),
  cria_masculino(Lista).

cria_masculino([]).
cria_masculino([(Feminino,Masculino)|R]):-
  recordb(masculino,Feminino,Masculino),
  cria_masculino(R).


plural_singular($os$,$o$).
plural_singular($as$,$a$).
plural_singular($ens$,$em$).
plural_singular($res$,$r$).

                                % define os sufixos femininos
feminino_masculino($a$,$o$).

/***********************
  Entrada de clausulas para o tratamento de verbos
***********************/

trata_verbo(Pal_a,F_Canonica) :-
      atom_string(Pal_a,Pal_s),
      string_length(Pal_s,Tam),
      canonizar_verbos(Pal_s,Canonica,Tam,1),
      atom_string(F_Canonica,Canonica).

canonizar_verbos(Palavra,Radical,Tam,Ext) :-
      Pos is Tam-Ext,
      substring(Palavra,Pos,Ext,Pal),
      retrieveb(arvore_verbos,Pal,Extensao),
      substring(Palavra,0,Pos,Radical).

canonizar_verbos(Palavra,Canonica,Tam,Ext) :-
```

```
    Ext2 is Ext+1,!,
    Ext2 =< Tam,
    canonizar_verbos(Palavra,Canonica,Tam,Ext2).


monta_arv_verbos:-

monta_verbos(terc_pess_sing,[$a$,$ava$,$ou$,$ara$,$ado$,$ada$,$i$,$ia$,$ira$,$ido$]),

monta_verbos(terc_pess_pl,[$am$,$avam$,$aram$,$ados$,$adas$,$iram$,$iam$,$iram$,$
idos$]).

monta_verbos(X,[]).
monta_verbos(Conj,[X|Cauda]):-
  recordb(arvore_verbos,X,Conj),
  monta_verbos(Conj,Cauda).


/*********************
 Clausulas de tratamento e entrada do verbo ser
*********************/

v_ser_negado-->negacao,verbser(X).
verbser(S,[X|Cauda],Cauda):-
   retrieveb(verbo_ser,X,S).

cria_ser:-
   monta_ser([(eh,s),(foi,s),(era,s),(sera,s),(sao,p),(foram,p),
                                                (eram,p),(serao,p)]).

monta_ser([]).
monta_ser([(X,Y)|Cauda]):-
  recordb(verbo_ser,X,Y),
  monta_ser(Cauda).

/* Listagem do programa principal */


/*************************************************************************

  Modulo de Tratador de linguagem Natural

  Objetivo - Transcrever perguntas do tipo Qu_ em Linguagem Natural para
        formulas que obedecem a logica de Primeira Ordem.
```

```
**********************************************************************/

main:-                  % Carrega arquivos de biblioteca
    ['leitor'],
    ['tratamen'],
    ['substant'],
    ['win'],
    ['abertura'],
    recupera_substantivos,
    recupera_masculino,
    recupera_singular,
    monta_arv_verbos,
    cria_ser,
    janela_entrada,
    x,      % inicio de laco de interacao
    janela_saida.


/*************************
 Rotinas de saida de dados
*************************/

escreva_c(X,'Ntem',Z,Saida):-
        atom(X),
      concat([Z,'(',X,')'],Saida).
escreva_c(X,Y,Z,Saida):-
        atom(X),
      concat([Z,'(',X,',',Y,')'],Saida).

escreva1([],[],[],A,A).
escreva1([X|Cauda1],[Y|Cauda2],[Z|Cauda3],S1,Saida):-
      escreva_c(X,Y,Z,S2),
      escreva1(Cauda1,Cauda2,Cauda3,[S2|S1],Saida).

escreva([X|Cauda1],[Y|Cauda2],[Z|Cauda3],Saida):-
      escreva_c(X,Y,Z,S1),
      escreva1(Cauda1,Cauda2,Cauda3,[S1],Saida).

escreva([],X,Y,A,A).
escreva(X,[],Y,A,A).
escreva([X|La],'Ntem',Z,S1,Saida):-
      concat([Z,'(',X,')'],S2),
      escreva(La,'Ntem',Z,[S2|S1],Saida).
```

```
escreva([X|La],Y,Z,S1,Saida):-
        atom(Y),
        concat([Z,'(',X,',',Y,')'],S2),
        escreva(La,Y,Z,[S2|S1],Saida).
escreva(Y,[X|Lp],Z,S1,Saida):-
        atom(Y),
        concat([Z,'(',Y,',',X,')'],S2),
        escreva(Y,Lp,Z,[S2|S1],Saida).
escreva_a([X|La],'Ntem',Ap,Saida):-
         concat([Ap,'(',X,')'],S1),
         escreva(La,'Ntem',Ap,[S1],Saida).
escreva_a([X|La],Paciente,Ap,Saida):-
         concat([Ap,'(',X,',',Paciente,')'],S1),
         escreva(La,Paciente,Ap,[S1],Saida).

escreva_b(Agente,[X|Lp],Ap,Saida):-
          concat([Ap,'(',Agente,',',X,')'],S1),
          escreva(Agente,Lp,Ap,[S1],Saida).

escreva1(X,'Ntem',Z,Saida):-
           concat(['V x,',X,'(x)-->',Z,'(x)'],Saida).
escreva1(X,Y,Z,Saida):-
           concat(['V x,',X,'(x)-->',Z,'(x,',Y,')'],Saida).
escreva2(X,Y,Z,Saida):-
           concat(['V x,',Y,'(x)-->',Z,'(',X,',x)'],Saida).
escreva3(X,'Ntem',Z,Saida):-
           concat(['V x,',X,'(x)--> ~',Z,'(x)'],Saida).
escreva3(X,Y,Z,Saida):-
           concat(['V x,',X,'(x)--> ~',Z,'(x,',Y,')'],Saida).
escreva4(X,Y,Z,Saida):-
           concat(['V x,',Y,'(x)--> ~',Z,'(',X,',x)'],Saida).


