UNIVERSIDADE NOVA DE LISBOA

Faculdade de Ciências e Tecnologia
Departamento de Informática

Analisador por Grafo: superação de incorrecções na escrita do Português

Maria Aline da Silva Gomes Camacho Baião

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática

Lisboa

1994

Resumo

Nesta tese é apresentado um sistema constituído por três módulos (um analisador, um formulador de hipóteses e um corrector) que detecta e corrige vários tipos de erros morfo-sintácticos e que, de algum modo, supera o seu próprio desconhecimento.

Assim, dada uma cadeia de caracteres, o sistema tenta classificá-la como uma frase portuguesa gramaticalmente correcta ou sugere correcções baseadas na análise sintáctica que realizou.

Este sistema, construído em Prolog, utiliza um analisador por grafo (chart-parser) para pesquisa de constituintes da frase que, em caso de falha, interage com um formulador de hipóteses permitindo assim a superação da mesma.

A gramática utilizada, de movimentação e ancoragem, permite a identificação de constituintes fora da ordem canónica e possibilita a sua recolocação na estrutura sintáctica canónica de uma frase tendo em conta alguns tipos de barreiras.

O módulo corrector, face ao grafo obtido, considera como mais plausíveis as soluções com menor número de erros.

Se existir uma palavra desconhecida com categoria sintáctica atribuída, o módulo corrector invocará um corrector ortográfico muito simplificado (tratando apenas os erros tipificados por Damerau [Dam 64] e utilizando um conjunto de regras que permitem o estabelecimento de preferências relacionadas com confusões fonéticas) que tentará propôr correcções à palavra ou a sua admissão como neologismo.

Abstract

This thesis describes a simple syntactic checker for Portuguese. This system is made up of three modules: a parser, a hypothesis formulator and an error correcting device.

The system detects and corrects different types of morpho-syntactic errors and somehow goes beyond its own lack of knowledge.

The system tries to classify a string as a grammatically correct Portuguese sentence or suggests corrections based on its syntactic analysis.

This system was built in Prolog. It uses a chart-parser to analyse the constituents of a string. If it fails to classify the string as a sentence it interacts with a hypothesis formulator to find possible reasons for the error(s) and to provide possible solutions for its (their) correction.

A grammar formalism for describing the movement of syntactic constituents in a sentence and for taking into account the barriers that those constituents should not cross was used. It was named GMA after the Portuguese name Gramáticas de Movimentação e Ancoragem.

The correcting device considers the obtained chart and the hypothesis formulated for correcting the detected failure(s). It accepts corrections that take a minimum number of errors.

A simple spelling corrector based on Damerau's [Dam 64] typified errors is invoked for every nonword with a syntactic category attributed. This spelling checker uses a set of rules that takes into account preferences based on possible phonetic confusions. It ranks alternative corrections and treats also the case where the written word is a new word not known by the system.

.

Abreviaturas

cap. - capítulo

et al. (et allii) - entre outros

d - determinativo

ex. - exemplo

f - frase

fig. - figura

n - nome comum

np - nome próprio

nº - número

orel - oração relativa

pag. - página

prep - preposição

pron - pronome

PLN - Processamento de língua natural

sn - sintagma nominal

sv - sintagma verbal

sp - sintagma preposicional

v - verbo

Glossário

acceptance-based - baseadas em consentimento

bigrams - bigramas

case frames - enquadramentos baseados em casos

chart-parser - analisador por grafo

collocation - colocação

context free grammars - gramáticas independentes do contexto

context sensitive grammars - gramáticas dependentes do contexto

frames - enquadramentos

fitting parser - analisador por encaixe

expectation-based - baseadas em expectativas

parse tree - árvore de análise ou árvore sintáctica

relaxation-based - baseadas em relaxamento

scanner - reconhecedor óptico de escrita

slots - entradas

triphones - trifones

trigrams - trigramas

wh-question - interrogativa-qu

Simbologia

$e \in B$ - e é elemento do conjunto B

$A \neq B$ - A é diferente de B

$A = B$ - A é igual a B

$A \cap B$ - intersecção dos conjuntos A e B

$A \cup B$ - união dos conjuntos A e B

$\varnothing$ - conjunto vazio

$A \geq B$ - A é maior ou igual a B

A^* - conjunto de todas as cadeias de A (incluindo a cadeia de comprimento nulo)

ε - cadeia vazia

Índice de matérias

Introdução

O avanço das novas tecnologias permitindo, por um lado a construção de computadores cada vez mais potentes e por outro, a utilização de unidades para entrada e saída de dados cada vez mais sofisticadas onde se destacam, por ex., a utilização de reconhecedores ópticos de escrita e canetas para introdução de dados e de unidades de voz para entrada/saída de dados, obriga ao desenvolvimento de ferramentas que auxiliem a construção rápida de sistemas que requeiram o uso de línguas naturais (no nosso caso, o Português).

As áreas de aplicação destes sistemas são, entre outras:

- Correctores morfo-sintáctico-semânticos,
- Construção de interfaces de língua natural
- Construção de sistemas automatizados para aprendizagem de línguas naturais
- Reconhecimento de texto manuscrito ou impresso
- Reconhecimento e síntese de fala
- Tradução auxiliada por computador
- Etc., etc., etc.

Nestes sistemas, a categorização de uma sequência de caracteres como uma frase ou sequência de frases, será conseguida recorrendo a ferramentas que também permitam detectar e corrigir erros[1].

Os primeiros sistemas construídos com esse objectivo incidiam sobre erros existentes em palavras isoladas (caso dos correctores ortográficos incorporados na maior parte dos processadores de texto existentes no mercado) não considerando, no entanto, o contexto em que as palavras se situavam. Isso acarretava:

- a não correcção de palavras que, por existirem no léxico, são consideradas correctas (ex: nós, noz; nos e nus; como) embora sintáctica e semanticamente possam não fazer sentido na posição onde se encontram;

[1]Doravante, em vez da expressão *sequência de caracteres* passarei a utilizar a expressão *sequência de palavras* onde *palavra* será utilizada independentemente da palavra estar bem ou mal escrita ou ser desconhecida.

- a impossibilidade do estabelecimento de preferências nas opções para uma palavra mal escrita. Por ex., as propostas de correcção para ve, poderiam ser, entre outras, {ave, vê, vês, vez}.

Segundo Kukich [Ku 92], os poucos estudos existentes sobre ocorrência de erros em textos ingleses, indiciam que 25 a 50% dos erros cometidos num texto correspondem a palavras existentes no léxico e, 75% destes, são ocasionados por violações sintácticas.

Estes estudos, ainda que não conclusivos, justificam a necessidade da utilização do contexto onde as palavras se situam para construir sistemas que permitam a superação contextual de erros morfo-sintáctico-semânticos.

Por outro lado, é necessária uma maior pesquisa nessa área (no que se refere ao português) pois, dos estudos sobre a ocorrência, caracterização e tipificação de erros, depende o aperfeiçoamento dos sistemas que os pretendam superar.

Nesta tese, que tem como base a língua portuguesa, o sistema construído, cuja aplicação foi limitada à ocorrência de alguns erros morfo-sintácticos ou de desconhecimento em frases de documentos escritos, utiliza a análise por grafo e recorre a uma gramática de movimentação e ancoragem (Lopes [Lop 92]) para detecção de irregularidades.

A tese está dividida em duas partes. Na primeira parte trato do enquadramento e delimitação do tema proposto. Na segunda parte descrevo o sistema construído e analiso os resultados que obtive.

A primeira parte está dividida em três capítulos, respectivamente, os capítulos um, dois e três. e a segunda parte é constituída por mais dois capítulos, respectivamente, os capítulos quatro e cinco.

No primeiro capítulo, são introduzidas algumas noções linguísticas necessárias à delimitação da temática e são apresentadas as gramáticas de movimentação e ancoragem (Lopes [Lop 92]) e as técnicas para pesquisar constituintes de frases, detalhando a análise por grafo e a sua aplicação às gramáticas de movimentação e ancoragem (Paulo [Pau 93]).

No segundo capítulo, apresenta-se uma panorâmica sobre os diferentes tipos de sistemas de detecção e correcção de erros já construídos noutras línguas, com incidência nos sistemas baseados em relaxamento, pois este será o modelo adoptado para o sistema construído.

No terceiro capítulo são descritos os vários tipos de erros morfo-sintácticos e o tipo de desconhecimento tratados e é apresentada a metodologia que foi utilizada para a construção do sistema desenvolvido.

Na segunda parte e no quarto capítulo, utilizando a descrição dos tipos de erros tratados, procede-se à descrição detalhada dos módulos desenvolvidos para formulação de hipóteses e para detecção e correcção de erros e das adaptações efectuadas na gramática de movimentação e ancoragem de Lopes [Lop 92] e no analisador por grafo de Paulo [Pau 93] para satisfazerem as necessidades do sistema desenvolvido.

No quinto capítulo faz-se a análise e interpretação dos resultados obtidos, tendo em conta que, por não existirem neste momento quaisquer sistemas construídos usando a mesma ou outras técnicas para a superação de erros morfo-sintácticos em Português, os testes de eficiência limitam-se a uma descrição de tempos de execução para cada frase testada e tempos comparativos quando é introduzida uma nova regra na gramática.

Finalmente, no capítulo das conclusões, após ser feita uma resenha dos tipos de frases e erros tratados, são analisadas as situações não contempladas, alguns problemas encontrados e algumas linhas de trabalho possível para o aperfeiçoamento dum sistema do tipo proposto.

I - Conceitos necessários à delimitação e compreensão do tema

1

Conceitos necessários à delimitação e compreensão do tema

Neste capítulo caracteriza-se a norma utilizada pelo sistema construído e apresentam-se alguns conceitos necessários à compreensão do tema desta tese.

Assim, define-se o conceito de gramática e apresentam-se as gramáticas de cláusulas definidas (Pereira e Warren [PW 80]) e as gramáticas de movimentação e ancoragem (Lopes [Lop 92]).

Finalmente, descrevem-se as diferentes técnicas utilizadas para pesquisa de constituintes incidindo especialmente sobre os analisadores por grafo e a sua adaptação a uma gramática de movimentação e ancoragem (Paulo [Pau 93]).

1.1 Língua e correcção

A língua, sendo condicionada pelo espaço geográfico, pela camada sociocultural dos falantes e pelo tipo de modalidade expressiva (língua falada, escrita, etc.), entre outros factores, admitirá várias normas, ou seja, várias noções de correcção (Cunha e Cintra [CuC 85] e Mateus et al. [MBDF 89]).

Assim, face a uma norma, considera-se que uma frase não é correcta se contiver desvios à norma, ou seja, erros ou incorrecções.

Sendo objectivo desta tese a detecção e correcção de alguns tipos de erros morfo-sintácticos e de desconhecimento, teremos que considerar a norma, descrita por uma gramática que integra um conjunto de regras que definem as categorias sintácticas existentes num dado domínio aplicativo da língua e um léxico que caracterizará cada palavra existente nesse domínio.

Sendo a norma condicionada pelo domínio de aplicação restringiu-se o sistema construído à escrita de documentos e considerou-se como norma-padrão a que é adoptada e difundida nas escolas.

É de salientar que a gramática construída para ser utilizada neste protótipo não descreve a globalidade das estruturas sintácticas do Português.

1.2 Representação de gramáticas

Dada uma descrição ou gramática de uma língua, um sistema de detecção e correcção de erros necessitará de um formalismo que permita a descrição formal das estruturas permitidas na língua.

1.2.1 Gramáticas formais

Uma gramática formal (G) é um quádruplo:

G = (N,T,S,P), onde N - conjunto de símbolos não terminais
T - conjunto de símbolos terminais
S - símbolo inicial
$N \cap T = \varnothing$ e $S \in N$
P - conjunto de regras de formação, do tipo α --> β, onde $\alpha \neq \varepsilon$, $\alpha, \beta \in (N \cup T)^*$ e α contém pelo menos um símbolo de N

Dada uma gramática é possível:

- gerar, por derivação, todas as cadeias de terminais que podem ser classificadas numa dada categoria
- analisar uma cadeia de caracteres, ou seja, tentar atribuir-lhe uma categoria

Por ex., dada a gramática da Fig. 1.1, a derivação de 'A Maria comeu um bolo' é a sequência:

f => sn sv => d np sv => **a** np sv => **a Maria** sv => **a Maria** v sn => **a Maria comeu** sn => **a Maria comeu** d n => **a Maria comeu um** n => **a Maria comeu um bolo**

N = {f, sn, sv, d, np, n, v}
T = {a, um, Maria, bolo, comeu}
S = f
P= {f -->sn sv, sn-->d np, sn-->d n,
sv-->v sn, d-->a, d-->um, n-->bolo,
np-->Maria, v-->comeu}

Fig. 1.1 Exemplo de Gramática

A seta (=>) deve ler-se: 'aplicando uma das regras da gramática obtém-se'

Neste tipo de gramáticas (independentes do contexto) o lado esquerdo das regras é composto por um único não terminal e a análise de uma sequência de palavras originará uma árvore sintáctica (Fig. 1.2) representando os seus constituintes.

Repare-se que, no âmbito de uma língua natural:

- o símbolo inicial é a categoria sintáctica de nível mais alto de uma dada gramática
- os símbolos não terminais são categorias sintácticas
- os símbolos pré-terminais são categorias sintácticas que não podem ser decompostas noutras
- os símbolos terminais são palavras

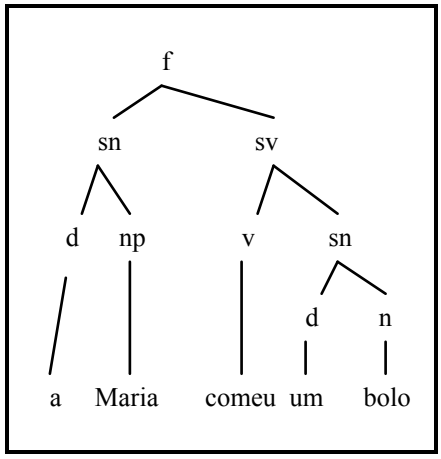

Fig. 1.2 Árvore sintáctica

Estas gramáticas são bastante potentes para descrever, duma forma clara e modular, a maior parte da estrutura de uma língua natural e suficientemente restritas para que a pesquisa de constituintes duma sequência de palavras seja eficaz.

No entanto, se considerarmos, na gramática anterior:

$T = T \cup \{uns\}$

$P = P \cup \{d \text{ --> } uns\}$

esta geraria, entre outras, 'A Maria comeu uns bolo', que, dentro da norma assumida, consideramos incorrecta. Donde, para evitar este tipo de incorrecções algumas regras teriam que ser sujeitas a restrições relacionadas com o contexto.

1.2.2 Gramáticas de cláusulas definidas

Neste tipo de gramáticas, consideradas como uma extensão das gramáticas independentes do contexto (Pereira e Warren [PW 80]), as regras são do tipo

A --> β, onde A é um único não terminal

β é uma cadeia de símbolos terminais e/ou não terminais que pode conter restrições cuja resolução pode ser entendida ou como invocação de procedimentos ou como invocação de uma tese que terá que ser demonstrada

Estas gramáticas, constituem um formalismo mais poderoso que o das gramáticas independentes do contexto, pois:

- poder-se-á ter em conta o contexto em que determinados constituintes ocorrem numa cadeia
- durante a análise, poder-se-ão construir as árvores de análise que não são condicionadas pela estrutura recursiva da gramática
- permitindo restrições em regras pode condicionar-se a pesquisa através de procedimentos auxiliares

Na Fig. 1.3 é apresentada a codificação em Prolog de uma gramática muito simples de cláusulas definidas onde se exemplifica a utilização de restrições para detectar as concordâncias em género e número entre os constituintes de uma frase.

Note-se que:

- Os símbolos adjacentes da parte direita da regra são separados por uma vírgula (,)
- A seta e '-->'
- As restrições são escritas entre chavetas ({})
- Os terminais são definidos através de listas
- Esta gramática, quando executada, comportar-se-á como um analisador descendente

```
f(N,G) -->  sn(N1,G),sv(N2),{concorda (N1,N2, N)}.
sn(N,G)--> d(N1,G1),np(N2,G2),
            {concorda(N1,N2,N),concorda(G1,G2,G)}.
sn(N,G) --> d(N1,G1), n(N2,G2),
            {concorda(N1,N2,N),concorda(G1,G2,G)}.
sv(N) --> v(N), sn(_,_).
d(sin,fem) --> [a].
d(sin,masc) --> [um].
np(sin,fem) --> [maria].
n (sin,masc)--> [bolo].
v(sin) --> [comeu].
concorda(X,X,X).
```

Fig. 1.3 Gramática de cláusulas definidas (Prolog)

1.2.3 Gramáticas de movimentação e ancoragem (GMAs)

As gramáticas de movimentação e ancoragem (Lopes [Lop 92]) são uma extensão do formalismo das gramáticas de extraposição (Pereira [Per 81]). As GMAs são caracterizadas como a seguir se enuncia.

Do lado esquerdo das regras, para além dos termos possíveis em gramáticas de cláusulas definidas, também podem existir termos do tipo:

> ∅ ***Movimento*** γ
>
> sendo ∅, γ símbolos não terminais da gramática e ***Movimento*** um operador que identifica o tipo de movimentação do material linguístico e fixa, de algum modo, a extensão desse movimento.

Numa GMA este operador pode ser instanciado com um dos valores seguintes:

rel - se ∅ for identificado como resultado de movimentação de material linguístico em orações relativas. Este operador refere-se especificamente ao sintagma nominal ou preposicional que encabeça uma oração relativa.

slash - se ∅ for identificado como resultado de um movimento de topicalização. Este operador refere-se tipicamente à topicalização de sintagmas preposicionais, núcleos verbais, orações subordinadas, etc.

quest - se ∅ for identificado como resultado de um movimento de interrogação. Este operador refere-se tipicamente à movimentação de sintagmas nominais e sintagmas preposicionais interrogativos.

Nota: Podem ainda ser definidos outro tipos de movimentação em frases portuguesas, nomeadamente, a movimentação de clíticos.

As regras que contêm estes termos são do tipo

> ∅ ***Movimento*** γ --> α onde α é uma cadeia de símbolos (pelo menos um) da gramática.

Estas regras descrevem situações em que um constituinte de categoria sintáctica α é contextualmente categorizado como ∅ e o material linguístico correspondente é reclassificado com a categoria γ e movimentado até que, na estrutura sintáctica padrão, seja encontrado um vestígio também de categoria γ. Nessa altura, o constituinte movimentado é ancorado. (Consultar o exemplo ilustrado nas Fig. 1.4 e Fig. 1.5)

A realização desta operação é implementada recorrendo a canais específicos para cada tipo de movimentação que são designados por canais ***rel***, canais ***slash*** e canais ***quest***.

Ex: A regra simplificada *comp* ***slash*** *v* --> *v* significa que um verbo pode aparecer fora do seu local habitual na frase que é, normalmente, a seguir ao sintagma nominal que desempenha o papel de sujeito.

Nessas circunstâncias o verbo é analisado e será transportado nos canais ***slash*** até que a sua presença seja requerida na estrutura sintáctica padrão do Português (*f -> sn, sv*).

Do lado direito das regras de uma GMA, para além dos termos possíveis em gramáticas de cláusulas definidas, também podem existir termos do tipo:

Tipo_barreira ∅

significando que "para a análise de um constituinte da categoria não terminal ∅ (dado que numa gramática de movimentação e ancoragem há possibilidade da movimentação de material linguístico a distâncias não pré-fixadas) é impedida a importação e exportação de material linguístico através dos canais relativos àquele tipo de *barreira*"

Tipo_barreira é operador que pode ter os valores seguintes:

> ***ilha*** - impede a importação e exportação de material linguístico via todo o tipo de canais
>
> ***ilha_slash*** - impede a importação e exportação de material linguístico via canais ***slash***
>
> ***ilha_relquest*** - impede a importação e exportação de material linguístico via canais ***rel*** e canais ***quest***
>
> ***ilha_slashquest*** - impede a importação e exportação de material linguístico via canais ***slash*** e canais ***quest***
>
> ***ilha_rel*** - impede a importação e exportação de material linguístico via canais ***rel***

As regras que contêm termos deste tipo condicionarão a ancoragem de material linguístico da categoria γ existente num determinado canal a determinadas localizações na estrutura sintáctica padrão.

Ex: A existência de uma regra do tipo

> *f --> **ilha_slashquest** sn, sv.*

numa gramática de movimentação e ancoragem (representação simplificada da estrutura sintáctica padrão do Português) impossibilita a ancoragem, para desempenhar a função de sujeito da frase, de qualquer material linguístico já existente nos canais ***slash*** e nos canais ***quest***. Também impede a exportação de material linguístico através daqueles canais.

Na Fig. 1.4 são apresentadas algumas regras de uma gramática de movimentação e ancoragem muito simplificada que permitiria analisar a frase 'ardeu o livro' cuja árvore sintáctica está representada na Fig. 1.5. Assim:

```
f ---> ilha_slashquest sn, sv.
f ---> comp, f.
comp slash v ---> ilha v.
sn ---> ilha d, ilha n.
sv ---> ilha_relquest v, args.
args---> [].
...
```

Fig. 1.4 GMA muito simplificada

- A primeira regra não permite a análise de 'ardeu' porque não o conseguimos classificar como um sintagma nominal.
- Por aplicação da segunda e terceira regras 'ardeu' é categorizado como um verbo
- Por aplicação da terceira regra essa informação é guardada no canal ***slash*** até que o processo de análise permita a sua ancoragem
- a existência de uma ***ilha_slashquest*** *sn* impedirá a importação do material existente nos canais ***slash*** e nos canais ***quest*** (neste caso apenas existe o verbo 'arder' nos canais ***slash***) e permitirá a análise de 'o livro'
- a existência de uma ***ilha_relquest*** *v* permitirá a importação/ancoragem do verbo já analisado e guardado no canal de ***slash*** determinando, assim, a inserção do verbo na estrutura sintáctica canónica da frase

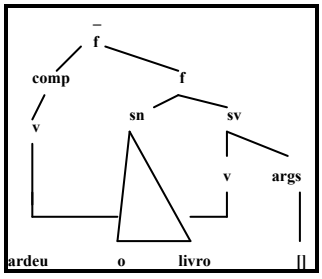

Fig. 1.5 Árvore sintáctica simplificada

Na Fig. 1.6 são apresentadas algumas regras de uma gramática de movimentação e ancoragem muito simplificada que permitiria analisar a frase interrogativa 'a quem deu o João os livros' cuja árvore sintáctica é apresentada na Fig. 1.7. Assim:

```
r1:f ---> ilha_slashquest sn, sv.
r2:f---> perg,f.
r3:perg quest sp ---> sp(i).
r4:f ---> comp, f.
r5:comp slash v ---> ilha v.
r6:sn ---> ilha d, ilha np.
r7:sn ---> ilha d, ilha n.
r8:sp(i)--->ilha p,ilha pron(i).
r9:sv ---> ilha_relquest v, args.
r10:args---> sn,sp.
...
```

Fig. 1.6 GMA muito simplificada

- A regra **r1** não permite a análise de 'a quem' porque não o conseguimos classificar como um sintagma nominal.
- Por aplicação das regras **r2**, **r3** e **r8** 'a quem' é classificado como um *sp* interrogativo
- Por aplicação da regra **r3** é guardado um *sp* no canal ***quest*** até que o processo de análise permita a sua ancoragem
- As regras **r1**, **r2, r3** e **r8** não permitem a análise de 'deu' porque não o conseguimos classificar nem como um sintagma nominal (**r1**) nem como um sintagma preposicional (**r2**, **r3** e **r8**).

- Por aplicação das regras **r4** e **r5** 'deu' é categorizado como um verbo
- Por aplicação da regra **r5** essa informação é guardada no canal ***slash*** até que o processo de análise permita a sua ancoragem
- a existência de uma ***ilha_slashquest*** *sn* na regra **r1** impedirá a importação de material existente nos canais ***slash*** e nos canais ***quest*** (neste caso existe o verbo 'dar' nos canais ***slash*** e o sintagma preposicional 'a quem' nos canais de ***quest***) e permitirá a análise de 'o joão' (regras **r1** e **r6**)
- a existência de uma ***ilha_relquest*** *v* (regra **r9**) permitirá a importação/ancoragem do verbo já analisado e guardado no canal de ***slash*** determinando, assim, a inserção do verbo na estrutura sintáctica canónica da frase
- Por aplicação das regras **r10** e **r7** a sequência de palavras 'os livros' é categorizada como um *sn* (complemento do verbo).

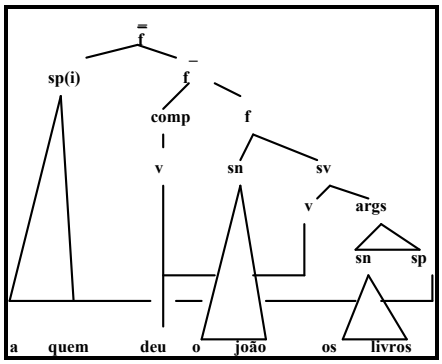

Fig. 1.7 Árvore sintáctica simplificada

- Ainda por aplicação da regra **r10** é permitida a importação/ancoragem do sintagma preposicional já analisado e existente no canal ***quest*** determinando assim, a inserção do sintagma preposicional, com função de complemento indirecto, na estrutura sintáctica canónica da frase.

Note-se que estas gramáticas necessitam um interpretador específico, pois não são interpretadas directamente pelo Prolog.

1.3 Analisadores

Dada uma representação da estrutura de uma língua e uma sequência de palavras correspondendo ao seu símbolo inicial, é tarefa do analisador a decomposição dessa sequência nos seus constituintes não terminais, obtendo assim uma árvore de constituintes correspondente à estrutura dessa sequência.

1.3.1 Estratégias de pesquisa

As estratégias de pesquisa de constituintes utilizadas em analisadores são, para línguas europeias, essencialmente efectuadas da esquerda para a direita podendo, no entanto combinar vários graus de profundidade/largura com estratégias ascendentes/descendentes.

1.3.1.1 Pesquisas em profundidade/largura

Um analisador, no caso da pesquisa em profundidade, tendo várias hipóteses para um constituinte, segue uma das hipóteses e, só em caso de falha, repete a pesquisa utilizando a segunda hipótese e assim sucessivamente, até obter a solução ou não existirem mais hipóteses a explorar.

No caso da pesquisa em largura, o analisador pesquisa todas as hipóteses a um dado nível e, só quando não existirem aí mais hipóteses, avança para o nível de profundidade seguinte.

1.3.1.2 Pesquisas ascendentes/descendentes

Um analisador que utilize uma estratégia ascendente inicia a pesquisa ao nível da categoria de cada palavra numa frase que depois irá agrupar-se com outras categorias construindo novas categorias até atingir a categoria final pretendida, no nosso caso, a frase.

Este tipo de estratégia pode ser pouco eficaz (se não houver cuidados redobrados com a eficiência) pois pode originar a proliferação de estruturas que não serão depois utilizadas por corresponderem a categorizações de palavras que ocupam posições onde não podem ocorrer.

No caso da estratégia ser descendente, o analisador inicia a pesquisa na representação da estrutura de mais alto nível (símbolo inicial da gramática) que vai decompondo nos seus constituintes até chegar ao nível da categoria pré-terminal que será então testada com a categoria pré-terminal que categoriza a palavra existente na frase.

No caso de gramáticas complexas este tipo de estratégia poderá ser pouco eficiente pois, como as falhas serão muitas, a derivação de regras será muito elevada.

Note-se que, existem uma infinidade de estratégias de pesquisa que resultam da combinação de estratégias do tipo esquerda/direita, com ascendentes/descendentes e profundidade/largura.

1.3.2 Analisadores por grafo

Estes tipo de analisadores (Gazdar e Mellish [GM 89], Allen [AL 87] e Rosenblueth [Ros 89]), concebidos para gramáticas independentes do contexto, são caracterizados pela existência de um grafo para armazenamento das sub-análises já realizadas, evitando a repetição das mesma e permitindo que, em caso de falha, possa ser detectada com maior facilidade a causa da mesma auxiliando, assim, o processo de reparação.

Os estados do processo da pesquisa dos constituintes de uma frase, ou seja, as sub-análises já efectuadas serão traduzidas por **arcos**:

- **arcos completos** representam a análise completa de constituintes. São descritos nesta tese pelo facto

  ac(U1,U2,Cat)

  onde *U1* e *U2* são os vértices inicial e final da sequência de palavras que constituí a categoria e *Cat* é a categoria cuja análise já foi concluída

- **arcos activos** representam a análise parcial de um dado constituinte. São descritos nesta tese pelo facto

  aa(U1,U2,Cat,Cat1,[Cat2,...,Catn])

  onde *Cat* é a categoria a ser analisada, *U1 e U2* são os vértices inicial e final da sequência de palavras já analisada como sendo parte da categoria *Cat*, *Cat1* é a primeira categoria a ser pesquisada e *[Cat2,...,Catn]* é a lista do resto das categorias a completar

Note-se que o lado direito das regras constituintes duma gramática sujeita a uma análise por grafo é reescrito sob a forma de lista (Ex. da Fig. 1.8).

Ex: Dada a gramática simplificada da Fig. 1.8 (Consultar também a Fig. 1.9), o resultado da análise de 'a maria comeu um bolo' seria representada pelo arco completo:

ac(1,6,f(f(sn(d(a),np(maria)),sv(v(comeu),sn(d(um),n(bolo))))))

Uma sub-análise não completa de *f* corresponderia, por ex., ao arco activo:

aa(1,3,f(f(sn(d(a),np(maria)),Y)),sv(Y),[])

que expressa a existência de um sn categorizando 'a maria' e a expectativa de análise de um *sv* para que a análise da categoria *f* possa ser completada.

```
f (f(X,Y)) --->  [sn(X), sv(Y)].
sn(sn(X,Y)) ---> [d(X), np(Y)].
sn(sn(X,Y)) ---> [d(X), n(Y)].
sv(sv(X,Y)) ---> [v(X), sn(Y)].
d(d(a)) ---> [a].
d(d(um)) ---> [um].
np(np(maria)) ---> [maria].
n(n(bolo)) --->  [bolo].
v(v(comeu)) ---> [comeu].
```

Fig. 1.8 Gramática simplificada

As sub-análises parciais e completas já efectuadas serão constituintes do **grafo** que é do tipo:

Grafo = As/Cs

onde *As* é a lista constituída por todas as análises parciais já efectuadas (arcos activos) e *Cs* é a lista constituída por todas as análises completas já efectuadas (arcos completos)

Inicialização

O algoritmo base da análise por grafo pressupõe a inicialização de algumas estruturas. Assim:

- *Cs* é inicializado com os arcos completos resultantes da categorização sintáctica das palavras e *As* é inicializado com a lista vazia ([]).

  Por ex., dada a gramática da Fig. 1.8 e a sequência de palavras 'a maria comeu um bolo' o **grafo inicial** (Fig. 1.9) será

  []/[ac(1,2,d(d(a))),ac(2,3,np(np(maria))),ac(3,4,v(v(comeu))),ac(4,5,d(d(um))),ac(5,6,n(n(bolo)))]

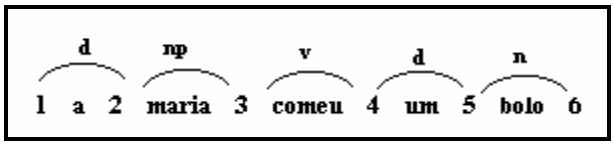

Fig. 1.9 Grafo correspondente à análise morfológica da frase

- **Propõe-se**, no vértice inicial da sequência de palavras, uma lista de **arcos activos** atribuídos ao símbolo inicial da gramática.

