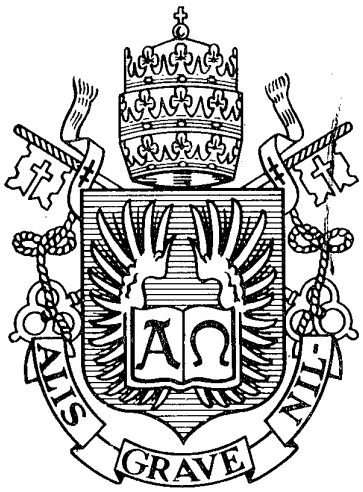


PUC

Série: Monografias em Ciência da Computação
No.1/89

Lettera : um gerador de textos em Português

Maria das Graças Volpe Nunes

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 – CEP 22453

RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Série : Monografias em Ciência da Computação, No. 1/89

Editor : Paulo A. S. Veloso Janeiro, 1989

Lettera : um gerador de textos em Português *

Maria das Graças Volpe Nunes

* Trabalho apresentado como Qualificação à Profa. Clarisse S. de Souza.
Parcialmente financiado pela FINEP.

Para obter cópias :
Rosane T. L. Castilho
Assessoria de Biblioteca, Documentação e Informação
Rua Marquês de São Vicente, 225 - Gávea
22.453 - Rio de Janeiro, RJ.
Brasil

ABSTRACT : A text generator in Portuguese is proposed. It syntactically realises answers in a question-answer system. The semantic representation of an answer is previously constructed by the planner, based on the Rhetorical Structure Theory of Mann and Thompson. Stylistic and focus-driven transformations are applied on basic sentences generated by a unification grammar.

KEY-WORDS : text generation, unification grammar, syntactic and semantic subordination and coordination, syntactic transformations on sentences, RST schema.

RESUMO : É proposto um gerador de textos em Portugês, que é parte de um sistema que responde a perguntas do usuário sobre um certo domínio. Lettera é o componente linguístico que realiza, sintaticamente, as respostas previamente planejadas por um componente estratégico, que usa a Rhetorical Structure Theory, de Mann e Thompson, Rhetorical Structure Theory, para construir a representação semântica de cada uma delas. Transformações estilísticas como coordenação aditiva, elipse e anaforização, bem como relativas a focalização como topicalização, passivização e emolduração, são aplicadas às sentencas básicas, previamente geradas por uma gramática de unificação.

PALAVRAS-CHAVES : geração de texto, gramática de unificação, subordinação e coordenação sintáticas e semânticas, transformações sintáticas sobre sentenças, esquemas RST.

ÍNDICE

Lettera : um gerador de textos em Portugês

1. INTRODUÇÃO.

Lettera é um gerador que produz textos em Português a partir de uma representação semântica. Ele é o componente linguístico do gerador de respostas de um sistema que aceita perguntas em linguagem natural sobre um certo domínio. Parte deste sistema já está implementada em Arity Prolog - os componentes estratégico e linguístico - e outra parte está em desenvolvimento - compreensão de linguagem natural e tratamento do diálogo.

A entrada para Lettera é uma estrutura retórica que indica as relações válidas entre os fatos nela contidos e seus efeitos desejáveis sobre as crenças do usuário; suas saídas são sentenças em Português. A representação semântica da resposta segue o modelo proposto por [Mann & Thompson-86] na sua Rhetorical Structure Theory - RST.

O conjunto de sentenças geradas inclui sentenças simples e aquelas resultantes de transformações tais como subordinação, coordenação, passivização e uso de anáforas pronominais e não pronominais . Para obter tais características, Lettera divide seu trabalho em 4 fases distintas : 1) Determinação da estrutura do período; 2) Geração de sentenças básicas; 3) Transformações sobre as sentenças básicas e 4) Morfologia.

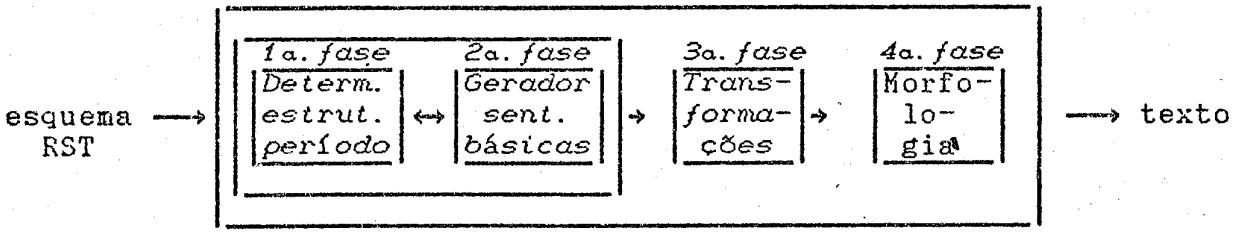


fig.1.: O Componente Linguístico Lettera

A entrada para a Determinação da estrutura do período é uma instância de esquema RST, contendo relações entre núcleos (orações principais) e satélites (orações subordinadas) e que é construída pelo componente estratégico. Nesta primeira fase são escolhidas a forma ou estrutura do período e as conjunções ou locuções que sinalizam sintaticamente a dependência entre as sentenças. Em seguida, o núcleo e o satélite dão origem a uma sentença básica cada, a partir do predicado que carrega a informação a ser veiculada. A estrutura da sentença básica é uma lista cujos elementos denotam os elementos sintáticos da sentença. A terceira fase realiza transformações sobre as sentenças básicas, se obedecidas restrições sintáticas e contextuais. O resultado é uma lista de sentenças simples ou coordenadas e sinais de pontuação

que será objeto de uma finalização que inclui consultas a dicionários e aplicacões de regras morfológicas, resultando no texto final em Português.

Todas as fases são discutidas em detalhes nas secões seguintes.

A seção 2 dá mais detalhes sobre o sistema de perguntas e respostas que abriga **Lettera** e, principalmente, do componente estratégico que o antecede.

A seção 3 discute as principais metodologias para construção de geradores de linguagem natural e justifica a metodologia escolhida para **Lettera**.

As seções 4,5,6 e 7 detalham as quatro fases de **Lettera**. Exemplos e conclusões estão nas seções 8 e 9 respectivamente.

2. O SISTEMA QUE ABRIGA Lettera

O sistema de perguntas e respostas, do qual Lettera é parte, tem como função responder a perguntas feitas em linguagem natural - L.N. - sobre um determinado domínio, de maneira cooperativa, segundo os critérios de Grice [Grice-75]. Para tal, ele conta com um módulo de compreensão de L.N., um módulo de determinação do foco da pergunta, um componente estratégico que determina o conteúdo semântico da resposta e o componente linguístico Lettera (estes dois últimos completamente implementados; os demais, em desenvolvimento).

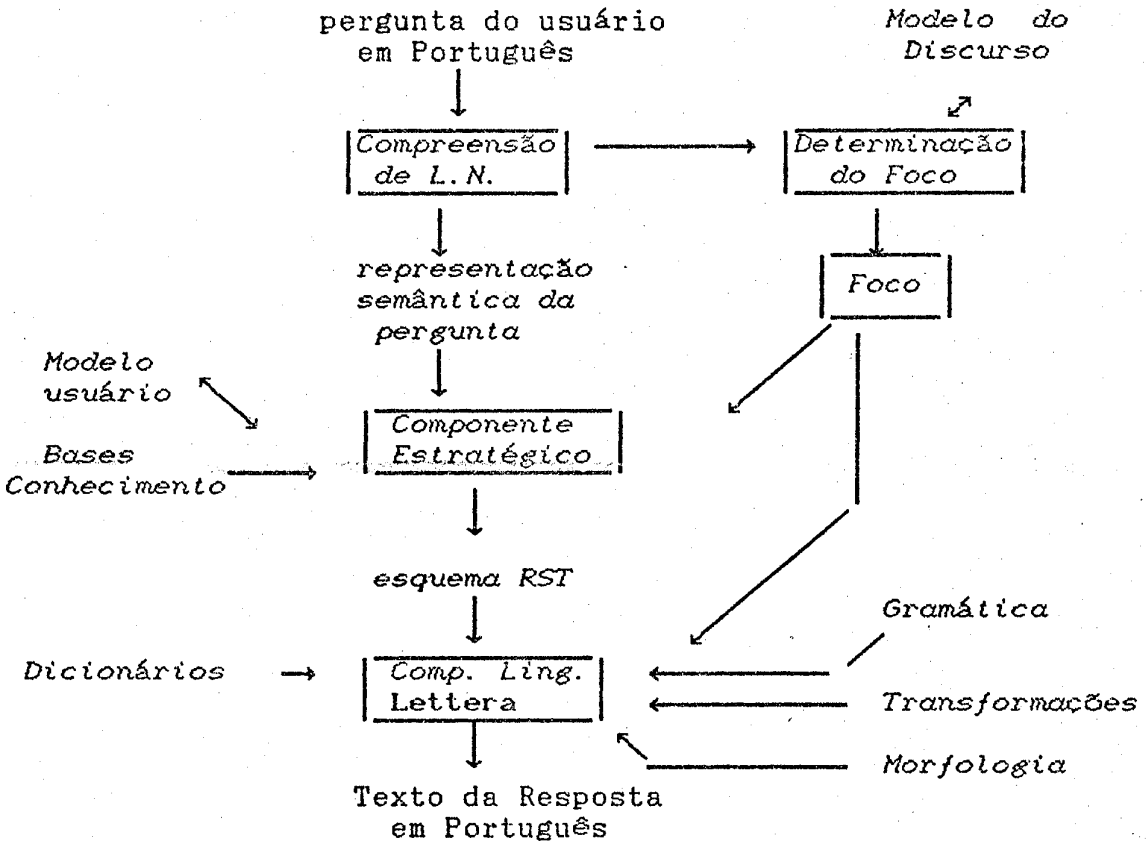


fig.2.: Sistema de Perguntas e Respostas

Até o momento, o sistema trata apenas de perguntas do tipo Sim/Não. Esta escolha se deve ao fato de que, sob o ponto de vista de geração de respostas, a diferença mais significante entre uma pergunta Qu__ e uma do tipo Sim/Não, é que enquanto a primeira sinaliza explicitamente quais aspectos da pergunta o usuário desconhece ou sobre os quais tem dúvidas, a segunda costuma ser bastante obscura quanto a este aspecto. Outra diferença importante é que, em geral, perguntas do tipo Sim/Não deixam mais expostas as pressuposições do usuário em relação ao domínio.

Estas características nos levaram a prever um módulo que determina o foco da pergunta, levando em conta um modelo do diálogo sistema/usuário, para o primeiro caso, e a desenvolver um tratamento de *misconceptions* seguindo as idéias de K. McCoy [McCoy-87], no segundo caso.

2.1. O COMPONENTE ESTRATÉGICO

Ao componente estratégico (ou planejador) cabe o planejamento da resposta, isto é, a decisão sobre "o quê dizer" e "quando dizê-lo". As informações selecionadas ou derivadas das bases de conhecimento devem ser organizadas visando uma apresentação textual. Isto implica adicionar especificações retóricas e determinar a sequência na qual estas informações serão apresentadas. Uma forma intermediária da resposta é produzida pelo componente estratégico.

Para alcançar um bom nível de cooperação na resposta, quatro diferentes fontes de conhecimento foram necessárias :

- Bases de Conhecimento do Domínio;
- Modelo de Senso Comum no Domínio : útil no tratmento de *misconceptions*, na provisão de informações relevantes adicionais e na determinação de um foco-*default* no caso de o Modelo do Discurso ser incapaz de determinar o foco da pergunta;
- Modelo do Usuário : determina quais informações relevantes devem ser omitidas, por já serem de conhecimento do usuário;
- Modelo do Discurso : deve produzir a informação sobre o foco da pergunta, baseado nos *status* do diálogo e do modelo do usuário e no domínio.

O domínio escolhido para testar o sistema contém conhecimento sobre um subconjunto do Código Penal Brasileiro e sobre um conjunto de fatos criminais supostamente ocorridos.

A representação semântica da resposta é uma estrutura retórica hierárquica, contruída segundo os moldes da teoria RST de W.Mann e S.Thompson, usada neste contexto como método gerativo, ao contrário de como é originalmente proposta, ou seja, como método descritivo.

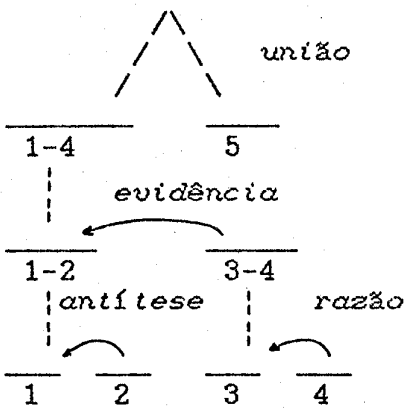
RST é um método linguisticamente útil para descrever textos em L.N., caracterizando suas estruturas principalmente em termos de relações que ocorrem entre partes do texto e como elas interagem entre si para alcançar o objetivo final da geração.

Como método descritivo para textos, esta teoria possui boas características : identifica estruturas hierárquicas no texto; descreve as relações entre partes do texto em termos funcionais; possibilita uma análise abrangente (vários tipos de textos) e é insensível ao tamanho do texto.

RST, como método gerativo, tem como meta mostrar a intenção

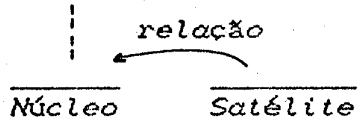
do escritor em relação ao leitor, e deve ser alcançada através de sucessivas seleções (submetas) de esquemas até obter uma estrutura "profunda" que corresponda a um texto coerente e coeso [Hobbs-79], isto é, que possua uma única relação no mais alto nível.

Por exemplo, o planejador poderia dar origem ao seguinte esquema RST, apos sua ativação pela pergunta : "Ana foi morta por um estranho ?" (Os números correspondem ao texto da resposta) :



ao qual corresponderia o (possível) texto :
(1)Não. (2)Foi João que a matou (3)com três tiros (4) por considerá-la infiel. (5)Ela era sua esposa.

Representamos, graficamente,



a relação conceitual entre a parte principal da informação, que contém a meta - o *Núcleo* - com a parte menos importante, complementar, de suporte - o *Satélite*.

Maiores detalhes do componente estratégico podem ser encontrados em [Nunes-88].

3. O COMPONENTE LINGUÍSTICO

Cabe a este componente-realizador decidir "como" realizar, através de texto em L.N., o objetivo de transferir as informações selecionadas pelo planejador para o usuário. Portanto, são decisões linguísticas aquelas envolvidas nesta fase. Algumas delas são :

- Escolha de vocabulário;
- Estilo de prosa : independe do domínio;
- Detalhes sintáticos e morfológicos : trasnformações sentenciais, conjugações verbais, etc.;
- Reestruturação da estrutura interna forçada pela gramática;
- Manifestação das intenções do escritor (sistema) : regras de foco;
- Manutenção da coesão do texto.

Podemos determinar, na literatura, quatro modelos de integração entre os componentes estratégico e linguístico de um gerador de textos.

No modelo sequencial, o planejamento do texto é totalmente feito antes da realização. Uma vez terminado o planejamento, o planejador já não apresenta qualquer utilidade para o realizador. Exemplos deste modelo são os sistemas de [McDonald & Pustejovsky-85], [McKeown-82] e o nosso, entre outros. Este modelo se ressente das consequências de uma estrutura estritamente sequencial : o planejador não leva em conta características sintáticas e nem o realizador pode retroceder à fase anterior, caso se depare com algum imprevisto.

No modelo integrado, cujo maior representante é o sistema KAMP/TELEGRAM [Appelt-83], planejamento e realização constituem um processo contínuo : o sistema manipula restrições sintáticas da mesma maneira que manipula informações sobre foco ou modelo do usuário. A única diferença é que as restrições sintáticas tendem a aparecer mais tarde no processo de geração. Este método, por sobrecarregar a interface estratégia/tática tornou impraticável o sistema, do ponto de vista computacional, embora seja conceitualmente atraente.

O terceiro modelo, intercalado, é o meio termo entre o sequencial e o integrado : o planejador é ativado pelo realizador quando este achar necessário. Assim, ele intercala planejamento e realização e é caracterizado por uma comunicação nos dois sentidos em pontos de decisão do realizador. As vantagens, segundo o autor da proposta [Hovy-88], são : separação das tarefas de planejamento e realização; o planejamento pode levar em conta certas oportunidades e também inadequações sintáticas.

Finalmente, no modelo conciliatório, embora estruturalmente sequenciais, o planejador e o realizador compartilham conhecimento cognitivo e linguístico. Com isto, o planejador é capaz de guiar suas decisões também por informações linguísticas, e o realizador,

tendo acesso aos elementos que influenciam o planejamento, é capaz de escolher alguma outra opção, quando aquela que se apresenta não for apropriada do ponto de vista sintático. Esta nova escolha não ofenderia, assim, os critérios de planejamento. De qualquer forma, o planejador não é ativado para uma nova tentativa, ou seja, não há *backtracking*. Este modelo foi proposto no contexto do sistema GEMA, para geração de descrições [Scott & Souza-88].

3.1. Lettera

A relação do componente estratégico descrito em 2.1. com Lettera é de total independência.

Seguindo a teoria de C.Mathiessen e S.Thompson [Mathiessen & Thompson-87], segundo a qual as regras sintáticas que regem a combinação de sentenças (*Clause Combining*) constituem exatamente a realização gramatical da organização retórica de discurso proposta pela teoria RST, identificamos os fenômenos da hipotaxe e da parataxe como realizações das relações nucleares (contendo o par núcleo-satélite) e multinucleares dos esquemas RST, respectivamente.

Mais especificamente, as hipotaxes do tipo expansionista que envolvem relações circunstanciais (de razão, condição, propósito, causa, modo, etc.) corresponderiam às relações retóricas nucleares. A coordenação aditiva, por outro lado, corresponderia à relação multinuclear "sequência", que abriga informações igualmente importantes, isto é, não subordinadas entre si.

Se, por um lado, o planejador nos fornece a estrutura da resposta na forma de relações retóricas e estas nos indicam quais orações podem ser principais ou subordinadas e ainda quais podem ser coordenadas, por outro lado, a gramática da língua portuguesa prevê regras sintáticas que permitem a realização destas subordinações e coordenações por meio de elementos de ligação - conjunções, locuções e pontuação.

Assim, temos detectados todos os elementos necessários para nosso objetivo. Mas, como organizá-los de forma eficiente ?. Bem, a resposta a esta questão comprometeu a estrutura e, consequentemente, o desempenho de **Lettera**.

Para justificar a organização escolhida é necessário esclarecer que o projeto e a implementação dos componentes estratégico e linguístico ocorreram, por força de várias circunstâncias, em épocas consecutivas, porém, totalmente independentes. Isto resultou no fato de que estes componentes não interagem entre si, com exceção da interface que faz da saída do planejador a entrada do componente linguístico.

Uma das consequências desta imposição foi que a representação de conhecimento, mais especificamente, a escolha dos predicados que abrigam toda a informação manipulável, que pareceu adequada ao primeiro projeto, possivelmente comprometeu a gramática usada no

segundo. Isto não quer dizer que de outro modo teríamos, com certeza, chegado a melhores resultados, mas não nos deixou outra alternativa senão usarmos uma gramática de unificação que consiste em uma ou mais regras para cada predicado que possa aparecer na estrutura interna da resposta.

Todavia, o uso de predicados para representar as informações que o sistema detém, facilitaria a utilização de uma gramática de inspiração léxico-funcionalista, onde informações lexicais, semânticas e sintáticas seriam encontradas em cada entrada do léxico. Mais importante seria sua aplicação também no módulo de compreensão de L.N. O que notamos é que uma escolha cuidadosa dos predicados seria indispensável. Tomar este caminho significava, para o nosso projeto, refazer grande parte do componente estratégico que nos fornece a entrada para o sistema.