/************************************

 Rotinas para tratamento de Linguagem Natural

************************************/


/*********

 Construcoes para o tratamento de quantificador universal

*********/
```

```
%###### Sentencas com o foco no antecedente #######

% Construcoes com o termo quantificado universalmente
% como o
% termo em aposicao

% construcao 1
% <verbo ser> <categoria> <pron.rel.> <senteca c/ agente
% ante_posto>

sentenca(Saida,S1)--> {ret('1')}, verbser(X),
                      categoria(Categoria,Cc), pronrel,
                      sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp),
                      { !,compat_1(Cc,Cr1),
                      compat_2(Cp,Cr2),
                      escreva1(Categoria,Paciente,Ap,S1),
                      unir(' o foco e o antecedente',Saida)}.

% construcao 2
% <artigo definido> <categoria> <verbo ser> <artigo
% definido> <predicado>

sentenca(Saida,S1)--> {ret('2')},artdefinido(Gen,X),
                      categoria(Categoria,Cc), verbser(X2),
                      artdefinido(Gen,X3),
                      predicado(Ap,Cr1,Cr2,Paciente,Cp),
                      { !,compat_1(Cc,Cr1),
                      compat_2(Cp,Cr2),
                      escreva1(Categoria,Paciente,Ap,S1),
                      unir(' o foco e o antecedente',Saida)}.

% construcao 3
% <verbo ser> <categoria> <pron.rel.> <sentenca c/ agente
% na passiva ante_posto>


sentenca(Saida,S1)--> {ret('3')},verbser(X),
                      categoria(Categoria,Cc), pronrel,
                      s_ap_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                      { !,compat_2(Cc,Cr2),
                      compat_1(Ca,Cr1),
                      escreva2(Agente,Categoria,Ap,S1),
                      unir(' o foco e o antecedente',Saida)}.
```

```
% construcao 4
% <verbo ser> <preposicao> <categoria> <pron.rel.>
% <sentenca c/ paciente ante_posto preposicionada>

sentenca(Saida,S1)--> {ret('4')},verbser(X),prep,
                      categoria(Categoria,5),pronrel,
                      sp_ante_posto_p(Ap,Cr1,Cr2,Agente,Ca),
                      { !,compat_2(Cc,Cr2),
                      compat_1(Ca,Cr1),
                      escreva2(Agente,Categoria,Ap,S1),
                      unir(' o foco e o antecedente',Saida)}.

% construcao 5
% <verbo ser> <categoria> <pron.rel> <sentenca com o
% paciente ante_posto>

sentenca(Saida,S1)--> {ret('5')},verbser(X),
                      categoria(Categoria,Cc),pronrel,
                      sp_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                      { !,compat_2(Cc,Cr2),
                      compat_1(Ca,Cr1),
                      escreva2(Agente,Categoria,Ap,S1),
                      unir(' o foco e o antecedente',Saida)}.

%######## Sentecas com o foco no consequente #########

% Construcoes default com o termo quantificado
% universalmente como o
% sujeito da sentenca

% construcao 6
% <categoria> <verbo negado> [<paciente>]


sentenca(Saida,S1)--> {ret('6a')},categoria(Categoria,Cc),
                      verbnegado(Ap,Cr1,Cr2),
                      paciente(Paciente,Cp),
                      { !,compat_2(Cp,Cr2),
                      compat_1(Cc,Cr1),
                      escreva3(Categoria,Paciente,Ap,S1),
                      unir(' o foco e o consequente',Saida)}.

sentenca(Saida,S1)--> {ret('6b')},categoria(Categoria,Cc),
                      verbnegado(Ap,Cr1,Cr2),
                      { !,compat_1(Cc,Cr1),
```

```
                        compat_2(0,Cr2),
                        escreva3(Categoria,'Ntem',Ap,S1),
                        unir(' o foco e o consequente',Saida)}.

% construcao 7
% <categoria> <verbo ser negado> <predicado>

sentenca(Saida,S1)--> {ret('7')},categoria(Categoria,Cc),
                        v_ser_negado,
                        predicado(Ap,Cr1,Cr2,Paciente,Cp),
                        { !,compat_1(Cc,Cr1),
                        compat_2(Cp,Cr2),
                       escreva3(Categoria,Paciente,Ap,S1),
                        unir(' o foco e o consequente',Saida)}.

% construcao 8
% <categoria> <locucao verbal negada> <pron.apass>
% <agente>

sentenca(Saida,S1)--> {ret('8')},categoria(Categoria,Cc),
                        loc_pass_neg(Ap,Cr1,Cr2),
                        p_apass,agente(Agente,Ca),
                        {!,compat_2(Cc,Cr2),
                        compat_1(Ca,Cr1),
                        escreva4(Agente,Categoria,Ap,S1),
                        unir(' o foco e o consequente',Saida)}.

% construcao 9
% <categoria> <verbo ser> <predicado>

sentenca(Saida,S1)--> ret('9')},categoria(Categoria,Cc),
                        verbser(p),
                        predicado(Ap,Cr1,Cr2,Paciente,Cp),
                        { !,compat_2(Cp,Cr2),
                        compat_1(Cc,Cr1),
                        escreva1(Categoria,Paciente,Ap,S1),
                        unir(' o foco e o consequente',Saida)}.