  Esta lista de arcos propostos, considerada **lista de arcos pendentes iniciais**, permitirá o desencadeamento da análise por grafo de acordo com o **algoritmo base** enunciado na pag. 26.

Proposta de arcos

A proposta de arcos para uma dada categoria *Cat* e para um dado vértice *U* consiste na procura de todas as regras gramaticais encabeçadas por *Cat* e na consequente criação de uma lista constituída por tantos arcos quantas as regras gramaticais encabeçadas por *Cat.*

Assim, para cada regra encabeçada por *Cat* cujo lado direito é vazio, cria-se um **arco completo**

- com categoria *Cat*
- vértices inicial e final iguais ao vértice dado (*U*)

e, para cada regra encabeçada por *Cat* cujo lado direito não é vazio, cria-se um **arco activo**

- com categoria *Cat*
- vértices inicial e final iguais ao vértice dado (*U*)

- com primeira categoria em resto igual à primeira categoria do lado direito da regra
- com resto das categorias a completar igual à lista das restantes categorias existentes no lado direito da regra.

Por ex., dada a gramática da Fig. 1.8, a lista de **pendentes inicial** será constituída pelo arco activo ***aa(1,1,f(f(SN,SV)),sn(SN),[sv(SV)])***.

Expansão de um arco num vértice

A expansão de um arco num dado vértice consiste na **proposta** de novos arcos para a **primeira categoria em resto**[2] desse arco.

Por ex: dada a gramática da Fig. 1.8 obter-se-ão como resultado da expansão do arco activo *aa(1,1,f(f(SN,SV)),sn(SN),[sv(SV)])* no vértice **1** os arcos

- *aa(1,1,sn(sn(X,Y)),d(X),[np(Y)])*
- *aa(1,1,sn(sn(X,Y)),d(X),[n(Y)])*

A expansão de arcos num dado vértice tem que ser condicionada para evitar, não só a repetição desnecessária de operações já efectuadas como também, a proliferação de arcos repetidos no grafo.

Consequentemente, só é efectuada a expansão de um arco num vértice se:

- o arco for activo
- a categoria a expandir ainda não tiver sido proposta para expansão nesse vértice.

Uma das formas de conseguir realizar esta operação é utilizando uma estrutura de armazenamento auxiliar (lista) que contenha todas as **categorias já propostas** em vértices, onde cada **categoria proposta** num vértice será armazenada como:

- *Cat/U1* onde *Cat* denota a categoria expandida e *U1* o vértice onde foi realizada a expansão

No ex. anterior, a **lista inicial de categoria já propostas** será constituída por ***f(f(SN,SV))/1***.

[2]Note-se que só é possível **propôr** novos arcos para categorias não terminais (ver **proposta de arcos**)

Algoritmo base para a análise por grafo

O **algoritmo base** para a análise por grafo (Gazdar e Mellish [GM 89], Allen [AL 87] e Rosenblueth [Ros 89]) é o seguinte:

1. Escolhe-se um **arco pendente** e tenta-se combiná-lo com todos os arcos do grafo. Este processo, designado por **incorporação**, pode originar novos arcos activos ou completos que são adicionados à lista de pendentes.

2. O arco escolhido em 1. é adicionado ao grafo.

3. Se o arco escolhido é activo **expande-se** originando novos arcos activos que serão adicionados à lista de pendentes.

4. Repete-se o processo de 1 a 4 até se conseguir obter um arco completo da categoria inicial ou, até não existirem mais arcos pendentes.

Note-se que, na pesquisa de constituintes, são utilizadas as estratégias referidas anteriormente, ou seja, combinações de largura/profundidade com pesquisas ascendentes/descendentes.

A estas estratégias são, por vezes, adicionadas outras que conduzam à preferência de uns ramos face a outros, (por ex., o conhecimento sobre as estruturas mais utilizadas num dado domínio de aplicação da língua) tentando suprir a principal desvantagem da utilização de um grafo, o espaço ocupado pelo mesmo.

Incorporação

Dados dois arcos:

um arco activo *A = aa(U1,U2,Cat,ACat,Resto_Cats)*,

e um arco completo *B = ac(U3,U4,BCat)*

o arco activo *A* **incorpora** o arco completo *B* se e só se *U2 = U3* e *ACat = BCat*.

Nessas condições obtêm-se:

- Ou um arco completo *ac(U1,U4,Cat)* se *Resto_Cats = []*
- Ou um arco activo *aa(U1,U4,Cat,Cat2,[Cat3,...,Catn])* se *Resto_Cats = [Cat2,Cat3...Catn]*

Por ex., dada a gramática da Fig. 1.8 e a sequência de palavras 'a maria comeu um bolo' ao incorporar:

1. o arco completo *ac(1,2,d(d(a)))* no arco activo *aa(1,1,sn(sn(X,Y)), d(X),[np(Y)])* obtém-se o arco activo *aa(1,2,sn(sn(d(a),Y)),np(Y),[])*

2. o arco completo *ac(2,3,np(np(maria)))* no arco activo *aa(1,2,sn(sn(d(d(a)),Y)), np(Y),[])* obtém-se o arco completos *ac(1,3,sn(sn(d(a),np(maria))))*

1.3.2.1 Analisador por grafo para GMAs simples

Uma adaptação do analisador por grafo às gramáticas de movimentação e ancoragem foi proposta por Paulo ([Pau 93]).

Para a utilização deste analisador Paulo ([Pau 93]) propôs que, na gramática de movimentação e ancoragem utilizada (Lopes [Lop 92]), as regras fossem prefixadas com um identificador permitindo, em cada instante, saber qual a regra utilizada.

A Fig. 1.10 ilustra uma gramática muito simples de movimentação e ancoragem onde cada regra é prefixada com um identificador e as restrições permitem o tratamento da concordância em género e número entre os constituintes duma frase.

Ex: *r4:sn(N,G,sn(X,Y))--->*
[ilha d(N1,G1,X),ilha np(N2,G2,Y),
{concorda(N1,N2,N),concorda(G1,G2,G)].

Esta regra, identificada como *r4*, determina que um sintagma nominal (*sn*) poderá ser constituído por um determinativo X e um nome próprio Y devendo existir concordância em género e número entre os seus constituintes.

```
r1:f (f(X,Y)) ---> [ilha_slashquest sn(N1,_,X),
                    sv(N2,Y),{concorda(N1,N2,N)}].
r2:f(F) ---> [comp,f(F)].
r3:comp slash nv(N,V) ---> [ilha nv(N,V)].
r4:sn(N,G,sn(X,Y)) --->
   [ilha d(N1,G1,X),ilha np(N2,G2,Y),
    {concorda(N1,N2,N),concorda(G1,G2,G)}].
r5:sv(N,sv(X,Y)) ---> [ilha_relquest nv(N,X),
                       args(Y)].
r6:nv(N,X) --->  [ilha v(N,X)].
r7:args([]) ---> [].
d(_,_,d([])) ---> [].
d(sin,fem,d(a)) ---> [a].
np(sin,fem,np(maria)) ---> [maria].
v(sin,v(comeu)) ---> [comeu].
concorda(X,X,X).
```

Fig. 1.10 Gramática muito simples de movimentação e ancoragem

Neste analisador que opera basicamente de acordo com o algoritmo enunciado na secção anterior, são introduzidas algumas alterações para poder ser utilizado sobre gramáticas de movimentação e ancoragem, ou seja, para poder tratar a existência de:

- categorias prefixadas
- infixos de movimentação
- restrições.

A) Alterações nas estruturas de armazenamento

Pretendendo operar com movimentações de material linguístico os arcos terão que incluir a informação relativa ao material linguístico existente nos canais. Assim:

- Os **arcos completos** serão do tipo *ac(U1,U2,Rg:Cat,**Q**,**S**,**R**,**V**)*

  onde ***Q***, ***S***, ***R***, ***V*** são diferenças de listas traduzindo a existência de material linguístico contido nos canais de movimentação de material linguístico interrogativo (***Q*** e ***V***), das topicalizações (***S***) e das relativas (***R***).

  Note-se que, sendo os sintagmas nominais interrogativos tratados como qualquer outro sintagma, a identificação, após a análise de uma frase interrogativa-qu, será conseguida através da detecção dos referentes interrogativos que foram guardados no canal ***V***[3].

  Rg, identificador da regra, pode ainda ser denotado com:

  pal, se o arco completo resultar da consulta do léxico referente a uma categoria concretizada obtendo arcos do tipo *ac(U1,U2,**pal**:Cat,Q,S,R,V)* onde *Cat* denota um pré-terminal concretizado

  lambda, se o arco completo resultar da consulta do léxico referente a uma categoria não concretizada obtendo arcos completos do tipo *ac(U,U,**lambda**:Cat,Q,S,R,V)* onde *Cat* denota um pré-terminal não concretizado (correspondendo, por ex., à ausência de determinativo)

  restr, se o arco completo resultar da expansão de uma restrição obtendo arcos do tipo *ac(U,U,**restr**:{Restrição},Q,S,R,V)* (ver secção referente às **Alterações à proposta de arcos**)

[3]Isto permite, inclusivé, a detecção de múltiplas perguntas numa única frase. Por ex: 'Quem deu o quê a quem?'

slash*, *quest ou ***rel*** se o arco completo resultar da expansão de um arco cuja primeira categoria em resto é **γ** e cujos canais (***slash*, *quest*** ou ***rel)*** transportem um constituinte também de categoria **γ**. (ver secção referente às **Alterações à proposta de arcos**).

Note-se que este arco completo traduz a **ancoragem de um constituinte γ na estrutura sintáctica padrão** (ver **GMAs** na secção **1.2.3** deste capítulo).

- Os **arcos activos** serão do tipo *aa(P1,P2,Rg:Cat,Cat1,[Cat2,...,Catn],**Q**,**S**,**R**,**V**)* onde *Rg* denotará somente o identificador da regra utilizada

A **lista das categorias já propostas para os vértices** incluirá, para além da categoria e do vértice a que se refere, a informação sobre o material linguístico existente nos canais da mesma. Assim, cada categoria já proposta para um vértice será denotada como:

*Cat/U1/**Q/S/R/V***

permitindo que, uma dada categoria *Cat* possa ser proposta para um mesmo vértice *U1* quando a informação existente nos canais é distinta.

O **grafo inicial** conterá:

1. Os arcos completos correspondentes a cada palavra numa dada categoria onde:
   - o vértice inicial e o final corresponderão às posições inicial e final da palavra
   - os canais serão não instanciados mas restringidos (parte esquerda e direita iguais) por uma possível movimentação de material linguístico indicado na regra.

     Nota1: As restrições dirão apenas respeito à ocorrência de pronomes interrogativos ou determinativos interrogativos que deverão ser guardados nos canais *V*.

     Nota2: A não instanciação dos canais será determinante no **processo de incorporação** (ver secção correspondente)
2. Os arcos completos correspondentes a cada uma das categorias que são derivadas do vazio onde:
   - o vértice inicial e final são iguais e são não instanciados permitindo maior versatilidade na sua futura **incorporação** (ver secção correspondente)

- a parte esquerda e direita dos canais de movimentação são não instanciadas mas restringidas (parte esquerda e direita iguais) para não permitirem ocorrência de movimentações de material linguístico (ver secção referente às **Alterações ao processo de incorporação**)

Por ex., dada a gramática da Fig. 1.10 e a frase 'a maria comeu' o **grafo inicial** (As/Cs) será:

[]/[ac(U,U, lambda:d(_,_,d([])),Q-Q,S-S,R-R,V-V), ac(1,2,pal:d(sin,fem,d(a)),Q1-Q1,S1-S1, R1-R1,V1-V1), ac(2,3,pal:np(sin,fem,np(maria)),Q2-Q2,S2-S2,R2-R2,V2-V2), ac(3,4,pal:v(sin,v(comeu)), Q3-Q3, S3-S3, R3-R3, V3-V3)]

B) Alterações ao processo de expansão

A expansão de um arco num vértice que consiste na **proposta** (ver secção referente às **Alterações à proposta de arcos**) de novos arcos para a primeira categoria em resto desse arco realizar-se-á, não só, se o arco for activo e se a sua primeira categoria ainda não tiver sido **proposta** para expansão nesse vértice (ver **processo de expansão** na secção **1.3.2**), mas também, se a sua primeira categoria for uma restrição que ainda não tenha sido **proposta** para esse vértice.

A adição de arcos aos pendentes é feita duma forma ordenada reflectindo uma busca preferencial em largura.

C) Alterações à proposta de arcos

Dada um arco activo ***aa(U1,U2,Rg:Cat,Cat1,[Cat2,...,Catn],Q-Q0,S-S0,R-R0,V-V0)*** a lista de arcos propostos para a categoria **não terminal *Cat1*** será constituída por:

- **arcos completos** do tipo ***ac(U2,U2,Rg1:Cat1,Q0-Q0,S0-S0,R0-R0,V0-V0)*** onde ***Rg1*** é o identificador de todas as regras encabeçadas por ***Cat1*** e cujo lado direito é vazio.

  Ex: Dada a gramática da Fig. 1.10, a frase 'comeu a maria' e o arco activo *aa(4,4, r5:sv(sin,sv(v(comeu),Y)), args(Y),[],[]-[],[nv(sin,v(comeu))]-**[]**,[]-[],[]-[]),* obter-se-á como **proposta** para a categoria ***args*** no vértice ***4*** o arco completo *ac(4,4,r7:args([]),[]-[],**[]-[]**,[]-[],[]-[])*

  Note-se que a **informação relativa a movimentações** existente na parte direita dos canais do arco a expandir deverá ser **propagada para os canais dos arcos resultantes da expansão** cujas partes esquerda e direita deverão ser iguais.

- **arcos activos** do tipo ***aa(U2,U2,Rg1:Cat1,Cat11,[Cat12...Cat1n],Q0-Q0,S0-S0,R0-R0,V0-V0)*** onde ***Rg1*** é o identificador de todas as regras encabeçadas por ***Cat1***, ***Cat11*** é a primeira categoria do lado direito da regra e ***[Cat12...Cat1n]*** as restantes categorias existentes do lado direito da regra.

  Ex: Dada a gramática da Fig. 1.10, a frase 'comeu a maria' e o arco activo *aa(1,2,r2:f(F),f(F),[],[]-[],[]-**[nv(sin,v(comeu))]**,[]-[],[]-[])*

  obter-se-ão como **propostas** para a categoria ***f*** no vértice ***2*** os arcos activos

  *aa(2,2,r1:f(f(X,Y)),ilha_slashquest sn(N1,G,X),[sv(N2,Y),{concorda(N1,N2,N)}],[]-[],**[nv(sin,v(comeu))]-[nv(sin,v(comeu))]**,[]-[],[]-[])*

  *aa(2,2,r2:f(F),comp,[f(F)],[]-[],**[nv(sin,v(comeu))]-[nv(sin,v(comeu))]**,[]-[],[]-[])*

  Note-se que, se *Cat1* traduzir movimentação de material, a proposta de arcos deve considerar as regras cujo **lado esquerdo** seja encabeçado por *Cat1* e obter-se-á um arco com *categoria* igual ao **lado esquerdo** da regra considerada.

  Ex: Dada a gramática da Fig. 1.10, a frase 'comeu a maria' e o arco activo *aa(1,1,r2:f(F),**comp**,[f(F)],[]-[],[]-[],[]-[] ,[]-[])* obter-se-á como **proposta** para a categoria ***comp***[4] no vértice ***1*** o arco activo *aa(1,1,r3:**comp slash nv(N,V)**,ilha nv(N,V),[],[]-[],[]-[],[]-[],[]-[])*

São também constituintes da lista dos propostos os **arcos completos** que correspondam à **ancoragem de um constituinte** de categoria **γ** por ter sido detectado, na estrutura sintáctica padrão, um vestígio também de categoria **γ**.

Estes arcos completos são prefixados por um identificador que referencia o tipo de movimentação efectuada (***slash***, ***rel*** ou ***quest***) e têm os seus canais iguais aos canais do arco activo com excepção da parte direita do canal relativo ao tipo de movimentação efectuada (donde será retirado o constituinte categorizado com γ).

Ex: Dada a gramática da Fig. 1.10, a frase 'comeu a maria' e o arco activo *aa(4,4,r5:sv(N, sv(V,Args)),ilha_relquest nv(N,V),[args(Args)],[],[]-[],[nv(sin,v(comeu))]**-[nv(sin,v(comeu))]**,[]-[],[]-[])* obter-se-ão como **propostas** para a categoria ***nv*** no vértice ***4*** o arco completo *ac(4,4,slash:nv(sin,v(comeu)),[]-[],[nv(sin,v(comeu))]**-[]**,[]-[],[]-[])* e o arco activo *aa(4,4, r6:nv(N,v(V)),v(N,V),[],[],[]-[],**[nv(sin,v(comeu))]-[nv(sin,v(comeu))]**,[]-[],[]-[])*

[4] ***comp*** é categoria não terminal e encabeça o lado esquerdo da regra **r3**

Se a proposta de novos arcos para um vértice incluir a categoria que corresponde ao conjunto de **restrições**, cria-se um conjunto de arcos completos correspondendo à categoria proposta com argumentos (restrições) instanciados onde:

- Os vértices iniciais e finais são iguais
- A informação relativa a movimentações existente na parte direita dos canais do arco a expandir é propagada para os canais dos arcos completos cujas partes direita e esquerda são iguais

Ex: Dada a gramática da Fig. 1.10 e a frase 'comeu a maria', a **expansão** do arco activo *aa(2,4,r4:sn(N,G,sn(d(a),np(maria))),{concorda(sin,sin,N),concorda(fem,fem,G)},[],[]-[],[nv (sin,v(comeu))]-**[nv(sin,v(comeu))]**,[]-[],[]-[])* **corresponderá à lista** constituída pelo arco completo *ac(4,4,restr:{concorda(sin,sin,sin),concorda(fem,fem,fem)},[]-[],**[nv(sin,v(comeu))]-[nv(sin,v(comeu))]**,[]-[],[]-[])*

A lista **inicial dos arcos pendentes** será constituída pelos arcos propostos com base no símbolo inicial da gramática cujos vértices inicial e final são iguais (ver secção **1.3.2**) e cujos canais são inicializados a vazio ([]).

Ex: Dada a gramática da Fig. 1.10 a lista inicial dos arcos pendentes será constituída pelos arcos activos *aa(1,1,r1:f(f(X,Y)), ilha_slashquest sn(N1,G,X),[sv(N2,Y),{concorda(N1,N2,N)}],[]-[],[]-[],[]-[],[] -[])* e *aa(1,1,r2:f(F), comp, [f(F)],[]-[],[]-[],[]-[],[] -[])*

D) Alterações ao processo de incorporação

Às **condições necessárias e suficientes** para se realizar a incorporação de um arco completo B num arco activo A enunciadas na secção **1.3.2** deve acrescentar-se:

- Parte direita de cada canal do arco activo A deve ser idêntica à parte esquerda dos respectivos canais no arco completo B
- **Canais do arco completo B** devem ser **compatíveis com a categoria**, possivelmente **restrita** por uma **ilha**, do arco activo A

Ex1: Dada a gramática da Fig. 1.10 e a frase 'comeu a maria' a incorporação dos arcos completos iniciais

(1) *ac(2,3,pal:d(sin,fem,d(a)),Q-Q,S-S,R-R,V-V)*

(2) *ac(U,U,lambda:d(_,_,[]),Q-Q,S-S,R-R,V-V)*

no arco activo

aa(2,2, r4:sn(N,G, sn(D,NP)),ilha d(N1,G1,D), [ilha np(N2,G2,NP),{concorda(N1, N2,N),concorda(G1,G2,G)}],[]-[],[nv(sin,v(comeu))]-[nv(sin,v(comeu))],[]-[],[]-[])

será possível porque:

1. a primeira categoria em resto (***d***) do arco activo é igual às categorias dos arcos completos

2. o vértice final do arco activo (vértice **2**) é igual ao vértice inicial do arco completo (1) e é idêntico ao vértice inicial do arco completo (2) (pois, em Prolog, uma variável não instanciada é idêntica a qualquer termo)[5]

3. a parte direita de cada canal do arco activo é idêntica à parte esquerda dos respectivos canais nos arcos completos (não instanciados)

4. Os canais dos arcos completos (*Q-Q, S-S, R-R, V-V*) **não permitem a ocorrência de movimentações de material linguístico** sendo portanto compatíveis com **a existência** de uma ***ilha*** *d* como primeira categoria do arco activo

   Note-se que

   - o canal de *slash* do arco activo referente a *sn* transporta o constituinte n*v(sin,v(comeu))* e só a não instanciação dos canais dos arcos completos iniciais permitirá a satisfação da condição 3.

   - a parte esquerda e direita dos arcos completos iniciais é igual permitindo assim a sua compatibilização com a existência de uma *ilha d* como primeira categoria em resto do arco activo

O **arco resultante da incorporação** será um **arco activo** obtido de acordo com a descrição da secção **1.3.2** que terá

- a parte esquerda dos seus canais igual à parte esquerda dos canais do arco A

- a parte direita dos seus canais igual à parte direita dos canais do arco B.

[5]Note-se que a **incorporação de um arco completo num arco activo deve ser efectuada de maneira a não alterar o arco completo nem o arco activo**, mas apenas o novo arco resultante da incorporação.

No ex. anterior, os arcos resultantes da incorporação dos arcos completos iniciais (1) e (2) no arco activo serão, respectivamente,

aa(2,3,r4:sn(N,G,sn(d(a),NP)),ilha np(N2,G2,NP), [{concorda(sin,N2,N),concorda(fem,G2,G)}],[]-[],[nv(sin,v(comeu))]-[nv(sin,v(comeu))] ,[]-[],[]-[])

aa(2,2,r4:sn(N,G,sn([],NP)),ilha np(N2,G2,NP),[{concorda(N1,N2,N),concorda(G1,G2,G)}],[]-[],[nv(sin,v(comeu))]-[nv(sin,v(comeu))],[]-[],[]-[]).

Os exemplos seguintes (Ex2, Ex3 e Ex4) ilustram o que se passa, respectivamente, com a incorporação de um conjunto de restrições e com a ancoragem e movimentação de um constituinte.

Ex2: Dada a gramática da Fig. 1.10 e a frase 'comeu a maria' por **incorporação** do arco completo *ac(4,4,restr:{concorda(sin,sin,sin),concorda(fem,fem,fem)},[]-[],[nv(sin,v(comeu))]-[nv(sin,v(comeu))],[]-[],[]-[])* no arco activo *aa(2,4,r4:sn(N,G,sn(d(a),np(maria)),[{concorda(sin,sin,N),concorda(fem,fem,G)}],[],[]-[],[nv(sin,v(comeu))]-[nv(sin,v(comeu))],[]-[],[]-[])* obtém-se o arco completo ***ac(2,4,r4:sn(sin,fem,sn(d(a),np(maria)),[]-[],[nv(sin,v(comeu))]-[nv(sin,v(comeu))],[]-[],[]-[])***

Ex3: Dada a gramática da Fig. 1.10 e a frase 'comeu a maria' por **incorporação** do arco completo *ac(4,4,slash:nv(sin,v(comeu)),[]-[],[nv(sin,v(comeu))]-[],[]-[],[]-[])* no arco activo *aa(4,4,r5:sv(N,sv(V,Args)),ilha_relquest nv(N,V),[args(Args)],[],[]-[],[nv(sin,v(comeu))]-[nv(sin,nv(comeu))],[]-[],[]-[])* obtém-se o arco activo ***aa(4,4,r5:sv(sin,sv(v(comeu),Args)),args(Args),[],[]-[],[nv(sin,v(comeu))]-[],[]-[],[]-[])***

Ex4: Dada a gramática da Fig. 1.10 e a frase 'comeu a maria' a incorporação do arco completo inicial *ac(1,2,r6:nv(sin,v(comeu)),[]-[],[]-[],[]-[],[]-[])* no arco activo *aa(1,1,r3:comp slash nv(N,V), ilha nv(N,V) ,[],[]-[],[]-[],[]-[],[]-[],[]-[])* será possível porque:

1. a primeira categoria em resto (n***v***) do arco activo é igual à categoria do arco completo
2. o vértice final do arco activo (vértice **1**) é igual ao vértice inicial do arco completo (vértice **1**).
3. a parte direita de cada canal do arco activo é idêntica à parte esquerda dos respectivos canais no arco completo
4. Os canais do arco completo (*[]-[],[]-[],[]-[],[]-[]*) não permitem a ocorrência de movimentações de material linguístico sendo portanto **compatíveis** com a **existência** de uma ***ilha*** *nv* como primeira categoria do arco activo

O **arco resultante da incorporação** será um arco completo obtido de acordo com a descrição da secção **1.3.2** que terá:

- a parte esquerda dos canais igual à parte esquerda dos canais do arco A
- a parte direita dos seus canais igual à parte direita dos canais do arco B.

Nesta situação, se **a categoria do arco activo A traduzir uma movimentação de material linguístico** este deve ser **adicionado à parte direita dos canais relativos ao tipo de movimentação** possibilitando assim o seu posterior transporte.

No Ex. 4, o arco resultante da **incorporação** do arco completo inicial no arco activo será *ac(1,2,r3:comp **slash** nv(sin,v(comeu)),[]-[],[]-[**nv(sin,v(comeu))**],[]-[],[]-[])*

Note-se que a categoria do arco activo (*comp **slash** nv(N,V))* traduz uma movimentação do tipo **slash** donde o constituinte ***nv(sin,v(comeu))*** é adicionado à parte direita do canal **slash**.

2

Sistemas para detecção e correcção de erros utilizando o contexto

Como vimos na introdução 25 a 50% dos erros existentes em textos ingleses (Kukich [Ku 92]) correspondem a palavras existentes no léxico e, 75% destes erros, são ocasionados por violações sintácticas.

Daí a necessidade de utilização do contexto para a correcção desses erros ou para melhoramento das técnicas de correcção e de aprendizagem de palavras não existentes no léxico.

Esta área de pesquisa e desenvolvimento implica a utilização de:

- analisadores robustos para pesquisa de constituintes com base numa gramática
- aproximações estatísticas do modelo da língua.
- uma integração das duas técnicas

2.1 Modelos utilizando analisadores com base em gramáticas

O universo de aplicação da maior parte dos sistemas construídos até agora é bastante restrito e dificilmente adaptável a aplicações com domínio mais vasto.

Contudo, com o desenvolvimento de analisadores sintácticos mais robustos, foram criados alguns sistemas de detecção e correcção de erros em domínios mais amplos (texto escrito). Destacam-se por ex., o sistema Epistle/Critique (Heidorn et al. [HJMB 82] e Richardson e Braden-Harder [RB 88]) como auxiliar da escrita em inglês e o sistema criado por Kempen e Vosse [KV 90] como auxiliar da escrita em holandês.

Em geral, todos estes sistemas têm como módulo principal um analisador contemplando análise sintáctica[6] ou em alguns casos semântica[7], pragmática[8] ou de modelos de estrutura de discurso[9].

[6]Entre outros, o sistema Epistle/Critique (Heidorn et al. [HJMB 82] e Richardson e Braden-Harder [RB 88]) e o sistema criado por Kempen e Vosse [KV 90]
[7]Entre outros, os sistemas CASPAR/DYPAR/MULTIPAR (Carbonell e Hayes [CaH 83]
[8]Entre outros, o sistema construído por Ramshaw [Ram 89] e o sistema construído por McCoy [Mc 89]
[9]Entre outros, o sistema construído por Ramshaw [Ram 89]

Para além do analisador, motor de toda a análise, recorrem a um léxico que contém informação morfológica sobre as palavras necessárias ao domínio da aplicação e a uma gramática que será o conjunto de regras que descrevem como as palavras, partes do discurso ou categorias sintácticas se organizam para gerar frases ou segmentos de frase correctos.

Por vezes, o conhecimento dos vários níveis linguísticos não é estratificado e, em algumas variantes, existe uma gramática do tipo semântico incorporando restrições à justaposição das palavras ou é incorporado no léxico o conhecimento sintáctico e semântico.

O analisador[10] pesquisará a existência de uma palavra no léxico e tentará, por aplicação das regras da gramática, construir novas categorias que, porque uma palavra pode ser constituinte de várias categorias sintácticas, serão exploradas em paralelo até que posteriores restrições obriguem à resolução da ambiguidade existente.

Note-se que, em sistemas onde exista à partida conhecimento semântico sobre os constituintes, este poderá ser aproveitado para restringir o espaço de pesquisa das categorias.

Em todos estes sistemas existem, pelo menos, dois tipos de situações problemáticas que terão que ser consideradas na detecção e correcção de erros: a incompletude da gramática e do léxico e a dificuldade de localização de uma falha (por ex., uma falha na pesquisa de uma palavra no léxico pode significar a existência de um erro ortográfico ou a existência de um neologismo).

Por exemplo, segundo Kukich [Ku 92], para resolver a elevada existência de neologismos numa dada categoria (por ex., nomes próprios), muitos sistemas optam por lhes atribuir uma categoria (geralmente nome comum[11]) embora, em caso de falha posterior, seja necessário decidir a sua causa: se se deveu à violação de uma regra, à incompletude da gramática ou à existência de uma categoria erradamente atribuída.

Kukich [Ku 92] considera que existem três tipos de aproximações para a resolução de incorrecções em frases:

1. Aproximações baseadas em consentimento
2. Aproximações baseadas em relaxamento
3. Aproximações baseadas em expectativas

[10]Por ex., no sistema Epistle/Critique (Heidorn et al. [HJMB 82] e Richardson e Braden-Harder [RB 88]) e no sistema criado por Kempen e Vosse [KV 90] como auxiliar da escrita em holandês)

[11]Por ex., no sistema Epistle/Critique (Heidorn et al. [HJMB 82] e Richardson e Braden-Harder [RB 88]

2.1.1 Aproximações baseadas em consentimento

Constatando que em caso de erros nós, seres humanos, conseguimos interpretá-los, neste tipo de aproximação os erros serão ignorados até que se consiga interpretar o que foi escrito no âmbito da aplicação que se propõe, utilizando para isso bastante conhecimento semântico em detrimento de qualquer outro nível de conhecimento linguístico.

Dois dos primeiros sistemas construídos de acordo com esta abordagem centravam-se na análise semântica (Waltz [Wal 78] e Schank et al. [SchLB 80]). Aí as restrições relativas à concordância, por exemplo, ou não existiam ou existiam sob a forma de regras acrescentadas à gramática para permitir lidar com certos tipos de erros mais comuns.

Uma variante destes sistemas, construída por Fass e Wilks [FW 83], baseava-se no facto de alguns textos serem considerados incorrectos por estarem efectivamente incorrectos ou por corresponderem a metáforas.