A gramática de predicados é usada na primeira fase, ou seja, na geração das sentenças básicas : cada predicado que aparece no esquema RST dá origem a uma sentença básica, através da ativação da gramática gerativa. Mas, antes disso, já teremos determinado a estrutura do período, uma vez que, pelo que já foi exposto acima, sabemos exatamente o que pode ser oração principal, subordinada ou coordenada.

Uma ressalva deve ser feita. Sob circunstâncias especiais, uma relação nuclear pode dar origem não a uma oração principal e uma subordinada, como se espera, mas a uma oração, correspondente ao núcleo, tendo como adjuntos adverbiais as informações do satélite. Outra decisão que o componente linguístico toma é a de omitir todo um núcleo se este já foi expresso em algum trecho anteriormente gerado. Estas são as únicas interferências feitas pelo componente linguístico que , pode-se dizer, alteram não a estrutura vinda dc planejador mas o critério de se realizar toda relação nuclear como orações principal e subordinada.

Finalmente, percebemos que a correlação entre proposição e sentença (cada predicado dá origem uma sentença básica) além de ser fracamente suportada pela prática, acarreta numa estrutura muito rígida do texto, onde as únicas variações foram excepcionalmente introduzidas coforme descrito acima.

A seguir discutiremos, com mais detalhes, cada uma das fases de **Lettera**.

4. DETERMINAÇÃO DA ESTRUTURA DO PERÍODO

A entrada para esta fase é a estrutura retórica hierárquica proveniente do planejador. Esta estrutura contém intâncias de relações retóricas nucleares e multinucleares. De acordo com a correspondência entre estas relações e os fenômenos de hipotaxe e parataxe, esta fase é responsável por identificar e determinar a forma do período, escolhendo os sinais sintáticos que indicam a subordinação, ou não, entre as sentenças correspondentes ao núcleo e satélite de cada relação.

O período resultante é produto da geração das sentenças simples, ou não, a partir de cada relação. Percebe-se, desde já, a necessidade de uma integração sintática entre estas sentenças, uma vez que, por ser hierárquica, a estrutura dará origem a uma forma que conterá outra forma e assim por diante. Pois bem, grande parte desta integração é feita por meio de sinais de pontuação como vírgulas, pontos e dois pontos. Sua determinação se dá, nesta primeira fase, por meio de informações do contexto na hierarquia no momento de geração daquela relação - através de uma pilha do contexto - e sua correspondência no domínio.

A forma sintática da realização de cada relação é determinada por dois fatores : 1) a semântica da relação propriamente dita, que determinará o conjunto das conjunções possíveis e, 2) a forma sintática escolhida para a relação que a precede no esquema RST (isto significa que algumas combinações sintáticas são prefenciais). Muitas vezes o último critério pode ser relaxado, nunca o primeiro. No máximo, deixa-se de sinalizar uma subordinação com uma conjunção ou locução, mas, neste caso, uma pontuação adequada relaciona semanticamente duas sentenças

Quando várias formas servem para uma determinada intância, é feita uma escolha pelo sistema, baseada na frequência com a qual ela tem sido usada no texto : escolhe-se a menos frequente. Esta prática garante uma grande variação das formas em textos com relações ocorrendo mais de uma vez. E o pequeno número de relações possíveis na aplicação (10) requer uma variação tanto na forma como na escolha das conjunções. Neste aspecto, Lettera é bastante eficiente.

Vejamos como exemplo algumas regras (descritas informalmente aqui) para determinação da forma sintática a partir das relações de *antítese*, *concessão* e *sequência*, as quais consideramos suficientemente ilustrativas : (Usamos Rn e Rs para indicar os textos gerados a partir do núcleo e do satélite, respectivamente).

Regras para relação Antítese :

A.1. Se Núcleo consiste do átomo "Não" (negação da pergunta) e, portanto, Satélite contém toda a resposta correta, então escolha a forma : "Não. Rs"

A.2. Se tanto Núcleo quanto Satélite consistem de uma relação

"sequência" e, portanto, esta relação se dá entre as diferenças entre dois conceitos do domínio,
então escolha a forma : "enquanto Rn, Rs"

A.3. Caso contrário, escolha a forma : "Rn Conj-ant Rs"
onde Conj-ant é a conjunção de menor frequência do conjunto {*ao passo que*, *enquanto que*}.

Regras para a relação Concessão :

C.1. Se o Núcleo descreve as diferenças e o Satélite as semelhanças entre dois tipos de crime,
então escolha a forma : "embora Rs, Rn"

C.2. Se a relação-pai desta concessão for de antítese e a regra ativada tiver sido A.1.,
então escolha uma das duas formas :
"Rn Conj-conc1 Rs" ou "Conj-conc2 Rs, Rn"
onde Conj-conc1 deve ser a menos frequente dentre {*embora*, *ainda que*, *mesmo*, *apesar de*, *se bem que* } e Conj-conc2 a menos frequente dentre {*mesmo*, *apesar de*}.

C.3. Caso contrário, acrescente às opções de C.2 as seguintes formas : "Conj-conc Rs, Rn" e "Rs se bem que Rn"
onde Conj-conc deve ser a menos frequente dentre {*embora*, *ainda que*}.

Regras para a relação multinuclear Sequência :
Esta relação possui como argumento uma lista de predicados do domínio : (Denotamos por Sp a sentença básica gerada a partir do predicado p)

S.1. Se a lista possui um único elemento [p],
então ativar a gramática gerativa e a forma será : "Sp"

S.2. Se a lista possui mais de um elemento [p1,p2,...pn],
então a forma será : "coord(Sp1, Sp2,...Spn)"
numa indicação que tais sentenças devem ser coordenadas.

Podemos notar que a geração de sentenças básicas não é uma fase que ocorre necessariamente após a determinação da forma do período, mas, sim, é ativada por esta toda vez que, ao percorrer recursivamente a estrutura profunda, for encontrado um predicado do domínio.
Apesar da integração exposta acima, discutiremos a geração de sentenças básicas separadamente, na seção seguinte.

A implementação da primeira fase encontra-se no apêndice 1.

5. GERAÇÃO DE SENTENÇAS BÁSICAS

Como vimos na seção anterior, cada vez que o núcleo ou satélite de uma relação nuclear consistir de um predicado, ou ainda, quando o argumento de uma relação multinuclear é acessado, a gramática gerativa é ativada.

Para cada predicado deve ser gerado o que chamaremos de um "nó sintático" : uma lista cujos elementos denotam os componentes sintáticos da sentença correspondente. Isto é feito através da unificação do predicado com sua(s) regra(s) gramatical(is). O conjunto destas regras forma a Gramática de Geração de Sentenças Básicas.

Cada regra desta gramática tem a seguinte forma : (ent:entrada; saída:argumento de saída)

```
regra(Predicado, Arg1,...Argn, Modo_verbal, Afirm_neg, Nó-Sint.)
```

onde :

Predicado (ent) é o nome do predicado do domínio;

Arg1,...Argn (ent) são os argumentos ordenados do Predicado;

Forma_verbal (ent) é a informação sobre o modo do verbo (indicativo, subjuntivo) ou forma de gerúndio ou infinitivo. O valor *default* é o indicativo mas ele pode ser alterado pela escolha de uma conjunção (por exemplo, *embora*, *mesmo que*) que precederá esta sentença. Isto é sinalizado assim que a conjunção é escolhida.

Afirm_neg é a informação de que a sentença estará na afirmativa ou negativa. Todos os predicados dão origem a sentenças afirmativas a menos que estejam precedidos pelo operador Prolog "*not*". Isto é detectado na fase anterior e, neste caso, a gramática é ativada mudando-se o traço de afirmativo para negativo. Tanto este traço como o Modo_verbal são automaticamente transferidos para o elemento "verbo" do nó sintático.

Nó-Sint. é o único argumento de saída, contendo o nó sintático correspondente a este predicado. Trocamos a estrutura usual de árvore sintática por uma lista especial : cada elemento é pré-fixado por seu papel sintático na sentença e a posição que ocupa na lista reflete diretamente sua posição relativa na sentença.

A opção por uma estrutura linear para representar as sentenças eliminou uma fase, usual em geradores de L.N., chamada de Linearização, que trasformaria estruturas hierárquicas - sintagmas nominais, verbais, preposicionais, etc. - em estruturas lineares exatamente como as propostas por nós. Ou seja, sequências de elementos os quais, num último passo, são analisados pela morfologia, transformados em palavras.

O nó sintático é, em geral, da forma :

[sujC..), verboC..), predC..)/odC..), complC..), (,compl)]

onde :
sujC Nucleo_suj, Artigo, Anafora, Prep, Compl_suj) :

Núcleo_suj - carrega o átomo que é o núcleo do sujeito;
Artigo - sinaliza sobre o uso de artigo antes do sujeito : definido, indefinido, inexistente ou indeterminado (nada pode ser dito). É possível determinar o artigo durante a geração de sentenças, quando temos controle sobre o domínio ou quando o sujeito é uma constante da regra; caso contrário, a Morfologia (um módulo do sistema, não a subárea da Linguística) deveria ser capaz de determiná-lo.
Anáfora - sinaliza se o sujeito está anforizado ou não. É um sinal para a fase de aplicação de regras morfológicas;
Prep - carrega a preposição que separa o núcleo do complemento do sujeito. Nada conterá caso o sujeito não possua complemento;
Compl_suj - carrega o complemento.

verboC Infinitivo, Tempo, Modo, Numero, Afirm_neg) :

Infinitivo - verbo no infinitivo;
Tempo - para o projeto, os tempos verbais possíveis são : presente, pretérito perfeito e pretérito imperfeito. O tempo é determinado pela semântica do predicado no domínio. Por exemplo, quando se refere a um fato criminal ocorrido, o tempo é o pretérito perfeito. Para a definição de um tipo de delito usa-se o presente. O pretérito imperfeito sinaliza relações válidas antes da ocorrência do fato narrado.
Modo - indicativo, subjuntivo, gerúndio ou infinitivo. A morfologia deverá ser capaz de tratar toda possível combinação tempo/modo.
Número - singular ou plural. Refere-se, sempre, à 3a. pessoa, por imposição do domínio.
Afirm_neg - negativo ou afirmativo. Indica se o verbo deve, ou não, vir precedido pelo átomo "não".

predC Predicativo, Artigo)

Esta classe foi criada especialmente porque é grande o número de sentenças cujo verbo principal é o verbo de ligação "ser".

Predicativo - carrega o predicativo da sentença.
Artigo - valem os mesmos comentários do sujeito.

odC Objeto, Artigo, Anáfora)

Objeto - carrega o objeto direto do verbo da sentença.
Artigo - valem os comentários anteriores.

Anáfora - valem os comentários anteriores.

compl(Complemento, Artigo, Preposição, Anáfora)

Complemento - carrega o complemento preposicionado do verbo. Pode ser um objeto indireto.
Artigo - valem as considerações anteriores.
Preposição - carrega a preposição que acompanha o verbo e precede o complemento.
Anáfora - o complemento também pode estar anaforizado.

Ainda pode aparecer na lista um elemento do tipo :

coord(suj(...), suj(...))

que representa uma coordenação de sujeitos, ou seja, um caso de sujeito composto. Com isto, o verbo ganha o traco plural. "coord" avisa a morfologia que um conectivo deve ser colocado entre as duas realizações de sujeito.

Após as transformações, que serão vistas na próxima seção, um nó sintático pode ter sua ordem inicial alterada : podemos encontrar mais de um verbo, uma sentença pode ser iniciada por um complemento ou verbo, etc.

No apêndice 2, listamos a implementação do módulo correspondente a esta fase.

6. TRANSFORMAÇÕES SOBRE AS SENTENÇAS BÁSICAS

As transformações sobre as sentenças básicas, representadas por nós sintáticos, visam criar estruturas mais complexas do que sentenças simples, tais como sentenças relativas, coordenadas, passivizadas, uso de elipses e anáforas.

São dois os fatores que guiam tais transformações : o foco e o estilo.

6.1. REGRAS DE FOCO

As regras de foco visam ressaltar a informação mais importante. O foco, como tivemos oportunidade de mostrar, é fornecido pelo componente estratégico.

Os mecanismos mais usuais em Português, para este fim, são : (a) aqueles que transferem a informação mais importante, na forma de seu constituinte, para a posição inicial da sentença : a topicalização e a passivização;
(b) "emolduração" através da expressão :
"verbo(ser) informação principal que ...".

A topicalização é o mecanismo que torna um complemento verbal preposicionado o foco da sentença.

Suj Verbo Compl ---*topicalização*---> Compl Suj Verbo

A passivização se dá quando queremos focalizar o objeto direto da sentença. Neste caso, ao contrário da topicalização, o objeto focalizado pode fazer papel de sujeito da sentença.

Suj Verbo Obj.Dir ---*passivização*---> Obj.Dir Parte-Verbal por Suj

onde Parte-Verbal é Verbo Ser conjugado como o principal + Verbo principal no particípio passado.

Quando é o sujeito que deve ser focalizado, pode-se usar um artifício a mais além daquele que o coloca na posição inicial. Mesmo porque, isso já é verdade neste caso. Este artifício é "emoldurar" o sujeito, por um lado com o verbo ser, conjugado segundo o verbo principal e concordando com o sujeito, por outro lado, com o pronome relativo "que".

Suj Verbo Compl ---*realce do sujeito*---> "Ser" Suj que Verbo Compl

No nosso sistema de perguntas e respostas, o escopo de sentenças que são passíveis de serem focalizadas é completamente determinado.

Depois que o planejador determina o foco da pergunta do usuário, esta informação guia todo o processo de seleção das informações que farão parte da resposta. No caso de resposta afirmativa, tudo o mais na resposta, além do "Sim", são informações complementares, guiadas pelo foco, mas não necessariamente envolvendo-o. No caso de resposta negativa, mais importante que as informações adicionais, é a parte da resposta que corrige os fatos errados da pergunta. Chamamos esta parte da resposta de Resposta Correta. É exatamente a Resposta Correta que merece atenção quanto à focalização.

O sistema sabe quando tem em mãos a sentença correspondente à Resposta Correta, através da pilha do contexto que mantém durante a primeira fase, aquela que analisa a estrutura hierárquica vinda do planejador. Assim, ele remete a porção da estrutura profunda correspondente à Resposta Correta a uma gramática especial de geração : aquela que leva em conta o foco e as regras de topicalização, passivização e emolduração.

Vejamos um exemplo de cada caso, considerando perguntas do usuário onde sublinhamos seu foco.

1) Perg. : Pedro matou Maria ?
   Resp. : Não. Foi José que matou Maria. (emolduração)

2) Perg. : Pedro matou Ana ?
   Resp. : Não. Joana foi morta por Pedro com dois tiros. (passiva)

3) Perg. : João matou Mário a facadas ?
   Resp. : Não. Com cinco tiros, João o matou. (topicalização)

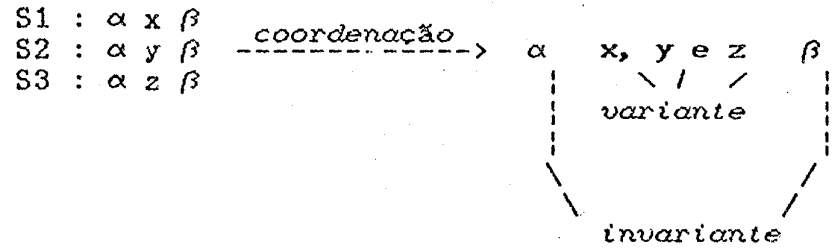
A resposta do último exemplo é, claramente, anti-natural. No entanto, se aplicarmos a emolduração sobre a topicalização obtemos uma alternativa sensivelmente melhor :

Não. Foi com cinco tiros que João o matou.

6.2. REGRAS DE ESTILO

As regras de estilo empregadas são : Coordenação, Elipse, Anaforização e Relativização.

A **coordenação aditiva** ou simplesmente **coordenação** de sentenças ocorre quando são identificadas uma parte variante e uma parte invariante entre elas.



O escopo da coordenação no nosso projeto é determinado em função das características, já discutidas, da estrutura retórica envolvida. Se considerarmos as sentenças candidatas oriundas de uma relação nuclear, ou seja, frutos de um núcleo e um satélite, percebemos dois casos : (a) se já existe uma conjunção subordinativa entre as duas, obviamente a coordenação não se aplica; (b) mesmo que não haja uma conjunção, sabemos que existe uma subordinação semântica entre o satélite e o núcleo e, portanto, poderia tornar-se obscura caso uma coordenação aditiva fosse aplicada, mesmo que as condições sintáticas fossem obedecidas. Uma outra razão para não interferir nessa correspondência é o fato de querermos medir a potencialidade da teoria RST como metodologia de geração.

Por outro lado, a relação multinuclear "sequência" possui características para uma coordenação sintática : esta relação é instanciada pelo planejador sempre que ele detecta um grupo de informações não dependentes entre si, ou seja, de igual importância. Logo, se as sentenças básicas relativas a estas informações obedecerem as restrições sintáticas acima, a coordenação é aplicada. A regra de coordenação é ativada sempre que uma relação "sequência" for detectada e, para ela, forem geradas as sentenças básicas.

Entretanto, numa lista-argumento de uma relação "sequência", nem sempre a ordem das sentenças correspondentes pode ser alterada sem que provoque uma alteração no significado a ser transmitido. Ora, isto parece uma contradição com a semântica definida anteriormente. E, de fato, é. Na realidade, trata-se de uma exceção : nosso sistema utiliza a base de conhecimento sobre Código Penal como programas Prolog, isto é, ao mesmo tempo em que as regras podem ser vistas como declarações de conhecimento, elas podem ser interpretadas pelo provador Prolog, afim de derivarem novos conhecimentos a partir de antigos. Devido a esta última característica, as regras devem ser escritas de modo que, durante sua interpretação, o fluxo de informações no corpo da regra se dê de maneira efetiva e eficiente. Isto requer, muitas vezes, uma ordem pré-estabelecida das condições, muito embora elas possam ser totalmente independentes entre si. Por outro lado, há momentos em que o sistema seleciona, para a resposta ao usuário, este caminho de dedução da nova informação. A relação trivial entre a cabeça e o corpo de uma cláusula Prolog é a de "condição" e, portanto ela é estabelecida pelo planejador. Finalmente, a relação conceitual entre as condições do corpo da cláusula - se mais de uma, é, sem dúvida a de "sequência". Temos aí o problema : neste caso específico, os elementos da sequência devem aparecer na ordem em que se encontram na estrutura. No máximo, podem ser combinados (coordenados) os que forem consecutivos. A escolha da "sequência ótima", do ponto de vista de melhor apresentação, é dependente de conhecimento do senso comum no domínio e não foi tratada por nós neste estágio de desenvolvimento.

Entretanto, há uma maneira simples de saber quando a sequência que se tem em mãos se enquadra neste caso e, portanto, deverá haver uma preocupação com a ordem de seus elementos : basta

consultar a pilha de contexto para saber se esta sequência é satélite de uma "condição"

Assim, desenvolvemos dois programas de coordenação : um, onde a coordenação é feita livremente, ou seja, basta que as restrições da regra de coordenação sejam obedecidas; o outro, obedecendo a ordem sequencial pré-estabelecida.

Antes de detalharmos os dois programas, é necessário salientar que optamos por coordenar sentenças duas a duas. Ou seja, o escopo da regra de coordenação será sempre duas sentenças.

(1) Coordenação Livre

Dada uma lista de sentenças na forma de nós sintáticos, a idéia pode ser resumida na execução dos seguintes passos :

(a) Tome como pivô o primeiro elemento da lista;
(b) Determine as partes invariantes deste com respeito a todos os demais elementos da lista;
(c) Escolha a sentença com o "melhor" invariante para ser coordenada com o pivô;
(d) Classifique o invariante segundo a matriz de casos (veja a seguir) e aplique a regra correspondente;
(e) Elimine da lista o pivô e o elemento selecionado;
(f) Repita este processo até que a lista fique com um único elemento ou vazia.

Há dois novos elementos introduzidos pelo método acima : o "melhor invariante" e a matriz de casos.