% construcao 10
% <categoria> <locucao verbal> <pron.apass.> <agente>

sentenca(Saida,S1)--> {ret('10')},categoria(Categoria,Cc),
                        loc_passiva(Ap,Cr1,Cr2),
                        p_apass,agente(Agente,Ca),
                        { !,compat_2(Cc,Cr2),
```

```
                        compat_1(Ca,Cr1),
                        escreva2(Agente,Categoria,Ap,S1),
                        unir(' o foco e o consequente',Saida)}.

% construcao 11
% <categoria> <verbo> [<paciente>]

% para verbos transitivos
sentenca(Saida,S1)--> {ret('11a')},categoria(Categoria,Cc),
                        verbo(Ap,Cr1,Cr2),
                        paciente(Paciente,Cp),
                      {!,compat_1(Cc,Cr1),
                        compat_2(Cp,Cr2),
                        escreva1(Categoria,Paciente,Ap,S1),
                        unir(' o foco e o consequente',Saida)}.

% para verbos intransitivos
sentenca(Saida,S1)--> {ret('11b')},categoria(Categoria,Cc),
                        vintransitivo(Ap,Cr1,Cr2),
                        {!,compat_1(Cc,Cr1),
                  compat_2(S,Cr2),
                        escreva1(Categoria,'Ntem',Ap,S1),
                        unir(' o foco e o consequente',Saida)}.

% construções com o termo quantificado universalmente
% dentro das sentenças com termos ante_postos

% construcao 12
% <verbo ser> <agente> <pron.rel.> <sentenca c/ agente
% ante_posto>

sentenca(Saida,S1)--> {ret('12')},verbser(X),
                        agente(Agente,Ca),pronrel,
                        sa_ante_posto(Ap,Cr1,Cr2,Categoria,5),
                        vazio,{ !,compat_2(5,Cr2),
                      compat_1(Ca,Cr1),
                      escreva2(Agente,Categoria,Ap,S1),
                        unir(' o foco e o 1o. argumento do
                                                consequente',Saida)}.

% construcao 13
% <agente> <verbo ser> <artigo definido> <predicado>

sentenca(Saida,S1)--> {ret('13')},agente(Agente,Ca),
                        verbser(X),artdefinido(Gen,Num),
```

```
                    predicado(Ap,Cr1,Cr2,Categoria,5),
                    { Cr2 \= [0],!,compat_2(5,Cr2),
                    compat_1(Ca,Cr1),
                    escreva2(Agente,Categoria,Ap,S1),
                    unir(' o foco e 1o. argumento do
                                        consequente',Saida)}.

% construcao 14
% <verbo ser> <paciente> <pron.rel> <sentenca c/ paciente
% ante_posto na passiva>

sentenca(Saida,S1)--> {ret('14')},verbser(X),
                    paciente(Paciente,Cp),pronrel,
                    s_ap_ante_posto(Ap,Cr1,Cr2,Categoria,5),
                    { !,compat_1(5,Cr1),compat_2(Cp,Cr2),
                    escreva1(Categoria,Paciente,Ap,S1),
                    unir(' o foco e o 2o. argumento do
                    consequente',Saida)}.

% construcao 15
% <verbo ser> <preposicao> <paciente> <pron.rel.>
% <sentenca c/ paciente ante_posto preposicionado>

sentenca(Saida,S1)--> {ret('15')},verbser(X),prep,
                    paciente(Paciente,Cp),pronrel,
                    sp_ante_posto_p(Ap,Cr1,Cr2,Categoria,5),
                    { !,compat_1(5,Cr1),compat_2(Cp,Cr2),
                    escreva1(Categoria,Paciente,Ap,S1),
                    unir(' o foco e o 2o. argumento do
                    consequente',Saida)}.

% construcao 16
% <verbo ser> <paciente> <pron.rel.> <sentenca c/ paciente
% ante_posto>

sentenca(Saida,S1)--> {ret('16')},verbser(X),
                    paciente(Paciente,Cp),pronrel,
                    sp_ante_posto(Ap,Cr1,Cr2,Categoria,5),
                    { !,compat_1(5,Cr1),compat_2(Cp,Cr2),
                    escreva1(Categoria,Paciente,Ap,S1),
                    unir(' o foco e o 2o. argumento do
                    consequente',Saida)}.

% construções default com o termo quantificado
% universalmente
```

```
% como objeto das sentencas

% construcao 17
% <agente> <verbo negado> <categoria>

sentenca(Saida,S1)--> {ret('17')},agente(Agente,Ca),
                      verbnegado(Ap,Cr1,Cr2),
                      categoria(Categoria,5),
                      { !,compat_2(5,Cr2),compat_1(Ca,Cr1),
                      escreva4(Agente,Categoria,Ap,S1),
                      unir(' o foco e consequente',Saida)}.

% construcao 18
% <agente> <verbo ser negado> <predicado>

sentenca(Saida,S1)--> {ret('18')},agente(Agente,Ca),
                      v_ser_negado,
                      predicado(Ap,Cr1,Cr2,Categoria,5),
                      { !,compat_2(5,Cr2),compat_1(Ca,Cr1),
                      escreva4(Agente,Categoria,Ap,S1),
                      unir(' o foco e o consequente',Saida)}.

% construcao 19
% <paciente> <locucao verbal negada> <pron.apass>
% <categoria>

sentenca(Saida,S1)--> {ret('19')},paciente(Paciente,Cp),
                      loc_pass_neg(Ap,Cr1,Cr2),p_apass,
                      categoria(Categoria,5),
                      {!,compat_1(5,Cr1),compat_2(Cp,Cr2),
                      escreva3(Categoria,Paciente,Ap,S1),
                      unir(' o foco e o consequente',Saida)}.