Neste último sistema era utilizado um formalismo para representar as relações semânticas entre constituintes e um procedimento que permitia a codificação de cada sequência de palavras de forma a evidenciar os conflitos semânticos ou a violação das suas preferências. Durante o processamento era ainda efectuada a escolha da análise que contivesse o menor número de alterações relativas à sequência de palavras de entrada ou ao sistema semântico, tendo como objectivo minimizar os conflitos.

No sucesso destes sistemas foi determinante a restrição do domínio das aplicações onde as entradas não eram ambíguas no que diz respeito à semântica. Contudo, são pouco eficazes no que diz respeito à detecção e correcção de erros pois, em geral, não os detectam e, por outro lado, obrigam à existência de um modelo semântico do discurso que, no caso de aplicações com domínios menos restritivos, pode não estar ainda completamente definido.

2.1.2 Aproximações baseados em relaxamento

Estes sistemas assumem o oposto dos anteriores, ou seja, nenhum erro deve ser ignorado pois as falhas dos sistemas de PLN que em geral são construídos sobre regras sintácticas, são devidas a violação dessas regras.

Quando o analisador falha, o sistema localiza o erro identificando a regra ou regras que o poderão ter originado e tenta determinar se o seu relaxamento pode conduzir a uma análise bem sucedida (permitindo assim a sua detecção e correcção).

Segundo Kukich [Ku 92] este tipo de aproximação, sendo a que utiliza menor conhecimento incorporado, poderá ser a mais bem sucedida em texto de domínio menos restritivo.

O sistema IBM Epistle (Heidorn et al. [HJMB 82]) que, mais tarde é revisto e designado por Critique (Richardson e Braden-Harder [RB 88]), usa a aproximação definida anteriormente e é considerado um dos melhores sistemas para utilizadores da língua inglesa.

O Epistle diagnostica alguns erros de estilo (por ex., ocorrência de frases muito longas) e, fundamentalmente, cinco tipos de erros sintácticos, no âmbito da correspondência comercial escrita em inglês:

1. Concordância sujeito-verbo.
2. Concordância nos sintagmas nominais.
3. Utilização apropriada dos pronomes pessoais como complementos de verbos
4. Utilização de formas verbais compostas padronizadas
5. Concordância quando se utilizam várias estruturas paralelas

O Critique é considerado pelos seus autores como um processador para texto extensivo a quatro domínios da escrita em inglês: composição, comercial, técnico e utilizadores de inglês como segunda língua desconhecendo-se os tipos de erros sintácticos e de estilo tratados (pois os seus autores não os referem no artigo de apresentação do mesmo).

O Epistle e o Critique, iniciam o processamento de uma frase consultando um dicionário para detectar e corrigir palavras. Se a palavra for desconhecida, é-lhe atribuída, em geral, a categoria de nome comum. Note-se a falha da análise no caso da palavra desconhecida ser um verbo.

Em seguida, procedem à analise dos constituintes que, em caso de falha, será efectuada em várias etapas. Assim:

1. A análise é efectuada utilizando as regras gramaticais com todas as restrições.
2. Se falhou 1. são relaxadas as restrições das regras gramaticais e são activadas regras gramaticais de substituição de palavras que se confundem (por ex., whose e who's).
3. Se falhou 2. é ainda desencadeado um outro procedimento de análise, a análise por encaixe, que se baseia no facto da análise dever ser, por natureza, ascendente.
4. As árvores sintácticas intermediárias são produzidas por segmentos de uma frase que podem ser encaixados obtendo, como resultado, uma análise incompleta da frase. Para cada segmento da frase analisado deve ter-se presente a detecção de erros gramaticais e de estilo.

Finalmente, se se obtêm múltiplas análises, é seleccionada uma só que corresponderá à melhor solução (Heidorn et al. [HJMB 82]); por outro lado, se o número de análises exceder um certo limite, o sistema informa o utilizador que o seu texto não é "suficientemente claro" e, em caso de falha do analisador, o utilizador será informado que o seu segmento de texto era "muito difícil de processar".

Um outro sistema que se revelou eficaz e tem como domínio a edição de texto em holandês foi construído por Kempen e Vosse [KV 90] e revisto por Vosse [Vos 92]. Este sistema detecta e corrige erros tipográficos, ortográficos e morfo-sintácticos.

Os erros morfo-sintácticos tratados são os seguintes:

1. Questões de concordância

2. Homófonas (só são detectadas aquelas que corresponderem a categorias sintácticas diferentes)

3. Palavras com flexões homófonas: caso do *d/t* em holandês (todas as palavras terminadas em *d*, *t* ou *dt* têm o mesmo som ainda que correspondam a conjugações verbais diferentes)

4. Palavras duplicadas

5. Expressões idiomáticas que não têm uma estrutura sintáctica regular podendo, algumas das palavras, ser consideradas correctas fora do contexto idiomático.

   Se estas ocorrerem em frases normais poderão ser considerados erros ortográficos mas, se existirem erros ortográficos na expressão idiomática, até poderão ser consideradas sintacticamente correctas

6. Palavras compostas que, com excepção dos neologismos, se compõem, em holandês, por aglutinação Vosse [Vos 92]

Este sistema processa o documento em dois níveis:

1. **Nível da palavra**

   Nesta fase são detectados e apresentadas as alternativas de correcção para erros ortográficos ou tipográficos utilizando um analisador baseado na análise de trifones de Berkel e DeSmedt [BDS 88] e na análise de trigramas de Angell et al. [AFW 83] gerando, para cada palavra do documento, um conjunto ordenado de alternativas para posterior correcção.

Em seguida é desencadeado um pré-processamento que combinará as palavras e suas correcções numa rede de palavras

2. **Nível da frase**

   Desencadear-se-á um analisador ao nível da frase e posteriormente um corrector sintáctico que, face a erros sintácticos assinalados, efectuará a sua correcção.

   O analisador é uma extensão do algoritmo de Tomita [Tomi 86] e utiliza duas estruturas para armazenamento que permitem controlar o estado da análise e o da entrada.

   O analisador incorpora um procedimento para instanciamento de variáveis e actua sobre uma gramática independente do contexto mas parametrizada.

   No caso de questões relacionadas com a concordância, neste tipo de análise, o local onde existiu a incongruência será assinalado, a regra relaxada e a análise prosseguirá.

   Por outro lado, quando não foi possível atribuir uma categoria sintáctica à palavra corrente da sequência de palavras de entrada, a análise prosseguirá relaxando a regra e substituindo a palavra por outra pertencente ao seu conjunto de correcção.

   Este sistema ainda inclui, ao nível da gramática, algumas regras para lidar com erros estruturais ou de ordenação de elementos na frase.

   A selecção ou ordenação das várias análises, caso existam, é feita atribuindo pesos a cada regra de acordo com o tipo de violações mais frequentes na língua (pesos mais baixos correspondem a erros mais frequentes) permitindo assim uma melhor selecção das mesmas.

Note-se que, em qualquer destes sistemas acima especificados, não se recorre a nenhum conhecimento semântico, pragmático ou estrutural do discurso e também não são corrigidos erros existentes a esses níveis e, por outro lado, as diversas análises parciais existirão em paralelo até que posteriores restrições permitam resolver as ambiguidades sintácticas detectadas.

Estes tipos de sistemas foram também propostos por Trawick [Tra 83], Weischedel e Sondheimer [WS 83], Suri [Su 91] e Suri e McCoy [SuMC 91].

No sistema REL (Trawick [Tra 83]), são utilizadas numa primeira fase, para além do relaxamento, estratégias de ordenação e, em caso de falha, é então tentado o conjunto ordenado de palavras alternativas utilizado por Kempen e Vosse [KV 90] e referido na pag. 40.

No sistema de Weischedel e Sondheimer [WS 83] é utilizado basicamente o relaxamento e um conjunto de meta-regras que, em caso de falha, guiam a pesquisa no conjunto de regras que poderão ter originado o erro.

Estas meta-regras são aplicadas quando ocorrem alguns erros específicos, como por ex., no caso de omissão de artigos ou violação de testes (questões de concordância).

O analisador utilizado no sistema de Weischedel e Sondheimer [WS 83] é uma rede de transições aumentada que, é citado por Mellish [Mel 89], para documentar a motivação na construção de um sistema baseada num analisador por grafo permitindo, em caso de falha, a utilização das análises anteriores e a optimização da exploração do contexto sintáctico na decisão da escolha da melhor análise para as incorrecções existentes numa frase.

Suri e McCoy [SuMC 91] começaram recentemente a trabalhar um sistema (Kukich [Ku 92]) que corrigirá textos ingleses escritos por utilizadores da American Sign Language (ASL), cuja gramática difere da inglesa, mas onde os erros corresponderão necessariamente a determinados padrões.

Assim, Suri [Su 91] desenvolveu uma taxonomia destes erros analisando um corpus de amostras da ASL e, essa taxonomia, servirá de guia para um corrector inserido numa aplicação de aprendizagem da língua.

Sendo este tipo de aproximação promissora, repare-se que a sua fraqueza está sempre relacionada com a robustez do analisador que, para gramáticas bastante completas, implicarão, entre outros problemas inerentes à escolha do tipo de analisador, grandes espaços de pesquisa e necessariamente várias correcções possíveis.

Consequentemente, serão factores determinantes para a construção destes sistemas, a utilização de analisadores permitindo o reaproveitamento das análises já efectuadas (utilizando estruturas, por ex. do tipo do grafo, para armazenamento das mesmas) bem como, a utilização de estratégias que colmatem os pontos fracos da análise (por ex., no que diz respeito à delimitação do espaço de pesquisa).

2.1.3 Aproximações baseadas em expectativas

Este tipo de aproximação situa-se entre as referidas em **2.1.1** e **2.1.2**, ou seja, baseia-se na constatação de que os erros ocorrem e no que fazemos para os corrigir.

O analisador, à medida que o sistema avança, constrói uma lista de palavras que espera existirem na próxima categoria sintáctica, semântica e, por vezes, até pragmática ou de

estrutura do discurso. Se o próximo termo a ser analisado não existe na lista dos esperados, o sistema assume a existência de um erro e tenta corrigi-lo face à lista existente.

Estes sistemas têm vindo a ser subdivididos para explorarem os vários níveis de conhecimento linguístico.

Um sistema que explora sintaxe ou semântica baseado num analisador que recorre a enquadramentos baseados em casos é designado respectivamente por CASPAR/DYPAR/MULTIPAR, designações correspondentes às várias versões construídos por Carbonell e Hayes [CaH 83].

Os enquadramentos baseados em casos são estruturas para representação de predicados (normalmente realizados por verbos) e seus argumentos. Por ex., num quadro relativo ao predicado DAR existirão três entradas (Dar significa que alguém deu alguma coisa a alguém) correspondentes aos casos nominativo (Quem), acusativo (O_Quê) e dativo (A_Quem).

As estruturas sintácticas e semânticas do discurso são modeladas em enquadramentos recursivos que são colocadas na gramática e no léxico. Cada palavra pesquisada na lista de entrada preencherá uma entrada dum quadro já existente ou originará um novo quadro. Assim, à medida que o analisador progride, uma lista de potenciais termos esperados pode ser gerada para cada palavra da sequência de palavras de entrada.

Por ex., em Carbonell et al. [CBMA 83], *prot* é substituído por *port* na frase "Add a dual *prot* disk" pois, embora a pesquisa no dicionário originasse 13 hipóteses de correcção, a única correspondente ao esperado (descrição das características do disco) era *port*.

O tamanho da lista de potenciais esperados poderá ser do tamanho:

- do dicionário quando um quadro ainda não foi seleccionado
- de pequenos conjuntos de termos potenciais quando um quadro incompleto está a ser processado
- de uma palavra quando esta é esperada.

Os enquadramentos baseados em casos que foram aplicados no Excalibur (Carbonell et al. [CBMA 83] e [CBMA 83a]), são ainda bastante úteis para detecção de palavras omissas ou pesquisa de anáforas ou elipses em texto.

Este tipo de aproximação é utilizada simultaneamente com relaxamento na última versão deste sistema (MULTIPAR - Minton et al. [MHF 85]) que, quando o analisador encontra um obstáculo, invoca tantas estratégias (realizadas em paralelo) para ultrapassá-lo quantas as necessárias para encontrar, pelo menos, uma análise válida.

A invocação das estratégias, neste sistema, é controlada por um procedimento que as ordena de acordo com os erros correspondentes ao menor desvio.

Um sistema baseado nesta aproximação, mas explorando pragmática e conhecimento sobre estrutura do discurso, foi desenvolvido por Ramshaw [Ram 89] onde os erros na entrada são detectados quando o sistema não é capaz de construir uma interpretação completa da entrada compatível com um ou vários planos do domínio.

Ramshaw [Ram 89] substitui cada palavra da entrada por uma área em aberto combinando depois, as restrições sintácticas e semânticas da interpretação parcial da entrada, com o que pode ser predito pelo modelo pragmático, numa tentativa de obter um conjunto ordenado de correcções.

Dois outros sistemas, McCoy [Mc 89] e Carberry [Car 84] descrevem métodos para gerar respostas relativas a entradas incorrectas sob o ponto de vista pragmático. Ambos usam heurísticas para construir um modelo dos planos do utilizador e as soluções são baseadas na informação que pode ser inferida da estrutura do discurso precedente.

Note-se ainda que a utilização da estrutura do discurso esperado, pode ser utilizado para corrigir erros e resolver ambiguidades em sistemas aplicados ao domínio do reconhecimento da fala.

Este tipo de aproximação tem o mesmo tipo de inconvenientes que as aproximações baseadas em consentimento pois, as técnicas para representar a semântica, a pragmática e a estrutura do discurso a um nível de detalhe eficaz, são dificilmente conseguidas mesmo em domínios restritos.

2.2 Modelos baseados em métodos estatísticos

Os modelos estatísticos da língua baseiam-se em tabelas/matrizes de probabilidades condicionais estimadas para algumas ou todas as palavras da língua, traduzindo a probabilidade de ocorrência de uma palavra num dado contexto de palavras.

Algumas medidas sobre as quais se constroem estes modelos:

- um trigrama de palavras especifica a probabilidade da ocorrência duma palavra condicionada pelas duas anteriores
- um bigrama de partes do discurso especificará a probabilidade de ocorrência da parte do discurso que contém a próxima palavra condicionada pela probabilidade de ocorrência da parte do discurso que contém a palavra anterior

- uma colocação especificará as probabilidades de certas palavras ocorrerem na vizinhança (limitada a cinco palavras à esquerda ou à direita) de outras relacionadas, sob o ponto de vista linguístico, com estas.

Este tipo de abordagem necessitará, obviamente, de grandes recursos em termos de memória para armazenamento do corpus e do cálculo das probabilidades condicionais necessárias ao modelo e, por outro lado, restrições efectuados no corpus podem invalidar as análises obtidas face a esse modelo.

Os sistemas que utilizam modelos probabilísticos baseiam-se em aproximações sobre o que se espera, ou seja, uma probabilidade condicional de ocorrência de uma palavra muito baixa poderá ser usada para ser detectado um erro e probabilidades muito altas de ocorrência de palavras poderão determinar alternativas ordenadas para possíveis correcções.

Nestes sistemas, se forem utilizados modelos baseados em probabilidades referentes a contextos sintácticos de partes de discurso, poderão ser detectadas alguns erros relacionados com semântica sem a necessidade de existência de um modelo formal da mesma (por ex., *faca* associada a *corte*).

Kukich [Ku 92] descreve os resultados obtidos por três sistemas desenvolvidos nesta área que utilizam modelos diferentes (em função dos objectivos pretendidos), onde:

- Um modelo baseado em bigramas de partes do discurso (Atwell e Elliott [AtE 87]) e aplicado à detecção de erros em palavras existentes no léxico detectou 62% dos 420 erros existentes respeitantes a violações locais e globais de restrições sintácticas e diagnosticou incorrectamente cerca de duas vezes o número total de erros

- Um modelo baseado em bigramas de palavras (Gale e Church [GC 90]) e Church e Gale [CG 91]) tentou melhorar a correcção de palavras desconhecidas revelando-se eficaz pois, num universo de 329 erros, obteve-se uma melhoria de correcção da ordem dos 87 a 90%

- Um modelo baseado em trigramas de palavras (Mays et al. [MDM 91]) pretendendo detectar e corrigir palavras existentes no léxico detectou 76% dos erros cometidos e corrigiu correctamente 74% destas embora com grandes custos computacionais pois necessitou gerar e testar um grande número de alternativas de frases para cada hipotético erro na entrada

Conclui-se assim que, embora os resultados sejam promissores, pertencem ainda a uma área que necessita de maior investigação pois, existem problemas relacionados com limitação de recursos e com as técnicas usadas para calcular as probabilidades

Por outro lado, modelos baseados em probabilidades condicionais referentes a bigramas e trigramas de palavras ou partes do discurso, limitam o seu universo de representação a relações locais.

Kiyono e Tsujii [KT 93] desenvolveram um sistema misto onde não é assumido que existem erros na entrada, mas sim que o conhecimento linguístico do mesmo é incompleto. Assim, no caso de falha do analisador, várias regras são formuladas como hipóteses acarretando um grande incremento no espaço de pesquisa que será reduzido utilizando um corpus da língua, para permitir a decisão sobre a regra mais plausível.

2.3 Modelos baseados em redes neuronais

Uma outra área de pesquisa actual é a das redes neuronais que, utiliza um princípio semelhante ao utilizado nas técnicas probabilísticas, ou seja, ambos pretendem obter expectativas sobre o contexto duma dada palavra modelando a distribuição do relacionamento entre palavras.

A principal diferença entre estes dois tipos de modelos é que, enquanto nos anteriores as probabilidades eram explicitamente representadas, nestes são implicitamente representadas através de pesos na rede.

Sob o ponto de vista computacional, estes modelos são muito mais pesados que os probabilísticos pois, se nestes últimos centenas de milhares de probabilidades sobre léxicos de milhares de palavras podem ser calculados e armazenados em disco para utilização posterior, nas redes, os milhares de nós que representam as palavras e as centenas de milhares de pesos que representam as suas associações terão que estar presentes na memória durante a fase de treino e de processamento.

Pode, então, afirmar-se que estes modelos, face aos recursos actuais, só são eficazes para léxicos limitados e só poderão ser mais abrangentes quando existirem recursos máquina e programas que melhorem a eficácia das redes neuronais.

3

Caracterização dos erros tratados e metodologia utilizada nesta tese

Neste capítulo caracterizam-se os erros morfo-sintácticos e o desconhecimento de palavras que foram tratados nesta tese. Apresenta-se também a metodologia que foi utilizada no sistema de detecção e correcção de erros desenvolvido por mim.

3.1 Caracterização dos erros tratados nesta tese

Considerando a frase como referencial e, tendo como base um conjunto de regras sintácticas e um dado léxico, os erros morfo-sintácticos tratados vão ser agrupados em função do conhecimento ou desconhecimento das palavras, ou seja, com a existência ou não dessas palavras no léxico do sistema desenvolvido.

3.1.1 Palavra desconhecida

Dois tipos de situações justificam o desconhecimento da palavra:

A) Palavra mal escrita

Existem dois tipos de erros (Berkel e DeSmedt [BDS 88]): os tipográficos e os ortográficos.

Erros tipográficos

Os erros tipográficos são causados por problemas na digitação sendo, portanto, de origem motora.

Dado não existirem estudos feitos sobre a ocorrência de erros em Português, vão ser considerados os quatro tipos de erros tipográficos que Damerau [Dam 64] considera representarem 80% dos erros cometidos em inglês:

1. Substituição de uma letra por outra (ex: 'Msria')

2. Omissão de uma letra (ex: escve)

3. Existência de uma letra a mais (ex: Masria)

4. Troca de letras consecutivas (ex: cmoia)

Erros ortográficos

Os erros ortográficos devidos à substituição de uma letra por outra(s) com o mesmo som, são cognitivos e dependem da correspondência entre a pronúncia e a escrita acontecendo por deficiente pronúncia, confusão de som ou ignorância.

Em Português, os erros resultantes da confusão de sons correspondem, em geral, às situações do tipo 1 a 3 acima referidas (por ex., espande, conosco e têem), e as excepções correspondem a substituições de uma letra por duas letras ou vice-versa (por ex: oxipital e óccido).

A Fig 3.1 especifica alguns erros passíveis de ser cometidos em português devido a confusão de sons.
Por ex., a linha **z** tem um ***x*** nas colunas **s** e **x** pois pode confundir-se o som da letra **z** com o som das letras **s**, **x**.

Ex: ezame e vasa (vazar)

	a	c	cc	ç	ch	e	ee	g	ha	he	hi	ho	hu	i	j	m	mm	n	nn	o	s	ss	u	x	z
a						x			x																
c			x	x																	x	x			
cc		x																						x	
ç		x																			x	x			
ch																								x	
e	x					x				x				x											
ee						x																			
g															x										
ha	x																								
he						x																			
hi														x											
ho																				x					
hu																							x		
i						x																			
j								x																	
m																	x	x							
mm																x									
n																x			x						
nn																		x							
o												x											x		
s		x		x																		x		x	x
ss		x		x																	x				
u													x							x					
x			x		x																				x
z																					x			x	

Fig. 3.1 Algumas trocas de letra(s) devidos a confusões de sons

As situações correspondentes à existência ou inexistência de um acento (Fig. 3.2) serão sempre substituições de um carácter por outro.

Exs: fôr e lexico

a	**á**	**à**	**â**	**ã**
e	**é**	**ê**		
i	**í**			
o	**ó**	**ô**	**õ**	
u	**ú**			

Fig. 3.2. Acentos

B) Neologismos em sentido lato (não existência da palavra no léxico)

Estas palavras terão que ser adquiridas pelo sistema que as desconhece (Lopes et al. [LMR 94]).

3.1.2 Palavra existente no léxico

As situações tratadas correspondem a:

A) Palavras a mais

a) Por duplicação (ex: 'O João João foi à praia')

b) Não duplicadas (ex: 'O João foi ia passear')

B) Questões de concordância

a) Pessoa (ex: 'Eu comeu um bife')

b) Género (ex: 'A Pedro comprou o livro' ou 'O ovo foi comida pela Maria'

c) Número (ex: 'Eu comeram um bife')

Nota: Não foram tratados nesta tese as questões de concordância referentes a construções predicativas onde o sujeito não só concorda com o verbo predicativo mas também com o predicativo do sujeito. Por ex., os erros referentes ao predicativo do sujeito nas frases 'os livros são linda' e 'O João continua parvas'

C) Palavra com categoria ou forma diferente da requerida

Ex: 'O João deu o livro há Maria' ou 'O livro foi dar pelo João à Maria'

Note-se que as homófonas com categorias sintácticas diferentes conduzirão a situações deste tipo.

3.1.3 Palavras omissas

Foram tratadas as duas situações seguintes:

A) Pode propor-se a categoria gramatical da(s) palavra(s) que falta(m)

Nesta situação a árvore sintáctica corresponderá a uma frase

Ex: 'foi à praia' ou 'O João o livro à Ana'

B) Frase incompleta

Nesta situação a árvore sintáctica não corresponderá a uma frase e as hipóteses serão formuladas para a primeira categoria pré-terminal em falta (Ex: 'O João')

Nota: Num texto, é usual a elisão de palavras que, no entanto, podem ser contextualmente recuperadas, por ex., 'Quem foi à praia? O João'. Este, no entanto, não foi o enfoque desta tese.

3.2 Metodologia utilizada nesta tese

O sistema desenvolvido por mim utiliza uma aproximação baseada em relaxamento para a resolução de incorrecções ao nível morfo-sintáctico, existentes em frases.

Este sistema utiliza um analisador por grafo que é uma extensão do analisador de Paulo [Pau 93] à problemática da detecção e correcção de erros, e actua sobre uma gramática de movimentação e ancoragem (Lopes [Lop 92]) que foi adaptada para impedir a falha do analisador face à existência de erros de concordância.

Assim, existindo um erro de concordância este é assinalado e a análise prosseguirá.

A detecção e correcção dos erros processa-se da seguinte forma:

- Inicialmente cada palavra é categorizada de acordo com um léxico ou considerada desconhecida
- É desencadeado um analisador por grafo que tentará analisar a frase
- Por falha do analisador desencadear-se-á um formulador de hipóteses que, utilizando técnicas de relaxamento, tentará formular hipóteses que permitam superar os erros ou desconhecimento tipificados na secção anterior.
- As técnicas de relaxamento foram utilizadas para permitir incorporações de terminais ou palavras desconhecidas em arcos que expressam uma expectativa de um dado constituinte. Assim:

    1. Na formulação de hipóteses relativas à existência de **palavras a mais,** foi utilizado o relaxamento de um vértice, ou seja, permitiu-se a incorporação de um pré-terminal concretizado num dado vértice com o mesmo pré-terminal esperado num vértice anterior.

Na Fig. 3.3 apresenta-se a árvore sintáctica da sequência de palavras 'a maria maria comeu um bolo'.

A análise poderá ser concluída por ter sido formulada, entre outras, a hipótese de existência de uma **palavra a mais**, ou seja, permitiu-se a incorporação do verbo 'comer' existente no vértice **4** com o verbo esperado no vértice **3**.

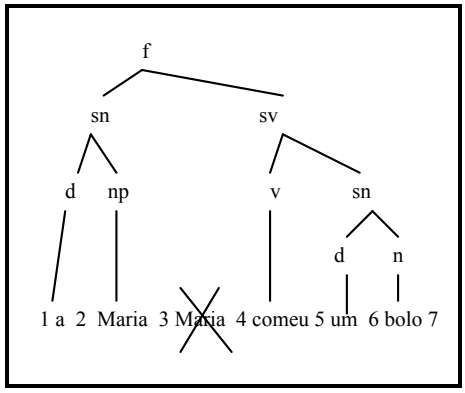

Fig. 3.3 Árvore sintáctica

2. Na formulação de hipóteses relativas à existência de **palavras omissas** ou á incorporação de **palavras desconhecidas** foi utilizado o relaxamento do constituinte esperado, ou seja, foi atribuída a categoria pré-terminal esperada num dado vértice a uma palavra desconhecida ou a uma palavra omissa nesse vértice.

   Na Fig. 3.4 apresenta-se a árvore sintáctica da sequência de palavras 'a comeu um bolo'.

   A análise poderá ser concluída por terem sido atribuídas as categorias pré-terminais esperadas no vértice **2** (*np* ou *n*) a uma palavra **omissa** existente no mesmo vértice

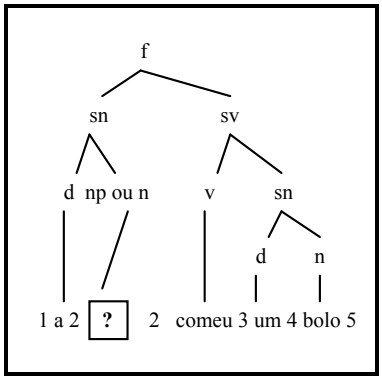

Fig. 3.4 Árvore sintáctica

   Na Fig. 3.5 apresenta-se a árvore sintáctica da sequência de palavras 'a mria comeu um bolo'.

   A análise poderá ser concluída por terem sido atribuídas as categoria pré-terminais esperadas no vértice **2** (*np* ou *n*) à **palavra desconhecida** existente no mesmo vértice

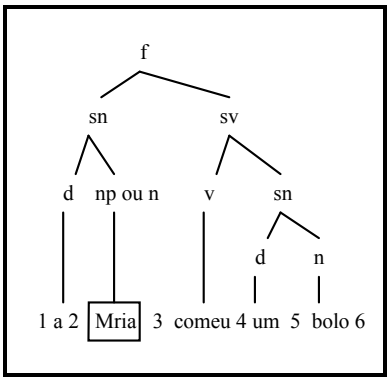

Fig. 3.5 Árvore sintáctica

- Finalmente é desencadeado um detector e corrector de erros que, face às análises obtidas para o símbolo inicial da gramática, informará o utilizador sobre as incorrecções encontradas e as alternativas possíveis de correcção.

- Este corrector incidirá sobre as análises que contêm um menor número de incorrecções e, em casos de palavras desconhecidas com categoria atribuída,

desencadeará um corrector muito simples que corrigirá os erros ortográficos ou tipográficos existentes nas palavras desconhecidas, considerando sempre como hipótese de correcção a possibilidade da palavra desconhecida poder ser um neologismo.

II - Descrição do Sistema Construído e Análise dos Resultados Obtidos

4

Descrição do sistema de detecção e correcção de erros

Neste capítulo, após a descrição genérica dos sistema desenvolvido, detalham-se todos os seus componentes utilizando, em todos os exemplos apresentados, a gramática existente no Apêndice I.

4.1 Descrição genérica do sistema

Na fase actual, o sistema desenvolvido testa a gramaticalidade ou agramaticalidade de uma sequência de palavras[12] representada através de uma lista de átomos Prolog que, em princípio, denotam palavras existentes no léxico.

Assim, dado um número, correspondente a uma lista de átomos existente no ficheiro de teste (eventualmente uma frase), o sistema fornece ao utilizador, via écran, informação sobre a gramaticalidade da frase testada. Em caso de agramaticalidade apresenta as possíveis correcções.

O sistema fornece ainda o tempo de execução, o número de arcos expandidos, o total de arcos activos e completos e o total de arcos hipotéticos considerados. Esta informação poderá ser utilizada em posteriores melhorias do sistema ou em análises comparativas com os resultados obtidos por correctores sintácticos construídos com base em estratégias diferentes da adoptada neste trabalho.

O programa a que corresponde este sistema foi desenvolvido em Arity Prolog (versão 6) e a sua execução é desencadeada pelo predicado *main*, obrigatório para a criação de módulos executáveis nesta linguagem.

O predicado *main* (ver secção **1** do Apêndice III) terá como funções:

1. a leitura de uma lista de palavras correspondente a um dado número fornecido pelo utilizador

2. o desencadeamento do sistema de detecção e correcção de erros

[12]Note-se que, de acordo com a terminologia adoptada nesta tese, *palavra* designará uma palavra bem ou mal escrita ou ainda uma palavra desconhecida

3. a informação, via écran, do tempo de execução, número de arcos expandidos e totais acima referidos.

Note-se que esta descrição corresponde ao programa na fase de teste. Se se pretendesse corrigir um texto, o programa teria que ser ligeiramente alterado para fazer a leitura das frases aí existentes e os resultados seriam apresentados frase a frase.

O sistema de detecção e correcção de erros (denotado por *corrector* na secção **1** do Apêndice III) invoca sequencialmente:

1. A inicialização que, baseando-se no léxico e na gramática, actua sobre a sequência de átomos, procedendo à criação das estruturas necessárias para se efectuar a análise por grafo e à identificação das palavras desconhecidas (ver secção **4.3** deste capítulo)

2. O analisador por grafo que, baseando-se na gramática e no grafo inicial, pesquisará todos os constituintes possíveis (ver secção **4.3** deste capítulo)

3. A formulação de hipóteses que, em caso de falhas, formulará hipóteses sobre os arcos invocando de novo o analisador, para que sejam detectados o maior número possível de constituintes (ver secção **4.4** deste capítulo)

4. A detecção e correcção de erros que, com base no grafo obtido, na lista das hipóteses consideradas pelo formulador para palavras omissas e na lista das palavras desconhecidas, pesquisará os erros existentes e proporá correcções informando finalmente o utilizador sobre as possíveis correcções para esses erros ou sobre a gramaticalidade da lista de átomos (ver secção **4.5** deste capítulo)

4.2 Léxico e gramática utilizados no sistema construído

Léxico

O léxico utilizado (Apêndice II), semelhante ao fornecido na cadeira de PLN, é constituído pela descrição exaustiva de palavras onde cada palavra é caracterizada pela sua categoria gramatical e parâmetros relativos ao género, número, subcategorização, etc.