Definimos como "melhor", um invariante que possua mais elementos em relação a outro. Isto significa priorizar a coordenação entre sentenças com maior número de elementos comuns.
Mas ainda resta o caso, muito frequente por sinal, em que este número coincide e, ainda assim, uma escolha deve ser feita.

Uma rápida análise dos elementos sintáticos que podem fazer parte de um invariante, nos levou a uma escala de prioridades entre eles. Esta lista está em ordem decrescente de prioridade que aqui, significa uma melhor qualidade na coordenação (vide respectivos exemplos).

(1) Sujeito. Ex. João matou Maria e foi preso.
(2) Predicativo. Ex. José e João são suspeitos.
(3) Complementos (mesma preposição). Ex. O autor foi João e o motivo foi ciúmes, no assassinato de Maria.
(4) Objeto direto. Ex. João assaltou e José matou Maria.
(5) Verbo. Ex. João matou Maria e José, Pedro.

Esta análise baseia-se na qualidade da coordenação quando só aquele elemento é invariante. Mas ela também é usada no desempate quando a intersecção for mais de um elemento.

A matriz de casos possíveis, análoga àquela utilizada pelo sistema GEMA [Scott & Souza-88], porém estendendo o número de

elementos a serem comparados, isola todas as possibilidades de combinação, facilitando a determinação da regra a ser aplicada.

	Sujeito	Verbo	Predicativo	Obj.Direto	Compl
Idênticos	A	C	E	G	I
Não Idênticos	B	D	F	H	J

fig.3. Matriz de casos possíveis

Esta matriz é usada da seguinte forma : dadas duas sentenças candidatas, compara-se cada par de elemento com mesmo papel sintático. O fato de serem iguais ou não, os situa numa certa região da matriz. Tomando os rótulos de todas as regiões classificadas, temos o rótulo da regra que deve ser aplicada. Pois então, vejamos :

CASO	REGRA
ACE/GJ	Elipse do suj., verbo e ob.dir. ou predic. na 2a. sent.
ACF/HJ	Elipse do sujeito e verbo da 2a. sentença.
ACF/HI	Elipse do compl. da 1a. e do sujeito e verbo da 2a.
ADGI	Elipse do ob.dir. e compl da 1a. e sujeito da 2a.
ADHI	Elipse do compl. da 1a. e do sujeito da 2a. sentença.
ADGJ	Elipse do sujeito da 2a. e anáfora do ob.dir. da 2a.
ADHJ	Elipse do sujeito da 2a. sentença.
ADFI	Não ocorre.
ADFJ	Não ocorre.
BCE/GI	Elipse do verbo, pred. ou ob.dir. e compl. da 1a. Alteração do número do verbo da 2a.
BCE/GJ	Elipse do verbo e pred. ou ob.dir. da 2a. sentença.
BCF/HI	Elipse do complemento da 1a. sentença.
BCF/HJ	Nada a fazer.
BDFI	Não ocorre.
BDFJ	Não ocorre.
BDGI	Elipse do ob.dir. e compl. da 1a. sentença.
BDGJ	Anáfora do ob.dir. da 2a. sentença.
BDHI	Elipse do complemento da 1a. sentença.

Algumas considerações sobre as regras acima :

- A não ocorrencia do padrão $\alpha DF\beta$ deve-se ao fato de que dois predicativos, no Lettera, ocorrem necessariamente com verbos idênticos (verbo ser) e, portanto a região D não é acessada neste caso.

- Quando apenas os verbos coincidem, padrão BCF/HJ, nada é feito devido à baixa qualidade da coordenação neste caso. Ainda, quando ambos, verbo e complemento, coincidem com seus pares, padrão BCF/HI, apenas o complemento deixa de ser repetido.

- Repare a detecção de anáfora nos padrões ADGJ e BDGJ. Desta

maneira, o fenômeno da anáfora tornou-se , assim como a elipse, um subproduto da coordenação.

- Finalmente, esta detecção oportuna de fenômenos sintáticos é uma consequência bastante desejável do planejamento conciliatório [Scott & Souza-88], não implementado totalmente no Lettera, mas aproveitado na medida do possível.

A elipse é efetivada retirando-se do nó sintático o elemento indicado. Já a anáfora é indicada através da instanciação de um argumento para este fim, com o valor "a".

A coordenação entre duas sentenças, S1 e S2, será indicada por coord(S1, S2). Isto implicará na inclusão do conectivo "e" entre a última palavra de S1 e a primeira de S2.

(2) Coordenação Sequencial

A coordenação que leva em conta a posição das sentenças na lista-argumento é uma simplificação da coordenação livre acima : ao invés de escolher uma sentença, dentre todas as da lista, para coordenar com o pivô, verificamos o invariante deste em relação à sentença contígua. Se não for vazio, classificamos segundo a matriz de casos possíveis e aplicamos a respectiva regra. Eliminamos da lista o pivô e a sentença contígua, se houve coordenação, e repitimos o processo até que a lista fique com um ou nenhum elemento.

O fenômeno da **relativização do sujeito** ocorre entre duas sentenças contíguas, S1 e S2, se as seguintes pré-condições forem obedecidas :
- o último elemento de S1 (pode ser objeto direto ou complemento) e o sujeito de S2 coincidem.

Neste caso, faz-se elipse do sujeito de S2 e acrescenta-se o pronome relativo "que" entre S1 e S2.

S1: α x
S2: x β ---*relativização*---> α x que β

Novamente, o escopo de uma relativização será uma relação "sequência". Isto porque, já vimos que em outros contextos sempre haverá ou uma conjunção subordinativa ou uma razão semântica para não fazê-lo. É o caso da relação "união" : nesta relação não-nuclear, não há conjunção entre seus dois argumentos. Em compensação, ela relaciona, no domínio, duas porções da resposta que são conceitualmente independentes entre si : é a resposta propriamente dita e a complementação das informações que o usuário tem sobre aquele fato específico. Sinalizamos esta separação semântica por meio de um ponto final entre as sentenças correspondentes.

A regra de coordenação é aplicada anteriomente à

relativização. Logo, teremos como possível elemento, dentre os candidatos a uma relativa, a coordenação de duas sentenças. Sob o risco de sobrecarregar, cognitivamente, o texto com uma relativa anexada a uma coordenação, não tomaremos como sentenças-candidatas aquelas sob "coord".

Assim, para uma lista com duas sentenças ou mais :

(a) Tome como pivô a primeira sentença simples (não-coordenada);
(b) Se sua contígua também for simples, verifique se o sujeito desta coincide com o último elemento do pivô;
(c) Se sucesso, faça elipse do sujeito coincidente e adicione o pronome relativo "que" entre ambas;
(d) Elimine da lista o pivô e sua sentença contígua, se houve relativização;
(e) Repita o processo até que a lista fique com um ou zero elementos ou que só possua elementos do tipo "coord".

Na prática, quando ocorre uma relativização, os dois nós sintáticos envolvidos dão origem a um único nó sintático que contém todos elementos do nó correspondente ao pivô, mais o pronome "que", mais os elementos do segundo nó sintático, com exceção do sujeito.

A implementação das transformações aqui expostas encontra-se no apêndice 3. Exemplos ilustrativos aparecem na seção 8.

7. O COMPONENTE MORFOLÓGICO

A ênfase maior na implementação de Lettera foi nas três fases expostas acima. Não nos dedicamos, pelo menos por ora, a um projeto do componente morfológico com o cuidado que lhe é devido. Entretanto, como queríamos ver os resultados obtidos, construímos um último passo que chamamos de Morfologia, mas que é na realidade, um componente morfológico muito diferente daquilo que se denomina Morfologia.

Na realidade, nosso último passo é um grande programa recursivo que analisa cada elemento nuclear da lista de entrada e aplica regras apropriadas com o fim de fazer corresponder a ele uma palavra ou cadeia de palavras, que é, automaticamente, concatenada ao texto final.

É entrada para nosso componente morfológico uma lista linear contendo nós sintáticos, símbolos de pontuação e conjunções. Internamente aos nós, encontramos elementos sintáticos como sujeito, verbo, objeto direto, pronomes, etc. que carregam informações das mais variadas. Por exemplo, o verbo carrega traços que devem reger a flexão; o sujeito pode indicar uma anáfora, além do uso de um artigo, etc. Estas informações alimentam uma série de regras morfológicas que procuraremos mostrar a seguir.

Dentre as funções que atribuímos ao nosso componente morfológico, temos :

- Efetivação de coordenações.
- Determinação de artigos e pronomes.
- Contração de preposição + artigo.
- Conjugação de verbos.

(1) Efetivação de coordenações

Há dois tipos de coordenação neste nível : aquela entre duas sentenças sob "coord", e aquela entre uma lista de sentenças, simples ou não, que fazem parte de uma "sequência".

No primeiro caso, já vimos que um conectivo "e" deve ser inserido entre as duas. No segundo caso, mais de duas sentenças podem fazer parte da lista, sendo necessário portanto, sua separação por meio de vírgulas, com exceção da última, antecedida por um "e".

(2) Determinação de artigos e pronomes

Elementos como sujeito, predicativo, objeto direto e complemento possuem um argumento que indica a presença ou não de um artigo antes de seu núcleo, e o tipo deste artigo, se presente: definido, indefinido ou indeterminado.

A definição dos artigos definido (o,a) ou indefinido (um,uma) necessita da informação do gênero (feminino, masculino) e do número (singular, plural) do núcleo do sujeito, objeto ou

complemento. Isto é obtido através da consulta a um dicionário do domínio, usando como chave o valor do argumento deste núcleo.

O dicionário, muito simples, tem a entrada com a seguinte forma :

dic(Chave, Categoria, Gênero, Número)

onde a categoria pode ser nome, adjetivo, pronome, etc.

Quando não é possível em tempo de geração de sentença básica, decidir a existência e/ou o tipo do artigo, cabe ao componente morfológico obter subsídios para decidi-lo. Infelizmente, não implementamos esta tarefa. Entretanto, percebemos que seria necessário um conhecimento do texto já gerado para esta decisão.

Também a determinação de pronomes, quando um sinal de anáfora é encontrado nos elementos sujeito, objeto direto ou complemento, é feita após a consulta ao dicionário do gênero e número do núcleo do elemento.

(3) Contração de preposição + artigo

Estas regras de contração são necessárias quando tanto a preposição quanto um artigo estiverem presentes junto ao elemento complemento. Elas aparecem de forma particularizada porque a preposição, mesmo que composta, aparece como valor único de uma variável do programa. Nossas regras excluem as preposições que não ocorrem em nossa aplicação :

(a) preposição "de" + artigo definido : contrai "d+artigo"
(b) preposição "de" + artigo indefinido: concatena "de artigo"
(c) preposição "por" + artigo definido : contrai "pel+artigo"
(d) preposição "por" + art. indefinido : concatena "por artigo"
(e) preposição "antes de" + definido : contrai "antes d+artigo"
(f) preposição "antes de" + indefinido : concatena "antes de art"
(g) preposição "depois de" + definido : contrai "depois d+artigo"
(h) preposição "depois de" + indefinido: concatena "depois de art"

(4) Flexão de verbos

Não há propriamente regras de flexão de verbos em nosso componente, mas sim, tabelas de flexão dos verbos encontrados no domínio com algumas regras de formação de verbos compostos :

Seja "Verbo-principal" o verbo no infinitivo.

(a) O gerúndio composto é
"tendo" + particípio passado de Verbo-principal
(b) O infinitivo composto é
"ter (traço número)" + particípio passado de Verbo-principal
(c) O pretérito perfeito do subjuntivo é
presente subjuntivo de Ter + particípio passado do Verbo_principal
(d) A negação de uma sentença se dá através da negação de seu verbo :
"não" + Verbo-principal flexionado

Todos os demais casos são obtidos diretamente em tabelas.

O apêndice 4 mostra o módulo do componente morfológico e o dicionário.

8. EXEMPLOS

Mostraremos agora alguns exemplos que ilustram o desempenho de Lettera. Seguem-se as consideracões sobre os resultados.

Algumas convencões : P = Pergunta do usuário; R = Resposta do sistema; entre parênteses, colocamos as palavras ou sinais que gostaríamos de poder ter gerado, mas que o sistema ainda não está apto para tal; o foco da pergunta é sinalizado pelo grifo no elemento correspondente.

(1) P : O assassino de Ana é um estranho ?
R : Não. foi(Foi) João que matou Ana com três tiros por causa de infidelidade. (O) autor era marido de Ana.

Este é um caso onde ocorre uma excecão : as informacões de "modo" e "causa" são realizadas como adjuntos adverbiais e não como subordinadas. O foco da resposta, o verdadeiro autor do crime, está devidamente claro pelo uso da "moldura" foi...que. O uso de maiúsculas no início de uma nova sentenca do período ainda não está sendo considerado. Repare que Lettera não se decide por colocar um artigo quando a gramática não o fez.

(2) P : O crime de João é homicídio simples culposo ?
R : Não. o(O) crime é homicídio simples doloso, desde que a ação do crime foi homicídio e (o) autor assumiu o risco(,) enquanto que(,) embora a ação seja matar em ambos, eles diferem em: enquanto o autor assume o risco em homicídio doloso, a causa é por imprudência do autor e por negligência do autor em homicídio culposo.

Repare, neste exemplo, a variação dos tempos verbais : presente do indicativo para classificacão do crime; pretérito perfeito para fatos do crime; emprego do subjuntivo devido ao uso da conjuncão "embora". Temos os dois tipos de coordenação : entre duas sentencas que compartilham apenas um sequenciamento (a ação do crime...e (o) autor...risco), e pela aplicacão da matriz de casos possíveis onde ocorre a elipse do complemento da primeira sentenca e do sujeito e do verbo da segunda sentenca (a causa é ... e por negligência....culposo). A repeticão de "do autor" deve-se ao fato de que esta expressão está ligada ao predicativo na representação interna do conhecimento.

(3) P : Pedro foi morto no Leme, dia 22 ?
R : Não. (Um) desconhecido feriu Pedro em(no) Leme(,) no dia 21(,) com facadas. Pedro foi ferido pelo autor (,) que fugiu.

Temos, aqui, o uso de uma relativa (que fugiu). A redundância da informacão de que Pedro foi ferido se deve ao fato de que, ao ter que informar sobre a situacão atual da vítima, não há uma verificação de que ela é consequência da ação já expressa pelo verbo. Conhecimento do domínio seria suficiente para tal.

(4) P : José é suspeito de ter ferido Pedro ?
R : Não. José foi testemunha do crime, porque José viu o crime(,) enquanto que José não foi (considerado) suspeito, se bem que o local do crime foi (o) Leme, o dia foi 21 e José esteve no Leme no dia 21.

Veja que a forma sentencial "natural" foi conservada para mostrar o foco sobre o papel de José no crime. Observe ainda a falta do uso de anáfora na segunda ocorrência de José na resposta. Isto se deve ao fato de que limitamos a previsão do fenômeno de anáfora como um subproduto da coordenação aditiva de duas sentenças, e as sentenças "José foi testemunha do crime" e "José viu o crime" não são consideradas como canditadas, uma vez que se relacionam subordinativamente. Uma extensão para o tratamento generalizado de anáforas não seria problemática, muito embora muito esforço se faz necessário para contornar problemas de ambiguidade e decidimos não implementá-la por ora.

(5) P : Carlos matou Marcio ?
R : Sim, com um tiro(,) por causa de uma discussão. Carlos foi preso.

Novamente, temos o uso de adjuntos adverbiais no lugar de orações subordinadas. Repare que poderíamos usar o pronome "ele" na segunda sentença,no lugar de Carlos, sem causar ambiguidade, usando conhecimento de senso comum no domínio.

(6) P : Maria cometeu infanticídio ?
R : Não. o(O) crime é aborto resultante de estupro, uma vez que a ação do crime foi aborto, a(o) praticante do aborto foi (um) médico e o motivo foi (uma) gravidez resultante de estupro(,) ao passo que(,) embora a ação seja matar e (o) autor seja (a) mãe da vítima em ambos, eles diferem em: enquanto a vítima é embrião e a(o) praticante do aborto é (um) médico em aborto resultante de estupro, o motivo de aborto resultante de estupro é gravidez resultante de estupro e aborto resultante de estupro não é crime e ocorre antes do parto, a vítima é recem nascido em infanticídio(,) que ocorre depois do parto.

Aqui, notamos que, conforme a estrutura profunda gerada pelo planejador vai se tornando mais complexa (trata-se de uma estrutura hierárquica), mais problemático fica o controle dos limites entre as partes geradas independentemente. Ou seja, é crítico o problema de pontuação: a sentença ficou proibitivamente longa. Temos, neste exemplo, dois casos de coordenação e a aplicação de uma relativização. Compare esta solução com a do exemplo 2 e veja a variação das conjunções para uma estrutura idêntica.

(7) P : Maria cometeu aborto ?
R : Sim, por causa de (uma) gravidez resultante de estupro.

(8) P : Antônio matou Maria ?
R : Não. Joana foi morta por Antônio(,) por causa de ciúme. Antônio espancou (a) vítima e Ana ouviu-a gritar.

Aqui, detectamos três casos de anáforas: as duas ocorrências de Antônio poderiam ser substituídas por "ele", e "vítima" poderia ser substituída por "a" (embora, a não aplicação pareça mais adequada neste caso). Por outro lado, notamos seu uso no último trecho do período. Novamente, o argumento para isto é aquele apresentado no exemplo (4). Ainda assim, duas considerações devem ser feitas: a primeira ocorrência de Antônio teria que levar em consideração a sentença do usuário, o que é impossível nas condições atuais do sistema; a aplicação da anáfora também é sensível ao senso comum, ou seja, ainda que haja ambiguidade sintática, ela poderia ser resolvida via senso comum no domínio. Finalmente, repare que o foco sobre a vítima levou ao uso da voz passiva.

9. CONCLUSÕES

Foram mostradas, nas seções anteriores, as principais características - boas ou restritivas, e o real desempenho de Lettera como gerador de respostas planejadas como estruturas retóricas.

Acreditamos que a maioria das restrições apresentadas pelos exemplos é facilmente eliminada, uma vez que suas fontes foram também detectadas. Por exemplo, seria necessário o uso de conhecimento de senso comum no domínio, útil para a determinação de artigos e anáforas não ambíguas. Um controle maior sobre a "história da geração", ou seja, sobre o que foi gerado anteriormente - palavras e estrutura do período, sanaria os problemas dos limites das porções realizadas individualmente (núcleo e satélite) - emprego de maiúsculas e, sobretudo, pontuação, que são sensíveis ao tamanho do texto. No entanto, esta opção seria bastante onerosa do ponto de vista operacional. Talvez uma melhor alternativa fosse dispor, já a tempo de planejamento, de informações quanto à estrutura mais adequada do período, nos moldes do modelo conciliatório.

Por outro lado, a obediência quase estrita à teoria da correspondência entre estruturas retóricas e hipotaxe e parataxe, resultou em textos carregados de orações subordinativas. Apesar de considerarmos algumas exceções, uma influência ainda maior do componente linguístico parece altamente desejável.

A despreocupação com o componente morfológico influenciou os resultados, pois as palavras - substantivos e adjetivos, não recebem qualquer tratamento morfológico depois de acessadas no dicionário.

Aliás, uma grande lição foi aprendida : a independência entre os projetos (e não entre funções) dos vários módulos do gerador, se é atraente em tese, na prática mostrou-se, no mínimo, comprometedora.

Finalmente, o domínio utilizado parece ter contribuído para tornar os textos "de difícil digestão". Esperamos que a troca de domínio contribua na descoberta de outras qualidades de Lettera.

BIBLIOGRAFIA

[Appelt-83] Appelt, D.E., *Telegram : A Grammar Formalism for Language Planning*. Proceedings of the Eighth IJCAI Conference, Karlsruhe, 1983.

[Barros-85] Barros, E.M., *Nova Gramática da Língua Portuguesa*. Editora Atlas S.A. São Paulo, 1985.