% cosntrucao 20
% <agente> <verbo ser> <predicado>

sentenca(Saida,S1)--> {ret('20')},agente(Agente,Ca),
                      verbser(X),
                      predicado(Ap,Cr1,Cr2,Categoria,5),
                      {  Cr2 \= [0],!,compat_2(5,Cr2),
                      compat_1(Ca,Cr1),
                      escreva2(Agente,Categoria,Ap,S1),
                      unir(' o foco e o consequente',Saida)}.

% construcao 21
```

```
% <paciente> <locucao verbal> <pron.apass.> <categoria>

sentenca(Saida,S1)--> {ret('21')},paciente(Paciente,Cp),
                      loc_passiva(Ap,Cr1,Cr2),
                      p_apass,categoria(Categoria,5),
                      { !,compat_1(5,Cr1),compat_2(Cp,Cr2),
                      escreva1(Categoria,Paciente,Ap,S1),
                      unir(' o foco e o consequente',Saida)}.

% construcao 22
% <agente> <verbo> <categoria>

sentenca(Saida,S1)--> {ret('22')},agente(Agente,Ca),
                      verbo(Ap,Cr1,Cr2),
                      categoria(Categoria,5),
                      {!,compat_2(5,Cr2),compat_1(Ca,Cr1),
                      escreva2(Agente,Categoria,Ap,S1),
                      unir(' o foco e o consequente',Saida)}.

/*********

 Construcoes para o tratamento de construcoes elementares

*********/


%######## sentenças com foco no agente #########

% construção 1
% <verbo ser><agente><pron.rel.><sentenca c/ sujeito
% ante_posto>

sentenca(Saida,S1)--> {ret('23')},verbser(X),
                      agente(Agente,Ca),pronrel,
                      sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp),
                      vazio,{ !,compat_2(Cp,Cr2),
                      compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e o primeiro argumento',
                                                    Saida)}.


% construção 2
% <agente><verbo ser><art.definido><predicado>
```

```
sentenca(Saida,S1)--> {ret('24')},agente(Agente,Ca),
                      verbser(X),artdefinido(Gen,Num),
                      predicado(Ap,Cr1,Cr2,Paciente,Cp),
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e o primeiro argumento',
                                                    Saida)}.

%####### sentenças com o foco no paciente ##########
% construção 3
% <verbo ser><paciente><pron.rel><sentença com objeto
% ante_posto>
% sentenca com objeto indireto

sentenca(Saida,S1)--> {ret('25a')},verbser(X),prep,
                      paciente(Paciente,Cp),pronrel,
                      sp_ante_posto_p(Ap,Cr1,Cr2,Agente,Ca),
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e o segundo argumento',
                                                    Saida)}.

% sentenca com objeto direto

sentenca(Saida,S1)--> {ret('25b')},verbser(X),
                      paciente(Paciente,Cp),pronrel,
                      sp_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                      vazio,{ !,compat_2(Cp,Cr2),
                      compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e o segundo argumento',
                                                    Saida)}.

% construção 4
% <verb ser><paciente><pron.rel.><sentença na passiva c/
% sujeito ante_posto>

sentenca(Saida,S1)--> {ret('26')},verbser(X),
                      paciente(Paciente,Cp), pronrel,
                      s_ap_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                      vazio,
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e o segundo argumento',
                                                    Saida)}.
```

```
%####### sentenças com o foco na ação ########

% construção 5
% <agente><verbo negado>[<paciente>]

sentenca(Saida,S1)--> {ret('27a')},agente(Agente,Ca),
                      verbnegado(Ap,Cr1,Cr2),
                      paciente(Paciente,Cp),vazio,
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e a acao',Saida)}.

sentenca(Saida,S1)--> {ret('27b')},agente(Agente,Ca),
                      verbnegado(Ap,Cr1,Cr2),vazio,
                      { !,compat_2(0,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],'Ntem',Ap,S1),
                      unir(' o foco e a acao',Saida)}.

% construção 6
% <agente><verbo ser negado><predicado>

sentenca(Saida,S1)--> {ret('28')},agente(Agente,Ca),
                      v_ser_negado,
                      predicado(Ap,Cr1,Cr2,Paciente,Cp),
                      vazio,
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e a acao',Saida)}.

% construcao 7
% <paciente><locucao verbal
% negada><part.apass.><agente>

sentenca(Saida,S1)-->{ret('29')},paciente(Paciente,Cp),
                      loc_pass_neg(Ap,Cr1,Cr2),
                      p_apass,agente(Agente,Ca), vazio,
                      {!,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e a acao',Saida)}.


% construção 8
% <agente><verbo ser><art.indefinido><predicado>
```

```
sentenca(Saida,S1)--> {ret('30')},agente(Agente,Ca),
                      verbser(X),artindefinido(Gen,Num),
                      predicado(Ap,Cr1,Cr2,Paciente,Cp),
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e o predicado',Saida)}.

%#########construções default ##############
% construção 9
% <agente><verbo ser><predicado>

sentenca(Saida,S1)--> {ret('31')},agente(Agente,Ca),
                      verbser(X),
                      predicado(Ap,Cr1,Cr2,Paciente,Cp),
                      vazio,
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e o predicado',Saida)}.

% construção 10
% <paciente><locução verbal><part.apassivador><agente>

sentenca(Saida,S1)--> {ret('32')},paciente(Paciente,Cp),
                      loc_passiva(Ap,Cr1,Cr2),
                      p_apass,agente(Agente,Ca),vazio,
                      { !,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e a acao',Saida)}.

% construcao 11
% <agente><verbo>[<paciente>]

% para verbos transitivos diretos
sentenca(Saida,S1)--> {ret('33a')},agente(Agente,Ca),
                      verbo(Ap,Cr1,Cr2),
                      paciente(Paciente,Cp),vazio,
                      {!,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
                      escreva_a([Agente],Paciente,Ap,S1),
                      unir(' o foco e a acao',Saida)}.