Optou-se por este tipo de léxico para simplificar a construção do corrector ortográfico que se limita a corrigir os erros tipificados por Damerau [Dam 64].

A consulta ao léxico é feita recorrendo a regras

Categoria ---> [[Palavra], dic_X(Palavra,N,G,..)].

Categoria ---> [].

onde X se relaciona com a categoria sintáctica procurada. Por ex., na regra

*np(pes=3,num=N,gen=G,np(pes=3,num=N,gen=G,**W**)) ---> [[**W**],{dic_np(**W**,N,G)}].*

é pesquisado um *np* à custa da palavra ***W.***

O segundo tipo de regras é aplicada a categorias que podem não ter realização, por ex., a regra

d(_,_,_,d([])) ---> [].

corresponderá à possibilidade da ausência de determinativos.

Cada categoria sintáctica pré-terminal, denotada por *Categoria*, incluirá como um dos seus argumentos a representação sintáctica da própria *Categoria*, tendo como objectivo facilitar a detecção final das categorias onde existem erros.

Nos parâmetros respeitantes à informação sobre concordância foi introduzida a identificação do tipo de concordância a que se referem (ex: **num** = sin) facilitando assim a detecção e correcção dos erros de concordância.

Gramática

Na gramática utilizada (Apêndice I), tal como na gramática utilizada em Paulo [Pau 93], as regras são prefixadas com um identificador da regra (ver secção **1.3.2.1** do Cap. **1**/Parte **I**).

Ex: **r14**:*sn(P,N,G,_,Classe,sint(sn):[conc:[P,N,G],X,Y]) ---> [ilha d(N1,G1,Classe,X),*
ilha np(P,N2,G2,Y),
{concorda(N1,N2,N),
concorda(G1,G2,G)}].

Esta regra, identificada como **r14**, determina que um sintagma nominal (sn) poderá ser constituído por um determinativo X e um nome próprio Y devendo existir concordância em género e número entre os seus constituintes.

Cada categoria sintáctica não terminal incluirá como um dos seus argumentos a representação sintáctica da própria categoria tendo como objectivo facilitar a detecção final das categorias onde existem erros.

Assim, a representação sintáctica duma categoria não terminal (*Cat*), existirá sobre as formas seguintes:

- *sint(Cat):[conc:Conc,Pré_Terminal1,...,Pré_Terminaln]* onde *Conc* denota uma lista constituída pelos parâmetros de concordância necessários para a detecção e correcção posterior de erros de concordância relativos à categoria *Cat*

  Ex: A representação sintáctica de *sn* no ex. anterior é *sint(sn):[conc:[P,N,G],X,Y]*

- *sint(Cat):[Pré_Terminal1,...,Pré_Terminaln]* se os parâmetros de concordância relativos a *Cat* não forem necessários para a correcção posterior de erros de concordância em categorias não terminais que se situem, na árvore sintáctica, a um nível superior a *Cat.*

  Ex: Dadas as regras da Fig. 4.1

```
r25:sv(P,N,G,sint(sv):[conc:[P,N,G],NV,ARGS])--->[ilha_relquest
                                nv(P,N,G,Scat,Voz,NV),args(Voz,Scat,ARGS)].
r34:args(Voz,Scat,sint(args):[ARGS])--->[{v_sc(Scat,S),transform_v_sc(Voz,S,S1),
                                select(S1,T)},arg(T,ARGS)].
r40:arg(sn,sint(arg):[X]) ---> [sn(_,_,_,_,_,X)].
```

Fig. 4.1 Algumas regras da gramática utilizada

A regra *25* traduz a existência de um sintagma verbal (**sv**) constituído por um núcleo verbal (**nv**) e por complementos (**args**).

Na regra *34* a categoria não terminal **args** tem como representação sintáctica ***sint(args):[ARGS]*** donde na, correcção posterior de erros de concordância ao nível do sintagma verbal, o único interveniente será o núcleo verbal[13].

Note-se que, por ex., em 'o João deu os livros à Maria' deve verificar-se a concordância entre os constituintes de cada complemento mas, ao nível da frase, os únicos intervenientes em questões de concordância são o sujeito e o núcleo verbo.

A gramática incluí também a identificação através de factos do seu símbolo inicial, dos não terminais e dos pré-terminais (Paulo [Pau 93]).

Por ex: *simbolo_inicial(f(_,_,_)).*
nao_terminal(f(_,_,_)).

[13]Como já foi dito anteriormente (ver secção **3.1.2** do cap. **3**/Parte **I**) não são contempladas as situações de concordância relativas a **args** com função de **predicadores do sujeito**.

pre_terminal(d(_,_,_,_)).

As regras correspondentes à concordância de Paulo [Pau 93] foram adaptadas tendo como objectivos principais:

- não impedir a análise de uma sequência de palavras por existir um erro de concordância entre constituintes, ainda que este deva ser assinalado para correcção posterior.

  Note-se que em Paulo [Pau 93] a existência de um erro de concordância determinava a falha do analisador.

- efectuar a correcção posterior incidindo apenas sobre o constituinte não terminal onde o erro foi detectado. Assim, assinalaram-se os erros de concordância nos não terminais a que se referem.

O valor obtido para a concordância entre dois constituintes cujos valores referentes a um dado parâmetro (Param) sejam, respectivamente, X e Y obter-se-á de acordo com as regras descritas da Fig. 4.2.

```
concorda(Param=X,Param=Y,Param=err):-X\=Y,!.
concorda(Param=X,Param=X,Param=X):-X \== err,!.
concorda(Param=X,Param=Y,Param=err):-equiv(X,Y).
```

Fig. 4.2 Regras gerais de concordância

Ex1: Em 'Os joão morreu' obter-se-á em *sn* o parâmetro *num* = *err*, em *sv* o parâmetro *num* = *sin* e em *f* o parâmetro *num* = *err* (*err* é diferente de *sin*) sendo o utilizador informado que {'Os' terá que ser singular ou 'joão' e 'morreu' terão que ser plural}

Ex2: Em 'O ana morreu' obter-se-á em *sn* o parâmetro *gen* = *err*, em sv o parâmetro *gen* = _ e em *f* o parâmetro *gen* = *err* (última regra descrita na Fig. 4.2 que, em situações de erro, impede a instanciação de parâmetros não instanciados anteriormente) sendo o utilizador informado que {'O' deverá ser feminino ou 'ana' deverá ser masculino}[14]

Note-se que a segunda regra descrita na Fig. 4.2 permite a instanciação de parâmetros não instanciados necessária à correcção de palavras desconhecidas atribuídas a uma dada categoria.

Ex: Em 'O prgrama termina' a correcção posterior de 'prgrama' incidirá sobre as categorias *np* e *n* com os parâmetros *num* = *sin* e *gen* = *masc* resultantes da instanciação obtida para os parâmetros de *sn* (valor obtido para a concordância de um determinativo com os parâmetros

[14]No futuro dever-se-iam filtrar as hipóteses obtidas impedindo que o utilizador seja informado que, por exemplo, {'ana' deverá ser masculino}

num = *sin* e *gen* = *masc* com um *np* ou um *n* atribuídos a uma palavra desconhecida com os parâmetros *gen* = _ e *num* = _).

Neste sistema optou-se por informar o utilizador sobre todas as hipóteses de correcção de erros de concordância pois, em algumas situações, é difícil optar-se por um dada hipótese de correcção.

Ex: Em 'Dois carro ardeu' é difícil decidir sobre uma das duas correcções possíveis:

1. singular de 'dois' (admitindo que o singular de 'dois' deveria ser o ordinal 'um')
2. plural de 'carro' e de 'ardeu'.

A concordância em género e número ao nível da frase introduz problemas específicos pois:

- a passiva obriga à concordância em género e número do sintagma nominal com o sintagma verbal seguindo, portanto, a lei geral

  Ex1: 'a maçã foi comida pela ana' e 'o bolo foi comido pela ana'

  Ex2: 'as maçãs foram comidas pela ana'

- os tempos compostos que utilizam o verbo **ter** determinam que o verbo principal (existente no particípio passado) seja masculino e singular.

  Ex1: 'a ana tinha comido a maçã'

  Ex2: 'os exames tinham terminado'

Assim, para assinalar erros de concordância em género e número originados pela existência de tempos compostos conjugados com o verbo **ter** são utilizados parâmetros nas regras de concordância que especificam a voz (v*oz=activa*) e o género ou o número do particípio passado (regra **r29** da Fig. 4.3).

```
r27:verbo(P,N,G,T,F,M,SCat,Voz,sint(verbo):[conc:[P,N,G],NV,Resto]) --->
                                                [ilha v(P1,N1,G1,T1,F,M,S,Vf,NV),
                                                 ilha resto(P2,N2,G2,conj=NV,S,SCat,Voz,Resto),
                                                {tv(NV,Resto,T1,T,M),concorda(P1,P2,P),concorda(N1,N2,N),
                                                 concorda(G1,G2,G)}].
r28:resto(pes=_,num=_,gen=_,conj=V,S,S,voz=activa,sint(resto):[]) ---> [].
r29:resto(P,N1,G1,conj=v(_,_,_,_,_,_,_,_,ter),S,Scat,voz=activa,sint(resto):[conc:[P,N1,G1],NV]) --->
                                                [v(P,N,G,tempo=preterito,forma=participio,M,Scat,V,NV),
                                                 {concorda(voz=activa,N,N1),concorda(voz=activa,G,G1)}].
```

Fig. 4.3 Regras relativas à existência de verbos na voz activa

Os valores obtidos para a concordância em género e número relativos à ocorrência de tempos compostos conjugados com o verbo **ter,** será obtido em duas fases:

1. Numa primeira fase obtêm-se os valores para a concordância referentes ao género e número do particípio passado do verbo principal (concordância ao nível da categoria não terminal *resto* - **r29** da Fig. 4.3) que serão:

   Concordância em género:

   - *gen=emasc* se o particípio passado for feminino (deveria ser ***masc***)
   - *gen*=_ se o particípio passado for masculino

   Concordância em número:

   - *num=esin* se o particípio passado for plural (deveria ser ***sin***)
   - *num*=_ se o particípio passado for singular

2. Finalmente, face aos resultados obtidos em 1., obtêm-se os valores para a concordância em género e ao número referentes ao verbo (concordância ao nível da categoria não terminal *verbo* - **r27** da Fig. 4.3) que serão:

   Concordância em género:

   - *gen*=_ se o género da categoria não terminal *resto* for e*masc*
   - caso contrário segue a lei geral de concordância descrita anteriormente

   Concordância em número:

   - o número do verbo auxiliar **ter** se o número da categoria não terminal *resto* for *esin*
   - caso contrário segue a lei geral de concordância descrita anteriormente

Ex1: Em 'a ana comeu um bolo' obter-se-á em *sn* o parâmetro *gen = fem,* nas categorias não terminais *resto*, *verbo* e *sv* o parâmetro *gen* = _ e em *f* o parâmetro *gen = fem* sendo o utilizador informado que não existem erros de concordância.

Ex2: Em 'a ana tinha comida um bolo' obter-se-á em *sn* o parâmetro *gen = fem,* na categoria não terminal *resto* o parâmetro *gen* = ***emasc***, nas categorias não terminais *verbo* e *sv* o parâmetro *gen* = _ e em *f* o parâmetro *num = fem* sendo o utilizador informado que {'comida' deveria ser masculino}

Ex3: Em 'o bolo foi comida pela ana' obter-se-á em *sn* o parâmetro *gen = masc,* nas categorias não terminais *resto*, *verbo* e *sv* o parâmetro *gen = fem* e em *f* o parâmetro *num* = ***err*** sendo o utilizador informado que {'o bolo' deveria ser feminino[15] ou 'comida' deveria ser masculino}

4.3 Analisador utilizado no sistema construído

Foi utilizado neste sistema de detecção e correcção de erros o analisador construído por Paulo [Pau 93] (ver secção **1.3.2.1** do Cap. **1**/Parte **I**) com as alterações necessárias para ser utilizado num sistema de detecção e correcção de erros.

As adaptações efectuadas (além das que já assinalei anteriormente) foram as seguintes:

- Na inicialização (denotada por *inicializa* na secção **2** do Apêndice III) são detectadas as palavras desconhecidas, palavras não existentes no léxico, que serão armazenadas numa lista sobre a forma de arcos completos cujo identificador da regra será denotado por ***desc***, a Categoria será ***pal_dsc*** e os canais serão:

  a) não instanciados para permitirem futuras incorporações em arcos que transportem material linguístico

  b) restringidos para não permitirem movimentações de material linguístico

  Por ex., em 'u maria comeu um bolo' a palavra **u** corresponderá ao arco completo

  ac(1,2,desc:pal_dsc(u),Q-Q,S-S,R-R,V-V)

- No analisador de Paulo [Pau 93] a pesquisa terminava se se encontrasse um arco completo do símbolo inicial com vértice inicial 1 e final igual ao último vértice da sequência de palavras ou se não existissem mais arcos pendentes.

  No analisador utilizado por este sistema (denotado por *analisa_por_grafo* na secção **3** do Apêndice III) a pesquisa de constituintes só termina quando não existem mais arcos pendentes permitindo assim que, existindo erros, se possam obter o maior número de hipóteses de correcção do mesmo.

  Por ex., em 'o **prgrama** termina'

  - a análise falha devido à existência da palavra desconhecida **prgrama**

[15]Como já foi referido, dever-se-iam filtrar as hipóteses obtidas impedindo que o utilizador seja informado que {'o bolo' deveria ser feminino}

- a formulação de hipóteses repara o erro considerando as hipóteses de **prgrama** poder ser categorizado como **np** ou **n**

- o desencadeamento posterior da análise só deve terminar após ter conseguido obter os dois arcos completos referentes ao símbolo inicial que corresponderão, respectivamente, às hipóteses de **prgrama** poder ser considerado um **nome próprio** ou um **nome comum**.

- Após a detecção e correcção de erros, o utilizador é informado que, entre outras possibilidades de correcção, **prgrama** poderá ser corrigido por **programa**.

Esta alteração, pesada em termos computacionais, permite uma maior eficácia do sistema de detecção e correcção de erros.

- Quando a pesquisa termina é retornado o vértice correspondente à última análise efectuada. Em Paulo [Pau 93], se a pesquisa de constituintes terminasse ao nível da frase era retornado o arco completo correspondente à frase ou, caso contrário, uma variável não instanciada.

  Esta alteração permite que, em caso de falha na análise dos constituintes se possa detectar quais os arcos activos causadores da falha que serão objecto de reparação posterior.

  No exemplo anterior o analisador falhará a análise da palavra desconhecida **prgrama** e retornará o vértice **2** que permitirá a formulação de hipóteses sabendo que a falha foi detectada no vértice **2**.

- A adição de arcos aos pendentes é feita duma forma ordenada reflectindo uma busca estritamente em largura que, num sistema de detecção e correcção de erros, permitirá fornecer ao utilizador um maior leque de opções para correcção.

  Este tipo de ordenação, redundante neste sistema pois a análise só termina quando não existirem mais pendentes, seria útil se se utilizassem prioridades na escolha de hipóteses de arcos pendentes porque, existindo uma falha, permitiria obter os completos que englobassem o maior número de palavras.

4.4 Formulador de Hipóteses

O formulador de hipóteses (denotado por *form_hip* na secção **4** do Apêndice III) recorre:

- ao grafo obtido

- ao vértice onde terminou a pesquisa de constituintes
- à lista dos arcos completos referentes às palavras desconhecidas obtidas na inicialização

e tem como objectivo a reparação de falhas para permitir o prosseguimento da análise.

Assim, face ao vértice retornado pelo analisador, duas situações são possíveis:

1. O vértice retornado é o vértice final da sequência de palavras.

   Nesta situação não **haverá necessidade de formular hipóteses** que permitam superar a falha pois:

   a) Se existir (no grafo) um arco completo do símbolo inicial com vértice final igual ao vértice final da sequência de palavras conclui-se-á que existe uma **frase gramaticalmente correcta**.

      Ex: Dada a sequência de palavras 'o joão morreu'

      - a análise de constituintes termina no vértice final (vértice **4**)
      - existe um arco completo com categoria frase e vértice final **4**

      donde o utilizador será informado que a frase está **correcta**.

   b) Se não existir (no grafo) um arco completo do símbolo inicial com vértice final igual ao vértice final da sequência de palavras concluir-se-á que a **frase está incompleta** (este tipo de erro corresponde ao tipo de erros descritos na secção **3.1.3 B** do Cap. **3**/Parte **I**).

      Nesta situação, os arcos activos cujos vértices finais sejam iguais ao vértice final da sequência de palavras permitirão informar o utilizador sobre a omissão detectada.

      Note-se que estes arcos activos corresponderam a expectativas sobre constituintes que foram goradas ocasionando a falha do analisador.

      Ex: Dada a sequência de palavras 'o joão'

      - a análise de constituintes termina no vértice final (vértice **3**)
      - não existe um arco completo com categoria **frase** e vértice final **3**

e, após a **detecção e correcção de erros**, o utilizador será informado que a frase está incompleta faltando-lhe, **pelo menos**[16], um **advérbio de negação** ou um **verbo** na posição **3**.

2. O vértice onde terminou a pesquisa não é igual ao vértice final.

   Nesta situação há que **proceder à reparação da falha**.

   Para **reparar uma falha** o formulador de hipóteses opera da seguinte forma:

   a) Selecciona os arcos activos necessários ao prosseguimento da análise (ver secção **4.4.1**)

   b) Com base na lista dos arcos referidos anteriormente **constrói uma lista de novos pendentes** (ver secção **4.4.5** deste capítulo).

      Esta lista é construída utilizando técnicas de relaxamento (ver secção **4.4.2**/**4.4.3**/**4.4.4** deste capítulo) para permitir incorporações de arcos completos referentes a palavras (conhecidas ou não) nos arcos activos necessários ao prosseguimento da análise e visando a formulação de hipóteses sobre erros referentes a **palavras a mais**, **palavras omissas** ou **palavras desconhecidas**.

   Se **não foi possível reparar a falha**, ou seja, se a **lista de novos pendentes é vazia** significa que existem palavras a mais no fim da frase[17].

   Nesta situação, a formulação de hipóteses termina e a detecção e correcção do erro que originou a falha será efectuada no módulo de **detecção e correcção de erros** (ver secção **4.5** deste capítulo) por comparação entre a sequência inicial de palavras e a árvore sintáctica obtida.

   Ex: Dada a sequência de palavras 'o joão morreu morreu morru' a análise por grafo retornaria o vértice **4** (o vértice final é **6**) e o formulador de hipóteses não conseguiria detectar novos pendentes porque o verbo **morrer** foi subcategorizado com tipo **0**, ou seja, é um verbo que não requer qualquer complemento.

   Após a **detecção e correcção de erros** o utilizador será informado que a palavra **morreu** na posição **4** e a palavra **morru** na posição **5** são palavras **a mais**.

[16]Optou-se por informar o utilizador sobre as categorias pré-terminais em falta mas poder-se-ia, indiferentemente, ter informado o utilizador sobre a primeira categoria não terminal incompleta ou omissa (neste caso, informar-se-ia o utilizador seria informado da inexistência de um **sintagma verbal)**.

[17]Note-se que a inexistência de *frase* corresponde à situação descrita em 1. (o vértice onde terminou a pesquisa é igual ao vértice final) ou à existência de uma lista de novos pendentes formulando, pelo menos, hipóteses relativas a erros de omissão (situação tratada na página seguinte).

Se **foi possível reparar a falha**, ou seja, se a lista de **novos pendentes não for vazia** é invocado o analisador que tentará continuar a pesquisa de constituintes.

Se o analisador não conseguir progredir na pesquisa de constituintes, ou seja, se se verificar novamente uma falha no mesmo vértice não há hipótese de superar a falha e a análise de constituintes bem como a formulação de hipóteses terminam.

Esta situação, tipificada pelo exemplo seguinte, corresponde à existência de palavra(s) que não corresponde(m) à(s) categoria(s) sintáctica(s) esperada(s).

Ex: Dada a sequência de palavras 'o joão casou morreu' o vértice correspondente à primeira falha é **4** e, como o verbo **casar** é subcategorizado com tipo ***4*** (admitindo a ausência de complementos ou a existência de um sintagma preposicional) é formulada como hipótese a existência de uma palavra omissa correspondente à categoria pré-terminal **prep** (preposição) e a expectativa de um *sn*.

Por invocação do analisador o novo vértice retornado será **4** pois não é conseguida a análise de nenhum *sn* e, após a **detecção e correcção de erros**, o utilizador será informado que a palavra **morreu** na posição **4** está **a mais**[18].

4.4.1 Selecção dos arcos activos necessários ao prosseguimento da análise

Há vários tipos de arcos activos necessários.

Uns serão aqueles cujo vértice final coincide com o vértice onde terminou a pesquisa de constituintes. Destes apenas nos vão interessar aqueles que têm um **pré-terminal**[19] como primeira categoria em resto.

Ex1: Em 'o jao morreu'

- o vértice onde terminou a pesquisa é o vértice **2**
- os arcos activos onde terminou a pesquisa têm como vértice final **2** e como primeiras categorias em resto os pré-terminais **n** e **np**.

os arcos activos necessários ao prosseguimento da análise têm como vértice final **2** e primeiras categorias em resto **n** e **np.**

[18]Note-se que **casar** permite a ausência de complementos

[19] Os arcos activos que têm restrições como primeira categoria em resto não originarão falhas (ver regras de **concordância** na secção **4.2** deste capítulo)

Note-se que **o vértice correspondente à falha** coincide com o vértice onde terminou a pesquisa.

Ex2: Em 'a pessoa que o jao desapareceu morreu'

- o vértice onde terminou a pesquisa é o vértice **5**
- os arcos activos onde terminou a pesquisa terão como vértice final **5** e como primeiras categorias em resto os pré-terminais **np** e **n**.

Nesta situação, sendo o verbo **desaparecer** subcategorizado com tipo **0** (não permitindo a existência de complementos), interessa alargar o conjunto dos arcos necessários prevendo a hipótese de 'o jao' poder vir a ser considerado **a mais** pois já existe um arco completo referente à categoria não terminal **sn** atribuído à palavra 'que'.

Assim, para permitir uma formulação de hipóteses mais alargada (situação tipificada pelo ex. 2), se existirem arcos completos referentes à categoria não terminal necessária[20] cujo vértice final é igual ao vértice inicial dos arcos activos referentes a essa categoria seleccionar-se-ão também os arcos activos cuja primeira categoria em resto seja um **pré-terminal** e cujos **vértice inicial e final** sejam iguais ao **vértice final** desses arcos completos.

No caso do Ex2, os arcos activos necessários ao prosseguimento da análise terão

- como vértice final **5** e primeiras categorias em resto **n** e **np**
- como vértice final **4** e primeira categoria em resto **v**

Note-se que, neste caso, o **vértice correspondente á falha** deverá ser o vértice **4** e não o vértice **5** (vértice onde terminou a pesquisa).

Finalmente, **se não existirem arcos activos cuja primeira categoria em resto seja um pré-terminal e cujo vértice final seja o vértice onde terminou a pesquisa**[21] (situação tipificada pelo exemplo seguinte) serão considerados arcos activos necessários ao prosseguimento da análise

- os arcos activos cujas primeiras categorias em resto sejam pré-terminais e cujo vértice inicial seja igual ao vértice final da categoria não resolvida existente no canal.

[20] No Ex2 existe um arco completo de categoria **sn** cujo vértice final é **4**

[21] Existência de constituintes não resolvidos nos canais de movimentação

Ex3: Dada a frase 'a pessoa que o joão desapareceu morreu'

- o vértice onde terminou a pesquisa é o vértice **7**
- o arco activo onde terminou a pesquisa corresponde à categoria não terminal **orel** com vértice inicial **3** e vértice final **7** que não foi conseguida pois, obrigando a subcategorização deste verbo à inexistência de complementos, não permitiu a **ancoragem do sn** (constituído pelo pronome relativo 'que') **existente no canal relativo**.

Nesta situação, dever-se-ia reanalisar a partir do vértice **4** (vértice final do **sn** existente no canal ***rel***) considerando o arco completo com categoria **sn** constituído pelo pronome relativo 'que' com função de sujeito da oração relativa e as palavras {o,joão} como palavras a mais.

Note-se que, neste caso, o vértice onde terminou a pesquisa é o vértice **7** e o **vértice correspondente à falha** será o vértice **4**.

4.4.2 Incorporação de palavras desconhecidas

A incorporação de um arco completo *ac(U4,U5,desc:**Pal**,Q1,S1,R1,V1)* num arco activo necessário ao prosseguimento da análise *aa(U3,U4,Cat,Cat1,Cats,Q,S,R,V)* é conseguida por relaxamento (ver secção **3.2** do Cap. **3**/Parte **I**) atribuindo a categoria pré-terminal ***Cat1*** à palavra desconhecida ***Pal***.

Ex: Dada a sequência de palavras 'O prgrama termina' o vértice correspondente à falha será o vértice **2**, os arcos activos necessários ao prosseguimento da análise serão:

(1) *aa(1,2,r14:sn(**Pes**, Num,Gen, caso=nom,cl=def, sint(sn):[conc:[**Pes**,Num,Gen],d(num=sin, gen=masc,cl=def,o),**NP**]), ilha **np(Pes, Num1, Gen1, NP)**, [{concorda(num=sin, **Num1**, Num), concorda(gen = masc,**Gen1**,Gen)}],[]-[],[]-[],[]-[],[]-[])*

(2) *aa(2,2,r15:resto_sn(**Pes**,**Num**,**Gen**, sint(resto_sn):[conc:[**Pes**,**Num**,**Gen**],**N**, Orel]), ilha **n(Pes, Num, Gen, N)**, [ilha orel(Orel)],[]-[],[]-[],[]-[],[]-[])*

O único arco completo inicial com vértice inicial superior ou igual a **2** correspondente a uma palavra desconhecida é *ac(2,3,desc:**pal_dsc(prgrama)**,Q-Q,S-S,R-R,V-V)*.

Da incorporação deste arco com o arco (1) obter-se-á

*aa(1,3,r14:sn(**pes=3**,Num,Gen,caso=nom,cl=def,sint(sn):[conc:[**pes=3**,Num,Gen], d(num=sin, gen=masc,cl=def,o),**np(pes=3,Num1,Gen1,pal_dsc(progrma))**]), {concorda(num=sin,**Num1**, Num),concorda(gen=masc,**Gen1**,Gen)},[],[]-[],[]-[], []-[],[]-[])*

Da incorporação deste arco com o arco (2) obter-se-á

*aa(2,3,r15:resto_sn(**pes=3**,**Num**,**Gen**,sint(resto_sn):[conc:[**pes=3**,**Num**,**Gen**],**n(pes=3,num= Num, gen=Gen,pal_dsc(prgrama))**,Orel]),ilha orel(Orel),[],[]-[],[]-[],[]-[],[]-[])*

Estes dois arcos que serão inseridos na lista de **novos pendentes** traduzem a categorização da palavra ***prgrama*** como nome próprio ou como nome comum permitindo que, no desencadeamento posterior da análise, se obtenha uma **frase** onde a palavra ***prgrama*** é categorizada como nome próprio e outra onde a palavra ***prgrama*** é categorizada como nome comum.

Assim, **a detecção e correcção de erros** informará o utilizador que

- ou ***prgrama*** pode ser corrigido por **programa**
- ou ***prgrama*** é um neologismo na categoria **np**
- ou ***prgrama*** é um neologismo na categoria **n**

4.4.3 Arcos referenciando a existência de palavras a mais

Os arcos referenciando a existência de palavras a mais resultam da incorporação de um arco completo inicial com vértice inicial superior ou igual ao vértice correspondente à falha nos arcos activos necessários ao prosseguimento da análise.

Deste processo de incorporação, conseguido por relaxamento de um vértice, ou seja, permitindo a incorporação de um pré-terminal concretizado num dado vértice com o mesmo pré-terminal esperado num vértice anterior (ver secção **3.2** do Cap. **3**/Parte **I**) resultam arcos cujo vértice final é o vértice final do arco completo inicial.

Note-se que os arcos referenciando as palavras a mais deverão ser resultantes da incorporação do arco completo inicial *mais próximo* cuja primeira categoria seja um pré-terminal necessário. (Em 'o o joão matou o pedro' o vértice correspondente á falha é **2**, e o arco completo inicial *mais próximo* cuja categoria seja um pré-terminal necessário corresponde à palavra 'joão').

Ex1: Dada a sequência de palavras 'o o joão morreu' os arcos activos necessários ao prosseguimento da análise serão:

(1) *aa(1,**2**,r14:sn(**Pes**,Num,Gen,caso=nom,cl=def, sint(sn):[conc:[**Pes**, Num, Gen], d(num = sin,gen = masc,cl = def, o), **NP**]), ilha **np(Pes, Num1, Gen1, NP)**, [{concorda(num = sin, **Num1**, Num), concorda(gen = masc, **Gen1**, Gen)}],[] - [],[] - [],[] - [],[] - [])*

(2) *aa(2,**2**,r15:resto_sn(**Pes**,**Num**,**Gen**, sint(resto_sn):[conc:[**Pes**,**Num**,**Gen**], **N**, Orel]), ilha **n(Pes,Num,Gen,N)**, [ilha orel(Orel)],[]-[],[]-[],[]-[],[]-[])*

Os arcos completos iniciais cujo vértice inicial é igual ou superior ao vértice correspondente à falha são:

(3) ac(2,3,pal:d(num=sin,gen=masc,d(num=sin,gen=masc,o)), Q-Q, S-S, R-R, V-V)

*(4) ac(3,**4**,pal:**np(pes=3,num=sin,gen=masc,np(pes=3,num=sin,gen=masc,joão))**, Q-Q, S-S, R-R, V-V)*

(5) ac(4,5,pal:v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,mod=indic,sc=0,morreu,v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,mod=indic,sc=0,morreu,morrer)),Q-Q,S-S,R-R,V-V)

Os arcos referenciando a existência de palavras a mais serão obtidos da seguinte forma:

- A tentativa de incorporação do arco completo (3) nos arcos activos necessários ao prosseguimento da análise falha
- Na incorporação relaxada do arco completo (4) no arco activo necessário (1) obter-se-á:

  *aa(1,**4**,r14:sn(**pes=3**,Num,Gen,caso=nom,cl=def,sint(sn):[conc:[**pes=3**,Num,Gen],d(num=sin,gen=masc,cl=def,o),**np(pes=3,num=sin,gen=masc,joão)**]),{concorda(num=sin,**num=sin**,Num),concorda(gen=masc,**gen=masc**,Gen)},[],[]-[],[]-[],[]-[],[]-[])*

- A tentativa de incorporação relaxada do arco completo (4) no arco activo necessário (2) falha pois a categoria pré-terminal a incorporar (categoria referente ao arco completo (4)) é **np** e a categoria pré-terminal esperada (categoria referente ao arco activo (2)) é **n**.
- As tentativas de incorporação do arco completo (5) nos arcos activos necessários (1) e (2) também falham.