[Grice-75] Grice, H.P., *Logic and Conversation*, in *Syntax and Semantics*. Vol.3. eds. P. Cole e J. Morgan, Academic Press, New York, 1975.

[Hobbs-79] Hobbs, J., *Coherence and Co-reference* in *Cognitive Science*, Vol.3., pp. 67-90. 1979.

[Hovy-88] Hovy, E.H., *Two Types of Planning in Language Generation*. Proceedings of the 26th Meeting of the ACL. Buffalo, New York. 1988.

[Mann & Thompson-86] Mann, W.C. & S.A. Thompson, *Rhetorical Structure Theory : Description and Construction of Text Structures*. Presented at the Third International Workshop on Text Generation. Nijmegen, The Netherlands. August. 1986.

[Matthiessen & Thompson-87] Matthiessen, C. & S.A. Thompson, *The Structure of Discourse and "Subordination"* in *Clause Combining in Discourse and Grammar*. eds. John Halman e Sandra A. Thompson. John Benjanins Publishing Company, 1987.

[McCoy-87] McCoy, K.F., *Highlighting a User Model to Respond to Misconceptions*. User Models in Dialogue System. eds. Alfred Kobsa e Wolfgang Walhster. Spring Verlag. 1987.

[McDonald & Pustejovsky-85] McDonald, D.D. & J.D. Pustejovsky, *Description-Directed Natural Language Generation*. Proceedings of the 9th. IJCAI. Vol.2. 1985.

[McKeown-82] McKeown, K., *Generating Natural Language Text in Response to Question about Database Strucutre*. Ph.D. dissertation, University of Pennsylvania, 1982.

[Nunes-88] Nunes, M.G.V., *Deep Generation in a Crime Knowledge-Based System*. Relatório Técnico. Dep. Informática, PUC-RJ. 1988.