% para verbos transitivos indiretos
sentenca(Saida,S1)--> {ret('33b')},agente(Agente,Ca),
                      verbo(Ap,Cr1,Cr2),prep,
                      paciente(Paciente,Cp),vazio,
                      {!,compat_2(Cp,Cr2),compat_1(Ca,Cr1),
```

```
                    escreva_a([Agente],Paciente,Ap,S1),
                    unir(' o foco e a acao',Saida)}.

% para verbos intransitivos
sentenca(Saida,S1)--> {ret('33c')},agente(Agente,Ca),
                    verbo(Ap,Cr1,Cr2),vazio,
                    {!,compat_2(Cr2,Cr2),compat_1(Ca,Cr1),
                    escreva_a([Agente],'Ntem',Ap,S1),
                    unir(' o foco e a acao',Saida)}.


/*********

   Construcao para o tratamento de composicao de sentencas com um termo
em comum ante-posto.

*********/

% construcao unica
% <verbo ser> <termo em comum> <lista de sentencas>

sentenca(Saida,S2)--> {ret('34'),abolish(saida1/4)},
                    verbser(X),termo(Termo,Ct),pronrel,
                     lista_sent_ante_posto(Termo,Ct),
                     {findall(A,saida1(A,_,_,_),L1),
                     findall(B,saida1(_,B,_,_),L2),
                     findall(C,saida1(_,_,C,_),L3),
                     findall(D,saida1(_,_,_,D),Saida),
                    escreva(L1,L2,L3,S1),
                     inverte_lista(S1,[],S2),
                    abolish(saida1/4),nl,nl}.

/*********

 Construcoes para o tratamento de relacoes de implicacao.

*********/

% construcao formadas com um termo em comum elipsado

% construcao 1
% <termo> <sentenca c/ elemento ante_posto> <adv. de causa>
%                                   <sentenca c/ elemento ante_posto>

sentenca(Saida,S3)--> {ret('35')},termo(Termo,Ct),
```

```
                    s_ante_posto(Ap,Cr1,Cr2,Paciente,Cp),
                    adv_causa,
                    s_ante_posto(Ap2,Cr3,Cr4,Paciente2,Cp2),
                    {!,compat_1(Ct,Cr1),compat_2(Cp,Cr2),
                    compat_1(Ct,Cr3),compat_2(Cp,Cr4),
                   escreva_c(Termo,Paciente2,Ap2,S1),
                    escreva_c(Termo,Paciente,Ap,S2),
                    concat([S1,'-->',S2],S3),
                    unir('o foco e o antecedente',Saida)}.

% construcao formuladas com sentencas independentes

% construcao 2
% <termo> <sentenca c/ elemento ante_posto> <adv. de causa>
% <termo>  <sentenca c/ elemento ante_posto>

sentenca(Saida,S3)--> {ret('36')},termo(Termo1,Ct1),
                    s_ante_posto(Ap,Cr1,Cr2,Paciente,Cp),
                    adv_causa,termo(Termo2,Ct2),
                    s_ante_posto(Ap2,Cr3,Cr4,Paciente2,Cp2),
                    {!,compat_1(Ct1,Cr1),compat_2(Cp,Cr2),
                    compat_1(Ct2,Cr3),compat_2(Cp2,Cr4),
                    escreva_c(Termo2,Paciente2,Ap2,S1),
                    escreva_c(Termo1,Paciente,Ap,S2),
                    concat([S1,'-->',S2],S3),
                    unir('o foco e o antecedente',Saida)}.

/*********

 Construcao para o tratamento de composicao de sentencas com um termo em comum

*********/

% construcao unica
% <termo> <lista sentencas>

sentenca(Saida,S2)--> {ret('37'),abolish(saida1/4)},
                    termo(Termo,Ct),ista_sent(Termo,Ct),
                    {findall(A,saida1(A,_,_,_),L1),
                    findall(B,saida1(_,B,_,_),L2),
                    findall(C,saida1(_,_,C,_),L3),
                    findall(D,saida1(_,_,_,D),Saida),
                    escreva(L1,L2,L3,S1),
                    inverte_lista(S1,[],S2),
                    abolish(saida1/4),nl,nl}.
```

```
/*********

Construcoes para o tratamento de sentencas com um termo composto

*********/

% construcao 1
% <verbo ser> <composicao de termo> <pron.rel.> <sentenca c/ agente ante_posto>

sentenca(Saida,S1)--> {ret('38')},verbser(X),comp(Lag,Lca),
                      pronrel,
                      sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp),
                      { !,compat_2(Cp,Cr2),compat_a(Lca,Cr1),
                      escreva_a(Lag,Paciente,Ap,S1),
                      unir(' os focos sao os primeiros
                                          argumentos',Saida)}.

% construcao 2
% <verbo ser> <composicao de termo> <pron.rel>
%                    <sentenca c/ agente ante_posto na passiva>

sentenca(Saida,S1)--> {ret('39')},verbser(X),comp(Lpa,Lcp),
                      pronrel,
                      s_ap_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                      { !,compat_1(Ca,Cr1),
                      compat_b(Lpa,Cr2),
                      escreva_b(Agente,Lpa,Ap,S1),
                      unir(' os focos sao os segundos
                                          argumentos',Saida)}.