Deste modo, o arco activo referenciando a existência de palavras a mais é

*aa(1,**4**,r14:sn(**pes=3**,Num,Gen,caso=nom,cl=def,sint(sn):[conc:[**pes=3**,Num,Gen],d(num=sin,gen=masc,cl=def,o),**np(pes=3,num=sin,gen=masc,joão)**]),{concorda(num=sin,**num=sin**,Num), concorda(gen=masc,**gen=masc**,Gen)},[],[]-[],[]-[], []-[],[]-[])*

Este arco que será colocado na lista de **novos pendentes** traduz a existência de um **sn** categorizando a sequência de palavras {o,joão} permitindo que, no desencadeamento posterior da análise, se obtenha uma **frase** constituída pelas palavras {o,joão,morreu}.

Por comparação da sequência de palavras inicial ('o o joão morreu') com a sequência de palavras obtidas ('o joão morreu') **a detecção e correcção de erros** informará o utilizador que a palavra **o** na posição **2** deverá ser considerada **a mais.**

Ex2: Dada a sequência de palavras 'a pessoa que o jao desapareceu morreu' os arcos activos necessários ao prosseguimento da análise serão:

(1) *aa(4,**5**,r14:sn(**Pes**,Num,Gen,caso=nom,cl=def, sint(sn):[conc:[**Pes**, Num, Gen], d(num=sin,gen=masc,cl=def,o),**NP**]), ilha **np(Pes,Num1,Gen1,NP)**,[{concorda(num=sin,**Num1**,Num), concorda(gen=masc,**Gen1**,Gen)}],[]-[],[]-[],[sn(pes=Pes2,num=Num2,gen=Gen2,caso=C,cl=rel,sint(sn):[conc:[pes=Pes2,num=Num2,gen=Gen2],pron(pes=Pes2,num=Num2,gen=Gen2,cl=rel,caso=C,que)])]-[sn(pes=Pes2,num=Num2,gen=Gen2,caso=C,cl=rel,sint(sn):[conc:[pes=Pes2,num=Num2,gen=Gen2],pron(pes=Pes2,num=Num2,gen=Gen2,cl=rel,caso=C,que)])],[] - [])*

(2) *aa(5,**5**,r15:resto_sn(**Pes**,**Num**,**Gen**, sint(resto_sn):[conc:[**Pes**,**Num**,**Gen**], **N**, Orel]), ilha **n(Pes,Num,Gen,N)**, [ilha orel(Orel)], []-[],[]-[], [sn(pes=Pes2,num=Num2,gen=Gen2,caso=C,cl=rel,sint(sn):[conc:[pes=Pes2,num=Num2,gen=Gen2],pron(pes=Pes2,num=Num2,gen=Gen2,cl=rel,caso=C,que)])],[sn(pes=Pes2,num=Num2,gen=Gen2,caso=C,cl=rel,sint(sn):[conc:[pes=Pes2,num=Num2,gen=Gen2],pron(pes=Pes2,num=Num2,gen=Gen2,cl=rel,caso=C,que)])],[]-[])*

*(3) aa(4,**4**,r27:verbo(Pes,Num,Gen,T,F,M,SCat,Voz,sint(verbo):[conc:[Pes,Num,Gen],**NV**,Resto]),ilha **v(Pes1,Num1,Gen1,T1,F,M,S,Vf,NV)**,[ilha resto(Pes2,Num2,Gen2,conj=NV,S,SCat,Voz,Resto),{tv(**NV**,Resto,**T1**,T,**M**),concorda(**Pes1**,Pes2,Pes),concorda(**Num1**,Num2,Num),concorda(**Gen1**,Gen2,Gen)}],[]-[],[]-[],[]-[],[]-[])]*

Os arcos completos iniciais cujo vértice inicial é igual ou superior ao vértice correspondente à falha (**4**) são:

(4) ac(4,5,pal:d(num=sin,gen=masc,d(num=sin,gen=masc,o)), Q-Q, S-S, R-R, V-V)

*(5) ac(**5**,6,desc:**pal_dsc(jao)**,Q-Q,S-S,R-R,V-V)*

*(6) ac(6,7,pal:**v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,mod=indic,sc=0,desapareceu,v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,mod=indic,sc=0,desapareceu,desaparecer)**),Q-Q,S-S,R-R,V-V)*

(7) ac(7,8,pal:v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,mod=indic,sc=0,morreu,v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,mod=indic,sc=0,morreu,morrer)),Q-Q,S-S,R-R,V-V)

Os arcos referenciando a existência de palavras a mais serão obtidos da seguinte forma:

- A tentativa de incorporação do arco completo (4) nos arcos activos necessários ao prosseguimento da análise falha
- Na incorporação relaxada do arco completo (6)[22] no arco activo necessário (3) obter-se-á:

  *aa(4,**7**,r27:verbo(Pes,Num,Gen,T,**forma=fin**,**modo=indic**,SCat,Voz,sint(verbo):[conc:[Pes,Num,Gen],**v(pes=3,num=sin,gen=Gen1,tempo=ppref,forma=fin,modo=indic,sc=0,desapareceu,desaparecer)**,Resto]),ilha resto(Pes2,Num2,Gen2,conj=v(pes=3,num=sin,gen=Gen1,tempo=ppref,forma=fin,modo=indic,sc=0,desapareceu,desaparecer),sc=0,SCat,Voz,Resto),[{tv(**v(pes=3,num=sin,gen=Gen1,tempo=ppref,forma=fin,modo=indic,sc=0,desapareceu,desaparecer)**,Resto,**tempo=ppref**,T,**modo=indic**),concorda(**pes=3**,Pes2,Pes),concorda(**num=sin**,Num2,Num),concorda(**Gen1**,Gen2,Gen)}],[]-[],[]-[],[]-[],[]-[])]*[23]
- A tentativa de incorporação do arco completo (6) nos arcos activos necessários (1) e (2) falham pois a categoria pré-terminal a incorporar (categoria referente ao arco completo (4)) é **v** e as categorias pré-terminais esperadas são, respectivamente, **np** e **n**.
- A tentativa de incorporação do arco completo (7) nos arcos activos necessários também falham.

Deste modo, o arco activo referenciando a existência de palavras a mais é o arco resultante da incorporação relaxada do arco completo (6) no arco activo necessário (3).

Este arco que será inserido na lista de **novos pendentes** traduz a existência de um **verbo** categorizando a palavra **desapareceu** permitindo que, no desencadeamento posterior da análise, se obtenha uma **frase** constituída pelas palavras {a,pessoa,que,desapareceu,morreu}.

[22]Das tentativas de incorporação do arco completo (5) nos arcos activos necessários (1) e (2) serão obtidos novos pendentes que formulam hipóteses sobre a categorização da palavra desconhecida **jao** como **np** ou como **n.**

[23]Note-se a inexistência do **sn** constituído pelo pronome relativo no canal **slash**

Por comparação da sequência de palavras inicial ('a pessoa que o jao desapareceu morreu') com a sequência de palavras obtidas ('a pessoa que desapareceu morreu') **a detecção e correcção de erros** informará o utilizador que a cadeia '**o jao**' deverá ser considerada **a mais.**

4.4.4 Arcos referenciando a existência de palavras omissas

Os arcos referenciando a existência de palavras omissas são obtidos incorporando em cada arco activo necessário ao prosseguimento da análise A um arco completo auxiliar B que terá:

- como categoria proposta a primeira categoria em resto de A atribuída a uma palavra denotada por *pal_omi* **prefixada** pelo **vértice final** de A

  A prefixação de *pal_omi* com o vértice final do arco A permitirá simplificar a pesquisa posterior do vértice onde ocorreu um erro relativo a uma omissão (pesquisa a efectuar na **detecção de erros referentes a palavras a mais, omissas e desconhecidas** - secção **4.5.1** deste capítulo).

- os vértices iniciais e finais iguais ao vértice final de A

- os canais serão:

  a) não instanciados para permitirem incorporações em arcos que transportem material linguístico

  b) restringidos para não permitirem movimentações de material linguístico

Deste processo de incorporação que utiliza o relaxamento de uma categoria (ver secção **3.2** do Cap. **3**/Parte **I)** resulta um arco cujo vértice final é igual ao vértice do arco activo necessário.

Ex: Dada a sequência de palavras 'o morreu' e dados os arcos activos necessários:

(1) *aa(1,2,r14:sn(**Pes**,Num,Gen,caso=nom,cl=def, sint(sn):[conc:[**Pes**, Num, Gen], d(num = sin,gen = masc,cl = def, o), **NP**], ilha **np(Pes, Num1, Gen1, NP)**, [{concorda(num = sin, **Num1**, Num), concorda(gen = masc, **Gen1**, Gen)}],[] - [],[] - [],[] - [],[] - [])*

(2) *aa(2,2,r15:resto_sn(**Pes**,**Num**,**Gen**, sint(resto_sn):[conc:[**Pes**,**Num**,**Gen**], **N**, Orel], ilha **n(Pes,Num,Gen,N)**, [ilha orel(Orel)],[]-[],[]-[],[]-[],[]-[])]*

os arcos correspondentes às palavras omissas seriam

(1) *ac(**2**,**2**,omi:**np(pes=3,Num1,Gen1,np(pes=3,Num1,Gen1,2:pal_omi))**,Q-Q,S-S,R-R,V-V)*

(2) *ac(**2**,**2**,omi:**n(pes=3,Num1,Gen1,n(pes=3,Num1,Gen1,2:pal_omi))**, Q-Q,S-S,R-R,V-V)*

e os resultados da incorporação serão, respectivamente,

(1) *aa(1,**2**,r14:sn(**pes=3**,Num,Gen,caso=nom,cl=def,sint(sn):[conc:[**pes=3**,Num,Gen],d(num=sin,gen=masc,cl=def,o),**np(pes=3,num=Num1,gen=Gen1,2:pal_omi)**]),{concorda(num=sin,N**um1**,Num),concorda(gen=masc,**Gen1**,Gen)},[],[]-[],[]-[],[]-[],[]-[])*

(2) *aa(2,**2**,r15:resto_sn(**pes=3**,**Num**,**Gen**,sint(resto_sn):[conc:[**pes=3**,**Num**,**Gen**],**n(pes=3,Num,Gen,2:pal_omi)**,Orel], ilha orel(Orel), [],[]-[], []-[],[]-[],[]-[])]*

Estes arcos que serão inseridos na lista de **novos pendentes** traduzem a existência de um **np** ou um **n** categorizando uma **palavra omissa** no vértice **2** permitindo que, à posteriori, o módulo de **detecção e correcção de erros** informe o utilizador da falta uma palavra categorizada pela categoria pré-terminal **np** ou pela categoria pré-terminal **n**.

4.4.5 Construção da lista dos novos pendentes

Os **novos pendentes** corresponderão a formulações (limitadas aos tipos de erros descritos na secção **3.1** do Capítulo **3**/Parte **I)** baseadas:

- na existência/ausência de arcos que incorporam **palavras desconhecidas**[24]
- na existência/ausência de **palavras conhecidas** com vértice igual ou superior à falha.

Assim, dada a lista dos arcos activos necessários ao prosseguimento da análise, a lista dos **novos pendentes** será constituída por **arcos formulando hipóteses** para reparação de **erros** devidos a:

1. **Desconhecimento** (tipo de erros descritos na secção **3.1.1** do Cap. **3**/Parte **I**) se **existirem** arcos que incorporam **palavras desconhecidas** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.2** deste capítulo) e **não existirem palavras** com vértice igual ou superior ao vértice correspondente à falha.
2. **Existência de palavras a mais** (tipo de erros descritos na secção **3.1.2** do Cap. **3**/Parte **I**) ou **omissas** (tipo de erros descritos na secção **3.1.3 A** do Cap. **3** da Parte **I**) se **não existirem** arcos que incorporam **palavras desconhecidas** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.2** deste capítulo) e **existirem palavras** com vértice igual ou superior ao vértice correspondente à falha.

[24]Recorde-se que os **arcos que incorporam palavras desconhecidas** são obtidos por incorporações de arcos completos correspondentes a palavras desconhecidas em arcos activos necessários ao prosseguimento da análise (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.2** deste capítulo).

3. **Desconhecimento** (tipo de erros descritos na secção **3.1.1** do Cap. **3**/Parte **I**) ou **existência de palavras a mais** (tipo de erros descritos na secção **3.1.2** do Cap. **3**/Parte **I**) se **existirem** arcos que incorporam **palavras desconhecidas** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.2** deste capítulo) e **existirem palavras** com vértice igual ou superior ao vértice correspondente à falha.

Nota1: Se não existirem arcos activos necessários ao prosseguimento da análise a **lista de pendentes** retorna **vazia**[25] e, como já foi referido anteriormente (ver **lista de pendentes vazia** na pág. 64), os erros existentes serão detectados e corrigidos no módulo de **detecção e correcção de erros** (ver secção **4.5** deste capítulo).

Nota2: Se existirem arcos activos necessários ao prosseguimento da análise e **não existirem palavras** com vértice igual ou superior ao vértice correspondente à falha nem **arcos incorporando palavras desconhecidas**[26] é porque o vértice onde terminou a pesquisa é igual ao vértice final da sequência de palavras e os erros corresponderão à situação de **frase incompleta** (tipo de erros descritos na secção **3.1.3 B** do Cap. **3**/Parte **I**).

Como já foi referido anteriormente (ver **vértice retornado é o vértice final da sequência de palavras** na pág. 63) não será desencadeada a **reparação da falha** pois não é possível prosseguir a análise e o erro será detectado no módulo de **detecção e correcção** de erros (ver secção **4.5** deste capítulo) recorrendo às árvores sintácticas existentes no grafo.

Pendentes formulando hipóteses sobre desconhecimento

Se não existirem arcos completos iniciais com vértice inicial superior ou igual ao vértice correspondente à falha a lista de novos pendentes será formada, unicamente, pelos arcos que incorporam **palavras desconhecidas** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.2** deste capítulo)

Ex: Dada a sequência de palavras 'o joão morru':

- o vértice correspondente à falha é o vértice **3**

[25]Existência de **palavras (conhecidas ou desconhecidas)** a mais no fim da frase. Ex. Em 'o joão morreu morreu morru' o vértice final é **6**, o vértice onde terminou a pesquisa é 4 e não existem arcos activos necessários ao prosseguimento da análise (**morreu** não admite complementos) donde a **lista de novos pendentes** é **vazia** e, posteriormente, o utilizador é informado que **morreu** na posição **4** e **morru** na posição **5** são palavras **a mais.**

[26]Neste caso não poderão existir palavras desconhecidas porque a sua existência permitiria a incorporação da primeira palavra desconhecida num arco activo necessário ao prosseguimento da análise (o vértice onde terminou a pesquisa seria igual ao vértice inicial dessa palavra).

- as categorias pré-terminais necessárias para prosseguimento da análise são um **advérbio de negação** ou um **verbo**

- a lista dos arcos completos iniciais com vértice inicial superior ou igual a **3** é vazia

- a lista dos arcos completos referentes a palavras desconhecidas com vértice inicial superior ou igual a **3** é **[*ac(3,4,desc:pal_dsc(morru),Q-Q,S-S,R-R,V-V)*]**

- a lista dos **novos pendentes** será constituída pelos arcos que incorporam a palavra desconhecida **morru**

Estes arcos pendentes corresponderão às hipóteses de **morru** poder ser categorizado como um **advérbio de negação** ou como um **verbo**.

Posteriormente, após a **detecção e correcção de erros**, o utilizador será informado que, entre outras correcções, **morru** poderá ser corrigido por **morreu** ou poderá ser um neologismo na categoria dos verbos[27].

Nota: Nesta situação não poderiam ser formuladas hipóteses sobre a existência de palavras a mais (estas não existem) mas poderia ter sido formulada, em paralelo a hipótese de existência de palavras omissas.

No entanto, foi assumido que, existindo palavras desconhecidas, deve tentar-se a sua incorporação como constituintes da frase, preterindo o recurso à utilização de palavras omissas.

Esta opção que restringe o número de hipóteses formuladas foi adoptada por se pensar que a existência de uma palavra desconhecida num documento deve ser entendida, em geral, como uma tentativa não conseguida de escrita de um vocábulo.

A desvantagem desta suposição é a tentativa de correcção de uma palavra desconhecida quando o sistema deveria informar o utilizador da existência de uma omissão.

Ex: Em 'o morru morreu' o sistema informará o utilizador que **morru** poderá corresponder a um neologismo[28] nas categorias **np** ou **n** e deveria considerar **morru** como cadeia a mais e a existência de uma palavra omissa categorizada por **np** ou **n** na posição **2**.

[27]Situação improvável em tempos de verbos portugueses

[28]Eventualmente podem ser apresentadas outras hipóteses de correcção. Por ex., o nome comum **morro**

Pendentes formulando hipóteses sobre a existência de palavras a mais ou omissas

Se existirem arcos completos iniciais com vértice inicial superior ou igual ao vértice correspondente à falha e não existirem arcos incorporando palavras desconhecidas a lista dos novos pendentes será constituída por:

1. arcos referenciando a existência de **palavras a mais** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.3** deste capítulo)

2. arcos referenciando a existência de palavras **omissas** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.4** deste capítulo) .

Ex1: Dada a sequência de palavras 'o o joão morreu'

- o vértice correspondente à falha é **2**
- as categorias pré-terminais necessárias para prosseguimento da análise são um *np* ou um *n*
- Os arcos completos iniciais com vértice inicial superior ou igual a **2** correspondem às palavras 'o', 'joão' e 'morreu'
- a lista dos arcos completos correspondentes a palavras desconhecidas com vértice inicial superior ou igual a **2** é vazia

a lista dos **novos pendentes** será constituída pelos arcos que referenciem hipóteses de erros devidos a omissões ou à existência de palavras a mais permitindo que, posteriormente, o utilizador seja informado da existência de uma **palavra a mais** (**o**).

Ex2: Dada a sequência de palavras 'o joão tem comer maçãs'[29]

- o vértice correspondente à falha é **4**
- as categorias pré-terminais necessárias para prosseguimento da análise são:

    a) a necessidade de um verbo no particípio passado (correspondendo à hipótese do verbo **ter** ser auxiliar)

[29]Esta situação (incorrecção verbal) corresponde a um **erro de concordância** que não foi tratado no âmbito desta tese. Neste caso, poder-se-ia considerar **comer** como palavra a mais mas por ex., em 'o joão tem tratar dos pássaros' **tratar** não deveria ser considerada palavra a mais mas sim forma verbal que deveria ocorrer no particípio passado.

b) *det*, *n* ou *np* (a subcategorização de **ter** obrigará à existência de um complemento directo - correspondendo à hipótese de **ter** ser o verbo principal)

- os arcos completos iniciais com vértice inicial superior ou igual a **4** correspondem aos terminais 'comer' e 'maçãs'
- a lista dos arcos completos correspondentes a palavras desconhecidas com vértice inicial superior ou igual a **4** é vazia
- a nova lista de pendentes será constituída por dois arcos:

  a) um referenciando a **omissão** de um verbo no particípio passado na posição **4**

  b) outro referenciando a existência de uma **palavra a mais** (**comer**).

permitindo que, posteriormente, o utilizador seja informado da existência de uma **palavra a mais** (**comer**).

Recorde-se que, a detecção deste tipo de erros é realizada no módulo de **detecção e correcção de erros** (ver secção **4.5** deste capítulo) e será conseguida por comparação entre a sequência inicial de palavras e a árvore sintáctica obtida.

Nota: São excluídas as hipóteses de palavras omissas referentes a pré-terminais que foram concretizados em arcos referenciando a existência de palavras a mais.

Com esta condição reduz-se o leque de hipóteses formuladas preterindo categorias omissas face à existência das mesmas categorias referenciando palavras a mais.

Assim, no ex. da Fig. 4.4 obter-se-iam:

1. um arco que referencia a existência de uma palavra a mais (**o**) resulta da incorporação do arco completo inicial referente ao terminal **joão** num arco activo necessário.
2. um arco que referencia a existência de uma palavra omissa categorizada por **n**[30].(o arco que referencia a existência de uma palavra omissa categorizada por **np**

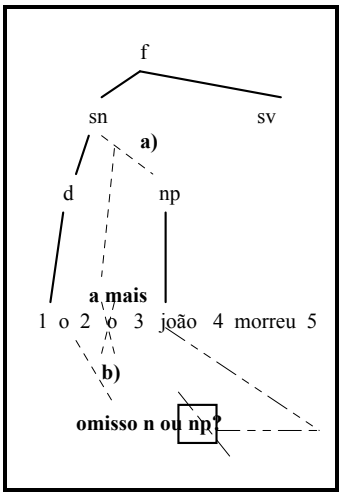

Fig. 4.4 Árvore sintáctica parcial

[30]Poder-se-iam ter excluído todas as hipóteses de omissão relativas a um não terminal que foi concretizado (neste ex. excluir-se-ia também a hipótese de omissão relativa a **n**) o que conduziria a que, por ex., em 'o a quem o joão comprou a casa morreu' o utilizador fosse informado que as palavras {a,quem,o,morreu} estão **a mais** considerando uincamente como *frase* 'o joão comprou a casa'.

não será considerado pois a categoria pré-terminal **np** foi concretizada em 1.).

Nota1: A formulação de hipóteses não diferencia a duplicação de palavras da existência de palavras a mais não duplicadas (tipo de erros descritos na secção **3.1.2** do Cap. **3**/Parte **I**).

Nota2: As situações referentes a palavras com categoria ou forma diferente da requerida (tipo de erros descritos na secção **3.2.C** do Cap. **3**/Parte **I**) são formuladas como uma omissão da categoria ou forma requerida e a existência de uma palavra a mais (correspondente à categoria ou forma apresentada).

Ex: Dada a sequência de palavras 'eu acto as cordas' o utilizador será informado da falta de um verbo e da existência de uma palavra a mais 'acto'[31].

Pendentes formulando hipóteses sobre desconhecimento ou existência de palavras a mais

Se existirem arcos completos iniciais com vértice inicial superior ou igual ao vértice correspondente à falha e arcos incorporando palavras desconhecidas a lista dos novos pendentes será constituída por:

1. arcos referenciando a existência de **palavras a mais** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.3** deste capítulo)

2. arcos incorporando **palavras desconhecidas** (ver secção **3.2** do Cap. **3**/Parte **I** e secção **4.4.2** deste capítulo)

Ex: Dada a sequência de palavras 'o joão nao[32] morreu'

- o vértice correspondente à falha é o vértice **3**

- as categorias pré-terminais necessárias para prosseguimento da análise serão um **advérbio de negação** ou um **verbo**.

- Os arcos completos iniciais com vértice inicial superior ou igual a **3** correspondem ao terminal 'morreu'

- a lista dos arcos completos atribuídos a palavras desconhecidas com vértice inicial superior ou igual a **3** é **[*ac(3,4,desc:pal_dsc(nao),Q-Q,S-S,R-R,V-V)*]**

[31]No futuro, poder-se-ia superar esta situação considerando que a existência de uma palavra a mais e uma omissão no mesmo vértice implicam a correcção da palavra considerada a mais na categoria pré-terminal da palavra omissa. Neste ex., o corrector proporia hipóteses de correcção para **acto** na categoria **verbo**, ou seja, informaria o utilizador que **acto** poderia ser corrigido por **ato**.

[32]Não confundir **não** com **nao** (palavra desconhecida)

- Os arcos resultantes da incorporação do arco completo referente à palavra desconhecida nos arcos activos necessários categorizarão 'nao' como **advérbio de negação** ou **verbo**

- a nova lista de pendentes será constituída por:

- o arco referenciando a existência de uma **palavra a mais** (**nao**).

- os arcos resultantes da incorporação da **palavra desconhecida nao** nos arcos activos necessários

permitindo que, posteriormente, o utilizador seja informado que uma das correcções para a palavra desconhecida **nao** poderá ser **não**.

Nota: Numa tentativa de **restringir o número de hipóteses formuladas** foram rejeitados da lista dos novos pendentes todos os arcos activos desta lista cuja primeira categoria em resto fosse um não terminal ou um pré-terminal que não incorporasse um arco completo inicial (referente a uma palavra ou a uma palavra desconhecida).

Ex: Dada a sequência de palavras 'o joão comu um bolo' a palavra desconhecida **comu** poderá ser categorizada como **advérbio de negação** ou **verbo**.

Sendo tentada a incorporação de um terminal verificar-se-ia que **comu** só poderia ser categorizado como verbo.

Esta estratégia pressupõe a inexistência de dois erros consecutivos, por ex., se considerássemos a sequência de palavras 'o joão comeu u com boolo' o utilizador seria informado que as palavras **u**, **com** e **boolo** são palavras **a mais**[33].

Ressalve-se a existência de dois erros consecutivos relativos a palavras desconhecidas. Neste caso, a segunda palavra desconhecida será incorporada no pré-terminal ou não terminal esperado.

Por ex. em 'o livro fi ddo pelo joão à maria' **fi** poderá ser categorizado como **advérbio de negação** ou **verbo** porque a incorporação do terminal seguinte no pré-terminal ou não terminal esperado será possível, ou seja, **ddo** pode ser categorizado, respectivamente, como **verbo** ou **particípio passado** de um verbo.

[33]Note-se que a subcategorização do verbo **comer** permite a inexistência de complementos ou a existência de um **sn.** Os novos pendentes formulariam hipóteses categorizando **u** como **determinativo m**as as tentativas de incorporação do terminal seguinte **(com)** nas categorias esperadas **np** e **n** falham.

4.5 Corrector de erros

O corrector de erros (denotado por *alerta* na secção **5** do Apêndice III) recorre:

- ao grafo obtido
- ao vértice onde terminou a pesquisa de constituintes
- à lista dos arcos completos referentes às palavras desconhecidas

e opera de acordo com o algoritmo seguinte:

1. Retira do grafo as representações sintácticas obtidas nos arcos completos de maior amplitude correspondentes ao símbolo inicial da gramática (ver *a representação sintáctica dos símbolos não terminais* referida na descrição da **Gramática** - secção **4.2** deste capítulo).

   Ex: Dada a sequência de palavras 'o joão comeu o bolo' e sendo o verbo 'comer' subcategorizado com tipo ***1*** (permitindo a inexistência de complementos ou a existência de um complemento directo) os arcos completos correspondentes ao símbolo inicial serão:

   a) um arco completo traduzindo a existência da frase 'o joão comeu' do vértice **1** ao vértice **4**

   b) um arco completo traduzindo a existência da frase 'o joão comeu o bolo' do vértice **1** ao vértice **6**

   donde a representação sintáctica sobre a qual incidirá a detecção e correcção de erros será a representação correspondente à análise da frase 'o joão comeu o bolo' porque o arco completo de maior amplitude é o arco referido em b)).

2. Se não existirem representações sintácticas correspondentes ao símbolo inicial da gramática, isto é, se não existirem arcos completos com categoria igual ao símbolo inicial da gramática ou, por outras palavras, se a análise não pôde ser completada, os erros existentes referenciarão situações de **frases incompletas** (tipo de erros descritos na secção **3.1.3 B** do Cap. **3**/Parte **I**).

   Assim, com base no vértice onde terminou a pesquisa, constrói uma lista de erros contendo

   - as categorias pré-terminais que são primeiras categorias em resto dos arcos activos cujo vértice final é o vértice onde terminou a pesquisa.

e informa o utilizador sobre os erros encontrados.

Ex: Dada a sequência de palavras 'o joão'

- a análise de constituintes terminará no vértice **3** (vértice final)
- as categorias pré-terminais necessárias para prosseguimento da análise seriam um **advérbio de negação** ou um **verbo**
- o utilizador será informado que a frase está incompleta faltando-lhe, pelo menos, um **advérbio de negação** ou um **verbo** na posição **3**.

3. Se existiram representações sintácticas correspondentes ao símbolo inicial da gramática:

   A) **Detecta**, em cada representação sintáctica, **os erros referentes a palavras a mais, omissas e desconhecidas** (ver secção **4.5.1** deste capítulo).

   Note-se que para cada representação sintáctica haverá uma lista de erros.

   B) Com base nas listas de erros obtidas em A) consideram-se:

   - listas de erros *mais plausíveis* (as que contêm o menor número de erros)
   - representações sintácticas *mais plausíveis* (as que correspondem às listas de erros *mais plausíveis*).

   C) **Detecta e corrige os erros de concordância** (ver secção **4.5.2** deste capítulo) existentes nas representações sintácticas *mais plausíveis* relatando de seguida a situação ao utilizador.

   D) **Corrige os erros ortográficos** (ver secção **4.5.3** deste capítulo) com base nas palavras desconhecidas[34] com categoria atribuída existentes nas listas de erros *mais plausíveis* e a situação relativa a erros em geral será relatada ao utilizador.

[34]Como foi referido anteriormente não são propostas para correcção as palavras conhecidas que estão mal escritas (Em 'eu acto as cordas' o utilizador é informado da **omissão** de um **verbo** e da existência de uma palavra **a mais** - **acto).**

4.5.1 Detecção de erros referentes a palavras a mais, omissas e desconhecidas

Dadas:

- uma representação sintáctica do símbolo inicial
- a lista dos arcos completos iniciais
- a listas dos arcos completos referentes a palavras desconhecidas

a pesquisa dos erros correspondentes a palavras a mais, omissas e desconhecidas (denotada por *erros_frase* na secção **5.1** do Apêndice III) opera de acordo com o algoritmo seguinte:

1. Compara a representação sintáctica do símbolo inicial com os arcos completos iniciais:

    a) constrói-se uma lista com todos os pré-terminais que referenciam erros ortográficos (caso de categorias atribuídas a palavras desconhecidas) ou omissões (casos de categorias atribuídas a palavras omissas).

    Ex1: Dada a sequência de palavras 'o joão comu o bolo'

    - Na representação sintáctica obtida **comu** é categorizado como um **verbo**
    - Nesta 1ª fase de detecção de erros obter-se-á a lista com todos os pré-terminais que referenciam erros ortográficos ou omissões que, neste caso, é constituída pela categoria pré-terminal **v** categorizando **comu**

    b) são inseridos na lista de erros todos os pré-terminais categorizando palavras que não pertençam à representação sintáctica (erros referentes à existência de palavras a mais) sob a forma

    (U1,U2,amais:LCat) onde *U1* e *U2* são os vértices inicial e final da palavra e *LCat* a lista de categorias pré-terminais atribuídas a essa palavra

    Ex2: Dada a sequência de palavras 'o o as joão comeu o boolo bolo'

    - A representação sintáctica obtida corresponderá à análise da frase 'o joão comeu o bolo'

- Nesta 1ª fase de detecção de erros as palavras:

  o na posição **2** e **as** na posição **3** são consideradas **a mais**

  porque nem o determinativo categorizando **o** na posição **2**, nem o determinativo categorizando **as** na posição **3** pertencem à representação sintáctica da frase.