[Scott & Souza-88] Scott, D.R. & C.S. Souza, *Conciliatory Planning for Extended Descriptive Texts*.

```
%
%                          APENDICE 1
%
%              DETERMINAÇÃO DA ESTRUTURA DO PERIODO
%
%
%       arquivo : comp_sint.ari
%       Programa que gera texto dado um esquema RST proveniente do
%       programa sist_cri.ari que os grava no arquivo interface.ari.
%       O programa processa_pergunta( Esquema, Arq.entrada, Arq.saida)
%       se dedica à primeira fase do componente linguistico, isto é,
%       à geração de sentenças básicas.



% programa inicio : carrega arquivos auxiliares e ativa a geracao de
% sentenças básicas.

inicio :-
        open(H1, 'interface.ari', r),   % arquivo com os esquemas de entrada
        open(H2, 'resultados.ari', w),  % arquivo com as estruturas resultantes
        write('Carregando dicionario....'), nl,
        consult('\ws\dicionario'),      % dicionario usado pelo componente morfologico
        write('Carregando base morfologica....'),nl,
        consult('\ws\base_morfol'),     % arquivo com regras verbais morfologicas
        write('Carregando base de predicados....'),nl,
        consult('\ws\base_pred'),       % arquivo com regras gramaticais de cada predicado (informacao)
        write('Carregando modelo do usuario....'),nl,
        consult(user_model),            % informacoes sobre o que o usuario sabe sao necessarias
        write('Carregando base de fatos....'), nl,
        consult('\ws\crimes_fa'),       % fatos criminais eventualmente sao consultados
        write('Carregando regras de transformacao....'), nl,
        consult('\ws\transf'),          % regras para coordenar sequencias.
        write('Lendo primeira entrada...'), nl,
        read(H1, Pergunta), write(H2, Pergunta), nl(H2),
        read(H1, Foco),
        assert(foco(Foco)),
        read(H1, Input),
        processa_pergunta( Input, H1, H2).      % ativa primeira fase do comp. linguistico


% programa processa_pergunta( Input, Arq.entrada, Arq.saida ) :
% Primeira fase do componente linguistico :
% determinação de orações principais e subordinadas, conjunções e geração de sentencas básicas
% Basicamente, cada predicado presente em núcleos e satélites de estruturas RST,
% gera uma sentenca básica. Exceções são feitas quando se tratam de predicados
% "do evento" no contexto de "Resposta Correta".

processa_pergunta( end_of_file, H1, H2 ) :-
        !,
        write('Encontrado fim de arquivo de entrada...Fim.'), nl,
        close(H1),
        close(H2).

processa_pergunta(Input, H1, H2) :-
```

```
        processa( Input, Result ),              % Determina estrutura, gera sentencas basicas e as transforma.
        formatar( Result, $$, Resposta ),       % Chama componente morfologico.
        concat( Resposta, $.$, Resp ),
        write('Gravando resultado...'), nl,
        write(H2, Result), nl(H2),
        write(H2, Resp), nl(H2),nl(H2),
        write('Resposta :  '), nl, write(Resp), nl,
        retract(foco(_)),
        write('Lendo nova entrada...'), nl,
        read(H1, Pergunta), write(H2, Pergunta), nl(H2),
        read(H1, Foco),
        assert(foco(Foco)),
        read(H1, Input1),
        processa_pergunta(Input1, H1, H2).

% processa( Input, Output ) :
% Consiste na 1a. fase : Geração e Transformação de Sentenças Básicas

processa( Input, Result ) :-
        atom( Input ), !,               % Se o esquema e' simplesmente "sim" ou "nao"
        regra(esquema, Input, Result).

processa( Input, Result ) :-
        Input =.. Lista,
        regra(esquema, Lista, Result).  % O esquema torna-se uma lista

% programa regra(esquema, Input, Result) :
% Encaminha as regras especificas da relação que encabeça cada esquema.
% Esta relação pode ser união, antítese ou evidência

regra(esquema, 'Sim', ['Sim'] ) :- !.

regra(esquema, 'Nao', ['Nao'] ) :- !.

regra(esquema, [uniao,El1,El2], Result ) :- !,
        regra(uniao, El1, El2, Result ).

regra(esquema, [antitese, nucleo(N), satel(S)], Result) :- !,
        regra(antitese, N, S, indicativo, Result).      % Introduzir traco verbal : indicativo

regra(esquema, [evidencia, nucleo(N), satel(S)], Result) :- !,
        regra(evidencia, N, S, indicativo, Result).

% programa regra(uniao, Arg1, Agr2, Result) :
% Uniao e' não-nuclear : Arg1 e Arg2 são retoricamente independentes. Arg2 e' sempre
% uma relação sequencia e Arg1 pode ser atomico ou uma relação nuclear. Sao separados por "."

regra(uniao, 'Sim', El2, ['Sim', '.'|Result] ) :- !,
        El2 =.. [sequencia,Lista],
        push(no_pai, (uniao,1)),
        regra(sequencia, Lista, indicativo, Result),
        pop(no_pai).

regra(uniao, El1, El2, Final ) :-
        El1 =.. [Rel, nucleo(N), satel(S)],
```

```
        push(no_pai, (uniao,2)),
        regra(Rel, N, S, indicativo, Result1),
        El2 =.. [sequencia, Lista],
        regra(sequencia, Lista, indicativo, Result2),
        concatena( [Result1, ['.'],Result2], Final ),
        pop(no_pai).

% programa regra(antitese, Nucleo, Satelite, Traco, Result) :
% As regras abaixo cuidam das instancias possiveis de antitese.
% "Traco" contem informacao sobre o modo verbal, uma vez que certas
% conjuncoes podem altera'-los.

regra(antitese, 'Nao', [], _, ['Nao', '.'] ) :- !.

regra(antitese, 'Nao', Satel, Traco, ['Nao', '.'|Result] ) :- !,       % Satel contem Resp.Correta
        analisa(Satel, (antitese,2), Traco, Result),
        pop(no_pai).

regra(antitese, 'Sim', S, Traco, [ 'Sim', Expr|Result2] ) :-
        S =.. [concessao, nucleo(N1), satel(S1)],              % Satel e' uma relação concessao no contexto
        push(no_pai, (antitese,3)),                            % de uma misconception.
        regra(concessao, N1, S1, Traco, Result2),
        conjuncao(antitese, Expr),                             % Escolhe arbitrariamente uma conjuncao de antitese
        pop(no_pai).

regra(antitese, 'Sim', sequencia(Lista), Traco, ['Sim', Expr|Result] ) :- !,
        push(no_pai, (antitese,4)),                            % Satel e' uma relação sequencia
        regra(sequencia, Lista, Traco, Result),
        conjuncao(antitese, Expr),
        pop(no_pai).

regra(antitese, N, sequencia(L), Traco, Final ) :-
        N =.. [evidencia, nucleo(N1), satel(S1)], !,    % Nucleo e'a evidencia para uma negativa
        push(no_pai, (antitese,5)),                     % Satel e'sequencia de justificativas para misconception.
        conjuncao(antitese, Expr),
        regra(evidencia, N1, S1, Traco, Result1),
        regra(sequencia, L, Traco, Result2),
        concatena( [Result1, [Expr], Result2], Final ),
        pop(no_pai).

regra(antitese, N, S, Traco, Final ) :-                 % Nucleo e' evidencia e Satel e' concessao.
        N =.. [evidencia, nucleo(N1), satel(S1)],
        S =.. [concessao, nucleo(N2), satel(S2)], !,
        push(no_pai, (antitese,6)),
        conjuncao(antitese, Expr),
        regra(evidencia, N1, S1, Traco, Result1),
        regra(concessao, N2, S2, Traco, Result2),
        concatena( [Result1, [Expr], Result2], Final ),
        pop(no_pai).

regra(antitese, N, sequencia(L), Traco, Final ) :-
        N =.. [antitese, nucleo(N1), satel(S1)], !,
        push(no_pai, (antitese,7)),
        regra(antitese, N1, S1, Traco, Result1),
        conjuncao(antitese, Expr),
```

```
        regra(sequencia, L, Traco, Result2),
        concatena( [Result1, [Expr], Result2], Final ),
        pop(no_pai).

regra(antitese, sequencia(L1), sequencia(L2),Traco, Final ):-
        push(no_pai, (antitese,8)),                    % Esta relação se da' entre as diferencas
        regra(sequencia, L1, Traco, Result1),          % entre dois tipos de crime, ou seja, L1 e L2
        regra(sequencia, L2, Traco, Result2),          % sao listas de caracterisiticas de 2 crimes distintos
        concatena( [[conj(enquanto)], Result1,[',',],Result2], Final),
        pop(no_pai).

regra(antitese, Pred1, [], Traco, Result1 ) :- !,      % No contexto de "diferencas",
        analisa_pred(Pred1, (antitese,9), Traco, af, Result1),  % o núcleo pode ser vazio.
        pop(no_pai).

regra(antitese, Pred1, Pred2, Traco, Final ) :-        % Nucleo e Satel sao predicados do dominio.
        analisa_pred(Pred1, (antitese,10), Traco, af, Result1),
        analisa_pred(Pred2, (antitese,10), Traco, af, Result2),
        concatena( [[conj(enquanto)],Result1,[',',],Result2], Final ),
        pop(no_pai), pop(no_pai).

% programa regra(evidencia, Nucleo, Satel, Traco, Result) :

regra(evidencia, 'Sim', S, Traco, ['Sim', '.'|Result2] ) :-
        S =.. [condicao, nucleo(N1), satel(S1)], !,    % Satelite e' relação condicao.
        push(no_pai, (evidencia,1)),
        regra(Rel, N1, S1, Traco, Result2),
        pop(no_pai).

regra(evidencia, 'Sim', [X|Y], Traco, ['Sim', ','|[Lista_adj]] ) :- !,  % Satel e'[modo e/ou motivo] ; indica que tais
        forma_adjuntos([X|Y], Lista_adj).              % evidencias devem adjuntos adverbias do verbo principal
                                                       % da pergunta do usuario.

regra(evidencia, 'Sim', Predicado, Traco, ['Sim', '.'|Result2] ) :-     % A evidencia 'a afirmacao
        analisa_pred(Predicado, (evidencia,2), Traco, af, Result2),     % e' um predicado do dominio.
        pop(no_pai).

regra(evidencia, N, sequencia(L), Traco, Final ) :-    % Nucleo e' a negação da resposta
        N =.. [antitese, nucleo(N1), satel(S1)],       % A sequencia-satélite mostra as condicoes que
        pertence(condicao(_,_), L), !,                 % evidenciam a Resposta Correta.
        push(no_pai, (evidencia,3)),
        regra(antitese, N1, S1, Traco, R1),
        regra(sequencia, L, Traco, R2),
        concatena( [R1,[',',],R2], Final ),
        pop(no_pai).

regra(evidencia, N, sequencia(L), Traco, ['Nao', '.'|Result] ) :-        % L = [modo/motivo]
        N =.. [antitese, nucleo(N1), satel(S1)],
        (pertence(modo(_,_), L) ; pertence(motivo(_,_), L) ), !,
        push(no_pai, (evidencia,4)),
        regra(antitese, N1, S1, Traco, R1),
        regra(sequencia, L, Traco, Lista_adj),         % Satelite contem informacoes que devem
        anexar(Lista_adj, R1, Result),                 % aparecer como adjuntos adverbiais do verbo do núcleo.
        pop(no_pai).
```

```
regra(evidencia, N, sequencia(L), Traco, Final ) :-
        N =.. [antitese, nucleo(N1),satel(S1)],
        push(no_pai, (evidencia,S)),
        regra(antitese, N1, S1, Traco, R1),
        regra(sequencia, L, Traco, Lista),
        concatena( [R1,[','],Lista], Final),
        pop(no_pai).


% programa regra(concessao, Nucleo, Satel, Traco, Resultado) :
% Ativa meta-regras que consideram o contexto em que a relação concessao aparece
% para determinar sua forma sintatica.

regra(concessao, Nucleo, Satel, Traco, Result) :-
        meta_regra(concessao, Nucleo, Satel, Traco, Result).

% programa meta_regra(concessao, Nucleo, Satel, Traco, Resultado) :
% faz a escolha entre as varias formas sintaticas desta relação
% baseado no núcleo e satélite e no contexto onde esta relação ocorre.

meta_regra(concessao, diferencas(S1), semelhancas(S2), T, R ) :- !,
        regra(c7, diferencas(S1), semelhancas(S2), T, R).

meta_regra(concessao, Predicado, Sequencia, Traco, Resultado) :-
        no_pai([(antitese,2)|R]), !,
        escolha(concessao, [c1,c2,c3,c5], Ci),                  % Escolhe qualquer das regras c1,c2,c3,c4.
        regra(Ci, Predicado, Sequencia, Traco, Resultado).

meta_regra(concessao, Predicado, Sequencia, Traco, Resultado) :-
        escolha(concessao, [c1,c2,c3,c4,c5,c6], Ci),
        regra(Ci, Predicado, Sequencia, Traco, Resultado).

regra(c1, Pred, Seq, Traco1, Final ) :-
        analisa_pred(Pred, (concessao,1), Traco1, af, Result1),
        Seq =.. [sequencia, Lista],
        regra(sequencia, Lista, indicativo, Result2),
        concatena( [Result1,[conj('se bem que')],Result2], Final ),
        pop(no_pai).

regra(c2, Pred, Seq, Traco1, Final ) :-
        conjuncao(concessao1, Conj),
        analisa_pred(Pred, (concessao,2), Traco1, af, Result1),
        Seq =.. [sequencia, Lista],
        regra(sequencia, Lista, subjuntivo, Result2),   % As conjuncoes do conj. concessao1 pedem subjuntivo a seguir
        concatena( [Result1,[Conj],Result2], Final ),
        pop(no_pai).

regra(c3, Pred, Seq, Traco1, Final ) :-
        conjuncao(concessao2, Conj),
        (Conj = mesmo, Traco = gerundio; Traco = infinitivo),   % "mesmo" pede gerundio; "apesar de" pede infinitivo
        analisa_pred(Pred, (concessao,3), Traco1, af, Result1),
        S =.. [sequencia, Lista],
        regra(sequencia, Lista, Traco, Result2),
        concatena( [Result1,[Conj],Result2], Final ),
        pop(no_pai).
```

```
regra(c4, Pred, Seq, Traco1, [Conj|Final]  ) :-
        conjuncao(concessao1, Conj),
        analisa_pred(Pred, (concessao,4), indicativo, af, Result1),
        S =.. [sequencia, Lista],
        regra(sequencia, Lista, subjuntivo, Result2),
        concatena( [Result2,[','],Result1], Final ),
        pop(no_pai).

regra(c5, Pred, Seq, Traco1, [Conj|Final] ) :-
        conjuncao(concessao2, Conj),
        (Conj = mesmo, Traco = gerundio; Traco = infinitivo),
        analisa_pred(Pred, (concessao,5), indicativo, af, Result1),
        S =.. [sequencia, Lista],
        regra(sequencia, Lista, Traco, Result2),
        concatena( [Result2, [','],Result1], Final ),
        pop(no_pai).

regra(c6, Pred, Seq, Traco1, Final ) :-
        analisa_pred(Pred, (concessao,6), indicativo, af, Result1),
        Seq =.. [sequencia, Lista],
        regra(sequencia, Lista, Traco1, Result2),
        concatena( [Result2,[conj('no entanto')],Result1], Final ),
        pop(no_pai).

regra(c7, diferencas(S1), semelhancas(S2), T, [conj(embora)|Final] ) :-
        S1 =.. [sequencia, L1],
        S2 =.. [sequencia, L2],
        push(no_pai, (concessao,7)),
        regra(sequencia, L1, T, R1),
        regra(sequencia, L2, subjuntivo, R2),
        regra(diferencas, R1, T, af, R11),
        regra(semelhancas, R2, T, af, R22),
        concatena( [[R22],[','],[R11]], Final ),
        pop(no_pai).

% programa regra(condicao, Nucleo, Satelite, Resultado) :
% Novamente, usa meta_regras para escolher sintaxes diferentes

regra(condicao, N, S, Traco, Result) :-
        meta_regra(condicao, N, S, Traco, Result).

meta_regra(condicao, N, S, Traco, Result) :-
        no_pai([(sequencia,_)|Resto]), !,       % Se faz parte de uma sequencia, aplicar regra cond3.
        regra(cond3, N, S, Traco, Result).

meta_regra(condicao, N, S, Traco, Result) :-
        escolha(condicao, [cond1,cond2], Cond1),        % Escolha regra cond1 ou cond2.
        regra(Cond1, N, S, Traco, Result).


regra(cond1, Pred, Seq, Traco, Final ) :-
        conjuncao(condicao_c, Conj),
        analisa_pred(Pred, (condicao,1), Traco, af, Result1),
        Seq =.. [sequencia, Lista],
```

```
        regra(sequencia, Lista, indicativo, Result2),
        concatena( [Result1,[Conj],Result2], Final ),
        pop(no_pai).

regra(cond2, Pred, Seq, Traco, Final ) :-
        analisa_pred(Pred, (condicao,2), indicativo, af, Result1),
        Seq =.. [sequencia, Lista],
        regra(sequencia, Lista, gerundio, Result2),
        concatena( [Result2,[Conj],Result2], Final ),
        pop(no_pai).

regra(cond3, Pred, sequencia(Lista), Traco, [Conj1|Result2] ) :-       % Nucleo ja' apareceu anteriormente
        conjuncao(condicao_c, Conj1),                                  % e nao deve ser repetido.
        regra(sequencia, Lista, indicativo, Result2).

% programa regra(resultado, Nucleo, Satelite, Traco, Resultado) :

regra(resultado, N, S, Traco, Final ) :-
        N =.. [sequencia, Lista],              % O núcleo e' sempre uma sequencia
        conjuncao(resultado, Conj),            % O satélite e' sempre um predicado
        analisa_pred(S, (resultado,1), Traco, af, Result2),
        regra(sequencia, Lista, Traco, Result1),
        concatena( [Result1,[Conj],Result2], Final ),
        pop(no_pai).

% programa regra(razao, Nucleo, Satelite, Traco, Resultado) :

regra(razao, N, S, Traco, Final ) :-
        N =.. [sequencia, Lista],
        conjuncao(razao_conj, Conj),
        analisa_pred(S, (razao,1), indicativo, af, Result2),
        regra(sequencia, Lista, Traco, Result1),
        concatena( [Result1, [Conj],Result2], Final ),
        pop(no_pai).

% programa regra(sequencia, Lista_argumento, Traco, Resultado) :

regra(sequencia, [], _, []) :- !.

regra(sequencia, [Elem], Traco, [Result] ):-
        Elem =.. [Rel, nucleo(N), satel(S)], !,        % Se o unico elemento e' uma relação.
        push(no_pai, (sequencia,1)),
        regra(Rel, N, S, Traco, Result),               % Rel = concessao,condicao,ou antitese.
        pop(no_pai).

regra(sequencia, [X|Y], Traco, Lista_adj ) :-
        no_pai([(evidencia,4)|R]),!,                   % E' uma lista com modo/motivo
        forma_adjuntos([X|Y], Lista_adj).

regra(sequencia, [Pred], Traco, [Result] ) :-          % Se unico predicado da lista nao e' do evento
        (not pertence(Pred, [autor(_,_),acao(_,_),vitima(_,_),local(_,_),dia(_,_)]);
         not no_pai([(antitese,2)|R]) ),
        analisa_pred(Pred, (sequencia,2), Traco, af, Result),
        pop(no_pai).
```

```
regra(sequencia, Lista, Traco, [Result] ) :-          % Neste caso o resultado e' uma lista
        not no_pai([(antitese,2)|R]), !,
        auxiliar( Lista, Traco, Lista_coord),         % de sentencas nucleares que devem ser coordenadas
        coordenacao(Lista_coord, Lista_result),       % Aplica regras de coordenacao.
        relativa(Lista_result, Result).               % Aplica regras de relativizacao.

regra(sequencia, Lista, Traco, [Seq] ) :-
%       no_pai([(antitese,2)|R]),       Lista compoe a Resposta Correta da pergunta
        push(no_pai, (sequencia,4)),
        separa(Lista, [autor(_,_),acao(_,_),vitima(_,_),local(_,_),dia(_,_)], Pred_evento, Extra_evento),
        regra(composta, Pred_evento, Traco, af, R1),  % Forma uma unica sentenca com informacoes do evento
        regra(sequencia, Extra_evento, Traco, R2),    % e extra-evento.
        forma_seq_final(R1, R2, Seq),                 % Nao coordena nem relativiza pois nao ocorrem, no projeto,
        pop(no_pai).                                  % circunstancias favoraveis.

% programa forma_seq_final(Sentenca_composta, Sequencia, Sequencia_final) :

forma_seq_final([], sequencia(R2), R2) :- !.

forma_seq_final(R1, [], R1) :- !.

%forma_seq_final(R1, sequencia(coord(R2)), coord([R1|R2]) ) :- !.

forma_seq_final(R1, sequencia(R2), [R1, R2] ).

% programa separa(Lista_origem, Restricoes, Lista_evento, Lista_extra_evento) :
% Separa da Lista_origem as Listas evento e extra_evento, usando para a classificacao, uma lista de Restricoes.

separa([], L, [], []) :- !.

separa([E1|R], L, [E1|R1], R2) :-
        pertence(E1, L), !,
        separa(R, L, R1, R2).

separa([E1|R], L, R1, [E1|R2]) :-
        separa(R, L, R1, R2).

% programa auxiliar(Lista, Traco, Lista_coord) :
% Para cada elemento da Lista, gera sua sentenca basica, formando Lista_coord.

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

auxiliar([Elem|Resto], Traco, [R1|RR]) :-             % forma lista de sentencas nucleares.
        analisa_pred(Elem, (sequencia,3), Traco, af, R1),
        pop(no_pai),
        auxiliar(Resto, Traco, RR).


% Operacoes sobre a pilha no_pai que empilha os nos pais da arvore.
% Esta pilha tem a finalidade de fornecer o contexto atual da estrutura RST que esta' sendo analisada
% Inicialmente, a pilha e' vazia.

no_pai( [] ).                 % Pilha inicialmente vazia.

% push( Pilha, Elemento ) :
```

```
% Realiza a operacao de empilhamento de Elemento na Pilha.

push(no_pai, Elem) :-
        no_pai(Pilha),
        Pilha1 = [Elem|Pilha],
        retract(no_pai(Pilha)),
        assert(no_pai(Pilha1)).

% pop( Pilha ) :
% Realiza a operacao de desempilhar a Pilha.

pop(no_pai) :-
        no_pai([Topo|Pilha]),
        retract(no_pai(X)),
        assert(no_pai(Pilha)).

% analisa( Rel, Fonte, Result)
% Result e' a arvore sintatica gerada das relacoes Sequencia, Evidencia, Razao ou Resultado ou de um predicado.
% Fonte e' o par (no_pai, numero da regra) que sera' empilhado.

analisa(Relacao, Fonte, Traco, Result) :-
        Relacao =.. [sequencia, Lista], !,
        push(no_pai, Fonte),
        regra(sequencia, Lista, Traco, Result).

analisa(Relacao, Fonte, Traco, Result) :-
        Relacao =.. [Rel, nucleo(N), satel(S)],
        pertence(Rel, [evidencia, razao, resultado]), !,
        push(no_pai, Fonte),
        regra(Rel, N, S, Traco, Result).

analisa(Predicado, Fonte, Traco, Result) :-
        not pertence(Predicado, [antitese,concessao,uniao,condicao]),
        analisa_pred(Predicado, Fonte, Traco, af, Result).


% analisa_pred( Pred, Fonte, Traco_modo_verbal, Traco_af_neg, Result)
% Encaminha para regras de predicados com 2, 3 ou 1 argumento (por ordem de probabilidade).
% Traco e' o modo do verbo exigido para a sentenca nuclear gerada por Pred.

analisa_pred(Pred, Fonte, Traco, AN, Result) :-
        Pred =.. [Predic, A1, A2], !,          % Predicado com 2 argumentos
        push(no_pai, Fonte),
        regra(Predic, A1, A2, Traco, AN, Result).

analisa_pred( Pred, Fonte, Traco, AN, Result ) :-
        Pred =.. [Predic, A1, A2, A3], !,      % Predicado com  3 argumentos
        push(no_pai, Fonte),
        regra(Predic, A1, A2, A3, Traco, AN, Result).

analisa_pred(Pred, Fonte, Traco, AN, [Result]) :-
        Pred =.. [not, A1],!,                  % Predicado negado : inverter traco do verbo
        (AN = af, AN1 = neg; AN1 = af),
        analisa_pred(A1, Fonte, Traco, AN1, Result).
```

```
analisa_pred(Pred, Fonte, Traco, AN, [Result]) :-
        Pred =.. [Predic, A1],                  % Predicado com um argumento
        push(no_pai, Fonte),
        regra(Predic, A1, Traco, AN, Result).


% Conjuncoes relativas as relacoes :

conj(antitese, ['ao passo que', 'enquanto que']).

conj(evidencia, [pois, 'uma vez que', 'desde que']).

conj(concessao1, [embora, 'ainda que']).

conj(concessao2, [mesmo, 'apesar de']).

conj(condicao_c, ['uma vez que', 'desde que', porque]).

conj(resultado, ['em consequencia disso', 'como resultado']).

conj(razao_conj, [porque, pois, 'uma vez que', 'ja que', 'visto que']).


% conjuncao( Relacao, Conj_escolhida )
% Escolhe a conjuncao relativa 'a Relacao, que menos tenha ocorrido no texto.
% A frequencia de cada uma e' aumentada cada vez que ela e' escolhida.

conjuncao(Relacao, conj(Conj_escolhida)) :-
        conj(Relacao, Conj),                    % Conj e' o conjunto de conjuncoes para Relacao.
        menos_frequente(Conj, Conj_escolhida),  % Escolhe a menos frequente do conjunto.
        atualiza_frequencia(Conj_escolhida).    % Atualiza frequencia da conjuncao escolhida.


% escolha( Relacao, Formas_possiveis, Forma_escolhida )
% Escolhe a regra, de concessao ou condicao, menos executada dentre as possiveis.

escolha(Relacao, Formas_possiveis, Forma_escolhida) :-
        menos_frequente(Formas_possiveis, Forma_escolhida),
        atualiza_frequencia(Forma_escolhida).


% freq(Elemento, Frequencia).
% Inicialmente, a frequencia de todas conjuncoes e regras e' zero.

freq(Elemento, 0).

% atualiza_frequencia( Elemento )
% Incrementa a frequencia e insere no inicio da lista de frequencia da base.

atualiza_frequencia(Elem) :-
        freq(Elem, 0), !,
        asserta( (freq(Elem, 1) :- !) ).

atualiza_frequencia(Elem) :-
        freq(Elem, Freq),
```

```
        New_freq is Freq + 1,
        retract( freq(Elem, Freq) ),
        asserta( (freq(Elem, New_freq) :- !) ).

% menos_frequente( Lista_elementos, Elemento_menos_frequente)
% Obtem o Elemento_menos_frequente da Lista_elementos.

menos_frequente( [Elemento], Elemento ) :- !.

menos_frequente( [Elem|Resto], Elem1 ) :-
        menos_frequente(Resto, Elem1),
        freq(Elem1, F1),
        freq(Elem, F),
        F1 < F, !.

menos_frequente( [Elem|_], Elem ).

% obter( Info, Fonte, Resposta)
% Info e' autor ou vitima e o objetivo e' obter o autor ou a vitima do crime se o usuario
% ja' a conhece, caso contrario devolver o atomo "vitima" ou "autor".

obter(Info, Num, Valor) :-
        integer(Num),
        Rel =.. [Info, Num, Valor],
        sabe(user, Rel), !.

obter(Info, Arg, Info).

% pertence( Elemento, Lista_elementos )

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

pertence(X, [_|R]) :-
        pertence(X, R).

% anexar(Lista_adjuntos, Satel_antitese, Resultado)
% Anexa uma lista de adjuntos adverbiais 'a sentenca que e' o satélite de uma antitese, obtendo Resultado.

anexar( Lista_adj, ['Nao', '.', L], [L1] ) :- !,
        append( L, Lista_adj, L1).

anexar(Lista_adj, ['Nao', '.', [X1|Y1],X,Y], [L1,X,Y] ) :- !,
        append([X1|Y1], Lista_adj, L1).

anexar(Lista_adj, ['Nao', '.', [X|Y]], [L1] ) :-
        append( [X|Y], Lista_adj, L1 ).

% append( Lista1, Lista2, Lista_Result )

append( L, [], L ) :- !.

append( [], L, L ) :- !.

append( [E|R], L, [E|RL] ) :-
        append( R, L, RL ).
```

```
% concatena( Lista_de_listas, Lista_resultante )

concatena( [Elem], Elem ) :- !.

concatena( [Elem1,Elem2|R], F ) :-
        append( Elem1, Elem2, E ),
        concatena( [E|R], F ).
```

```
%
%                          APENDICE 2
%
%                  GERAÇAO DE SENTENÇAS BASICAS
%
%
%  arquivo : Base_pred.ari
%
%  Base de regras sintaticas para predicados nucleares do dominio.
%
%  regra( nome_predicado, arg1, arg2,..., Traco_modo_verbal, Traco_afirmativo_negativo, Estrutura_sintatica)
%
%  Traco_modo_verbal : indicativo, subjuntivo, infinitivo, gerundio
%  Traco_afirmativo_negativo : neg ou af, indicando que o verbo deve ou nao ser negado.
%
%  Estrutura_sintatica : [ SUJ, VERBO, PRED/OD, COMPL {,COMPL,..} ],  onde
%
%  SUJ e' suj( Nucleo_suj, def/indef/n, Anafora, Prep, Compl_suj ) :
%         def:artigo definido; indef:artigo indefinido; n:artigo inexistente; _ :indeterminado
%         Prep e' a preposicao do complemento do sujeito, Compl_suj, se presente.
%      ou coord([suj(...),suj(...)]), que e' a coordenacao de dois sujeitos.
%  VERBO e' verbo( Verbo, Tempo, Modo, Numero, Traco_afirm_neg ).
%  PRED/OD e' predic( Predicativo, Artigo ) ou od( Objeto, Artigo, Anafora ).
%  COMPL e' compl(Complemento, Artigo, Preposicao, Anafora ) e pode aparecer mais de uma vez na lista.



% predicado pena(Valor)

regra(pena, Valor, T_modo, AN, [ suj(pena, def,_,_, _), verbo(ser,pres,T_modo,sg,AN), predic(Valor,n)] ) :- !.

% predicado pena(Tipo_crime, Valor)

regra(pena, Tipo_crime, Valor, T_modo, AN, [ suj(pena,def,_,para, Tipo_crime), verbo(ser,pres,T_modo,sg,AN),
                                             predic( Valor,n) ] ) :- !.

% predicado rel_autor_vitima(Num/Tipo_crime, Rel)

regra(rel_autor_vitima, Num, Rel, T_modo, AN, [ suj(Autor,_,_,_, _), verbo(ser,imperf,T_modo,sg,AN),
                                             predic( Rel,_), compl(Vitima, _,de, _)] ) :-
       integer(Num), !,
       obter(autor, Num, Autor),
       obter(vitima, Num, Vitima).

regra(rel_autor_vitima, Tipo_crime, Rel, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                                    predic(Rel,n), compl(Vitima,_,de,_), compl(Tipo_crime,n,em,_)] ) :-!,
       obter(autor, Num, Autor),
       obter(vitima, Num, Vitima).

% predicado situacao_autor(Num, SA)

regra(situacao_autor, Num, morto, T_modo, AN, [suj(Autor,_,_,n,n), verbo(morrer,perf,T_modo,sg,AN) ] ) :- !,
       obter(autor, Num, Autor).

regra(situacao_autor, Num, foragido, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(fugir,perf,T_modo,sg,AN) ] ) :- !,
```

```
        obter(autor, Num, Autor).

regra(situacao_autor, Num, SA, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(ser,perf,T_modo,sg,AN), predic(SA,n) ] ) :-!,
        obter(autor, Num, Autor).

% predicado situacao_vitima(Num/Tipo_crime, SV)

regra(situacao_vitima, Num, morta, T_modo, AN, [ suj(Vitima,_,_,n,n), verbo(morrer,per,T_modo,sg,AN) ] ) :-
        integer(Num), !,
        obter(vitima, Num, Vitima).

regra(situacao_vitima, Num, viva, T_modo, AN, [ suj(Vitima,_,_,n,n), verbo(estar,pres,T_modo,sg,AN), predic(viva,n) ] ) :-
        integer(Num), !,
        obter(vitima, Num, Vitima).

regra(situacao_vitima, Num, SV, T_modo, AN, [ suj(Vitima,_,_,n,n), verbo(ser,perf,T_modo,sg,AN), predic(SV,n),
                                              compl(Autor,def,por,_) ] ) :-
        integer(Num), !,
        obter(vitima, Num, Vitima),
        obter(autor, Num, Autor).

regra(situacao_vitima, Tipo_crime, morta, T_modo, AN, [ suj(vitima,def,_,n,n), verbo(morrer,pres,T_modo,sg,AN),
                                                        compl(Tipo_crime,n,em,_) ] ) :- !.

regra(situacao_vitima, Tipo_crime, viva, T_modo, AN, [ suj(vitima,def,_,n,n), verbo(morrer,pres,T_modo,sg,ANN),
                                                        compl(Tipo_crime,n,em,_) ] ) :-!,
        (AN = af, ANN = neg ; ANN = af).

% predicado modo(Num, Modo)

regra(modo, Num, espancamento, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(espancar,perf,T_modo,sg,AN),
                                    od(Vitima,_,_) ] ) :- !,
        obter(autor, Num, Autor),
        obter(vitima, Num, Vitima).

regra(modo, Num, Modo, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(ferir,perf,T_modo,sg,AN),
                                    od(Vitima,_,_), compl(Modo,n,com,_) ] ) :-
        acao(Num, Ac),
        (pertence(Ac, [ferir, 'tentativa de homicidio']) ; situacao_vitima(Num, ferida)),!,
        obter(autor, Num, Autor),
        obter(vitima, Num, Vitima).

regra(modo, Num, Modo, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(abortar,perf,T_modo,sg,AN),
                                    compl(Modo,n,com,_) ] ) :-