% construcao 3
% <verbo ser> <composicao de termo> <pron.rel.>
%                          <sentenca c/ paciente ante_posto>

sentenca(Saida,S1)--> {ret('40')},verbser(X),comp(Lpa,Lcp),
                      pronrel,
                      sp_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                      { !,compat_1(Ca,Cr1),
                      compat_b(Lpa,Cr2),
                      escreva_b(Agente,Lpa,Ap,S1),
                      unir(' os focos sao os segundos
                                          argumentos',Saida)}.
```

```
% construcao 4
% <verbo ser> <preposicao> <composicao de termo>
% <pron.rel.>
%       <sentenca c/ paciente ante_posto Preposicionado>

sentenca(Saida,S1)--> {ret('41')},verbser(X),prep,
                      comp(Lpa,Lcp),pronrel,
                      sp_ante_posto_p(Ap,Cr1,Cr2,Agente,Ca),
                      { !,compat_1(Ca,Cr1),
                      compat_b(Lpa,Cr2),
                      escreva_b(Agente,Lpa,Ap,S1),
                      unir(' os focos sao os segundos
                                          argumentos',Saida)}.

/*****************

 sentenca final
     Avisa do nao reconhecimento da sentenca

*****************/

sentenca(S1,Saida,A,[]):-
    unir('    por favor reformule-a.',S1),
    unir(' A sentenca nao foi reconhecida,',Saida).


%<<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>>>

% <composicao de termo> ::= (<agente> <conjuncao>
% <composicao de termo>| <agente>)

comp([A|La],[C|Lc])-->agente(A,C),conjuncao,comp(La,Lc).
comp([A],[C])-->agente(A,C).


% <termo> ::= <artigo> <substantivo>

termo(A,B)-->artigo(S),substantivo(A,B),!.
termo(A,B)-->substantivo(A,B),!.



% <categoria> ::= <artigo> <substantivo representando uma
% classe>
```

```
categoria(A,B)-->artigo(p),substantivo(A,Cc),!,{Cc == 5}.
categoria(A,B)-->substantivo(A,Cc),!,{Cc == 5}.


% <lista sentencas> ::= <sentenca negativa> <continuacao>

lista_sent(Termo,Ct)-->sentneg(Termo,Ct),cont(Termo,Ct).
lista_sent(Termo,Ct)-->sent(Termo,Ct),cont(Termo,Ct).


% <continuacao> ::= [<conjuncao> <lista sentencas>]

cont(Termo,Ct,Entrada,Saida):-
      conjuncao(Entrada,S1),
      lista_sent(Termo,Ct,S1,Saida).
cont(Termo,Ct,[],[]).


% <sentenca negativa> ::= <negacao> <sent>

sentneg(Termo,Ct)-->negacao,sent(Termo,Ct).


% <sent> ::= (<sentenca c/ agente ante_posto> |
%             <sentenca c/ agente na passiva ante_posto>)

sent(Termo,Ct)-->sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp),
                 {compat_1(Ct,Cr1),compat_2(Cp,Cr2),
                 assert(saida1(Termo,Paciente,Ap,'o foco e o
                                        predicado da clausula')), !}.
sent(Termo,Ct)-->s_ap_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                 {compat_1(Ct,Cr1),compat_2(Cp,Cr2),
                  assert(saida1(Agente,Termo,Ap,'o foco e o
                                        predicado da clausula')), !}.

% <lista sentencas c/ termo ante_posto> ::= <sentenca c/ termo
% ante_posto> <continuacao da lista de sentencas c/ termo
% ante_posto>

lista_sent_ante_posto(Termo,Ct)--> sante_posto(Termo,Ct),
                                   cont_ante_posto(Termo,Ct).


% <continuacao da lista de sentencas c/ termo ante_posto> ::=
% [<conjuncao> <lista de sentencas c/ termo ante_posto>]
```

```
cont_ante_posto(Termo,Ct,Entrada,Saida):-
      conjuncao(Entrada,S1),
      lista_sent_ante_posto(Termo,Ct,S1,Saida).
cont_ante_posto(Termo,Ct,[],[]).


% <sentenca c/ elemento ante_posto> ::= (<negacao> <sentenca
% c/ agente ante_posto> |
% <sentenca c/agente ante_posto> |
% <negacao> <sentenca c/ agente ante_posto na passiva> |
% <sentenca c/ agente ante_posto na passiva>)

s_ante_posto(Ap,Cr1,Cr2,Paciente,Cr2)--> negacao,
                        sa_ante_posto(Ap,Cr1,Cr2,'Ntem',Cp).
s_ante_posto(Ap,Cr1,Cr2,Paciente,Cp)--> negacao,
                        sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp).
s_ante_posto(Ap,Cr1,Cr2,Paciente,Cr2)-->
                        sa_ante_posto(Ap,Cr1,Cr2,'Ntem',Cp).
s_ante_posto(Ap,Cr1,Cr2,Paciente,Cp)-->
                        sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp).
s_ante_posto(Ap,Cr1,Cr2,Paciente,Cp)--> negacao,
                         s_ap_ante_posto(Ap,Cr1,Cr2,Paciente,Cp).
s_ante_posto(Ap,Cr1,Cr2,Paciente,Cp)-->
                         s_ap_ante_posto(Ap,Cr1,Cr2,Paciente,Cp).