- Numa segunda fase de detecção de erros a palavra desconhecida **boolo** também será considerada **a mais** (ver ponto seguinte - **2. b)**)

2. Com a lista obtida em 1. e a lista dos arcos completos correspondentes a palavras desconhecidas detectam-se os erros referentes a palavras desconhecidas com categoria atribuída. Assim:

   a) Todas as categorias referentes a palavras desconhecidas pertencentes à lista obtida em 1 a). serão armazenadas na lista de erros sob a forma

      (U1,U2,desc:Cat) onde *U1* e *U2* são os vértices inicial e final da palavra e *Cat* é o pré-terminal atribuído à palavra desconhecida

   Nota: *U1* e *U2* serão obtidos por consulta da lista inicial dos arcos completos referentes a palavras desconhecidas

   No Ex1 ('o joão comu o bolo') é inserido na lista de erros um erro relativo a desconhecimento na posição **3** referenciando a existência de uma palavra desconhecida, **comu**, categorizada pela categoria pré-terminal **v** (verbo).

   Após a correcção de erros referentes a palavras desconhecidas com categoria atribuída, o utilizador será informado das possíveis correcções para **comu**.

   b) As palavras desconhecidas que não pertencem à lista obtida em 1 a) corresponderão a erros referentes a **palavras a mais** e serão inseridos na lista de erros sob a forma,

      (U1,U2,amais:pal_dsc(PalDesc)) onde *U1* e *U2* são os vértices inicial e final da palavra desconhecida e *PalDesc* denota a palavra desconhecida

   No Ex2. ('o o as joão comeu o boolo bolo') é inserido na lista de erros um erro relativo a **palavra a mais** na posição **7** referenciando a palavra desconhecida **boolo**.

No final, o utilizador será informado da **existência de três palavras a mais**:

- determinativo **o** na posição 2
- determinativo **as** na posição **3**
- palavra desconhecida **boolo** na posição **7**.

3. Todos os pré-terminais da lista obtida em 1 a) que não foram consumidos em 2. referir-se-ão a categorias atribuídas a palavras omissas que serão inseridos na lista de erros sob a forma,

   (U3,U3,omissa:Cat) onde *Cat* denota um pré-terminal atribuído a uma palavra omissa e *U3* denota o vértice da omissão.

   Nota: *U3* é fornecido pelo prefixo da palavra omissa denotada por *pal_om* (ver pag. 72 deste capítulo).

   Ex: Dada a sequência de palavras 'o morreu' obter-se-ão duas representações sintácticas para a categoria **frase** (uma referenciando a omissão de um **np** e outra referenciando a omissão de um **n**).

   O pré-terminal obtido na primeira representação sintáctica e não consumido em 1. e 2. é

   - ***np(pes=3,Num1,Gen1,2:pal_omi)***

   determinando a inserção na lista de erros correspondente de um erro referenciando a omissão de uma palavra na posição **2** categorizada pelo pré-terminal **np**.

   O pré-terminal obtido na segunda representação sintáctica e não consumido em 1. e 2. é

   - ***n(pes=3,Num1,Gen1,2:pal_omi)***

   determinando a inserção na lista de erros correspondente de um erro referenciando a omissão de uma palavra na posição **2** categorizada pelo pré-terminal **n**.

   Assim, após a detecção e correcção de erros o utilizador será informado da **inexistência de um nome próprio ou nome comum** na posição **2**.

4.5.2 Detecção e correcção de erros de concordância

Os erros de concordância que foram assinalados durante a análise (secção **4.2** deste capítulo) serão detectados e corrigidos com base nas representações sintácticas *mais plausíveis* do símbolo inicial.

Assim, dadas:

- as representações sintácticas *mais plausíveis* do símbolo inicial
- a lista dos arcos completos iniciais
- a lista dos arcos completos referentes a palavras desconhecidas

a detecção e correcção de erros de concordância (denotada por *concordância* na secção **5.1** do Apêndice III) opera, sobre cada representação sintáctica do símbolo inicial, de acordo com o algoritmo seguinte:

1. Armazena numa lista as representações sintácticas das categorias não terminais de **maior amplitude** onde tenham sido assinalados erros de concordância, ou seja, onde existam os átomos *err*, *esin* e *emasc* atribuídos a parâmetros relativos a concordância (ver secção **4.2** deste capítulo).

   Recorde-se que os átomos *esin* e *emasc* assinalam erros de concordância em número e género relativos aos particípios passados conjugados com o verbo **ter**.

   Ex: Em 'os maria morreu' a representação sintáctica obtida é

   > *[tipo=declarativa,sint(f):[conc:[pes=3,**num=err**,**gen=err**],sint(sn):[conc:[pes=3,**num=err**,**gen=err**],d(num=plu,gen=masc,cl=def,os),np(pes=3,num=sin,gen=fem,maria)],sint(sv):[conc:[pes=3,num=sin,gen=_],sint(nv):[conc:[pes=3,num=sin,gen=_],sint(verbo):[conc:[pes=3,num=sin,gen=_],v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,modo=indic,sc=0,morreu,morrer),sint(resto):[]]],sint(args):[sint(arg):[]]]]*

   - As categorias não terminais onde foram assinalados erros de concordância são as categorias **f** e **sn**.
   - A categoria não terminal de maior amplitude é a categoria **f** (pois contém a categoria não terminal **sn)**.

   Note-se que, sendo o erro assinalado a vários níveis sintácticos para permitir múltiplas hipóteses de correcção, a detecção e correcção dos mesmos deverá efectuar-se sempre ao nível mais alto de ocorrência do erro.

2. Para cada representação sintáctica obtida em 1. a correcção dos erros de concordância efectuar-se-á da forma seguinte:

   a) Identificam-se os parâmetros a que se refere o erro

      No ex. anterior os erros de concordância dizem respeito à concordância em **número** (***num=err***) e à concordância em **género** (***gen=err***).

   b) Seleccionam-se os pré-terminais que foram intervenientes na concordância

      Recorde-se que a representação sintáctica dos não terminais não intervenientes na concordância a um dado nível é da forma *sint(Cat):[Pré_Terminal1,..., Pré_Terminaln]* onde *Cat* denota a categoria não terminal (ver secção **4.2** deste capítulo)

      No ex. anterior os pré-terminais intervenientes na concordância são

      - os pré-terminais constituintes de **sn**

        d(num=plu,gen=masc,cl=def,os) e *np(pes=3,num=sin, gen=fem,maria)*

      - os pré-terminais constituintes de **nv**

        v(pes=3,num=sin,gen=_,tempo=ppref,forma=fin,modo=indic,sc=0,morreu,morrer)

   c) Por comparação com os arcos completos iniciais e com os arcos completos referentes a palavras desconhecidas são obtidos os vértices de cada pré-terminal e os correspondentes pré-terminais iniciais.

      Nota: É necessário obter os pré-terminais iniciais porque a instanciação de parâmetros na situação geral de concordância (necessária para a correcção posterior de palavras desconhecida com categoria atribuída) obrigou à instanciação de parâmetros que inicialmente eram não instanciados.

      Assim, na utilização da passiva o verbo auxiliar cujos parâmetros referentes ao género e ao número são inicialmente não instanciados, por aplicação das regras de concordância ao nível da categoria não terminal *verbo,* passam a ser instanciados com o valor obtido para o género e número do particípio passado (ver *regra geral de concordância* na secção **4.2** deste capítulo).

      Donde, existindo erros de concordância relativos ao género ou ao número ao nível da frase seria também proposta uma correcção para o verbo auxiliar.

Ex: Dada a sequência de palavras 'a maria foi tratado no hospital' onde existe um erro de concordância em género seria obtido para o pré-terminal **v** categorizando 'foi' o género **masculino**.

Assim, o utilizador seria informado que {ou 'a maria' deveria ser masculino ou 'foi tratado' deveria ser feminino}

d) Utilizando a lista dos pré-terminais iniciais intervenientes no erro de concordância assinalado (obtida em **c**) e cada parâmetro a que se refere o erro efectuar-se-á a sua correcção (ver **secção seguinte**).

e) Se houver vários parâmetros com erro as listas de erros obtidas são combinadas entre si para permitir a formulação de hipóteses de correcção de erros onde existam, simultaneamente, hipóteses de correcção relativas aos vários parâmetros.

No ex. anterior ('os maria morreu') a combinação das várias listas permitirá informar o utilizador[35] que:

- Ou 'maria' deveria ser masculino e plural e 'morreu' deveria ser plural
- Ou 'maria' e morreu' deveriam ser plural e a palavra 'os' deveria ser feminino
- Ou 'Os' deveria ser singular e 'maria' deveria ser masculino
- Ou 'Os' deveria ser feminino e singular

4.5.2.1 Correcção de um erro de concordância

Dada a lista dos pré-terminais iniciais intervenientes no erro assinalado e um dado parâmetro referenciando o respectivo erro, a correcção processa-se da forma seguinte:

1. Armazenam-se numa lista todos os diferentes valores referentes ao parâmetro do erro que existam nos pré-terminais iniciais

   No ex. anterior ('os maria morreu'),

   1. para a correcção da concordância em género obter-se-ia a lista *[gen = masc,gen = fem]*

[35]Note-se, mais uma vez, que no futuro estas saídas deveriam ser filtradas para não permitir, por ex., informar o utilizador que 'maria deveria ser masculino'

2. para a correcção da concordância em número obter-se-ia a lista *[num = sin,num = plu]*

Nota: Se o parâmetro referenciar um erro em género ou número que ocorreu na utilização de tempos compostos com o verbo **ter** (*gen=emasc* ou *num=esin*) são inseridos na lista os átomos:

- *gen=masc* se *gen=emasc*
- *num=sin* se *num=esin*

Recorde-se que estes erros só serão assinalados na categoria não terminal *resto* (ver secção **4.2** deste capítulo).

2. Compara-se cada átomo da lista obtida em 1. com o átomo existente para o mesmo parâmetro em cada pré-terminal inicial interveniente no erro assinalado.

   Se os átomos são diferentes é assinalado um erro sobre a forma *(U1,U2,Erro:Cat)* onde *U1* e *U2* denotam os vértices inicial e final do pré-terminal, *Erro* denota o tipo de erro de concordância, ou seja:

   - *conc_pes* se o erro se refere a concordância em pessoa
   - *conc_gen* se o erro se refere a concordância em número
   - *conc_num* se o erro se refere a concordância em género

   e *Cat* denota o pré-terminal onde o valor do parâmetro é substituído pelo valor proposto para correcção

- No ex. anterior ('os maria morreu')[36]

  1. Na correcção do erro de concordância em género obter-se-ia a lista constituída por

     *[(2,3,conc_**gen**:np(pes=3,num=sin,gen=**masc**,maria))]* correspondente à hipótese de considerar o masculino de 'maria'

     *[(1,2,conc_**gen**:d(pes=3,num=plu,gen=**fem**,cl=def,os))]* correspondente à hipótese de considerar o feminino de 'os'

[36]Note-se, mais uma vez, que no futuro estas saídas deveriam ser filtradas para não permitir, por ex., informar o utilizador que um nome próprio deveria ser masculino ou plural.

2. Na correcção do erro de concordância em número obter-se-ia a lista constituída por

   *[(2,3,conc_**num**:np(pes=3,num=**plu**,gen=fem,maria)),(3,4,conc_**num**:v(pes=3,num=**plu**,gen=_,tempo=ppref,forma=fin,modo=indic,sc=0,morreu,morrer)]*
   correspondente à hipótese de considerar o plural de 'maria' e de considerar 'morrer' na 3ª pessoa do plural do pretérito perfeito

   *[(1,2,conc_**num**:d(pes=3,num=**sin**,gen=masc,cl=def,os))]* correspondente à hipótese de considerar o singular de 'os'

4.5.3 Correcção ortográfica

Para cada erro existente numa lista de erros *mais plausível* (referenciado pela atribuição de uma categoria pré-terminal a uma palavra desconhecida[37]) tentar-se-á a sua correcção (denotada por *orto* na secção **5.2** do Apêndice III) que se processa de acordo com o algoritmo seguinte:

1. Por consulta ao léxico obtêm-se as palavras pertencentes à categoria pré-terminal que categoriza a palavra desconhecida.

2. Da lista de palavras obtidas em 1. consideram-se apenas aquelas que possuem no máximo mais um ou menos um carácter que a palavra desconhecida.

   Nota: A limitação do tamanho das palavras restringe o universo de palavras a ser tratadas uma vez que a correcção se limitará aos erros tipográficos e ortográficos referidos na secção **3.1.1** do Cap. **3**.

   Recorde-se que:

   - os erros tipográficos tratados são:

     1. a substituição de uma letra por outra
     2. a omissão de uma letra
     3. a existência de uma letra a mais
     4. a troca de letras consecutivas

[37]Como já foi referido não será tentada a correcção de uma palavra conhecida mas mal escrita face ao contexto sintáctico (situação tipificda por 'Eu acto as cordas').

- os erros resultantes da confusão de sons e deficiente acentuação correspondem em geral aos três primeiros tipos de erros acima referidos.

  Contudo, existem alguns erros originados por confusão de sons que correspondem a substituições de uma letra por duas letras ou vice-versa.

3. Utilizando a lista de palavras obtida em 2. e a palavra desconhecida são detectadas as hipóteses possíveis de correcção referentes aos erros tipográficos tipificados por Damerau [Dam 64] de acordo com o seguinte algoritmo:

   a) Usando o algoritmo da pag. 489 de Berghel [Ber 87], detectam-se as palavras que serão hipóteses de correcção para os erros tipográficos existentes na palavra desconhecida

   b) Para cada palavra considerada como hipótese de correcção em a) tenta detectar-se se o erro cometido na palavra desconhecida coincide com algum erro ortográfico (resultante de confusão sonora ou de deficiente acentuação) retornando um identificador do mesmo.

      Assim,

      b1) Cada palavra pertencente ao conjunto de correcção é comparada com a palavra desconhecida obtendo os caracteres correspondentes às diferenças encontradas.

      Ex1: Dada a palavra desconhecida 'u' e o conjunto de correcção representado pela lista [o,um] os caracteres correspondentes às diferenças encontradas serão, respectivamente:

      - Para a 1ª hipótese de correcção [u] e [o]
      - Para a 2ª hipótese de correcção [] e [m]

      Ex2: Dada a palavra desconhecida 'jãoo' e o conjunto de correcção representado pela lista [joão] os caracteres correspondentes às diferenças encontradas são [ão] e [oã]

      b2) Por consulta de tabelas que enumeram os caracteres que têm o mesmo som e as possibilidades de acentuação de cada vogal tenta detectar-se se a diferença de caracteres resulta de confusão sonora ou de deficiente acentuação retornando um identificador do erro que denotará:

      - *corr_fon* se existe um erro ortográfico coincidente com um erro tipográfico do tipo 1 a 3

- *corr_ac* se existir um erro resultante de incorrecta acentuação coincidente com um erro tipográfico

- *corr_tip* se não puder ser atribuída nenhuma das causas anteriores

No ex1., a palavra 'o' corresponde à correcção de um erro devido a confusão sonora.

Nota: Este identificador permitirá a ordenação, posterior, das hipóteses de correcção

4. São detectadas ainda todas as hipóteses de correcção referentes aos erros ortográficos que não coincidem com os erros tipográficos tratados em 3., ou seja, erros referentes à substituição de um carácter por dois ou vice-versa (ver secção **3.2.1** do Cap. I).

   A correcção destes erros processar-se-á de acordo com o algoritmo seguinte:

   a) Substitui-se cada carácter ou cada sequência de dois caracteres existentes na palavra desconhecida pela cadeia de caracteres ou carácter correspondente existente na tabela que enumera os caracteres (Fig. 4.5) passíveis de originar este tipo de erro

mesmo som: **x** e **ch**
mesmo som: **x** e **cc**
mesmo som: **c** e **ss**
mesmo som: **c** e **ç**.

Fig. 4.5 Confusões Sonoras

   b) Se a palavra obtida pertencer ao conjunto de palavras recolhidas no léxico é considerada como hipótese de correcção e é-lhe atribuído o identificador *corr_fon*.

      Ex: Dada a palavra desconhecida *axo* substituindo *x* por *ch* obtenho a palavra *acho* que existe no conjunto das palavras obtidas por consulta ao léxico

5. As hipóteses de erros são ordenadas colocando, primeiramente, os erros identificados por *corr_fon* e *corr_ac* e finalmente os erros identificados por *corr_tip*.

   Esta ordenação corresponderá ao estabelecimento de preferências relacionadas com correcções ortográficas ou relativas a acentuações face a correcções tipográficas.

6. É sempre acrescentada à lista de erros a hipótese de neologismo correspondendo à existência do identificador do erro denotado como *neol*

7. A palavra desconhecida e as suas hipóteses de correcção serão elementos de uma lista de erros onde, cada hipótese de correcção, existirá com a seguinte estrutura:

(U1,U2,desc(X):[Tipo_err:Cat,...]) onde *X* denota a palavra desconhecida, *Cat* denota a categoria pré-terminal correspondente a uma hipóteses de correcção e *Tipo_err* denota o identificador referido anteriormente

No ex1., a lista de correcções da palavra desconhecida 'u' será

*(1,2,desc(u):[**corr_fon**:d(num=sin,gen=masc,cl=_,**o**),**corr_tip**:d(num=sin,gen=masc,cl=_,**um**), **neol**:d(num = sin,gen = masc,cl = _,**u**)])*

que permitirá informar o utilizador que as hipóteses de correcção para a palavra 'u' serão:

- a palavra 'o'
- a palavra 'um'
- o neologismo 'u'

Nota: Face ao elevado número de comparações entre cadeias de caracteres efectuadas neste módulo foram utilizadas no programa Prolog correspondente, as codificações ASCII para os caracteres constituintes das palavras, das cadeias desconhecidas e das tabelas utilizadas.

5

Saídas produzidas e análise dos resultados obtidos

5.1 Saídas produzidas

Não foram desenhados écrans especiais para as saídas produzidas nem os erros foram objecto de formatação cuidada pois, o único objectivo na produção das saídas foi mostrar que, após serem efectuadas a detecção e correcção dos erros, existe uma lista de erros que permite informar o utilizador sobre a gramaticalidade ou a gramaticalidade das frases.

A lista de erros é formada por estruturas do tipo:

*[(**U1**,**U2**,**Tipo_erro**:Cat),...]* onde

U1 e ***U2*** denotam os vértices inicial e final do erro

Tipo_Erro denota os vários tipos de erros detectados, ou seja:

- *conc_num, conc_gen, conc_pes* - assinalam erros referentes a concordância, respectivamente, em número, género e pessoa
- *omissa* - assinala um erro referente a uma omissão
- *amais* - assinala um erro referente à existência de uma palavra a mais
- *desc(Cadeia)* - assinala um erro referente a desconhecimento sobre a palavra denotada por *Cadeia*

e *Cat* denota:

- uma categoria atribuída a um terminal contendo as hipóteses de correcção para erros de concordância

  Ex: *(2,3, **conc_num**:n(**num = plu**,gen = masc,bolo))* devendo ler-se: "a palavra 'bolo' na posição 2 deveria ser plural"

- uma categoria atribuída a uma palavra omissa ou a um terminal

  Ex1: *(2,2, **omissa**:n(num = sin,gen = masc,**pal_omi**))* devendo ler-se: "falta, na posição 2, um substantivo comum masculino e singular"

Ex2: *(1,2,**amais** : pal_dsc(u))* devendo ler-se: "a palavra 'u' na posição 1 está a mais"

Ex3: *(2,3,**amais**:np(num = plu,gen = fem,maria))* devendo ler-se: "a palavra 'maria' na posição 2 está a mais"

- uma lista de hipóteses de correcção para uma palavra desconhecida

  Ex: *(1,2, desc(u):[**corr_fon**:d(num=sin,gen=masc,cl=_,o), **corr_tip**:d(num= sin,gen=masc,cl=_,um), **neol**:d(num = sin,gen = masc,cl = _,u)])* devendo ler-se: "A palavra desconhecida 'u' poderá ser corrigida por 'o' (correcção fonética) ou 'um' (correcção tipográfica) ou deve ser acrescentada ao léxico"

poder-se-ão construir, no futuro, écrans de saída bastante mais legíveis necessitando apenas de blocos de texto normalizados para cada situação de erro descrita anteriormente.

5.2 Análise dos resultados obtidos

Este programa foi testado num PC com um processador 486 DX2 a 66 MHz e com 4 MB de RAM, sendo a sua execução desencadeada pelo comando **corrige** (módulo executável) em ambiente DOS (versão 6.2).

Os quadros das figuras desta secção referem-se a informações obtidas para cada frase existentes no ficheiro de teste, com base na gramática do Apêndice I e para um léxico contendo cerca de 100 palavras. Os títulos das colunas destes quadros têm o seguinte significado:

Omissa(s): refere-se às regras que tiveram de ser suprimidas da gramática por não existir espaço suficiente na RAM para o teste da frase em questão

TExp: refere-se ao número total de arcos expandidos

TA: refere-se ao número total de arcos activos existentes no grafo final

TC: refere-se ao número total de arcos completos existentes no grafo

THip: refere-se ao número total de hipótese formuladas para a obtenção de uma solução

T '': refere-se ao número tempo de execução medido em segundos

Cada frase existente no ficheiro de teste foi representada por *ex(N,Frase)* onde *N* denota um número atribuído à frase e *Frase* denota a lista de palavras.

Ex: No quadro da fig. 5.1, *ex(1,[os maria morreu]).*

Os resultados de cada frase onde existia pelo menos um erro foi, quando possível, comparada com os resultados obtidos para uma das hipóteses de correcção da mesma (representada por ex(_,*Frase*)).

Ex: No quadro da fig. 5.3, em *ex(54,[o,joão,casou,cm,quem])* o utilizador é informado que a palavra desconhecida 'cm' terá como correcções 'com' e 'em' ou poderá ser considerado um neologismo na categoria das preposições. A frase correcta apresentada, correspondendo à primeira hipótese de correcção, é representada em *ex(_,[o,joão,casou,com,quem])*.

Note-se que, os resultados obtidos, dependem menos do número de palavras existentes na sequência de palavras do que do número de regras existentes na gramática e aplicáveis à expansão de uma dada categoria num vértice (ver exs. 63 e 64 do quadro da Fig. 5.4).

Nos diferentes quadros, nas sequências de palavras começadas por 'a' (de acordo com o léxico utilizado 'a' pode ser um artigo definido ou uma preposição) é pesquisada a existência de um *sp* topicalizado donde, os totais na frase correcta são muito superiores aos totais obtidos com a situação de erro e, por vezes, determinam a omissão de regras para poderem ser testadas (ver exs. 1 e 5 do quadro da Fig. 5.1).

5.2.1 Situações relativas a concordâncias

No quadro da Fig. 5.1, referente à detecção de erros de concordância pode observar-se que:

- A detecção de erros de concordância corresponde a variações nos tempos de execução da ordem das centésimas.
- O total de expandidos é, em geral, igual na versão correcta e na versão com erros.
- O total de hipóteses formuladas será 0 pois, o analisador não falha por existirem erros de concordância.

Os exemplos 12 e 13 expressam duas situações de concordância referente a tempos compostos conjugados com o verbo 'ter':

- No ex. 12 o utilizador é apenas informado que 'comidas' deveria ser masculino e singular
- No ex. 13, o utilizador é informado que {'comidas' deverá ser masculino singular e 'tinham' deverá singular ou 'comidas' deverá ser masculino singular e 'o joão' deverá ser plural}

Concorddâncias	Omissa(s):	TExp	TA	TC	THIP	T "
ex(_,[a,maria,morreu]).	r4	40	73	31	0	.88
ex(1,[os,maria,morreu]).		27	39	20	0	.5
ex(2,[o,maria,morreu]).		27	39	20	0	.5
ex(_,[a,maria,escreveu,o,livro]).	r4;r8	31	48	38	0	.61
ex(3,[a,maria,escreveu,a,livro]).	r8	36	61	40	0	.77
ex(_,[dois,programas,terminam]).		29	43	22	0	.49
ex(4,[dois,programas,termina]).		29	43	22	0	.49
ex(_,[a,maria,casou,com,o,joão]).	r4;r8	30	46	37	0	.6
ex(5,[o,maria,casou,com,a,joão]).		38	58	38	0	.83
ex(_,[o,joão,tinha,dado,o,livro,a,a,maria]).	r4;r8;r28;r35	39	68	40	0	.88
ex(6,[o,joão,tinha,dados,o,livro,a,a,maria]).	r4;r8;r28;r35	39	68	40	0	.88
ex(_,[o,livro,foi,dado,por,o,joão,a,a,maria]).	r4;r8;r35	41	71	50	0	1.04
ex(7,[o,livro,foi,dados,por,o,joão,a,a,maria]).	r4;r8;r35	41	71	50	0	1.04
ex(_,[a,ana,tinha,comido,a,maçã]).	r4;r8	46	70	57	0	1.16
ex(8,[a,ana,tinha,comida,a,maçã]).	r4;r8	46	70	57	0	1.16
ex(_,[a,maçã,foi,comida,por,o,joão]).	r4;r8	37	57	50	0	.88
ex(9,[a,maçã,foi,comido,por,o,joão]).	r4;r8	37	57	50	0	.88
ex(_,[a,maçã,tinha,sido,comida,por,o,joão]).	r4;r8;r28	37	59	48	0	.93
ex(10,[a,maçã,tinha,sido,comido,por,o,joão]).	r4;r8;r28	34	56	44	0	.82
ex(_,[o,joão,tinha,comido,a,maçã]).	r4;r8	46	70	56	0	1.15
ex(11,[o,joão,tinha,comida,a,maçã]).	r4;r8	46	70	56	0	1.16
ex(_,[o,joão,tinha,comido,as,maçãs]).	r4;r8	46	70	55	0	1.16
ex(12,[o,joão,tinha,comidas,as,maçãs]).	r4;r8	46	70	55	0	1.16
ex(13,[o,joão,tinham,comidas,as,maçãs]).	r4;r8	46	70	55	0	1.16
ex(_,[quem,morreu]).	r8	43	58	40	0	.88
ex(14,[quem,morreram]).	r8	43	58	40	0	.88
ex(15,[noz,escrevemos,este,livro]).	r4	36	56	38	0	.71

Fig. 5.1 Resultados referentes a erros de concordância

No ex. 15, foi testada uma situação de homófona 'nós' e 'noz' e, porque a ausência do determinativo é permitida pelas regras gramaticais, a frase é considerada correcta mas é detectado um erro de concordância em pessoa e em número ('noz' deveria ser primeira pessoa do plural ou 'escrevemos' deveria ser terceira pessoa do singular)[38].

5.2.2 Situações relativas a omissão

Os exemplos 21 22, 24, 25 e 29[39] (quadro da Fig. 5.2) terminam a pesquisa no último vértice e informam o utilizador que a frase está incompleta e quais as categorias possíveis para a continuação da pesquisa.

Omissões	Omissa(s):	TExp	TA	TC	THIP	T "
ex(20,[o,ardeu]).		30	49	24	2	.88
ex(21,[o,joão]).		17	29	6	0	.6
ex(22,[com,o,winograd,o,joão]).		32	61	14	0	.72
ex(23,[joão,deu]).	r4	35	57	38	0	.83
ex(24,[o,joão,deu,um,livro]).	r35	36	59	19	0	.66
ex(25,[o,joão,deu,a,a,maria]).	r35	37	69	21	0	.77
ex(26,[a,maria,casou,o,joão]).	r4;r8	31	46	37	1	.99
ex(27,[o,programa,termina,ardeu])./v. 1		29	43	23	0	.71
ex(27,[o,programa,termina,ardeu])./v. 2	r8;r22	42	61	36	1	1.42
ex(28,[o,programa,o,joao,termina,ardeu]).	r21	28	40	25	1	.65
ex(29,[morreu]).		23	40	9	0	.49

Fig. 5.2 Resultados obtidos referentes a erros de omissão

No ex. 20, a situação de erro é detectada na análise de 'ardeu' e o utilizador é informado da existência de uma frase onde é omissa uma palavra referente a um nome próprio ou comum.

[38]Estas hipóteses de correcção, tal como já foi referido anteriormente, deveriam no futuro ser filtradas pois um nome comum corresponde sempre à **terceira pessoa**.

[39]Nos exemplos 24 e 25 o utilizador é informado que a frase está incompleta porque a regra 35 (permitindo a inexistência de um complemento) foi suprimida.

O ex. 23 termina correctamente porque a gramática permite a inexistência de complementos em verbos (regras r35 e r39).

No ex. 26, a situação de erro é detectada na análise de 'o' e o utilizador será informado da existência de uma frase onde é omissa uma preposição (a subcategorização do verbo 'casar' determina a inexistência de complementos ou a existência de um sintagma preposicional[40]).

Neste exemplo, o utilizador não é informado que {o,joão} podem ser consideradas palavras a mais (correspondendo à possibilidade da inexistência de complementos) porque a informação ao utilizador refere apenas as soluções com menor número de erros.

O ex. 27 ([o,programa,termina,ardeu]), na sua versão 1, completa a frase em 'termina' e considera 'ardeu' como palavra a mais pois a situação de erro só é detectada depois de constituir um sv com 'termina'.

Na versão 2 do mesmo ex., porque foi eliminada a hipótese da oração relativa ser vazia, a situação de erro é detectada na análise de 'termina' correspondendo à omissão de um pronome relativo.

O ex. 28, sendo na gramática o verbo 'terminar' subcategorizado com tipo 0, correspondendo à inexistência de complementos, são consideradas palavras a mais {o,joão,ardeu}.

5.2.3 Situações relativas a categorias atribuídas a palavras desconhecidas

Note-se que, em geral, as incorrecções detectadas nas frases dos diferentes exemplos do quadro da figura 5.3 corresponderam a um acréscimo significativo no número de arcos activos (face às análises das frases consideradas correctas) o que determinou a necessidade de omissão de maior número de regras.

Nesse quadro e na coluna referente às regras omissas:

> **(1)** denota a omissão da regra referente à existência de determinativo vazio (regra d(_,_,_,d([])) ---> []. referida no **Apêndice II**).
>
> **(2)** denota a omissão dos factos referentes à subcategorização dos verbos (predicado *v_sc* referida no **Apêndice I**) com excepção da subcategorização 'esperada' para a palavra errada.
>
> Ex: Em *ex(54,[mrreu,o,joão]).* são omissos todas as subcategorizações com excepção do facto referente à subcategorização do verbo 'morrer' (*v_sc(sc=0,[]).* no **Apêndice I**).