        acao(Num, aborto), !,
        obter(autor, Num, Autor).

regra(modo, Num, Modo, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(matar,perf,T_modo,sg,AN),
                                    od(Vitima,_,_), compl(Modo,n,com,_) ] ) :- !,
        obter(autor, Num, Autor),
        obter(vitima, Num, Vitima).


% predicado modo_fuga(Num, Modo_fuga)
```

```
regra(modo_fuga, Num, Modo, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(fugir,perf,T_modo,sg,AN),
                                          compl(Modo,n,_,_) ] ) :- !,
        obter(autor, Num, Autor).

% predicado modo_acao(Num/Tipo_crime, Modo)

regra(modo_acao, Num, Modo, T_modo, AN, [ suj(acao,def,_,n,n), verbo(dar_se,perf,T_modo,sg,AN),
                                          compl(Modo,n,'atraves de', _) ] ) :-
        integer(Num), !.

regra(modo_acao, Tipo_crime, Modo, T_modo, AN, [ suj(acao,def,_,n,n), verbo(dar_se,pres,T_modo,sg,AN),
                                            compl(Modo,n,'atraves de',_), compl(Tipo_crime,em, _) ] ) :- !.

% predicado possui(Pessoa, Instrumento)

regra(possui, Pessoa, Inst, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(possuir,imperf,T_modo,sg,AN),
                                          od(Inst,indef,_) ] ) :- !.

% predicado ocorre(Tipo_crime, Valor)

regra(ocorre, Tipo_crime, Valor, T_modo, AN, [ suj(Tipo_crime,_,_,n,n), verbo(ocorrer,pres,T_modo,sg,AN),
                                            compl(parto, def,Valor,_) ] ) :- !.

% predicado coincide(Tipo_crime/ambos, A, V)

regra(coincide, Tipo_crime, A, V, T_modo, AN, [ coord([suj(A,n,_,n,n),suj(V,n,_,n,n)]), verbo(coincidir,pres,T_modo,pl,AN),
                                            compl(Tipo_crime, n, em,_) ] ) :- !.

% predicado decisao(Num/Tipo_crime, Des)

regra(decisao, Num, Dec, T_modo, AN, [ suj(terceiros,n,_,n,n), verbo(influenciar,perf,T_modo,sg,AN),
                                       od(decisao,def,_) ] ) :-
        integer(Num), !.

regra(decisao, Tipo_crime, Dec, T_modo, AN, [ suj(terceiros,n,_,n,n), verbo(influenciar,pres,T_modo,sg,AN),
                                         od(decisao,def,_), compl(Tipo_crime,n,em,_) ] ) :- !.

% predicado captura(Num, Modo)

regra(captura, Num, Modo, T_modo, AN, [ suj(policia,def,_,n,n), verbo(capturar,perf,t_modo,sg,AN), od(Autor,_,_),
                                       compl(Modo,n,_,_) ] ) :- !,
        obter(autor, Num, Autor).

% predicado tipo_crime(Num, Tipo)

regra(tipo_crime, Num, Tipo, T_modo, AN, [ suj(crime,def,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                         predic(Tipo,n) ] ) :- !.

% predicado testemunha(Num, Pessoa)

regra(testemunha, Num, Pessoa, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ser,perf,T_modo,sg,AN),
                                         predic(testemunha,n) ] ) :- !.

% predicado suspeito(Num, Pessoa)
```

```
regra(suspeito, Num, Pessoa, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ser,perf,T_modo,sg,AN),
                                           predic(suspeito,n) ] ) :- !.

% predicado motivo(Num/Tipo_crime, Mot)

regra(motivo, Num, Mot, T_modo, AN, [ suj(motivo,def,_,n,n), verbo(ser,perf,T_modo,sg,AN), predic(Mot,n) ] ) :-
       integer(Num), !.

regra(motivo, Tipo_crime, Mot, T_modo, AN, [ suj(motivo,def,_,de,Tipo_crime), verbo(ser,pres,T_modo,sg,AN),
                                           predic(Mot,n) ] ) :- !.

% predicado arrolado(Pessoa, Papel)

regra(arrolado, Pessoa, Papel, T, AN, [suj(Papel,def,_,do,crime),verbo(ser,perf,T,sg,AN),predic(Pessoa,n)] ) :-
       foco([arrolado,pessoa]), !.

regra(arrolado, Pessoa, Papel, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ser,perf,T_modo,sg,AN),
                                           predic(Papel,n), compl(crime,def,de,_) ] ) :- !.

% predicado tipo_vitima(Num,/Tipo_crime, Tipo)

regra(tipo_vitima, Num, Tipo, T_modo, AN, [ suj(vitima,def,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                          predic(Tipo,_) ] ) :-
       integer(Num), !.

regra(tipo_vitima, Tipo_crime, Tipo, T_modo, AN, [ suj(vitima,def,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                               predic(Tipo,_), compl(Tipo_crime,_,em,_) ] ) :- !.

% predicado praticante_aborto(Num/Tipo_crime, Pessoa)

regra(praticante_aborto, Num, Pessoa, T_modo, AN, [ suj('praticante do aborto',def,_,n,n), verbo(ser,perf,T_modo,sg,AN),
                                                predic(Pessoa,_) ] ) :-
       integer(Num), !.

regra(praticante_aborto, Tipo_crime, Pessoa, T_modo, AN, [ suj('praticante do aborto',def,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                                       predic(Pessoa,n), compl(Tipo_crime,n,em,_) ] ) :- !.

% predicado consentimento_mae(Num, Valor)

regra(consentimento_mae, Num, dado, T_modo, AN, [ suj(mae,def,_,n,n), verbo(dar,perf,T_modo,sg,AN), od(consentimento,n,_) ] ) :-



regra(consentimento_mae, Num, negado, T_modo, AN, [ suj(mae,def,_,n,n), verbo(dar,perf,T_modo,sg,ANN), od(consentimento,n,_) ] ) :-

       (AN = af, ANN = neg; ANN = af).

% predicado causa(Num/Tipo_crime, Causa)

regra(causa, Num, Causa, T_modo, AN, [ suj(causa,def,_,n,n), verbo(ser,perf,T_modo,sg,AN), compl(Causa,_,por,_) ]) :-
       integer(Num), !.

regra(causa, Tipo_crime, Causa, T_modo, AN, [ suj(causa,def,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                          compl(Causa,_,por,_), compl(Tipo_crime,n,em,_)] ) :- !.

% predicado risco_autor(Num/Tipo_crime, Val).
```

```
regra(risco_autor, Num, Val, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(assumir,perf,T_modo,sg,AN), od(risco,def,_)]) :-
        integer(Num), !,
        obter(autor, Num, Autor).

regra(risco_autor, Tipo_crime, Val, T_modo, AN, [ suj(autor,def,_,n,n), verbo(assumir,pres,T_modo,sg,AN), od(risco,def,_),
                                        compl(Tipo_crime,n,em,_) ]) :- !.

% predicado idade_autor(Num, Id)

regra(idade_autor, Num, Id, T_modo, AN, [ suj(Autor,_,_,n,n), verbo(ter,imperf,T_modo,sg,AN), od(Id,n,_),
                                        compl(anos,n,_,_) ] ) :- !,
        obter(autor, Num, Autor).

% predicado menor( Id, X )

regra(menor, Id, X, T_modo, AN, [ suj(Id,_,_,n,n), verbo(ser,pres,T_modo,sg,AN), predic(menor,n),
                          compl(X, n,que,_) ] ) :- !.

% predicado estado_mental_mae(Num, EM)

regra(estado_mental_mae, Num, EM, T_modo, AN, [ suj(mae,def,_,n,n), verbo(ser,pres,T_modo,sg,AN), predic(EM,n)] ) :- !.

% predicado meio(Tipo_crime, Valor).

regra(meio, Tipo_crime, Valor, T_modo, AN, [ suj(meio,def,_,n,n), verbo(ser,pres,T_modo,sg,AN), predic(Valor,n),
                                        compl(Tipo_crime,n,em,_)] ) :- !.

% predicado objetivo(Tipo_crime, Obj)

regra(objetivo, Tipo_crime, Obj, T_modo, AN, [ suj(objetivo,def,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                        predic(Obj,n), compl(Tipo_crime,n,em,_)] ) :- !.

% predicado crime(Tipo_crime, sim/nao)

regra(crime, ambos, sim, T_modo, AN, [ suj(ambos,n,_,n,n), verbo(ser,pres,T_modo,pl,AN), predic(crime,n)] ) :-!.

regra(crime, ambos, nao, T_modo, AN, [ suj(ambos,n,_,n,n), verbo(ser,pres,T_modo,pl,ANN), predic(crime,n)]) :- !,
        (AN = af, ANN = neg; ANN = af).

regra(crime, Tipo_crime, sim, T_modo, AN, [ suj(Tipo_crime,n,_,n,n), verbo(ser,pres,T_modo,sg,AN), predic(crime,n)]):- !.

regra(crime, Tipo_crime, nao, T_modo, AN, [ suj(Tipo_crime,n,_,n,n), verbo(ser,pres,T_modo,sg,ANN), predic(crime,n)]) :- !,
        (AN = af, ANN = neg; ANN = af).

% predicado estava(Pessoa, Local, Dia)

regra(estava, Pessoa, Local, Dia, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(estar,perf,T_modo,sg,AN),
                                        compl(Local,def,em,_), compl(dia,def,em,_), compl(Dia,n,_,_)]):-!.

% predicado not_humano(X)

regra(not_humano, X, T_modo, AN, [ suj(X,_,_,n,n), verbo(ver,pres,T_modo,sg,ANN), predic(humano,n) ]) :- !,
        (AN = af, ANN = neg; ANN = af).

% predicado autor_conhecido(Num, Autor)
```

```
regra(autor_conhecido, Num, Autor, T_modo, AN, [ suj(autor,def,_,n,n), verbo(ser,pres,T_modo,sg,AN),
                                                 predic(conhecido,n) ] ) :- !,
       obter(autor, Num, Autor1).

% predicado status(Pessoa, Valor)

regra(status, Pessoa, Valor, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ser,imperf,T_modo,sg,AN), predic(Valor,n)]) :- !.

% predicado relacao(P1, P2, Rel)

regra(relacao, P1, P2, Rel, T_modo, AN, [ suj(P1,n,_,n,n), verbo(ser,imperf,T_modo,sg,AN), predic(Rel,n),
                                          compl(P2,n,de,_) ] ) :- !.

% predicado arrolado(Pessoa, Papel, outro_crime)

regra(arrolado, Pessoa, Papel, Valor, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ser,perf,T_modo,sg,AN),
                                                    predic(Papel,n), compl(Valor,n,de,_)] ) :- !.

% predicado autor(Num, Valor)

regra(autor, Num, A,T_modo, AN, [suj(autor,def,_,do,crime), verbo(ser,perf,T_modo,sg,AN),
                                 predic(A,n)] ) :- !.

% predicado autor(Pessoa, 'outro crime')

regra(autor, Pessoa, 'outro crime', T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ser,perf,T_modo,sg,AN), predic(autor,def),
                                          compl('outro crime',n, de,_) ] ) :- !.

% predicado acao(Num/Tipo_crime, Valor)

regra(acao, Num, Valor, T_modo, AN, [suj(acao,def,_,do,crime), verbo(ser,perf,T_modo,sg,AN),
                                     predic(Valor,n)] ) :-
       integer(Num),!.

regra(acao, Tipo_crime, Valor, T_modo, AN, [ suj(acao,def,_,n,n), verbo(ser,pres,T_modo,sg,AN), predic(Valor,n),
                                          compl(Tipo_crime, n,ee, _) ] ) :- !.

% predicado vitima(Num, V)

regra(vitima, Num, V, T_modo, AN, [ suj(vitima,def,_,do,crime), verbo(ser,perf,T_modo,sg,AN),
                                    predic(V,n) ] ) :- !.

% predicado local(Num, L)

regra(local, Num, L, T_modo, AN, [ suj(local,def,_,do,crime), verbo(ser,perf,T_modo,sg,AN),
                                   predic(L,_) ] ) :- !.

% predicado dia(Num, D)

regra(dia, Num, D, T_modo, AN, [ suj(dia, def,_,do,crime), verbo(ser,perf,T_modo,sg,AN),
                                 predic(D,n) ] ) :- !.

% predicado instrumento(Num, Inst)
```

```
regra(instrumento, Num, Inst, T_modo, AN, [ suj(instrumento,def,_,do,crime), verbo(ser,perf,T_modo,sg,AN),
                                        predic(Inst,indef) ] ) :- !.

% predicado diferencas(sequencia(Lista))

regra(diferencas, [Lista_nucleares], T_modo, AN, [ suj(ambos,n,a,n,n), verbo(diferir,pres,T_modo,pl,AN),
                                            compl(Lista_nucleares,n, em, _) ] ) :- !.

% predicado semelhancas(sequencia(Lista))

regra(semelhancas, [Lista_nucleares], T_modo, AN, Lista_nucleares ) :- !.

% predicado viu(Pessoa,*Num)

regra(viu, Pessoa, Num, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ver,perf,T_modo,sg,AN),
                                    od(crime,def,_) ] ) :- !.

% predicado para testar anafora apenas : ouviu_vitima(Pessoa, Num, Oque)
regra( ouviu_vitima, Pessoa, Num, Oque, T_modo, AN, [ suj(Pessoa,n,_,n,n), verbo(ouvir,perf,T_modo,sg,AN),
                                    od(Vitima,n,_), compl(Oque,n,_,_)] ) :-
        obter(vitima, Num, Vitima).


% regra composta : agrupa as informacoes de autor,vitima,acao,local,dia que fazem
%                  parte da Resposta Correta de uma negacao, numa unica sentenca
%                  que tem a seguinte semantica : Autor verbo(Acao) Vitima em Local, Dia.
% Nem todas essas informacoes estao, necessariamente, presentes na lista de entrada :
% Cabe ao programa recupera'-las, quando conhecidas pelo usuario.
% Obs.: A semantica sera' outra, mais apropriada, quando um foco deve ser ressaltado.
% Ou seja, nao ha' regras de transformacoes de foco, na geracao da sentenca basica,
% esse fato e' previsto e a sentenca ja' e' construida de acordo. Esta decisao se deve ao fato
% de que o foco extraido da pergunta do usuario guia a escolha de informacoes pelo planejador.
% No caso de resposta negativa, a resposta correta deve refletir este foco. E isso e' feito pelo
% programa regra_foco que, por coerencia apenas, se encontra no arquivo Transf.ari juntamente com
% as demais regras de transformacoes.

regra(composta, [], _, _, [] ) :- !.

regra(composta, Lista, T_modo, AN, Result) :-
        pertence(autor(N,A), Lista), !,
        sentenca_autor(Lista, A, N, T_modo, AN, Result).

regra(composta, Lista, T, AN, Result) :-        % se vitima mas nao o autor esta' na lista
        pertence(vitima(N, V), Lista), !,
        sentenca_vitima(Lista, V, N, T, AN, Result).

regra(composta, Lista, T, AN, [ suj(crime,def,_,n,n), verbo(acontecer,perf,T,sg,AN),
                                compl(L,_,em,_),compl(dia,def,em,_),compl(D,n,_,_) ] ) :-
        pertence(local(N,L), Lista), !,         % se nem autor nem vitima aparecem na lista
        pertence(dia(N,D), Lista).

sentenca_autor(Lista, A, N, T, AN, Result ) :-
        (pertence(acao(N, Ac), Lista); obter(acao, N, Ac)),
        pertence(Ac, [homicidio, matar, ferir, roubar, assaltar, 'tentativa de homicidio']), !,
        obter(vitima, N, Vit),                  % Se acao admite vitima
```

```
        determina_verbo(Ac, V),
        obter_compl(Lista, Compl),
        foco(Foco),                             % Obtem foco da pergunta do usuario e ativa regra_foco
        regra_foco(Foco, [suj(A,_,_,n,n), verbo(V,perf,T,sg,AN),od(Vit,n,_)|Compl], Result).

sentenca_autor(Lista, A, N, T, AN, Result ) :-
        (pertence(acao(N, Ac), Lista) ; obter(acao, N, Ac)),
        obter_compl(Lista, Compl),              % se vitima e' o proprio autor
        foco(Foco),
        regra_foco(Foco, [suj(A,_,_,n,n),verbo(cometer,perf,T,sg,AN),od(Ac,n,_)|Compl], Result).


sentenca_vitima(Lista, Vit, N, T, AN, Result ) :-
        (pertence(acao(N, Ac), Lista); obter(acao, N, Ac)),
        pertence(Ac, [matar, ferir, roubar, assaltar, homicidio, 'tentativa de homicidio']), !,
        obter(autor, N, A),                     % obter o autor fora da lista
        determina_verbo(Ac, V),
        obter_compl(Lista, Compl),
        foco(Foco),
        regra_foco(Foco, [suj(A,n,_,n,n), verbo(V,perf,Ts,sg,AN), od(Vit,n,_)|Compl], Result).

sentenca_vitima(Lista, Vit, N, T, AN, [ suj(Vit,n,_,n,n), verbo(V,perf,T,sg,AN)|Compl ] ) :-
%       acao = suicidio, aborto
        (pertence(acao(N, Ac), Lista) ; obter(acao, N, Ac)),
        determina_verbo(Ac, V),
        obter_compl(Lista, Compl).


determina_verbo(homicidio, matar).
determina_verbo('tentativa de homicidio', ferir).
determina_verbo(aborto, abortar).
determina_verbo(suicidio, suicidar_se).
determina_verbo(Verbo, Verbo).

% obter_compl( Lista_informacoes, Complementos )
% Extrai da Lista_informacoes os Complementos existentes

obter_compl(Lista, [compl(L,n,em,_),compl(dia,def,em,_),compl(D,n,_,_)]) :-
        pertence(local(N,L), Lista),
        pertence(dia(N,D), Lista),!.

obter_compl(Lista, [compl(dia,def,em,_), compl(D,n,_,_)] ) :-
        pertence(dia(N,D), Lista),!.

obter_compl(Lista, [compl(L,n,em,_)]) :-
        pertence(local(N,L), Lista), !.

obter_compl(_, []).


% forma_adjuntos( Lista, Lista_adjuntos )
% Lista contem o predicado modo(Num, Modo) ou motivo(Num, Mot) ou ambos.
% Lista_adjuntos e' a lista contendo os respectivos complementos (adjuntos adverbiais)

forma_adjuntos( [modo(Num, M)], [compl(M,_,com,_)] ) :- !.
```

```
forma_adjuntos( [motivo(Num, Mot)], [compl(Mot,_,'por causa de', _)] ) :- !.

forma_adjuntos( [modo(Num, M), motivo(Num, Mot)], [compl(M,_,com,_), compl(Mot, _,'por causa de',_)] ).
```

```
%
%                          APENDICE 3
%
%           TRANSFORMAÇÕES DE SENTENÇAS BASICAS
%
%
% arquivo Transf. ari :
%
% Contém regras de coordenação , relativização, de foco (topicalização e passivização) e anaforização
%
% coordenacao( Lista_Sentencas, Lista_coordenada ) :
% decide e ativa uma coordenação livre ou sequencial, dependendo do
% contexto em que a lista de sentenças se encontra na estrutura RST.
% As sentenças são coordenadas duas a duas, quando for possivel coordenar.

coordenacao(Lista_coord, Lista_result) :-
        (no_pai([(concessao,7)|R1]) ;
         no_pai([(uniao,_)|R2]) ;
         no_pai([(antitese,8)|R3])), !,
        coordenacao_livre(Lista_coord, Lista_result).

coordenacao(Lista_coord, Lista_result) :-
        coordenacao_sequencial(Lista_coord, Lista_result).

% coordenacao_livre( Lista_Sent, Lista_Result ) :
% Lista_Result contem como elementos toda a sentenca de Lista_Sent, quer sob o sinal de coord(S1,S2),
% caso ela tenha sido coordenada com outra, quer na sua forma de entrada.

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

coordenacao_livre( [Prim|Resto], Result ) :-                    % Cada elemento da Lista
        forma_particao(Prim, Resto, Particao, Lista_result),    % e' um pivo para coordenacao
        coordenacao_livre(Lista_result, Restante),
        forma_result(Particao, Restante, Result).

% forma_result( [S1,S2], Resto Resultado ):
% Resultado marca uma coordenacao entre S1 e S2

forma_result( [[S1|R1], [S2|R2]], R, [coord( [S1|R1], [S2|R2] )|R] ) :- !.

forma_result( S, R, [S|R] ).    % Quando nao houve coordenacao, anexar candidato 'a lista final.

% forma_particao( Sent, Resto, Coordenacao, Lista_result ) :
% Encontra o melhor candidato para coordenacao , classifica invariante e aplica regra.

forma_particao(Prim, Resto, Lista_coord, Resto_par) :-
        maior_invariante(Prim, Resto, [Sent_par, Invariante], Resto_par),
        classifica_aplica(Invariante, Prim, Sent_par, Lista_coord).

% maior_invariante( Sent, Candidatos, [Escolhido, Invariante], Restantes ) :
% Selecao de candidato para coordenacao : acha invariante com respeito 'a sentence contigua.
% Compara-o com o maior invariante com respeito a todas as demais sentencas. Escolhe a sentenca com
% "melhor" invariante dentre essas duas.

maior_invariante( Prim, [], _, [] ) :- !.
```

```
maior_invariante(Pric, [Elem|Resto], [S, Maior], Restante) :-
        elem_comuns(Pric, Elem, Comuns),
        maior_invariante(Pric, Resto, [Sent, Invar], Resto1),
        escolhe_maior([Elem,Comuns], Resto, [Sent, Invar], Resto1, [S, Maior], Restante), !.

% escolhe_maior( [Sent1, Inv1],Resto1, [Sent2, Inv2],Resto2, [Sent_escolhida,Inv], Lista_result ) :
% Prioriza invariantes de candidatos : Tem mais baixa prioridade os candidatos com invariante
% vazio ou contendo apenas o verbo (neste caso nao coordena). Escolhe aquele com invariante
% de prioridade_maior (veja adiante).

escolhe_maior( [Elem, Inv1], Resto, [Sent, Inv2], Resto1, [_,_], [Elem|Resto] ) :-
        (Inv1 = [] ; Inv1 = [verbo(_,_,_,_,_)]),
        (Inv2 = [] ; Inv2 = [verbo(_,_,_,_,_)]), !.

escolhe_maior( [Elem, Inv1], Resto, [S, Inv], Resto1, [S, Inv], [Elem|Resto1] ) :-
        (Inv1 = [] ; Inv1 = [verbo(_,_,_,_,_)]) ,!.

escolhe_maior( [Elem, Com], Resto,[S, Inv], Resto1, [Elem, Com], Resto ) :-
        (Inv = [] ; Inv = [verbo(_,_,_,_,_)]), !.

escolhe_maior( [Elem1, Inv1], Resto, [Elem2, Inv2], Resto1, [Elem1, Inv1], Resto ) :-
        prioridade_maior(Inv1, Inv2), !.

escolhe_maior( [Elem1, Inv1], Resto, [Elem2, Inv2], Resto1, [Elem2, Inv2], [Elem1|Resto1] ) :- !.

% elem_comuns(Sent1, Sent2, Lista_elem_comuns ) :
% Verifica igualdade entre componentes sintaticos de S1 e S2, colocando os iguais
% na Lista de elementos comuns.

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

elem_comuns([Elem|Resto], Lista, [Elem|Resto1]) :-
        pertence(Elem, Lista), !,
        elem_comuns(Resto, Lista, Resto1).

elem_comuns([_|Resto], Lista, Resto1) :-
        elem_comuns(Resto, Lista, Resto1), !.

% prioridade_maior( Invariante1, Invariante2 ) :
% Invariante1 tem prioridade maior que Invariante2 se :

prioridade_maior([Elem1], [Elem2] ) :-  % Ambos tem um unico elemento e
        prioridade(Elem1, P1),          % prioridade do primeiro e' maior
        prioridade(Elem2, P2),          % que a do segundo
        P1 < P2, !.

prioridade_maior([Elem1|Resto], [Elem2] ) :-    % O conjunto invariante do primeiro
        Resto \== [].                           % e' maior que o segundo.

% prioridade( Elemento, Prioridade ) : Tabela Descendente de Prioridades.

prioridade( suj(_,_,_,_,_), 1 ) :- !.
prioridade( predic(_,_,_), 2 ) :- !.
prioridade( compl(_,_,_,_), 3 ) :- !.
```

```
prioridade( od(_,_,_), 4 ) :- !.
prioridade( verbo(_,_,_,_,_), 5 ).

% classifica_aplica( Invariante, Sent1, Sent2, Lista_coordenada ) :
% classifica o Invariante e aplica regra correspondente para Sent1
% e Sent2, obtendo Lista_coordenada.

classifica_aplica( Inv, Prim, _, Prim ) :-
        var(Inv), !.            % se nao ha' par para coordenar com Prim, Lista e' o proprio Prim.

classifica_aplica( Inv, Prim, Sent, Lista_coord ) :-
        classifica( Inv, Classe ),
        aplica(Classe, Prim, Sent, Lista_coord).

% classifica( Invariante, Classe ) :

classifica( [suj(_,_,_,_,_)], adhj ) :- !.      % Invariante = sujeito

classifica( [od(_,_,_)], bdgj ) :- !.           % Invariante = objeto direto

classifica( [compl(_,_,_,_)], bdhi ) :- !.      % Invariante = complemento

classifica( [verbo(_,_,_,_,_,_)], bcfhj ) :- !. % Invariante = verbo

classifica( Inv, acfhj ) :-                     % Invariante = sujeito + verbo
        coincide( Inv, [suj(_,_,_,_,_), verbo(_,_,_,_,_)] ) , !.

classifica( Inv, adhi ) :-                      % Invariante = sujeito + complemento
        coincide(Inv, [suj(_,_,_,_,_), compl(_,_,_,_)]), !.

classifica( Inv, adgj ) :-                      % Invariante = sujeito + objeto direto
        coincide(Inv, [suj(_,_,_,_,_), od(_,_,_)]), !.

classifica( Inv, bcegj ) :-                     % Invariante = verbo + predicativo/obj.direto
        coincide(Inv, [verbo(_,_,_,_,_), predic(_,_,_)]) ;
        coincide(Inv, [verbo(_,_,_,_,_), od(_,_,_)] ), !.

classifica( Inv, bcfhi ) :-                     % Invariante = verbo + complemento
        coincide(Inv, [verbo(_,_,_,_,_), compl(_,_,_,_)]), !.

classifica( Inv, bdgi ) :-                      % Invariante = objeto direto + complemento
        coincide( Inv, [od(_,_,_), compl(_,_,_,_)]), !.

classifica( Inv, acegj ) :-                     % Invariante = suj + verbo + obj.direto/predicativo
        coincide(Inv, [suj(_,_,_,_,_), verbo(_,_,_,_,_), od(_,_,_)]) ;
        coincide(Inv, [suj(_,_,_,_,_), verbo(_,_,_,_,_), predic(_,_,_)]), !.

classifica( Inv, acfhi ) :-                     % Invariante = sujeito + verbo + complemento
        coincide(Inv, [suj(_,_,_,_,_), verbo(_,_,_,_,_), compl(_,_,_,_)]), !.

classifica( Inv, adgi ) :-                      % Invariante = sujeito + obj.direto + complemento
        coincide(Inv, [suj(_,_,_,_,_), od(_,_,_), compl(_,_,_,_)]), !.

classifica( Inv, bcegi ) :-                     % Invariante = verbo + obj.direto/predicativo + complemento
        coincide(Inv, [verbo(_,_,_,_,_), od(_,_,_), compl(_,_,_,_)]) ;
```

```
        coincide(Inv, [verbo(_,_,_,_,_), predic(_,_,_), compl(_,_,_,_)]).

% coincide( Conj1, Conj2 ) : Igualdade entre conjuntos
% Conj1 = Conj2 <==> (Conj1 - Conj2) = [] e (Conj2 - Conj1) = [].

coincide( L1, L2 ) :-
        diferenca( L1, L2, [] ), !,
        diferenca( L2, L1, [] ).

% diferenca( Conj1, Conj2, Dif ) : Diferenca de Conjuntos

diferenca( [], L, [] ) :- !.

diferenca( L, [], [] ) :- !.

diferenca( [E|R], L, TR ) :-
        pertence(E, L), !,
        diferenca(R, L, TR).

diferenca( [E|R], L, [E|TR] ) :-
        diferenca(R, L, TR).

% aplica( Classe, S1, S2, Result ) :
% Result e' uma lista com as sentencas S1 e S2 possivelmente modificadas (elipses/anaforas)

% classe adhj : elipse do sujeito da 2a. sentenca
aplica(adhj, Prim, [suj(X,Y,Z,W,U)|Resto], [Prim, Resto] ) :- !.

% classe bdgj : anafora do objeto direto da 2a. sentenca
aplica(bdgj, Prim, [S, V, od(X,Y,Z)|Resto], [Prim, [S,V, od(X,Y,a)|Resto]] ) :- !.

% classe bdhi : elipse do complemento da 1a. sentenca
aplica(bdhi, Prim, Sent, [Prim1, Sent] ) :- !,
        troca_compl(Prim, Sent, Prim1).

% classe bcfhj : nada a fazer
aplica(bcfhj, Prim, Sent, [Prim, Sent] ) :- !.

% classe acfhj : elipse do sujeito e verbo da 2a. sentenca
aplica(acfhj, Prim, [suj(X,Y,Z,W,U), verbo(X1,Y1,Z1,W1,U1)|Resto],
                    [Prim, Resto]) :- !.

% classe adhi : elipse do complemento da 1a. e sujeito da 2a. sentenca.
aplica(adhi, Prim, [suj(X,Y,Z,W,U)|Resto], [Prim1, Resto] ) :- !,
        troca_compl(Prim, Resto, Prim1).

% classe adgj : elipse do sujeito da 1a. e anafora do obj.direto da 2a.
aplica(adgj, [suj(X,Y,Z,W,U)|R1], [S,V,od(X1,Y1,Z1)|R2], [R1, [S,V,od(X1,Y1,a)|R2]] ) :-!.

% classe bcegj : elipse do obj.direto ou predicativo da 2a.
aplica(bcegj, Prim, [S,verbo(X,Y,Z,W,U), P_OD|Resto], [Prim, [S,Resto]] ) :- !,
        P_OD =.. [od,X1,Y1,Z1] ;
        P_OD =.. [predic,X1,Y1,Z1].

% classe bcfhi : elipse do complemento da 1a. sentenca.
```

```
aplica(bcfhi, Prim, Sent, [Prim1, Sent] ) :- !,
       troca_compl(Prim, Sent, Prim1).

% classe bdgi : elipse do obj.direto e do complemento da 1a.
aplica(bdgi, [S,V,od(X,Y,Z)|Resto], Sent, [[S,V|Resto1], Sent] ) :- !,
       troca_compl(Resto, Sent, Resto1).

% classe acegj : elipse do sujeito, verbo e obj.direto ou predicativo da 2a.sentenca.
aplica(acegj, Prim, [suj(X,Y,Z,W,U), verbo(X1,Y1,Z1,W1,U1), P_OD|Resto],
                    [Prim, Resto] ) :- !,
       P_OD =.. [predic,X2,Y2,Z2];
       P_OD =.. [od,X2,Y2,Z2].

% classe acfhi : elipse do complemento da 1a. e sujeito e verbo da 2a.
aplica(acfhi, Prim, [Suj, Verbo|Resto], [Prim1, Resto] ) :- !,
       troca_compl(Prim, Resto, Prim1).

% classe adgi : elipse do obj.direto e complemento da 1a. e sujeito da 2a.
aplica(adgi, [S,V,od(X,Y,Z)|Resto1], [suj(X1,Y1,Z1,W1,U1)|Resto2],
                                      [[S,V|Resto11], Resto2] ) :- !,
       troca_compl(Resto1, Resto2, Resto11).

% classe bcegi : elipse do verbo obj.direto ou predicativo e complemento da 1a. Troca numero de verbo da 2a. para plural.
aplica(bcegi, [S, verbo(X,Y,Z,W,U),P_OD|Resto1], [S1, verbo(X1,Y1,Z1,W1,U1)|Resto2],
              [[S|Resto11], [S1, verbo(X1,Y1,Z1,pl,V1)|Resto2]] ) :- !,
       (P_OD =.. [od,X2,Y2,Z2];
       P_OD =.. [predic,X2,Y2,Z2]),
       troca_compl(Resto1, Resto2, Resto11).

% troca_compl( S1, S2, S ) :
% S e' S1 com o complemento-comum tendo sido eliminado devido a ocorrencia de elipse.

troca_compl(Prim, Sent, Prim1) :-
       pertence(Compl, Prim),
       pertence(Compl, Sent); !,
       elimina(Compl, Prim, Prim1).

troca_compl(Prim, Sent, Prim1) :-
       ultimo_compl(Prim, Compl),              % Ha' no maximo dois complementos numa sentenca
       pertence(Compl, Sent),
       elimina(Compl, Prim, Prim1).

% ultimo_compl( Lista, Complemento ) :
% verifica se Complemento e' o ultimo complemento da Lista

ultimo_compl( [X], X ) :- !.

ultimo_compl( [_|R], X ) :-
       ultimo_compl( R, X ).

% elimina( Compl, Lista, Result ) : Elimina Compl da Lista, resultando Result.

elimina(Compl, [Compl|Resto], Resto ) :- !.

elimina(X, [Elem|Resto], [Elem|T] ) :-
```

```
        elimina(X, Resto, T).

% coordenacao_sequencial( Lista_sentencas, Lista_coordenada ) :
% Os elementoss desta lista possuem uma escala de prioridades dadas por suas posicoes
% na lista. Isto se deve ao fato da Lista_sentnecas corresponder ao corpo de uma clausula
% Prolog, e entre os argumentos de cada sentenca.

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

coordenacao_sequencial( [Prim], [Prim] ) :- !.  % Se ha' uma unica sentenca, nada a coordenar.

coordenacao_sequencial( [Prim,Seg|Resto], [coord(Prim1,Seg1)|Resto1] ) :-      % Selecao de Candidato :
        elem_comuns(Prim, Seg, Comuns),         % Verifica se o invariante com respeito 'a sentenca contigua
        Comuns \= [],                           % nao e' nulo ou simplesmente um verbo.
        Comuns \= [verbo(_,_,_,_,_)], !,
        classifica_aplica(Comuns, Prim, Seg, [Prim1, Seg1] ),   % Classifica invariante e aplica regra.
        coordenacao_sequencial( Resto, Resto1 ).

coordenacao_sequencial( [Elem|Resto], [Elem|Result] ) :-        % Se nao ha' invariante valido, nao coordenar
        coordenacao_sequencial( Resto, Result ).                % a sentenca e buscar coordenacao para lista restante


%  Regras de relativizacao :

% Escopo : entre os elementos de uma sequencia, exceto aqueles sob "coord", pois seria
% arriscado carregar ainda mais uma sentenca ja' coordenada.
% Entre as sentencas de nucleo e satelite nao convem, pois ficaria obscura a subordinacao entre ambos.
% Tambem nao e' feito entre os argumentos de uniao, pois e' necessario ficar clara a intencao do
% segundo argumento de que o que segue sao informacoes adicionais, nao motivadas pelo conteudo do 1o. argumento.

% Relativa( Lista_sentencas, Lista_resultante ) :

relativa( [Unico], [Unico] ) :- !.      % Quando ha' um unico elemento, nada a fazer.

relativa( [coord(Lista), Elem], [coord(Lista, Elem)] ) :- !.    % Dois elementos, um deles sob "coord" :nada a fazer

relativa( [Elem, coord(Lista)], [Elem, coord(Lista)] ) :- !.

relativa( Lista, Result ) :-
        separa_coord( Lista, Sentencas ),       % Analisar apenas as Sentencas.
        rel( Sentencas, Result ).

% separa_coord( Lista, Sentencas ) : Sentencas e' Lista menos os elementos "coord".

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

separa_coord( [coord(L)|Resto], Resto1 ) :- !,
        separa_coord( Resto, Resto1 ).

separa_coord( [Elem|Resto], [Elem|Resto1] ) :-
        separa_coord( Resto, Resto1 ).

% rel( Sentencas, Result ) : Escopo para relativizacao e' Sentencas.

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