% <sentenca c/ termo ante_posto> ::= (<sentenca c/ agente
% ante_posto>| <sentenca c/ paciente ante_posto>|
% <sentenca c/ paciente ante_posto preposicionado>|
% <sentenca c/ agente ante_posto na passiva>)

sante_posto(Termo,Ct)-->
                          sa_ante_posto(Ap,Cr1,[X|Cauda],'Ntem',Cp),
                          {compat_1(Ct,Cr1),compat_2(X,[X|Cauda]),
                          assert(saida1(Termo,Paciente,Ap,'o
                                                  foco e o 1o. termo')),!}.
sante_posto(Termo,Ct)-->sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp),
                        {compat_1(Ct,Cr1),
                        compat_2(Cp,Cr2),
                        assert(saida1(Termo,Paciente,Ap,'o foco
                                                  e o 1o. termo')),!}.


sante_posto(Termo,Ct)-->sp_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
```

```
                        {compat_1(Ct,Cr1),
                         compat_2(Cp,Cr2),
                         assert(saida1(Agente,Termo,Ap,'o foco e
                                                    o 2o. termo')),!}.
sante_posto(Termo,Ct)-->sp_ante_posto_p(Ap,Cr1,Cr2,Agente,Ca),
                        {compat_1(Ct,Cr1),
                         compat_2(Cp,Cr2),
                         assert(saida1(Agente,Termo,Ap,'o foco e
                                                    o 2o. termo')),!}.
sante_posto(Termo,Ct)-->s_ap_ante_posto(Ap,Cr1,Cr2,Agente,Ca),
                        {compat_1(Ct,Cr1),compat_2(Cp,Cr2),
                         assert(saida1(Agente,Termo,Ap,'o foco e
                                                    o 2o. termo')),!}.


% <agente> ::= [<artigo>] <substantivo>

agente(A,B)-->artigo(S),substantivo(A,B),!.
agente(A,B)-->substantivo(A,B),!.

% <sentenca c/ agente ante_posto> ::= ( <vintransitivo> |
%                         <vtdiretp> <paciente> |
%                         <verbo ser> <predicado> |
%                         <vtindireto> <preposicao> <paciente>)

sa_ante_posto(Ap,Cr1,Cr2,'Ntem',0)-->vintransitivo(Ap,Cr1,Cr2),!.
sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp)--> vtdireto(Ap,Cr1,Cr2),
                                         paciente(Paciente,Cp),!.
sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp)--> verbser(X),
                                predicado(Ap,Cr1,Cr2,Paciente,Cp),!.
sa_ante_posto(Ap,Cr1,Cr2,Paciente,Cp)-->vtindireto(Ap,Cr1,Cr2),
                                      prep,paciente(Paciente,Cp),!.


% <sentenca c/ paciente ante_posto preposicionado> ::=
% (<agente> <vtindireto>|
%                             <agente> <verbo ser> <relaco>)

sp_ante_posto_p(Ap,Cr1,Cr2,Agente,Ca)--> agente(Agente,Ca),
                                         vtindireto(Ap,Cr1,Cr2),!.
sp_ante_posto_p(Ap,Cr1,Cr2,Agente,Ca)--> agente(Agente,Ca),
                                         verbser(X),
                                         relacao(Ap,Cr1,Cr2),!.
```

```
% <sentenca c/ paciente ante_posto> ::= <agente> <vtdireto>

sp_ante_posto(Ap,Cr1,Cr2,Agente,Ca)--> agente(Agente,Ca),
                                       vtdireto(Ap,Cr1,Cr2),!.


% <sentenca c/ agente ante_posto na passiva> ::= <verbo ser>
% <vtdireto>  <pron.apass> <agente>

s_ap_ante_posto(Ap,Cr1,Cr2,Agente,Ca)--> verbser(X),
                                         vtdireto(Ap,Cr1,Cr2),
                                         p_apass,
                                         agente(Agente,Ca),!.


% <verbo> ::= <negacao>
% (<vtdireto>|<vtindireto>|<vintransitivo>)

verbnegado(Ap,Cr1,Cr2)-->negacao,vtdireto(Ap,Cr1,Cr2).
verbnegado(Ap,Cr1,Cr2)--> negacao, vtindireto(Ap,Cr1,Cr2),
                          prep.
verbnegado(Ap,Cr1,Cr2)--> negacao,
                          vintransitivo(Ap,Cr1,Cr2).


% <verbo> ::= (<vtdireto>|<vtindireto>|<vintransitivo>)

verbo(Ap,Cr1,Cr2)-->vtdireto(Ap,Cr1,Cr2).
verbo(Ap,Cr1,Cr2)-->vtindireto(Ap,Cr1,Cr2),prep.
verbo(Ap,Cr1,Cr2)-->vintransitivo(Ap,Cr1,Cr2).


% <locucao verbal> ::= <verbo ser> <vtindireto>

loc_passiva(Ap,Cr1,Cr2)-->verbser(X),vtdireto(Ap,Cr1,Cr2).


% <locucao verbal negada> ::= <negacao> <locucao
% verbal>

loc_pass_neg(Ap,Cr1,Cr2)--> negacao,
                            loc_passiva(Ap,Cr1,Cr2).


% <predicado> ::= (<adjetivo> | <relacao> <preposicao> <paciente>)
```

```
predicado(Ap,Cr,[0],'Ntem',0)-->adjetivo(Ap,Cr).
predicado(Ap,Cr1,Cr2,Paciente,Cp)--> relacao(Ap,Cr1,Cr2),
                                     prep,
                                     paciente(Paciente,Cp).


% <paciente> ::= [<artigo>]<substantivo>.

paciente(A,Cp)-->artigo(S),substantivo(A,Cp),!.
paciente(A,Cp)-->substantivo(A,Cp),!.


% <vtdireto> ::= <verbo transitivo direto>

vtdireto(Y,C1,C2,[X|Resto],Resto):-
        trata_verbo(X,Rad),
        verbo_td(Rad,Y,C1,C2).


% <vtindireto> ::= <verbo transitivo indireto>

vtindireto(Y,C1,C2,[X|Resto],Resto):-
        trata_verbo(X,Rad),
        verbo_ti(Rad,Y,C1,C2).


% <vintransitivo> ::= <verbo intransitivo>

vintransitivo(Y,C1,C2,[X|Resto],Resto):-
        trata_verbo(X,Rad),
        verbo_i(Rad,Y,C1,C2).