[40]Note-se que a subcategorização de **casar** deveria também permitir a existência de um sintagma nominal

Palavras Desconhecidas	Omissa(s):	TExp	TA	TC	THIP	T "
ex(_,[o,joão,morreu]).		27	39	20	0	.44
ex(40,[u,joão,morreu]).	r8;(1)	35	50	25	2	.66
ex(_,[o,programa,termina]).		29	43	22	0	.49
ex(41,[o,progrma,termina]).		30	49	24	2	.93
ex(_,[o,joão,deu,o,shrdlu,a,a,maria]).	r35	43	71	36	0	.93
ex(42,[o,joão,deu,o,shrdlu,a,a,mria]).	r4;r8;r35;r44	37	68	46	2	1.76
ex(43,[o,joão,du,o,shrdlu,a,a,maria]).	r4;r8;r35;r44	41	75	48	1	1.11
ex(_,[morreu,o,joão]).		37	58	24	0	1.1
ex(44,[mrreu,o,joão]).	r4;r10;r35;(2)	38	66	31	5	1.37
ex(_,[com,a,maria,o,joão,casou]).	r4;r35;r44	35	62	32	0	.88
ex(45,[cm,a,maria,o,joão,casou]).	r4;r9;r13;r35;r44;(1)	33	50	32	3	1.21
ex(_,[quem,morreu]).	r8	43	58	40	0	.83
ex(46,[qem,morreu]).	r8;(1)	25	34	19	1	.55
ex(_,[o,programa,que,termina,ardeu]).	r4;r8	39	58	42	0	.77
ex(47,[o,programa,qe,termina,ardeu]).	r4;r8;r22	35	52	36	1	1.16
ex(_,[que,comeu,a,maria]).	r8;r44;r35;(1)	47	68	48	0	1.15
ex(48,[qe,comeu,a,maria]).	r8;r44;r35;(1)	32	51	26	3	.82
ex(_,[o,programa,que,terminou,ardeu]).	r4;r8	39	58	42	0	.77
ex(49,[o,programa,que,o,jao,terminou,ardeu]).	r4;r8;r21	20	28	26	1	.55
ex(_,[a,maria,tem,o,livro]).	r8	36	63	39	0	.77
ex(50,[a,maria,tem,o,livr]).	r4;r8	31	54	43	2	1.32
ex(_,[o,livro,foi,dado,por,o,joão,a,a,maria]).	r4;r8;r35	41	71	50	0	1.04
ex(51,[o,livro,fi,dado,por,o,joão,a,a,maria]).	r4;r8;r28;r29,r35;r44	45	82	42	3	1.21
ex(52,[o,livro,fi,ddo,por,o,joão,a,a,maria]).	r4;r8;r28;29	43	79	41	3	1.65
	r30;r35;r44;(2)					
ex(_,[a,maçã,foi,comida,por,o,joão]).	r8	42	70	51	0	1.04
ex(53,[a,massã,foi,comido,por,o,joão]).	r4;r8;r35;r44	36	62	44	3	1.65
ex(_,[o,joão,casou,com,quem]).		42	60	50	0	.99
ex(54,[o,joão,casou,cm,quem]).		43	60	49	1	1.59

Fig. 5.3 Resultados obtidos referentes a palavras desconhecidas

O teste do ex. 49 não é significativo porque teve que ser omitida a regra correspondente à hipótese de existência de oração relativa sendo o utilizador informado que são palavras a mais {que,o,joão,ardeu}.

Todos os outros resultados obtidos para cada frase onde existia pelo menos um erro corresponderam, em geral, ao esperado.

5.2.4 Situações relativas a palavras a mais

No ex. 64 da fig. 5.4, existirão duas soluções: uma onde ′'escrevu' é neologismo (advérbio de negação) e outra onde 'escrevu' é palavra a mais.

Palavras a Mais	Omissa(s):	TExp	TA	TC	THIP	T "
ex(_,[o,joão,morreu]).		27	39	20	0	.44
ex(61,[o,joão,joão,morreu]).		27	39	21	1	.65
ex(62,[o,joão,maria,joão,morreu]).		27	39	22	1	.65
ex(_,[o,joão,escreveu,um,livro]).		39	60	37	0	.82
ex(63,[o,joão,escreveu,escrevu,um,livro]).	r4;r8;r12;r21;r44	36	53	54	3	1.76
ex(64,[o,joão,escrevu,escreveu,um,livro]).	r4;r8;r35;r44	38	71	38	2	1.16
ex(_,[o,joão,termina]).		27	39	20	0	.44
ex(65,[o,joão,termina,termina,um,livro]).		27	39	23	0	.66
ex(66,[o,o,joão,termina,um,livro]).		35	50	27	2	.88

Fig. 5.4 Resultados obtidos referentes a palavras a mais

Conclusões

Neste capítulo é apresentado um resumo sobre os tipos de frases e erros tratados, analisam-se situações não contempladas e descrevem-se alguns problemas encontrados bem como algumas linhas de trabalho possível para o aperfeiçoamento de um sistema deste tipo.

Resenha dos tipos de frases e erros tratados

Foram tratados nesta tese alguns erros morfo-sintácticos ou devidos a desconhecimento de palavras existentes em frases declarativas e interrogativas.

No que respeita a palavras desconhecidas foram tratadas dois tipos de situações:

- Palavra mal escrita

  Nesta categoria foram apenas tratados os quatro tipos de erros tipográficos que Damerau [Dam 64] considera representarem 80% dos erros cometidos em inglês:

  1. Substituição de uma letra por outra
  2. Omissão de uma letra
  3. Existência de uma letra a mais
  4. Troca de letras consecutivas.

  Foram também tratados os erros resultantes da confusão de sons que correspondem, em geral, às situações do tipo 1 a 3 acima referidas e as excepções que correspondem a substituições de uma letra por duas letras ou vice-versa (por ex: ó<u>cc</u>ido).

- Neologismos em sentido lato (não existência da palavra no léxico)

No que respeita a palavras existentes no léxico foram tratadas as situações seguintes:

- Palavras a mais por duplicação ou não duplicadas

- Questões de concordância em pessoa, género e número com excepção das questões de concordância relativas a construções predicativas no que diz respeito ao predicativo do sujeito.

- Palavras com categoria ou forma diferente da requerida

No que respeita a omissões foram tratadas duas situações:

- Omissões que não condicionam a existência de uma frase e onde podem detectar-se todos os pré-terminais que categorizam as palavras ausentes.

- Omissões conduzindo à situação de frase incompletas

Sobre o sistema em geral

- A gramática é restrita a um conjunto limitado de regras. No entanto, pode ser aumentada embora se deva ter consciência que a detecção e correcção de erros incidirá sempre sobre gramáticas incompletas, pela incapacidade de definição exaustiva de uma gramática.

- O léxico também é muito reduzido e, tal como a gramática, pode ser aumentado embora seja impossível obter um léxico completo de uma língua.

  A não completude do léxico foi tratada neste sistema através da inclusão da possibilidade de se estar perante uma palavra bem escrita mas que o sistema desconhece (**neologismo** foi a designação utilizada).

- A grande desvantagem de um sistema deste tipo é o espaço e o tempo ocupado em pesquisas com base em hipóteses formuladas que poderão redundar em fracasso.

  Assim, para limitar o número de hipóteses formuladas foram utilizadas diversas estratégias:

  1. um erro devido à existência de uma palavra a mais é preferido relativamente a um erro devido a uma omissão

  2. rejeitam-se todas as hipóteses formuladas por arcos activos cuja primeira categoria em resto seja um não terminal ou um pré-terminal que não incorpore um arco completo inicial ou uma palavra desconhecida.

Note-se que esta estratégia só será eficaz se não existirem dois erros consecutivos com excepção dos erros ocasionados pela existência de palavras desconhecidas.

Priviligiou-se a ocorrência de erros consecutivos ocasionados por palavras desconhecidas por se pensar que são os erros mais frequentes na escrita de documentos utilizando processadores de texto.

Todavia, seriam necessários estudos mais aprofundados para permitir uma decisão mais fundamentada.

As estratégias referidas anteriormente e a opção de pesquisa em largura (resultante do tipo de ordenação feita pelo analisador) poderiam ser no futuro melhoradas se houvesse uma ordenação de arcos de acordo com preferências existentes na língua.

Esta informação que poderia ser fornecida por pesos atribuídos às regras (Vosse [Vos 92]) deveria resultar da constatação dos erros mais frequentes ou das estruturas sintácticas mais utilizadas pelos utentes da língua.

- Foi tentada a utilização de um analisador ascendente (uma das estratégias utilizada no Critique/Epistle) para tentar prosseguir a análise numa situação de falha.

  Esta tentativa resultou pouco eficaz pela necessidade de coexistência de regras gramaticais (para tentar impedir a proliferação de estruturas não utilizadas).

- As principais vantagens de um sistema deste tipo são:

  1. A vinculação de uma palavra desconhecida a uma categoria sintáctica permitindo uma pesquisa de léxico mais condicionada (o que é importante pois diminui o o espaço e o tempo de pesquisa no léxico) embora a pesquisa no léxico também possa ser bastante optimizada.

  2. A utilização de restrições nas regras que, tal como é proposto em Vosse [Vos 92], permitem a não falha do analisador face a erros de concordância.

- No sistema de Vosse [Vos 92] que também é baseado em técnicas de relaxamento é utilizada uma outra abordagem, ou seja, inicialmente é atribuída a cada palavra desconhecida um conjunto de correcção que será usado para superar as falhas do analisador.

  Esta abordagem permitirá reduzir o número de análises a efectuar mas, em contrapartida, obrigará a uma pesquisa no léxico não condicionada.

Deixa-se, portanto, em aberto a construção de um sistema deste tipo que permitiria obter resultados comparativos relativamente às questões de espaço e tempo.

- Foi assumido que, existindo palavras desconhecidas, deve tentar-se a sua incorporação como constituintes da frase, preterindo o recurso à utilização de palavras omissas.

  Esta opção que restringe o número de hipóteses formuladas foi adoptada por se pensar que a existência de uma palavra desconhecida num documento deve ser entendida, em geral, como uma tentativa não conseguida de escrita de um vocábulo.

  A desvantagem desta suposição é a tentativa de correcção de uma palavra desconhecida quando o sistema deveria informar o utilizador da existência de uma omissão.

  Assim, por ex. em 'o *çkj* comeu um bolo' o sistema informará o utilizador que a cadeia '*çkj*'[41] poderá ser um neologismo nas categorias sintácticas: nome comum ou nome próprio.

- Foi também assumido que se considerariam como mais plausíveis as soluções com um menor número de erros. Esta opção que parece bastante razoável conduz por vezes a resultados não muito óbvios.

  Por ex., em frases do tipo 'a maria tinha comprar livros'[42] o utilizador será informado que 'comprar' é uma palavra a mais pois a outra solução corresponderia a considerar a inexistência de um verbo no particípio passado e a existência de uma palavra a mais: 'comprar'.

- As situações referentes a palavras com categoria ou forma diferente da requerida são formuladas como uma omissão da categoria ou forma requerida e a existência de uma palavra a mais (correspondente à categoria ou forma apresentada).

  Esta opção origina, em casos de palavras mal escritas mas existentes no léxico, como por ex. em 'eu acto as cordas', a não proposta de correcção para acto.

  No futuro, poder-se-ia superar esta falha do sistema considerando que a existência de uma palavra a mais e uma omissão no mesmo vértice implicam a tentativa de correcção da palavra a mais na categoria pré-terminal da palavra omissa.

  Note-se que em Vosse [Voss 92] este tipo de situação é contemplado pois é proposto, à partida, um conjunto de alternativas/correcções para cada palavra da frase.

- Neste sistema não é contemplada análise semântica donde:

[41] Palavra tipicamente mal escrita

[42] Note-se que a incorrecção da forma verbal não foi tratada no âmbito desta tese.

1. frases do tipo 'o livro quem o João deu à Maria termina' são consideradas correctas pois a restrição dos pronomes relativos deveria ser feita a nível semântico.

2. As homófonas, tal como em Vosse [Vos 92], só serão detectadas quando existam em categorias sintácticas diferentes ou quando violem as restrições sintácticas consideradas (por ex., nós e noz).

- As questões de concordância contempladas dizem respeito apenas a género, número e pessoa, devendo no futuro ser alargadas para permitir tratar, entre outras, questões de tempo e modo.

  Note-se que também não foram tratados nesta tese as questões de concordância referentes a construções predicativas onde o sujeito não só concorda com o verbo predicativo mas também com o predicativo do sujeito.

  Não foram tratados, por ex., os erros referentes ao predicativo do sujeito nas frases 'os livros são linda' e 'O João continua parvas'

- Também não foram tratadas nesta tese erros devidos à aglutinação de palavras por omissão do espaço que deveria existir entre elas.

  Por ex: Em '**ojoão** morreu' o utilizador seria informado da omissão de um nome próprio ou nome comum (esta gramática permite a ausência de determinativo).

  Este tipo de erros[43] poderia ser tratado com uma abordagem semelhante à utilizada por Vosse [Vos 92] para a detecção de erros em palavras compostas[44].

- O sistema de Vosse ([Vos 92]) trata ainda erros devidos a duplicação de palavras e erros existentes em frases idiomáticas mas, por outro lado, não trata erros devidos à existência de uma ou várias palavras a mais (não duplicadas) nem os erros devidos a omissões.

- Comparando o sistema construído com o sistema Epistle (Heidorn et al. [HJMB 82]) e o sistema Critique (Richardson e Braden-Harder [RB 88])[45] verifica-se que o sistema construído, ainda que incompleto, consegue realizar grande parte das correcções propostas para os sistema referidos anteriormente (sistemas propondo-se, basicamente, realizar correcções para erros de concordância).

  Nestes sistemas as palavras desconhecidas são corrigidas à partida e, caso não seja possível a sua correcção, são categorizadas como nomes comuns.

[43] Muito frequente na escrita de documentos utilizando processadores de texto
[44] Em holandês, as palavras compostas podem ser escritas numa única palavra ou em duas palavras distintas
[45] Sistemas que também utilizam aproximações baseadas em relaxamento

Esta abordagem, tal como a de Vosse [Vos 92], permitirá reduzir o número de análises a efectuar mas, em contrapartida, obrigará a uma pesquisa no léxico não condicionada. Por outro lado, a atribuição da categoria **nome comum** a palavras desconhecidas obrigará à falha do analisador no caso da palavra desconhecida ser, por exemplo, um **verbo**.

Note-se ainda que, tal como no sistema construído nesta tese, as palavras existentes no léxico não serão corrigidas porque, as correcções propostas à partida, inicidirão sobre palavras desconhecidas.

Mas, por outro lado, em caso de falha do analisador são activadas regras gramaticais para substituição de palavras que se confundem (por ex., whose e who´s) o que permite uma resolução parcial dos erros relativos a palavras com categoria ou forma diferente da requerida.

- As mensagens de erro nas situações de concordância deveriam ser filtradas para não permitir considerar hipóteses como, por ex., {o masculino de Maria} ou {noz na 1ª pessoa}.

- Também não foi contemplada na construção deste sistema a situação de gramática poder estar incompleta. Essa situação obrigaria à admissão de regras novas.

Sobre a correcção de palavras desconhecidas

- A correcção de erros em palavras desconhecidas é muito rudimentar e baseada apenas nos erros descritos em Damerau [Dam 64] e nas confusões fonéticas.

- Por outro lado, o corrector proposto (construído em Prolog) seria muito pouco eficiente para um léxico de cento e vinte mil formas base (Lopes et al. [LMR 94]).

- Note-se que a construção de um sistema mais robusto, constitui, pela sua complexidade, uma área de pesquisa autónoma.

- A modularidade do sistema construído permite que, no futuro, possa ser incorporado um corrector ortográfico também baseado no contexto mas mais robusto.

Apêndice I

Gramática

```
r1:s(tipo=Tipo,sint(s):[tipo=Tipo,SF]) ---> [f(P,N,G,SF),vars(List),{struct(List,Tipo)}].
r2:vars([]) ---> [].
r3:vars([H|T]) ---> [varq(H),vars(T)].

r5:f(P,N,G,sint(f):[conc:[P,N,G],SN,VP])  ---> [ilha_slashquest sn(P1,N1,G1,caso=nom,_,SN),sv(P2,N2,G2,VP),
                                                {concorda(P1,P2,P), concorda(N1,N2,N), concorda(G1,G2,G)}].

r4:f(P,N,G,sint(f):[F])       ---> [perg,f(P,N,G,F)].
r6:perg quest sp(SP) ---> [int_sp(SP)].
r7:perg quest sn(P,N,G,C,cl=interrog,sn(SN)) ---> [ilha sn(P,N,G,C,cl=Classse,sn(SN)), {Classe == interrog}].

r8:f(P,N,G,sint(f):[SF])       ---> [comp, f(P,N,G,SF)].
r9:comp slash nv(P,N,G,Scat,Voz,NV) ---> [ilha nv(P,N,G,Scat,Voz,NV)].
r10:comp slash sp(SP) ---> [ilha sp(SP)].

r12:sn(P,N,G,C,Classe,sint(sn):[conc:[P,N,G],X]) ---> [ilha pron(P,N,G,Classe,C,X)].
r13:sn(P1,N,G,_,Classe,sint(sn):[conc:[P1,N,G],X,Y]) ---> [ilha d(N1,G1,Classe,X), ilha resto_sn(P1,N2,G2,Y),
                                                          {concorda(N1,N2,N), concorda(G1,G2,G)}].
r14:sn(P,N,G,_,Classe,sint(sn):[conc:[P,N,G],X,Y]) ---> [ilha d(N1,G1,Classe,X), ilha np(P,N2,G2,Y),
                                                        {concorda(N1,N2,N), concorda(G1,G2,G)}].

r15:resto_sn(P,N,G,sint(resto_sn):[conc:[P,N,G],Y,Orel] ---> [ilha n(P,N,G,Y), ilha orel(Orel)].

r21:orel(sint(orel):[conc:[P1,N,G1],X,F]) ---> [varv(P1,N1,G1,_,X), f(_,N2,_,F),concorda(N1,N2,N)}].
r22:orel(sint(orel):[]) ---> [].
r23:varv(P,N,G,C,X) rel sn(P,N,G,C,cl=rel,sint(sn):[conc:[P,N,G],VAR]) ---> [ilha pron(P,N,G,cl=rel,C,VAR)].
r24:varv(P,N,G,C,X) rel sp(sint(sp):[Prep,sint(sn):[conc:[P,N,G],VAR]]) --->
                                                          [ilha prep(Prep),ilha pron(P,N,G,cl=rel,C,VAR)].

r25:sv(P,N,G,sint(sv):[conc:[P,N,G],NV,ARGS]) ---> [ilha_relquest nv(P,N,G,Scat,Voz,NV),args(Voz,Scat,ARGS)].

r26:nv(P,N,G,SCat,Voz,sint(nv):[conc:[P,N,G],Neg,V]) ---> [ilha neg(Neg),ilha verbo(P,N,G,T,F,M,SCat,Voz,V)].
```

```
r27:verbo(P,N,G,T,F,M,SCat,Voz,sint(verbo):[conc:[P,N,G],NV,Resto) --->
                                                [ilha v(P1,N1,G1,T1,F,M,S,Vf,NV),
                                                 ilha resto(P2,N2,G2,conj=NV,S,SCat,Voz,Resto),
                                                 {tv(NV,Resto,T1,T,M),concorda(P1,P2,P),concorda(N1,N2,N),
                                                 concorda(G1,G2,G)}].

r28:resto(pes=_,num=_,gen=_,conj=V,S,S,voz=activa,sint(resto):[]) ---> [].
r29:resto(P,N1,G1,conj=v(_,_,_,_,_,_,_,_,ter),S,Scat,voz=activa,sint(resto):[conc:[P,N1,G1],NV]) --->
                                                [v(P,N,G,tempo=preterito,forma=participio,M,Scat,V,NV),
                                                 {concorda(voz=activa,N,N1),concorda(voz=activa,G,G1)}].
r30:resto(P,N,G,conj=v(_,_,_,_,_,_,_,_,ter),S,Scat,voz=passiva,sint(resto):[conc:[P,N,G],NV,NV0]) --->
                                                [v(P1,N1,G1,tempo=preterito,forma=participio,M,_,V,NV),
                                                 {functor(NV,_,Ari),arg(Ari,NV,ser)},
                                                 v(P2,N2,G2,tempo=preterito,forma=participio,M,Scat,V0,NV0),
                                                 {concorda(P1,P2,P),concorda(N1,N2,N),concorda(G1,G2,G)}].
r31:resto(P,N,G,conj=v(_,_,_,_,_,_,_,_,ser),S,Scat,voz=passiva,sint(resto):[conc:[P,N,G],NV]) --->
                                                [v(P,N,G,tempo=preterito,forma=participio,M,Scat,V,NV)].

r32:sp(sint(sp):[P,It]) ---> [ilha prep(P), ilha sn(_,_,_,caso=p,_,It)].
r33:int_sp(sint(sp):[P,It]) ---> [ilha prep(P),ilha sn(_,_,_,_,cl=Classe,It), {Classe == interrog}].

r34:args(Voz,Scat,sint(args):[ARGS]) ---> [{v_sc(Scat,S),transform_v_sc(Voz,S,S1),select(S1,T)},arg(T,ARGS)].
r35:arg(X,sint(arg):[]) ---> [{X\=[],X\=and([])}].
r37:arg(and([H|T]),sint(arg):[X|Y]) ---> [arg(H,sint(arg):[X]),arg(and(T),sint(arg):Y)].
r38:arg(and([H,H1|R]),sint(arg):[X|Y]) ---> [arg(H1,sint(arg):[X]), arg(and([H|R]),sint(arg):Y)].
r39:arg(and([H|T]),sint(arg):[[]|Y]) ---> [arg(H,sint(arg):[]), arg(and(T),sint(arg):Y)].
r44:arg([],sint(arg):[]) ---> [].
r40:arg(sn,sint(arg):[X]) ---> [sn(_,_,_,_,_,X)].
r42:arg(sp(P,sn),sint(arg):[sint(sp):[P,It]]) ---> [sp(sint(sp):[P,It])].
r45:arg(and([]),sint(arg):[[]]) ---> [].
```

```
tv(V,sint(resto):[],T,T,_):-!.
select(and(X),and(X)).
select([],[]).
select(or([]),[]).

tv(V,V1,T,T,_):-!.
select(or([H|_]),H).
select(or([_|T]),Y) :- select(or(T),Y).

v_sc(sc=0,[]).
v_sc(sc=1,or([sn])).
v_sc(sc=2,and([sn,sp(prep(_),sn)])).
v_sc(sc=3,and([sp(prep(por),sn),sp(_,sn)])).
v_sc(sc=4,or([sp(prep(_),sn)])).

transform_v_sc(voz=activa,S,S).
transform_v_sc(voz=passiva,or([]),_):-fail.
transform_v_sc(voz=passiva,or([sn|T]),or([sp(prep(por),sn)|T])).
transform_v_sc(voz=passiva,or([sn|T]),or([sp(prep(por),sn)|T])).
transform_v_sc(voz=passiva,or([H|T]),or([H|T1])):-transform_v_sc(voz=passiva,or(T),or(T1)).
transform_v_sc(voz=passiva,and([]),_):-fail.
transform_v_sc(voz=passiva,and([sn|T]),and([sp(prep(por),sn)|T])).
transform_v_sc(voz=passiva,and([sn|T]),and([sp(prep(por),sn)|T])).
transform_v_sc(voz=passiva,and([H|T]),and([H|T1])):-transform_v_sc(voz=passiva,and(T),and(T1)).

concorda(Param=X,gen=Gen,gen=emasc):-X==activa,Gen == fem,!.
concorda(Param=X,num=Num,num=esin):-X==activa,Num == plu,!.
concorda(Param=X,Param1=Y,Param1=_):- (X==activa;Y==emasc),!.
concorda(Param=X,Param=Y,Param=X):- Y=esin,!.
concorda(Param=X,Param=Y,Param=err):-X\=Y,!.
concorda(Param=X,Param=X,Param=X):-X \== err,!.
concorda(Param=X,Param=Y,Param=err):-equiv(X,Y).

struct([],declarativa).
struct(L,interrogativa):- L \= [],!.

simbolo_inicial(s(_,_)).

nao_terminal(s(_,_)).
nao_terminal(f(_,_,_,_)).
nao_terminal(sn(_,_,_,_,_,_)).
nao_terminal(resto_sn(_,_,_,_)).
nao_terminal(orel(_)).
nao_terminal(varv(_,_,_,_,_)).
nao_terminal(comp).
nao_terminal(perg).
nao_terminal(vars(_)).
nao_terminal(varq(_)).
nao_terminal(sv(_,_,_,_)).
nao_terminal(nv(_,_,_,_,_,_)).
nao_terminal(verbo(_,_,_,_,_,_,_,_,_,_)).
nao_terminal(resto(_,_,_,_,_,_,_,_)).
nao_terminal(args(_,_,_)).
nao_terminal(arg(_,_)).
nao_terminal(sp(_)).
nao_terminal(int_sp(_)).
```

```
nao_terminal(ter(_,_,_,_,_,_,_,_)).

pre_terminal(neg(_)).
pre_terminal(pron(_,_,_,_,_,_)).
pre_terminal(d(_,_,_,_)).
pre_terminal(n(_,_,_,_)).

nao_terminal(ser(_,_,_,_,_)).

pre_terminal(np(_,_,_,_)).
pre_terminal(prep(_)).
pre_terminal(v(_,_,_,_,_,_,_,_,_)).
```

Apêndice II

Léxico

Consulta

```
neg(neg(X)) ---> [[X],{dic_neg(X)}].

neg([]) ---> [].

pron(pes=P,num=N,gen=G,cl=Classe,caso=C,pron(pes=P,num=N,gen=G,cl=Classe,caso=C,X)) --->
                                                    [[X],{dic_pron(X,P,N,G,Classe,C),Classe\=interrog}].

pron(pes=P,num=N,gen=G,cl=interrog,caso=C,pron(pes=P,num=N,gen=G,cl=interrog,caso=C,X)) vrs varq(D) ---> [[X],
                                                    {dic_pron(X,P,N,G,interrog,C),functor(F,X,1),arg(1,F,D)}].

d(num=N,gen=G,cl=Classe,d(num=N,gen=G,cl=Classe,X)) ---> [[X],{dic_det(X,N,G,Classe),Classe\=interrog}].

d(num=N,gen=G,cl=interrog,d(num=N,gen=G,cl=interrog,X)) vrs varq(D) ---> [[X],
                                                    {dic_det(X,N,G,interrog),functor(F,X,1),arg(1,F,D)}].

d(_,_,_,d([])) ---> [].

n(pes=3,num=N,gen=G,n(pes=3,num=N,gen=G,X)) ---> [[X], {dic_n(X,N,G)}].

np(pes=3,num=N,gen=G,np(pes=3,num=N,gen=G,X)) ---> [[X],{dic_np(X,N,G)}].

prep(prep(P))---> [[P], {dic_prep(P)}].

v(pes=P,num=N,gen=G,tempo=T,forma=F,modo=M,sc=Scat,V,
  v(pes=P,num=N,gen=G,tempo=T,forma=F,modo=M,sc=Scat,V,NV)) ---> [[V], {dic_v(NV,V,P,N,G,T,F,M,Scat)}].
```

Parte do Léxico

```
dic_neg(não).

dic_pron(que,_,_,_,rel,_).

dic_pron(que,3,_,_,interrog,_).

dic_pron(eu,1,sin,_,pessoal,nom).
dic_pron(tu,2,sin,_,pessoal,nom).
dic_pron(ele,3,sin,masc,pessoal,nom).
dic_pron(ela,3,sin,fem,pessoal,nom).

dic_np(ana,sin,fem).
dic_np(joão,sin,masc).

dic_pron(quem,_,sin,_,rel,_).

dic_pron(quem,3,sin,_,interrog,_).

dic_pron(eles,3,plu,masc,pessoal,nom).
dic_pron(elas,3,plu,fem,pessoal,nom).
dic_pron(nós,1,plu,fem,pessoal,nom).
...

dic_np(maria,sin,fem).
dic_np(shrdlu,sin,masc).
```

```
dic_n(barro,sin,masc).
dic_n(livro,sin,masc).
dic_n(maçã,sin,fem).
dic_n(maçãs,plu,fem).
dic_n(noz,sin,fem).

dic_det(a,sin,fem,def).
dic_det(as,plu,fem,def).
dic_det(o,sin,masc,def).
dic_det(os,plu,masc,def).
dic_det(dois,plu,masc,ord).

dic_v(arder,ardeu,3,sin,_,ppref,fin,indic,0).
dic_v(casar,casou,3,sin,_,ppref,fin,indic,4).
dic_v(casar,casado,_,sin,masc,preterito,participio,indic,4).
dic_v(comer,comeu,3,sin,_,ppref,fin,indic,1).
dic_v(comer,comido,_,sin,masc,preterito,participio,indic,1).
dic_v(comer,comida,_,sin,fem,preterito,participio,indic,1).
dic_v(comer,comidas,_,plu,fem,preterito,participio,indic,1).
dic_v(dar,dado,_,sin,masc,preterito,participio,indic,2).
dic_v(dar,dada,_,sin,fem,pprefc,fin,indic,2).
dic_v(dar,dados,_,plu,masc,preterito,participio,indic,2).
dic_v(dar,dei,1,sin,_,ppref,fin,indic,2).
dic_v(dar,deu,3,sin,_,ppref,fin,indic,2).
dic_v(escrever,escreveu,3,sin,_,ppref,fin,indic,1).
...

dic_prep(a).
dic_prep(de).
dic_prep(com).

dic_n(óxido,sin,masc).
dic_n(peixe,sin,masc).
dic_n(pessoa,sin,fem).
dic_n(programa,sin,masc).
dic_n(programas,plu,masc).

dic_det(este,sin,masc,dem).
dic_det(nenhum,sin,masc,negindef).
dic_det(um,sin,masc,indef).
dic_det(umas,plu,fem,indef).

dic_v(escrever,escrevemos,1,plu,_,ppref,fin,indic,1).
dic_v(ser,é,3,sin,_,pres,fin,indic,5).
dic_v(ser,foi,3,sin,_,ppref,fin,indic,5).
dic_v(ser,foram,3,plu,_,ppref,fin,indic,5).
dic_v(ser,sido,_,_,_,preterito,participio,indic,5).
dic_v(ter,tem,3,sin,_,pres,fin,indic,1).
dic_v(ter,tido,_,sin,masc,preterito,participio,indic,1).
dic_v(ter,tinha,3,sin,_,pimp,fin,indic,1).
dic_v(terminar,termina,3,sin,_,pres,fin,indic,0).
dic_v(terminar,termino,1,sin,_,pres,fin,indic,0).
dic_v(terminar,terminam,3,plu,_,pres,fin,indic,0).
dic_v(terminar,terminado,_,sin,fem,preterito,participio,indic
,0).

dic_prep(em).
dic_prep(por).
```

Apêndice III

Programa

%%%Nomes de variáveis

%Arcos: A, A1,...

A_Nec, A_Descs, A_AMais, A_Dups,...

%Arcos Pendentes: Ps,P1s,NPs,...

%Grafo (Activos/Completos): Grafo, NGrafo, Grafo1, As/Cs, A1s/Cs,...

%Categorias já propostas: PRs, NPRs,...

%Lista de palavras: Frase, Pals, NPals, ...

%Categorias: Cat, NCat, Cat1, ...

%Lista dos arcos completos referentes a palavras desconhecidas: PalDescs

%Lista das hipóteses correspondentes a erros de omissão: PalOms

%Lista de erros e suas correcções: Erros, WErros, Erros_Conc, etc.

%Listas dos arcos completos correspondentes ao símbolo inicial: Finais, WFinais, ...

%Lista das categorias pré-terminais obtidas .à custa das palavras iniciais: ICats, WICats, ...

%Totais obtidos: THip, TExp, TAs, TCs, TTempo,...

%Vértices: U, U1, Ui, Uf,...