```
rel( [Prim|Resto], Result ) :-                  % Sempre havera' no minimo 2 sentencas, da 1a. vez
       busca_relativa( Prim, Resto, Prim1, Resto1),
       rel( Resto1, Restante ),
       acerta( Restante, Prim1, Result ).

% busca_relativa( Sent, Resto_sentnecas, Result, Resto_analisar ):
% Se e' possivel relativizar em relacao a sentenca contigua, aplicar regra
% segundo o elemento invariante ao sujeito desta ultima.

busca_relativa( Prim, [Elem|R], Result, R ) :-
       possivel_relat( Prim, Elem, Ob_comp ), !,
       aplica_relativa( Ob_comp, Prim, Elem, Result ).

busca_relativa( Prim, Resto, Prim, Resto ).     % Nao e' possivel relativizar.

acerta( [], Prim1, [Prim1] ) :- !.

acerta( Resto, Prim1, [Prim1|Resto] ).

% possivel_relat( A , B, Elem_rel ) :
% E' possivel relativizar A e B se o sujeito de B coincide com o objeto direto ou com o complemento de A.

possivel_relat( [S,V, od(X,Y,Z)|R], [suj(X,_,_,n,n)|R1], od(X,Y,Z)  ) :- !.     % O sujeito nao deve ter complemento.

possivel_relat( [S, V, O_pred|Compl], [suj(X,_,_,n,n)|R], compl(X,Y,Z,W) ) :-
       ultimo_compl( Compl, compl(X,Y,Z,W) ).

% aplica_relativa( Elem_relativizado, S1, S2, Sent_relativa ) :
% Sent_relatica e' S1 com o Elem_relativizado seguido de pronome (que), S2 menos o
% sujeito e Resto de S1 e S2.

aplica_relativa( od(X,Y,Z), [S, V, OD|R], [S1|R1], [S, V, OD, pron(que)|R2] ) :-
       !,                                % Elem_relat. = objeto direto
       append( R, R1, R2 ).

aplica_relativa( compl(X,Y,Z,W), Prim, [S1|R1], Result ) :-     % Elem_relat = complemento
       append( Prim, [pron(que)|R1], Result ).

% Regras de Foco :

% regra_foco( Foco, Sentenca_basica, Sentenca_modificada )

% se foco e' autor, portanto o sujeito da sentenca basica, realca'-lo usando "foi ... que... ".

regra_foco( [evento,autor], [Suj, verbo(V,T,M,N,AN), Ob|Compl],
           [verbo(ser,perf,indicativo,sg,AN), Suj, pron(que),verbo(V,T,M,N,af),Ob|Compl] ) :- !.

% se foco e' a vitima, portanto o objeto direto da sentenca basica, passivizar a sentenca.

regra_foco( [evento,vitima], [Suj, verbo(V,T,M,N,AN), Ob|Compl],
           [Ob, verbo(ser,perf,indicativo,sg,AN), verbo(V,_,passivo,_,af), prep(por), Suj] ) :- !.

% se foco e' a data, portanto um possivel complemento da sentenca basica, topicaliza-la, colocando-o,
% se existir, no inicio da sentenca.
```

```
regra_foco( [evento,dia], [S, V, O, compl(dia,Art,Prep,X),compl(O,Art1,_,_)],
            [compl(dia,Art,Prep,X),compl(O,Art1,_,_),S,V,O] ) :- !.

regra_foco( [evento,dia], [S, V, O, CL, compl(dia,Art,Prep,X),compl(O,Art1,_,_)],
            [compl(dia,Art,Prep,X),compl(O,Art1,_,_),S, V, O, CL] ) :- !.

% se foco e' o local do crime, portanto um possivel complemento da sentenca basica, topicaliza-la.

regra_foco( [evento,local], [S, V, O, compl(L,Art,Prep,X)|R],
            [compl(L,Art,Prep,X), S, V, O|R] ) :-
       L \== dia, !.

% demais casos : nao alterar a sentenca basica.

regra_foco( _, Sent, Sent ).
```

```
%
%                       APENDICE 4
%
%                 COMPONENTE MORFOLOGICO
%
%
% arquivo : base_morfologica.ari
%
% Dada a estrutura resultante do passo anterior, ou seja, um esquema RST
% com as sentenças transformadas, dar forma ao texto final, lexicalizando
% cada componente sintático e introduzindo pontuação interna 'as sequencias,
% uma vez que entre os núcleos e satélites já foi feito no passo anterior.

% formatar( Lista, String, String_result ) :

formatar( [], St, St ) :- !.

formatar( [X|R], St, Str ) :-
        linear( X, St, St1 ),
        formatar( R, St1, Str ).

% linear( Arg1, Str_anterior, Str_resultante ) :
% Acha texto correspondente 'a Arg1 e o concatena 'a Str_anterior formando Str_resultante.

linear( [], S, S ) :- !.                % Nada a fazer se Lista vazia.

linear( 'Sim', $$, $Sim$ ) :- !.

linear( 'Nao', $$, $Nao$ ) :- !.

linear( ',', St, Str ) :- !,
        concat( St, $,$, Str ).

linear( '.', St, Str ) :- !,
        concat( St, $.$, Str ).

linear( conj(Conj), St, Str ) :- !,
        atom_string( Conj, Conj1 ),
        concat( [St, Conj1, $ $], Str ).

linear( [conj(Conj), Lista], St, Str ) :- !,    % Se termo conj(Conjunção), converter
        atom_string( Conj, C1 ),                % conjunção em string e concatenar.
        concat( [St, C1, $ $], St1 ),
        linear( Lista, St1, Str ).

linear( [conj(Conj)|Resto], St, Str ) :- !,
        atom_string( Conj, C1 ),
        concat( [St, C1, $ $], St1 ),
        formatar( Resto, St1, Str ).

linear( [X|R], St, Str ) :-
        sentenca( [X|R] ), !,
        morfologia( [X|R], St, Str ).

linear( [X|Y], St, Str ) :- !,
```

```
        morfologia( X, St, St1 ),
        pontua( Y, Pont ),
        concat( St1, Pont, St2 ),
        linear( Y, St2, Str ).

% sentenca ( Candidato ) :

sentenca( [conj(C)|R] ) :- !,
        fail.

sentenca( [coord(S1, S2)|R] ) :- !,
        fail.

sentenca( [ [_|_]|R ] ) :- !,
        fail.

sentenca( _ ).

% pontua( Lista, Pontuacao ) :
% Faz a pontuação entre os elementos de uma sequencia (Lista). Antes do último
% elemento coloca um "e"; entre os demais, uma virgula.

pontua( [], $$ ) :- !.

pontua( [Elem], $e $ ) :- !.

pontua( Lista, $, $ ).


% morfologia( Lista, String_anterior, String_final ) :
% Lista = [ Sentenca e/ou coord( S1, S2 ) ]

morfologia( [], St, St ) :- !.

morfologia( coord( S1, S2 ), St, Str ) :- !,
        morfologia( S1, St, St1 ),
        concat( St1, $e $, St2 ),
        morfologia( S2, St2, Str ).

morfologia( [Elem|Resto], St, Str ) :-
        analise( Elem, Sta ),
        concat( [St, Sta, $ $], St1 ),
        morfologia( Resto, St1, Str ).