/**************
 Entrada de relacoes validas para a composicao de predicados
**************/

relacao(irmao,[4,5],[4,5])-->[irmao].
relacao(irmao,[4,5],[4,5])-->[irmaos].
relacao(irmao,[4,5],[4,5])-->[irma].
relacao(irmao,[4,5],[4,5])-->[irmas].
relacao(marido,[4,5],[4,5])-->[marido].
relacao(esposa,[4,5],[4,5])-->[esposa].
relacao(filho,[4,5],[4,5])-->[filho].
relacao(filho,[4,5],[4,5])-->[filha].
```

```
relacao(filho,[4,5],[4,5])-->[filhos].
relacao(filho,[4,5],[4,5])-->[filhas].


/***************
  clausulas de tratamento de adjetivo
***************/

adjetivo(A,C,[A|Cauda],Cauda):-
    adjet(A,C,Gen,Num).
adjetivo(X,C,[A|Cauda],Cauda):-
    trata_palavra(A,X),
    adjet(X,C,Gen,Num).


/*******************
  Entrada de adjetivos validos
*******************/

adjet(bonito,[1,2,3,4,5],m,s).
adjet(feio,[1,2,3,4,5],m,s).
adjet(lento,[2,4,5],m,s).

/*******************
  Entrada de artigos
*******************/

artigo(Num)-->artdefinido(Gen,Num).
artigo(Num)-->artindefinido(Gen,Num).

artdefinido(m,s)-->[o].
artdefinido(m,p)-->[os].
artdefinido(f,s)-->[a].
artdefinido(f,p)-->[as].
artindefinido(m,s)-->[um].
artindefinido(m,p)-->[uns].
artindefinido(f,s)-->[uma].
artindefinido(f,p)-->[umas].

/*******************
  Entrada dos verbos validos
*******************/

verbo_i(viaj,viajar,[1,2,3,4,5],[0]).
verbo_td(am,amar,[4,5],[1,2,3,4,5]).
```

```
verbo_td(ador,adorar,[4,5],[1,2,3,4,5]).
verbo_td(possu,possuir,[1,2,3,4,5],[1,2,3]).
verbo_ti(gost,gostar,[4,5],[1,2,3,4,5]).


/*******************
  Entrada do pronome relativo valido
*******************/

pronrel-->[que].

/*******************
  Entrada das preposicoes validas
*******************/

prep-->[de].
prep-->[do].
prep-->[dos].
prep-->[da].
prep-->[das].

/*********************
  Entrada de pronomes para formacao de passiva
*********************/

p_apass-->[por].
p_apass-->[pela].
p_apass-->[pelo].
p_apass-->[pelos].
p_apass-->[pelas].

/**********************
  Adverbio de negacao valido
**********************/

negacao-->[nao].

/***********************
  Teste para verificacao se lista retornada e vazia
***********************/

vazio([],[]).

/***********************
  Conjuncao valida
```

```
**********************/

conjuncao-->[e].

/************************
 Adverbios de causa para formacao de implicacoes
************************/

adv_causa-->[porque].
adv_causa-->[pois].

/************************
 Testes de compatibilidade de classes de elementos
************************/

% Classes de compatibilidade para as relações
% 1 acoes
% 2 animados
% 3 inanimados
% 4 pessoas
% 5 categoria
% note que 4 e compativel com 2, mas 2 nem sempre e compativel com 4

% Testa compatibilidade entre agente e verbo ou predicado

compat_a([],Cr).
compat_a([X|Lcpa],Cr):-
    compat_1(X,Cr),
    compat_a(Lcpa,Cr).
compat_1(A,[A|Cauda]).
compat_1(A,[X|Cauda]):-
    compat_1(A,Cauda).
compat_1(A,[]):-
    mensagem_erro1.

% Testa compatibilidade entre verbo ou predicado e o paciente

compat_b([],Cr).
compat_b([X|Lca],Cr):-
    compat_2(X,Cr),
    compat_b(Lca,Cr).
compat_2(A,[A|Cauda]).
compat_2(A,[X|Cauda]):-
    compat_2(A,Cauda).
compat_2(A,[]):-
```

```
    mensagem_erro2.

unir(X,X).

inverte_lista([],A,A).
inverte_lista([A|X],X1,Cauda):-
  inverte_lista(X,[A|X1],Cauda).
```

Indíce

Bibliografia.

*[Casanova-86] Casanova,M.A. - **Programação em Lógica** - V Escola de Computação, Belo Horizonte, 1986*

*[Clocksin-81] Clocksin,W.F. and Mellish,C.S. - **Programming in Prolog** - Springer - Verlag , New York, 1981.*

*[Monteiro Jr.-92] Monteiro Jr.,J. - **Adequação de um Sistema de Processamento de Linguagem Natural para Interpretação de Perguntas** - Relatório de Atividades de Iniciação Científica - UFSCar - São Carlos , 1992*

*[Nunes-91] Nunes,M.G.V. - **A Geração de Respostas Cooperativas em Sistemas Baseados em Lógica** - Dissertação de Doutorado - Departamento de Informática - PUC do Rio de Janeiro - Rio de Janeiro, março de 1991.*

*[Pereira-80] Pereira,F.C.N. and Warren,D.H.D - "Definite Clause Grammars for Language Analisys - A Survey of the Formalism And a Comparison with Augmented Transition Networks" - **Artificial Inteligence**, 13 (1980), 231-278.*

*[Rino-87] Rino,L.H.M.R. - **Uma Interface em Linguagem Natural para Recuperação de Conhecimento** - Dissertação de Mestrado - ICMSC - USP - São Carlos, 1987.*