% analise( Compon_sintatico, Cadeia ) :
% Constrói o item léxico correspondente ao componente sintático dado, segundo
% traços morfológicos do dicionário e traços sintáticos que ele carrega consigo.

analise( suj(I,A,Anafora,C,P), Nome ) :-
        var(Anafora),
        (var(C), var(P) ; C = n, P = n), !,
        dic(I,nome, Num, Gen ),
        artigo( A, Gen, Num, Art ),
        atom_string( I, I1 ),
        concat( [Art, $ $, I1], Nome ).
```

```
analise( suj(I,A,Anafora,C,P), Nome ) :-
        var(Anafora), !,                % Se sujeito nao anaforizado, cadeia = Artigo + Item.
        dic(I, nome, Num, Gen ),
        artigo( A, Gen, Num, Art ),
        atom_string( I, I1 ),
        atom_string( C, C1 ),
        atom_string( P, P1 ),
        concat( [Art, $ $, I1, $ $, C1, $ $, P1], Nome ).

analise( suj(I,A,a,C,P), Pron ) :- !,
        dic(I, nome, Num, Gen ),        % Se sujeito anaforizado, copnstruir pronome segundo
        pronome( Gen, Num, suj, Pron ). % tracos do item.

analise( verbo(I,T,M,N,AN), Verbo ) :- !, % Se verbo, conjuga'-lo segundo tracos do item.
        morf( I, verbo, T, M, N, AN, Verbo ).

analise( predic(I,A), Predic ) :- !,    % Se predicativo, obter tracos morfologicos do item
        dic(I, nome, Num, Gen ),        % no dicionario e contruir Artigo + Item.
        artigo( A, Gen, Num, Art ),
        atom_string( I, I1 ),
        concat( [Art, $ $, I1], Predic ).

analise( od(I,A,Anafora), OD ) :-       % Se objeto direto nao anaforizado,
        var(Anafora), !,                % cadeia = Artigo + Item.
        dic(I,nome, Num, Gen ),
        artigo( A, Gen, Num, Art ),
        atom_string( I, I1 ),
        concat( [Art, $ $, I1], OD ).

analise( od(I,A,a), Pron ) :- !,        % Se objeto direto anaforizado, construir
        dic(I, nome, Num, Gen ),        % pronome segundo tracos do item.
        pronome( Gen, Num, od, Pron1 ),
        concat( $-$, Pron1, Pron ).

% analise( compl(I,A,P,a), Pron )  nao acontece na aplicacao.

analise( compl( [X|Y], _, em, _], Cadeia ) :-  % Se complemento remete a uma relacao antitese
        formatar( [X|Y], $$, St1 ),
        concat( $em : $, St1, Cadeia).

analise( compl(I,A,P,_), Cadeia ) :-    % Se complemento preposicionado, construir
        dic(I, nome, Num, Gen ),        % preposicao + Artigo e concatenar 'a cadeia
        artigo( A, Gen, Num, Art ),     % correspondente ao item.
        prep_art( P, Art, Preart ),
        atom_string( I, I1 ),
        concat( [Preart, $ $, I1], Cadeia ).

analise( pron( P ), Pron ) :- !,        % Se pronome, converte-lo em string.
        atom_string( P, Pron ).

analise( prep( P ), Prep ) :- !,        % Se preposicao, converte-la em string.
        atom_string( P, Prep ).

analise( coord( S1, S2 ), Cadeia ) :-   % Se "coordencao interna" (de sujeitos, por ex).
```

```
        !,
        morfologia(S1, $$, S11),        % analisar recursivamente S1 e S2 e coordena'-los com "e".
        concat(S11, $e $, Si),
        morfologia(S2, Si, Cadeia).

%analise( [Elem|R], Cadeia ) :-
%       analise(Elem, C1),
%       analise(R, C2),
%       concat( [C1, $ $, C2], Cadeia ).

% pronome( Genero, Numero, Componente_sintatico, Cadeia_pronome ) :
% Tabela de pronomes retos e obliquos.

pronome(f,sg,suj, $ela$).
pronome(f,pl,suj, $elas$).
pronome(m,sg,suj, $ele$).
pronome(m,pl,suj, $eles$).
pronome(f,sg,od, $a$).
pronome(f,pl,od, $as$).
pronome(m,sg,od, $o$).
pronome(m,pl,od, $os$).

% artigo( def/indef/n/_, Genero, Numero, Cadeia_artigo ) :
% Tabela de artigos definidos e indefinidos.

artigo( Tipo, _,_, $$) :-               % este caso mereceria um estudo do nome que antecede o artigo
        var(Tipo), !.                   % para decidir se este deve aparecer e qual seu tipo.

artigo(def,f,sg, $a$):- !.
artigo(def,f,pl, $as$) :- !.
artigo(def,m,sg, $o$) :- !.
artigo(def,m,pl, $os$) :- !.
artigo(indef,f,sg, $uma$) :- !.
artigo(indef,f,pl, $umas$) :- !.
artigo(indef,m,sg, $um$) :- !.
artigo(indef,m,pl, $ums$) :- !.

artigo(n,_,_, $$).      % quando o artigo e' "n",ou seja, nao deve ser usado.

% prep_art( Preposicao, Artigo, Cadeia ) :
% Regras de contração (ou não) de Preposição + Artigo.

prep_art(Prep, Art, Art) :-             % Se nao ha' preposicao, cadeia = Artigo.
        var(Prep), !.

prep_art(antes, $$, $antes de$) :- !.   % Se preposicao "antes" e artigo inexistente, cadeia = antes de

prep_art(depois, $$, $depois de$) :- !. % Idem para "depois de".

prep_art(Prep, $$, Cadeia) :- !,        % Se artigo inexistente, cadeia = Preposicao.
        atom_string(Prep, Cadeia).

prep_art(em, Art, Cadeia) :- !,         % Ex: em + a = na
        concat($ n$, Art, Cadeia).
```

```
prep_art(de, Art, Cadeia) :- !,          % Ex: de + a = da
       concat($ d$, Art, Cadeia).

prep_art(por, $o$, $pelo$) :- !.         % por + o = pelo

prep_art(por, $a$, $pela$) :- !.         % por + a = pela

prep_art(antes, Art, Cadeia) :- !,       % por ex: antes + o = antes do
       concat( [$antes $, $d$, Art], Cadeia ).

prep_art(depois, Art, Cadeia) :- !,      % por ex: depois + a = depois da
       concat( [$depois $, $d$, Art], Cadeia ).

prep_art(Prep, Art, Cadeia) :-           % Prep = com, etc.: nao contrair : concatenar.
       atom_string( Prep, Prep1 ),
       concat( [Prep1, $ $, Art], Cadeia ).

% Regras de Conjugacao de Verbos
% morf( Item, verbo, Tempo, Modo, Numero, af/neg, Verbo_conjugado )

morf(Verbo, verbo, pres, infinitivo,_, af, Verbo) :- !. % regra para obter pres. infinitivo

morf(Verbo, verbo, Pret, gerundio, _, af, Result) :-          % regra para obter gerundio composto
       pertence(Pret, [perf, imperf]),
       morf(Verbo, verbo, _, part_pass, _, _, Str),
       concat($tendo $, Str, Result).

morf(Verbo, verbo, _, infinitivo,  sg, af, Result) :-         % regra para obter infinitivo composto singular
       morf(Verbo, verbo, _, part_pass, _, _, Str),
       concat($ter $, Str, Result).

morf(Verbo, verbo, _, infinitivo,  pl, af, Result) :-         % regra para obter infinitivo composto plural.
       morf(Verbo, verbo, _, part_pass, _, _, Str),
       concat($terem $, Str, Result).

morf(Verbo, verbo, perf, subjuntivo,  N, af, Result) :-       % regra para obter perf. subjuntivo
       morf(Verbo, verbo, _, part_pass, _, _, Str),
       morf(ter, verbo, pres, subjuntivo,  N, af, Str1),
       concat( [Str, $ $, Str1], Result).

% Tabela de Verbos conjugados segundo necessidades atuais do projeto :

morf(ser,verbo,pres,indicativo,sg,af, $eh$).
morf(ser,verbo,pres,indicativo,pl,af, $sao$).
morf(ser,verbo,perf,indicativo,sg,af, $foi$).
morf(ser,verbo,imperf,indicativo,sg,af, $era$).
morf(ser,verbo,pres,subjuntivo,sg,af, $seja$).
morf(ser,verbo,pres,subjuntivo,pl,af, $sejam$).
morf(ser,verbo,imperf,subjuntivo,sg,af, $fosse$).
morf(ser,verbo,pres,gerundio,sg,af, $sendo$).
morf(ser,verbo,_,part_pass,_,af, $sido$).

morf(morrer,verbo,pres,indicativo,sg,af, $morre$).
morf(morrer,verbo,perf,indicativo,sg,af, $morreu$).
morf(morrer,verbo,pres,subjuntivo,sg,af, $morra$).
```

```
morf(morrer,verbo,pres,gerundio,sg,af, $morrendo$).
morf(morrer,verbo,_,part_pass,_,af, $morrido$).

morf(estar,verbo,perf,indicativo,sg,af, $esteve$).
morf(estar,verbo,_part_pass,_,af, $estado$).

morf(ouvir,verbo,perf,indicativo,sg,af, $ouviu$).
morf(ouvir,verbo,_,part_pass,_,af, $ouvido$).

morf(fugir,verbo,perf,indicativo,sg,af, $fugiu$).
morf(fugir,verbo,_,part_pass,_,af, $fugido$).
morf(ver,verbo,perf,indicativo,sg,af, $viu$).
morf(ver,verbo,_,part_pass,_,af, $visto$).

morf(ferir,verbo,perf,indicativo,sg,af, $feriu$).
morf(ferir,verbo,_,part_pass,_,af, $ferido$).

morf(suicidar_se,verbo,perf,indicativo,sg,af, $suicidou_se$).
morf(suicidar_se,verbo,_,part_pass,_,af, $se_suicidado$).

morf(roubar,verbo,perf,indicativo,sg,af, $roubou$).
morf(roubar,verbo,_,part_pass,_,af, $roubado$).

morf(assaltar,verbo,perf,indicativo,sg,af, $assaltou$).
morf(assaltar,verbo,_,part_pass,_,af, $assaltado$).

morf(cometer,verbo,perf,indicativo,sg,af, $cometeu$).
morf(cometer,verbo,_,part_pass,_,af, $cometido$).

morf(abortar,verbo,perf,indicativo,sg,af, $abortou$).
morf(abortar,verbo,_,part_pass,_,af, $abortado$).

morf(matar,verbo,perf,indicativo,sg,af, $matou$).
morf(matar,verbo,_,part_pass,_,af, $matado$).
morf(matar,verbo,_,passivo,_,af, $morta$).

morf(dar_se,verbo,pres,indicativo,sg,af, $da_se$).
morf(dar_se,verbo,perf,indicativo,sg,af, $deu_se$).
morf(dar_se,verbo,pres,subjuntivo,sg,af, $da_se$).
morf(dar_se,verbo,pres,gerundio,sg,af, $dando_se$).
morf(dar_se,verbo,_,part_pass,_,af, $dado$).

morf(espancar,verbo,perf,indicativo,sg,af, $espancou$).
morf(espancar,verbo,_,part_pass,_,af, $espancado$ ).

morf(possuir,verbo,imperf,indicativo,sg,af, $possuia$).
morf(possuir,verbo,_,part_pass,_,af, $possuido$).

morf(ocorrer,verbo,pres,indicativo,sg,af, $ocorre$).
morf(ocorrer,verbo,pres,subjuntivo,sg,af, $ocorra$).
morf(ocorrer,verbo,pres,gerundio,sg,af, $ocorrendo$).
morf(ocorrer,verbo,_,part_pass,_,af, $ocorrido$).

morf(capturar,verbo,perf,indicativo,sg,af, $capturou$).
morf(capturar,verbo,perf,indicativo,pl,af, $capturaram$).
```

```
morf(capturar,verbo,_,part_pass,_,af, $capturado$).

morf(influenciar,verbo,pres,indicativo,sg,af, $influencia$).
morf(influenciar,verbo,perf,indicativo,sg,af, $influenciou$).
morf(influenciar,verbo,pres,subjuntivo,sg,af, $influencie$).
morf(influenciar,verbo,pres,gerundio,sg,af, $influenciando$).
morf(influenciar,verbo,_,part_pass,_,af, $influenciado$).

morf(dar,verbo,perf,indicativo,sg,af, $deu$).
morf(dar,verbo,_,part_pass,_,af, $dado$).

morf(diferir,verbo,pres,indicativo,pl,af, $diferem$).
morf(diferir,verbo,pres,subjuntivo,pl,af, $diferem$).
morf(diferir,verbo,pres,gerundio,pl,af, $diferindo$).
morf(diferir,verbo,_,part_pass,_,af, $diferido$).

morf(assumir,verbo,pres,indicativo,sg,af, $assume$).
morf(assumir,verbo,perf,indicativo,sg,af, $assumiu$).
morf(assumir,verbo,pres,subjuntivo,sg,af, $assuma$).
morf(assumir,verbo,pres,gerundio,sg,af, $assumindo$).
morf(assumir,verbo,_,part_pass,_,af, $assumido$).

morf(ter,verbo,pres,indicativo,sg,af, $tem$).
morf(ter,verbo,imperf,indicativo,sg,af, $tinha$).
morf(ter,verbo,perf,indicativo,sg,af, $teve$).
morf(ter,verbo,pres,subjuntivo,sg,af, $tenha$).
morf(ter,verbo,imperf,subjuntivo,sg,af, $tivesse$).
morf(ter,verbo,pres,gerundio,sg,af, $tendo$).
morf(ter,verbo,_,part_pass,_,af, $tido$).

morf(coincidir,verbo,pres,indicativo,sg,af, $coincide$).
morf(coincidir,verbo,pres,indicativo,pl,af, $coincidem$).
morf(coincidir,verbo,pres,subjuntivo,sg,af, $coincida$).
morf(coincidir,verbo,pres,subjuntivo,pl,af, $coincidam$).
morf(coincidir,verbo,_,part_pass,_,af, $coincidido$).

morf( Verbo,verbo, T,M,N, neg, Cadeia) :-          % Regra para negar o verbo :
       morf(Verbo,verbo,T,M,N, af, Result),        % concatenar 'a frente do verbo,
       concat($nao $, Result, Cadeia).             % a cadeia "nao".
```

```
% Dicionario

% Categoria NOME  ou  ADJETIVO  :
% formato : dic( entrada_lexical, categoria, numero, genero )

dic(autor, nome, sg, m).
dic(vitima, nome, sg, f).
dic(local, nome, sg, m).
dic(dia, nome, sg, m).
dic(acao, nome, sg, f).
dic(crime, nome, sg, m).
dic(instrumento, nome, sg, m).
dic(motivo, nome, sg, m).
dic(testemunha, nome, sg, f).
dic(suspeito, nome, sg, m).
dic(pena, nome, sg, f).
dic(decisao, nome, sg, f).
dic(praticante, nome, sg, _).
dic(consentimento, nome, sg, m).
dic(mae, nome, sg, f).
dic(causa, nome, sg, f).
dic(risco, nome, sg, m).
dic(idade, nome, sg, f).
dic(meio, nome, sg, m).
dic(objetivo, nome, sg, m).
dic(ambos, nome, pl, m).
dic(revolver, nome, sg, m).
dic(faca, nome, sg, f).
dic(ciume, nome, sg, m).
dic(infidelidade, nome, sg, f).
dic(vinganca, nome, sg, f).
dic('ser humano', nome, sg, m).
dic('auxilio de terceiros', nome, sg, m).
dic('influencia do estado puerperal', nome, sg, f).
dic('recem nascido', nome, sg, m).
dic(homicidio, nome, sg, m).
dic(suicidio, nome, sg, m).
dic(infanticidio, nome, sg, m).
dic(traicao, nome, sg, f).
dic(emboscada, nome, sg, f).
dic(dissimulacao, nome, sg, f).
dic(estupro,nome, sg, m).
dic(aborto, nome, sg, m).
dic(vida, nome, sg, f).
dic(gestante, nome, sg, f).
dic(medico, nome, sg, m).
dic(embriao, nome, sg, m).
dic(tiro, nome, sg, m).
dic('tres tiros', nome,pl, m).
dic('um tiro',nome,sg,m).
dic(assalto, nome, sg, m).
dic(feto, nome, sg, m).
dic(marido, nome, sg, m).
dic(mulher, nome, sg, f).
dic(esposa, nome, sg, f).
```

```
dic(esposo, nome, sg, m).
dic(discussao, nome, sg, f).
dic(irmao, nome, sg, m).
dic(irma, nome, sg, f).
dic(pai, nome, sg, m).
dic('Marcio', nome,sg,m).
dic('Alfredo',nome,sg,m).
dic('Carlos',nome,sg,m).
dic('Ana', nome, sg, f).
dic('Joao', nome, sg, m).
dic('Antonio', nome, sg, m).
dic('Pedro', nome, sg, m).
dic('Joana', nome, sg, f).
dic('Maria', nome, sg, f).
dic('Jose', nome, sg, m).
dic(sequestro, nome, sg, m).
dic(roubo, nome, sg, m).
dic(inimigo, nome, sg, m).
dic(rival, nome, sg, _).
dic(adversario, nome, sg, m).
dic(socio, nome, sg, m).
dic(concorrente, some, sg, _).
dic(veneno, nome, sg, m).
dic(fogo, nome, sg, m).
dic(explosivo, nome, sg, m).
dic(asfixia, nome, sg, f).
dic(tortura, nome, sg, f).
dic(pagamento, nome, sg, m).
dic('gravidez resultante de estupro', nome, sg, f).
dic('aborto provocado por terceiros com consentimento qualificado', nome, sg, m).
dic('aborto provocado por terceiros sem consentimento qualificado', nome, sg, m).
dic('aborto sem consentimento', nome, sg, m).
dic('aborto resultante de estupro', nome, sg,m).
dic('salvamento da vida da gestante', nome, sg, m).
dic('aborto provocado por terceiros sem consentimento', nome, sg, m).
dic('aborto provocado por terceiros com consentimento', nome, sg, m).
dic('suicidio induzido', nome, sg, m).
dic('homicidio doloso', nome, sg, m).
dic('homicidio qualificado', nome, sg, m).
dic('homicidio simples culposo', nome, sg, m).
dic('homicidio simples doloso', nome, sg, m).
dic('homicidio culposo', nome,sg,m).
dic('homicidio simples culposo',nome,sg,m).
dic('tentativa de homicidio', nome, sg, f).
dic('aborto necessario', nome, sg, m).
dic('aborto provocado', nome, sg, m).
dic('inobservancia de regra tecnica', nome, sg, f).
dic('imprudencia do autor', nome, sg, f).
dic('negligencia do autor', nome, sg, f).
dic('impericia do autor', nome, sg, f).
dic('praticante do aborto', nome, sg, _).
dic('outro crime', nome, sg, m).
dic('ocultacao de crime', nome, sg, f).
dic('muito importante', nome, sg, _).
dic('muito rica', nome, sg, f).
```

```
dic(parto, nome, sg, m).
dic(policia,nome,sg,f).
dic('Leme',nome,sg,m).
dic('Copacabana',nome,sg,m).
dic('21',nome,sg,m).
dic(terceiros,nome,pl,m).

% adjetivos

dic(futil, nome, sg, _).
dic(morto, nome, sg, m).
dic(assumido, nome, sg, m).
dic('debil mental', nome, sg, _).
dic(alienada, nome, sg, f).
dic(vivo, nome, sg, m).
dic(foragido, nome, sg, m).
dic(ferido, nome, sg, m).
dic(preso, nome, sg, m).
dic(rica, nome, sg, f).
dic(importante, nome, sg, _).
dic(desconhecido, nome, sg, m).
dic(fugitivo, nome, sg, m).
dic(conhecido, nome, sg, m).
dic(facadas, nome, pl, f).
dic(gritar, nome, sg,_).
dic(matar, nome, sg,_).
```