%Número da frase do ficheiro de teste: Numero

%Nota: Um 's' no fim do nome duma variável significa um conjunto ou lista. No inicio do nome de uma variável 'N'

%significa *novo,* 'W' significa *auxiliar*, 'L' significa *lista* e 'T' significa *total.*

%%% Operadores necessários à definição da gramática

```
:-op(999,xfx,--->).
:-op(990,xfx,binds).
:-op(300,xfx,rel).
:-op(300,xfx,slash).
:-op(300,xfx,quest).
:-op(300,xfx,vrs).
:-op(300,fx,ilha).
:-op(300,fx,ilha_rel).
:-op(300,fx,ilha_slash).
:-op(300,fx,ilha_slashquest).
:-op(300,fx,ilha_relquest).
```

1. Predicados gestores do sistema

%%%main/0 - Predicado principal que desencadeia o sistema de detecção e correcção de erros

%Frase - Lista de palavras

%Numero - Numero da frase no ficheiro de teste

% As/Cs - Grafo obtido

%TAs, TCs, TExp, THip - Totais de arcos activos, completos expandidos e hipotéticos criados em situações de falha do

%analisador

%Tempo_e, Tempo_s, TTempo - Tempo inicial, final e tempo de execução

%%

```
main:- cls,
        consult(gramatica),
        nl,
        write('Frase'),
        read(Numero),
        write(Numero),write(' ---  '),
        open(H,'frases.txt',r),
        ler(H,Numero,_,Frase),
        write(Frase),nl,
        time(Tempo_e),
        corrector(Frase,TExp,As/Cs,THip),!,
        time(Tempo_s),
        conta_tempo(Tempo_e,Tempo_s,TTempo),
        nl,nl,write('%%%%%%Estatiticas%%%%%%%%%%'),nl,
        write('Arcos exp. ---> '),write(TExp),nl,
        length(As,TAs),
        write('TActivos   ---> '),write(TAs),nl,
        length(Cs,TCs),
        write('TCompletos ---> '),write(TCs),nl,
        write('T Hip.     ---> '),write(THip),nl,
        write('Tempo      ---> '),write(TTempo),write(' segs').
```

%%%corrector/4 - Predicado principal do sistema de detecção e correcção de erros

%Frase - Lista de palavras

%TExp - Total de expandidos

%NGrafo - Grafo obtido

%THip - Total de arcos hipotéticos criados em situações de falha do analisador

%%

```
corrector(Frase,TExp,NGrafo,THip):-
        inicializa(Frase,[ICats|Vazios],Ps,PRs,Uf,PalDescs),
        concatena(ICats,Vazios,ICs),
```

```
        analisa_por_grafo(Ps,PRs,PRs1,[]/ICs,Grafo,Uf,U,1,WTExp),
        form_hip(U,[ICats|Vazios],PRs1,NPRs,Grafo,NGrafo,Uf,PalDescs,U1,WTExp,TExp,0,THip),!,
        alerta(ICats,NGrafo,U1,PalDescs).
```

2. Inicializações

%%%inicializa/6 - Inicialização das estruturas necessárias à análise por grafo e à detecção e correcção de erros

%Frase - Lista de palavras

%ICats - Lista dos completos iniciais

%Vazios - Lista dos completos derivados das categorias em vazio

%Ps - Lista dos pendentes

%Uf - Vértice Final

%PalDescs - Lista dos arcos completos correspondentes a palavras desconhecidas

%%

```
inicializa(Frase,[ICats|Vazios],Ps,[Cat/1],Uf,PalDescs):- criar_completos(Frase,ICats,PalDescs,1,Uf),
                                                          arcos_palavras_vazias(Vazios),
                                                          call(simbolo_inicial(Cat)),
                                                          propor(Cat,1,[]-[],[]-[],[]-[],[]-[],[],Ps).
```

%%criar_completos/5 - Retorna uma lista constituída pelos arcos completos correspondentes a palavras existentes no **%%**léxico, a lista constituída pelos arcos completos correspondentes a palavras desconhecidas e o vértice final (Uf)

```
criar_completos([Pal|Pals],Cs,PalDescs,U1,Uf) :- !, arcos_da_palavra(Pal,U1,U2,WCs,WPalDescs),
                                                  concatena(WCs,NCs,Cs),
                                                  concatena(WPalDescs,NPalDescs,PalDescs),
                                                  criar_completos(Pals,NCs,NPalDescs,U2,Uf).
criar_completos([],[],[],Uf,Uf).
```

%%arcos_da_palavra/5 - Retorna a lista constituída pelos arcos completos correspondendo às categorias pré-terminais **%%**que podem ser atribuídas a uma dada palavra impondo, caso necessário, restrições aos canais afectados e a lista dos **%%**arcos completos correspondentes a palavras desconhecidas

```
arcos_da_palavra(Pal,U1,U2,Cs,PalDescs):-
        U2 is U1 + 1,
        findall(ac(U1,U2,pal:Cat,Q0-Q,S0-S,R0-R,V0-V),
                (saca_pterminal(Cat,Rh--->[[Pal],{Restrs}]), call(Restrs),
                 restringe_canais(Rh,Q0-Q0,S0-S0,R0-R0,V0-V0, Q0-Q ,S0-S ,R0-R ,V0-V)), Cs1),
        arcos_descs(Pal,U1,U2,Cs1,Cs,PalDescs).
```

%%arcos_descs/6 - Retorna uma lista constituída por um arco completo correspondente a uma palavra desconhecida

```
arcos_descs(Pal,U1,U2,[],[],[ac(U1,U2,desc:pal_dsc(Pal),Q0-Q0,R0-R0,S0-S0,V0-V0)]):- !.
arcos_descs(Pal,U1,U2,Cs,Cs,[]).
```

%%arcos_palavras_vazias/1 - Cria os arcos completos correspondentes a categorias que podem derivar em vazio

%%arcos_palavras_vazias(Cs):- findall(ac(U,U,lambda:Rh,Q-Q,S-S,R-R,V-V), (saca_pterminal(Cat,Rh--->[])), Cs).

%%saca_pterminal/2 - Retorna as regras cuja cabeça é Cat

```
saca_pterminal(Cat,Cat rel Cat1 ---> Corpo):-     call(Cat rel Cat1 ---> Corpo),
                                                  call(pre_terminal(Cat)).
saca_pterminal(Cat,Cat slash Cat1 ---> Corpo):- call(Cat slash Cat1 ---> Corpo),
                                                  call(pre_terminal(Cat)).
saca_pterminal(Cat,Cat quest Cat1 ---> Corpo):- call(Cat quest Cat1 ---> Corpo),
                                                  call(pre_terminal(Cat)).
saca_pterminal(Cat,Cat vrs Cat1 ---> Corpo):-    call(Cat vrs Cat1 ---> Corpo),
                                                  call(pre_terminal(Cat)).
saca_pterminal(Cat,Cat ---> Corpo):-              call(Cat  ---> Corpo),
                                                  call(pre_terminal(Cat)).
```

3. Analisador por grafo

%%%analisa_por_grafo/9 - Predicado principal do analisador por grafo

%A -Arco

%Ps - Arcos pendentes

%PRs/NPRs - Lista das categorias já propostas

%As/Cs ou Grafo/NGrafo - Grafo

%Uf - Vértice final

%U - Vértice onde terminou a análise por grafo

%TExp/NTExp - Total de expandidos

%%

```
analisa_por_grafo([],PRs,PRs,As/Cs,As/Cs,Uf,U,TExp,TExp):-  maior_vert_f(As,U).
analisa_por_grafo([A|Ps],PRs,NPRs,Grafo,NGrafo,Uf,U,WTExp,NTExp):-
        combinar(A,Grafo,Ps,P1s),
        ad_grafo(A,Grafo,Grafo1),
        expandir_se(A,P1s,P2s,PRs,PRs1,WTExp,TExp),!,
        analisa_por_grafo(P2s,PRs1,NPRs,Grafo1,NGrafo,Uf,U,TExp,NTExp).
```

%%combinar/4 - Combina um arco com os arcos existentes no grafo produzindo novos pendentes

```
combinar(ac(Ui,Uf,Cat,Q,S,R,V),As/Cs,Ps,NPs) :- !,combinar_completo(As,ac(Ui,Uf,Cat,Q,S,R,V),Ps,NPs).
combinar(A,As/Cs,Ps,NPs) :- combinar_activo(Cs,A,Ps,NPs).
```

%%combinar_completo/4 - Combina um arco completo com todos os arcos activos existentes no grafo

```
combinar_completo([Aa|Aas],Ac,Ps,NPs) :- incorporar(Aa,Ac,NA),!,
                                         ad_ord(NA,Ps,P1s),
                                         combinar_completo(Aas,Ac,P1s,NPs).
```

```
combinar_completo([_|Aas],Ac,Ps,NPs) :- !, combinar_completo(Aas,Ac,Ps,NPs).
combinar_completo([],Ac,Ps,Ps).
```

%%combinar_activo/4 - Combina um arco activo com todos os arcos completos existentes no grafo

```
combinar_activo([Ac|Acs],Aa,Ps,NPs) :- incorporar(Aa,Ac,NA),!,
                                        ad_ord(NA,Ps,P1s),
                                        combinar_activo(Acs,Aa,P1s,NPs).
combinar_activo([_|Acs],Aa,Ps,NPs) :- !, combinar_activo(Acs,Aa,Ps,NPs).
combinar_activo([],Aa,Ps,Ps).
```

%%ad_grafo/3 - Adiciona um arco ao grafo retornando o novo grafo

```
ad_grafo(aa(U1,U2,Cat,Cat1,Cats,Q,S,R,V),As/Cs,A1s/Cs) :-!,ad(aa(U1,U2,Cat,Cat1,Cats,Q,S,R,V),As,A1s).
ad_grafo(Ac,As/Cs,As/C1s) :-!, ad(Ac,Cs,C1s).
```

%%expandir_se/7 - Expande um arco se este for activo, se Cat1 for não terminal ou restrição e se Cat1/Uf/Q/S/R/V não **%%**tiver sido proposto, incrementando o contador de expandidos

```
expandir_se(aa(_,Uf,_,Cat1,_,Q,S,R,V),Ps,P1s,PRs,[Cat1/Uf/Q/S/R/V|PRs],WTExp,TExp) :-
        (e_nt(Cat1);restricao(Cat1)),
        not membro(Cat1/Uf/Q/S/R/V,PRs),!,
        TExp is WTExp + 1,
        propor(Cat1,Uf,Q,S,R,V,Ps,P1s).
expandir_se(_,Ps,Ps,PRs,PRs,TExp,TExp).
```

%%e_nt/1 - Verifica se uma categoria, possivelmente modificada por uma ilha é não terminal

```
e_nt(Cat):- tira_ilha(Cat,NCat),!,
             call(nao_terminal(NCat)).
```

%%e_t/1 - Verifica se uma categoria, possivelmente modificada por uma ilha é pré-terminal

```
e_t(Cat):- tira_ilha(Cat,NCat),!,
            call(pre_terminal(NCat)).
```

%%restricao/1 - Verifica se uma categoria, possivelmente modificada por uma ilha é restrição

```
restricao({Restricao}):-!.
```

%%propor/8 - Propõe uma categoria ou restrição no vértice U produzindo novos pendentes

```
propor({Restrs},U,Q0-Q,S0-S,R0-R,V0-V,Ps,NPs) :-!, findall(ac(U,U,restr:{Restrs},Q-Q,S-S,R-R,V-V), call(Restrs), Ac),
                                                    conc_ord(Ac,Ps,NPs).
propor(X,U,Q0-Q,S0-S,R0-R,V0-V,Ps,NPs) :-
        tira_ilha(X,Cat),!,
        findall(Ac,consumos(Cat,U,Q,S,R,V,Ac),Pa),
```

```
        findall(aa(U,U,Rh,Cat1,Cats,Q-Q,S-S,R-R,V-V),saca_regra(Cat,Rh--->[Cat1|Cats]),P1),
        findall(ac(U,U,Rh,Q-Q,S-S,R-R,V-V),saca_regra(Cat,Rh--->[]),P2),
        conc_ord(Pa,P1,Pa1),
        conc_ord(Pa1,Ps,Pa1s),
        conc_ord(P2,Pa1s,NPs).
```

%%consumos/7 - Consome Cat no vértice U existente no canal Q, S, R ou V.

```
consumos(Cat,U,[Cat|Q],S,R,V,ac(U,U,quest:Cat,[Cat|Q]-Q,S-S,R-R,V-V)).
consumos(Cat,U,Q,[Cat|S],R,V,ac(U,U,slash:Cat,Q-Q,[Cat|S]-S,R-R,V-V)).
consumos(Cat,U,Q,S,[Cat|R],V,ac(U,U,rel:Cat,Q-Q,S-S,[Cat|R]-R,V-V)).
consumos(Cat,U,Q,S,R,[Cat|V],ac(U,U,vrs:Cat,Q-Q,S-S,R-R,[Cat|V]-V)).
```

%%saca_regra/2 - Retorna uma regra que tenha à cabeça Cat

```
saca_regra(Cat,Rg:Cat rel Cat1 ---> Corpo):- call(Rg:Cat rel Cat1 ---> Corpo).
saca_regra(Cat,Rg:Cat slash Cat1 ---> Corpo):- call(Rg:Cat slash Cat1 ---> Corpo).
saca_regra(Cat,Rg:Cat quest Cat1 ---> Corpo):- call(Rg:Cat quest Cat1 ---> Corpo).
saca_regra(Cat,Rg:Cat vrs Cat1 ---> Corpo):- call(Rg:Cat vrs Cat1 ---> Corpo).
saca_regra(Cat,Rg:Cat ---> Corpo):- call(Rg:Cat ---> Corpo).
```

%%tira_ilha/2 - Separa o operador *ilha* da categoria gramatical

```
tira_ilha(ilha Cat,Cat1):-!, Cat1=Cat.
tira_ilha(ilha_slash Cat,Cat1):-!, Cat1=Cat.
tira_ilha(ilha_rel Cat,Cat1):-!, Cat1=Cat.
tira_ilha(ilha_slashquest Cat,Cat1):-!,Cat1=Cat.
tira_ilha(ilha_relquest Cat,Cat1):-!, Cat1=Cat.
tira_ilha(Cat,Cat).
```

%%tira_canal/3 - Separa o operador *canal* da categoria gramatical

```
tira_canal(Cat rel Cat1,NCat,NCat1):-!, NCat=Cat, NCat1=Cat1.
tira_canal(Cat slash Cat1,NCat,NCat1):-!, NCat=Cat, NCat1=Cat1.
tira_canal(Cat quest Cat1,NCat,NCat1):-!, NCat=Cat, NCat1=Cat1.
tira_canal(Cat vrs Cat1,NCat,NCat1):-!, NCat=Cat, NCat1=Cat1.
tira_canal(Cat,Cat,[]).
```

%%restringe_ilhas/5 - Verifica se os canais estão em conformidade com o tipo de categoria em questão

```
restringe_ilhas(ilha Cat, Q-Q, S-S, R-R, V0-V).
restringe_ilhas(ilha_rel Cat, Q0-Q, S0-S, R-R, V0-V).
restringe_ilhas(ilha_slash Cat, Q0-Q, S-S, R0-R, V0-V).
restringe_ilhas(ilha_slashquest Cat, Q-Q, S-S, R0-R, V0-V).
restringe_ilhas(ilha_relquest Cat,Q-Q, S0-S, R-R, V0-V).
```

```
restringe_ilhas(Cat, Q0-Q, S0-S, R0-R, V0-V) :- tira_ilha(Cat,Cat).
```

%%incorporar/3 - Incorpora um arco completo num arco activo

```
incorporar(aa(Ui,U1,NCat,Cat1,Cats, Q0-Q1, S0-S1, R0-R1, V0-V1),
            ac(U2,Uf,Rg2:Cat2,  Q2-Q3, S2-S3, R2-R3, V2-V3),NA) :-
        podem_incorporar(Cat1,U1, Q0-Q1, S0-S1, R0-R1, V0-V1,Cat2,U2,Q2-Q3, S2-S3, R2-R3, V2-V3),!,
        (R1=R2,Q1=Q2, S1=S2, V1=V2, U1=U2,tira_ilha(Cat1,WCat),tira_canal(Cat2,WCat,_),
         completo_ou_activo(Cats,Ui,Uf,NCat, Q0-Q3, S0-S3, R0-R3, V0-V3,NA)) binds NA.
```

%%podem_incorporar/12 - Verifica se um arco completo pode ser incorporado num activo

```
podem_incorporar(Cat1,U1, Q0-Q1, S0-S1, R0-R1, V0-V1, Cat2,U2, Q2-Q3, S2-S3, R2-R3, V2-V3):-
        tira_ilha(Cat1,NCat1),
        tira_canal(Cat2,NCat2,_),
        equiv(NCat1,NCat2),
        equiv(U1,U2),
        equiv(Q1,Q2),
        equiv(S1,S2),
        equiv(R1,R2),
        equiv(V1,V2),
        not not restringe_ilhas(Cat1, Q2-Q3, S2-S3, R2-R3, V2-V3).
```

%%equiv/2 - Verifica se X é igual a Y impedindo a instanciação das variáveis

```
equiv(X,Y) :- not not X = Y,!.
```

%%completo_ou_activo/9 - Produz o arco resultante da incorporação

```
completo_ou_activo([Cat1|Cats],U1,U2,Rg:Cat,Q0-Q,S0-S,R0-R,V0-V,
                        aa(U1,U2,Rg:Cat,Cat1,Cats,Q0-Q,S0-S,R0-R,V0-V)) :-!.
completo_ou_activo([],U1,U2,Rg:Cat,Q0-Q1,S0-S1,R0-R1,V0-V1,ac(U1,U2,Rg:Cat,Q0-Q,S0-S,R0-R,V0-V)) :-
        restringe_canais(Cat, Q0-Q1, S0-S1, R0-R1, V0-V1,Q0-Q,  S0-S,  R0-R,  V0-V),!.
completo_ou_activo([],U1,U2,Rg:Cat,Q0-Q,S0-S,R0-R,V0-V,ac(U1,U2,Rg:NCat,Q0-Q, S0-S, R0-R, V0-V)) :-
        tira_canal(Cat,NCat,_).
```

%%restringe_canais/9 - Restringe os canais de acordo com a categoria em questão

```
restringe_canais(Cat, Q0-Q, S0-S, R0-R, V0-V, Q0-Q, S0-S, R0-R, V0-V):-tira_canal(Cat,Cat,[]).
restringe_canais(Cat rel Cat1, Q0-Q, S0-S, R0-R0, V0-V, Q0-Q, S0-S, R0-[Cat1|R0], V0-V).
restringe_canais(Cat slash Cat1, Q0-Q, S0-S0, R0-R, V0-V, Q0-Q, S0-[Cat1|S0], R0-R, V0-V).
restringe_canais(Cat quest Cat1, Q0-Q0, S0-S, R0-R, V0-V, Q0-[Cat1|Q0], S0-S, R0-R, V0-V).
restringe_canais(Cat vrs Cat1, Q0-Q, S0-S, R0-R, V0-V0, Q0-Q, S0-S, R0-R, V0-[Cat1|V0]).
```

%%binds/2 - Obriga a que, na incorporação de um arco B num arco A só sejam instanciados os argumentos (vértices,
%%categorias e canais de movimentação) do arco NA, resultante da incorporação

```
G binds T :- call(G),
        assert(binded(T)),
        fail.
G binds T :- retract(binded(T)).
```

4. Formulador de Hipóteses

%%%form_hip/13 - Predicado principal da formulação de hipóteses que, face ao grafo obtido e ao vértice U onde
%%terminou a análise por grafo, tenta formular hipóteses para que a análise por grafo prossiga

%U/NU - Vértice onde terminou a análise por grafo

%ICats - Lista dos arcos completos iniciais

%PRs/NPRs - Lista das categorias já propostas

%As/Cs, Grafo - grafo

%Uf - Vértice final

%PalDescs - Lista dos arcos completos correspondentes a palavras desconhecidas

%TExp/NTexp - Total de Expandidos

%THip/NTHip - Total de arcos hipotéticos

%%

```
form_hip(Uf,_,PRs,PRs,Grafo,Grafo,Uf,_,Uf,TExp,TExp,THip,THip):- !.
form_hip(U,[ICats|Vazios],PRs,NPRs,As/Cs,Grafo,Uf,PalDescs,NU,WTExp,NTExp,THip,NTHip):-
        precisa_cats(As,Vazios,U,A_Nec),
        trata_anec(WA_Nec,[ICats|Vazios],As/Cs,U,A_Nec,WU),
        separa_pals([ICats|Vazios],PalDescs,WU,WICats,WPalDescs),
        hipo_cats(A_Nec,WPalDescs,A_Descs),
        existem_cats(WICats,WPalDescs,A_Nec,A_Descs,Ps,TExp1),
        Ps \= [],
        length(Ps,NPs),
        WTExp1 is WTExp + TExp1,
        analisa_por_grafo(Ps,PRs,PRs1,As/Cs,NGrafo,Uf,U1,WTExp1,TExp),
        WU \= U1,!,
        WTHip is THip +NPs - TExp1,
        form_hip(U1,[ICats|Vazios],PRs1,NPRs,NGrafo,Grafo,Uf,PalDescs,NU,TExp,NTExp,WTHip,NTHip).
form_hip(U,_,PRs,PRs,Grafo,Grafo,Uf,PalDescs,U,TExp,TExp,THip,THip):- !.
```

%%trata_anecs/6 - Dada a lista de arcos activos necessários, a lista das categorias pré-terminais iniciais e as **%%**deriváveis em vazio e o vértice onde terminou a análise por grafo (U) constrói a lista dos arcos correspondentes à falha **%%**(A_Nec) e retorna o vértice efectivo de falha

```
trata_anec([],[ICats|Vazios],As/Cs,U,A_Nec,NU):- encontra_acanais(As,Cs,U,NU,As_Cs),!,
                                                  precisa_ocats(As,Vazios,NU,As_Cs,A_Nec).
trata_anec(WA_Nec,[ICats|Vazios],As/Cs,U,A_Nec,WU):-encontra_completos(U,As/Cs,Vazios,WA_Nec,A_Nec,WU).
```

%%encontra_acanais/5 - Dada a lista dos arcos necessários, a lista dos arcos completos e o vértice onde terminou a **%%**pesquisa retorna o vértice efectivo de falha e as categorias não terminais já existentes na lista dos arcos completos. **%%**Com este predicado pretende detectar-se se existe informação a mais nos canais (situações correspondendo a orações **%%**relativas/interrogativas)

```
encontra_acanais([aa(U1,U,Cat,Restrs,[],Q-Q0,S-S0,R-R0,V-V0)|As],Cs,U,U2,[NF]):-
        consumos(WCat,U1,Q0,S0,R0,V0,ac(U2,U2,NCat,Q1,S1,R1,V1)),
        not not membro(ac(U2,U2,NCat,Q1,S1,R1,V1),Cs),!,
        functor(WCat,NF,_).
encontra_acanais([A1|As],Cs,U,U2,As_Cs):-!,encontra_acanais(As,Cs,U,U2,As_Cs).
```

%%precisa_cats/4 - Dada a lista de arcos activos e o vértice onde terminou a análise por grafo (U) constrói a lista dos **%%**arcos correspondentes à falha (A_Nec)

```
precisa_cats([],_,_,[]):-!.
precisa_cats([aa(U1,U,Cat,Cat1,Cats,Q,S,R,V)|As],Vazios,U,[aa(U1,U,Cat,Cat1,Cats,Q,S,R,V)|A_Nec]):-
        tira_ilha(Cat1,NCat1),
        not membro(ac(_,_,_:NCat1,_,_,_,_),Vazios),
        call(pre_terminal(NCat1),!,
        precisa_cats(As,Vazios,U,A_Nec).
precisa_cats([A1|As],Vazios,U,A_Necs):-!, precisa_cats(As,Vazios,U,A_Necs).
```

%%encontra_completos/6 - Detecta se existem arcos completos relativos às categorias correspondentes à falha. Nessa **%%**situação produz novos arcos com categorias necessárias correspondentes ao vértice final dos completos encontrados

```
encontra_completos(U,As/Cs,Vazios,A_Nec,NA_Nec,NU):-
        completos(A_Nec,Cs,NU,As_Cs),
        As_Cs \= [],!,
        precisa_ocats(As,Vazios,NU,As_Cs,A_Nec1),
        concatena(A_Nec,A_Nec1,NA_Nec).
encontra_completos(U,As/Cs,Vazios,A_Nec,A_Nec,U):-!.
```

%%completos/4 - Detecta a existência de arcos completos correspondentes a uma categoria necessária retornando o **%%**vértice final dos arcos completos detectados bem como as categorias não terminais que lhe correspondem

```
completos([],_,U2,[]):-!.
```

```
completos([aa(U1,U,RH:Cat,_,_,Q,S,R,V)|As],Cs,U2,[NCat|As_Cs]):-
        functor(Cat,NCat,Ari),
        functor(WCat,NCat,Ari),
        membro(ac(U1,U2,RH1:WCat,_,_,_,_),Cs),
        completos(As,Cs,U,As_Cs),
        not membro(NCat,As_Cs).
completos([aa(U1,U,Cat,_,_,Q,S,R,V)|As],Cs,U,As_Cs):-!,completos(As,Cs,U,As_Cs).
```

%%precisa_ocats/5 - Produz novos arcos activos necessários (A_Nec1s) face a um vértice (U) e à lista dos arcos que
%%correspondiam à falha (A_Necs)

```
precisa_ocats([],_,_,_,[]):-!.
precisa_ocats([aa(U,U,RH:Cat,Cat1,Cats,Q,S,R,V)|As],Vazios,U,As_Cs,[aa(U,U,RH:Cat,Cat1,Cats,Q,S,R,V)|A_Nec1s]):-
        tira_ilha(Cat1,NCat1),
        not membro(ac(_,_,_:NCat1,_,_,_,_),Vazios),
        call(pre_terminal(NCat1)),
        functor(Cat,NCat,Ari),
        not membro(NCat,As_Cs),!,
        precisa_ocats(As,Vazios,U,As_Cs,A_Nec1s).
precisa_ocats([A1|As],Vazios,U,As_Cs,A_Nec1s):-!,precisa_ocats(As,Vazios,U,As_Cs,A_Nec1s).
```

%%separa_pals/5 - Produz uma lista contendo a lista dos arcos completos das categorias derivadas em vazio e a lista dos
%%arcos completos das categorias iniciais cujo vértice inicial seja maior ou igual a U produzindo ainda a lista dos arcos
%%completos correspondentes a palavras desconhecidas cujo vértice inicial seja superior ou igual a U.

```
separa_pals([ICats|Vazios],PalDescs,U,[Vazios|[WICats]],WPalDescs):-!,
        separa_pals(ICats,U,WICats),
        separa_pals(PalDescs,U,WPalDescs).
```

%%separa_pals/3 - Dada uma lista de arcos completos e um vértice U retorna a lista dos arcos completos cujo vértice
%%inicial é superior ou igual a U

```
separa_pals([],_,[]):-!.
separa_pals([ac(U1,U2,Cat,Q,S,R,V)|ICats],U,[ac(U1,U2,Cat,Q,S,R,V)|NICats]) :-
        U1 >= U,!,
        separa_pals(ICats,U,NICats).
separa_pals([IC|ICats],U,NICats):-!,separa_pals(ICats,U,NICats).
```

%%hipo_cats/3 - Constrói uma lista com os arcos resultantes da incorporação de arcos completos correspondentes a
%%palavras desconhecidas nos arcos activos correspondentes à falha

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

```
hipo_cats([aa(U3,U4,Cat,Cat1,Cats,Q,S,R,V)|A_Necs],[ac(U4,U5,desc:Pal,Q1,R1,S1,V1)|PalDescs],[A_Desc|A_Descs]):-
        tira_ilha(Cat1,NCat1),
        functor(NCat1,NNCat1,Ari),
        functor(Erro,NNCat1,Ari),
        saca_pterminal(Erro,Erro ---> [[Pal],{_}]),
        incorporar(aa(U3,U4,Cat,Cat1,Cats,Q,S,R,V),
        ac(U4,U5,desc:Erro,Q1,R1,S1,V1),A_Desc),!,
        hipo_cats(A_Necs,[ac(U4,U5,desc:Pal,Q1,R1,S1,V1)|PalDescs],A_Descs).
hipo_cats([A1|As],PalDescs,A_Descs):-!, hipo_cats(As,PalDesc,A_Descs).
```

%%existem_cats/6 - Constrói a lista de pendentes correspondentes às hipóteses de:

%%1. Só existirem arcos activos resultantes da incorporação de palavras desconhecidas em arcos que originaram a falha

%%2. Não existirem arcos na situação referida em 1.

%%3. Existirem arcos na situação 1. e existirem arcos completos iniciais com vértice igual ou superior ao vértice **%%**correspondente à falha (hipótese de existência de palavras a mais)

```
existem_cats([Cs_Vazios|[[]]],_,_,A_Descs,A_Descs,0):-!.
existem_cats([Cs_Vazios|[ICats]],WPalDescs,A_Nec,[],Ps,TExp):-!,
        existe_1cat(ICats,_,_,A_Nec,A_AMais,_),
        pal_oms(A_Nec,A_Oms),
        resolve_contr(A_AMais,A_Oms,WA_Oms),
        existe_1cat(ICats,WPalDescs,Cs_Vazios,WA_Oms,NA_Oms,TExp),
        concatena(NA_Oms,A_AMais,Ps).
existem_cats([Cs_Vazios|[ICats]],WPalDescs,A_Nec,A_Descs,Ps,TExp):-!,
        existe_1cat(ICats,WPalDescs,Cs_Vazios,A_Descs,NA_Descs,TExp),
        existe_1cat(ICats,_,_,A_Nec,A_AMais,_),
        concatena(A_AMais,NA_Descs,Ps).
```

%%existe_1cat/6 - Produz a lista de arcos correspondentes à hipótese de existência de palavras a mais ou à tentativa de **%%**incorporação de um arco completo inicial ou derivado em vazio nos arcos pendentes produzidos como hipóteses **%%**formuladas para desconhecimento ou omissão

```
existe_1cat([],_,_,_,[],0):-!.
existe_1cat([IC|ICats],WPalDescs,Cs_Vazios,As,Ps,TExp):- cat_nec(As,IC,WPalDescs,Cs_Vazios,Ps,TExp),
                                                         Ps \= [],!.
existe_1cat([IC|ICats],WPalDescs,Cs_Vazios,As,Ps,TExp):-!,existe_1cat(ICats,WPalDescs,Cs_Vazios,As,Ps,TExp).
```

%%cat_nec/6 - Produz novos pendentes resultantes de restrições impostas aos arcos formulados como hipotéticos

```
cat_nec([],_,_,_,[],0):-!.
cat_nec([aa(U1,U2,Cat,Cat_Nec,Cats,Q,S,R,V)|A_Necs],IC,WPalDescs,Cs_Vazios,[NA|Ps],TExp):-
        incorporar(aa(U1,U3,Cat,Cat_Nec,Cats,Q,S,R,V),IC,NA),!,
        cat_nec(A_Necs,IC,WPalDescs,Cs_Vazios,Ps,TExp).
```

```
cat_nec([aa(U1,U2,Cat,Cat_Nec,Cats,Q,S,R,V)|A_Necs],IC,WPalDescs,Cs_Vazios,
        [aa(U1,U2,Cat,Cat_Nec,Cats,Q,S,R,V)|Ps],TExp):-
        restricao(Cat_Nec),!,
        cat_nec(A_Necs,IC,WPalDescs,Cs_Vazios,Ps,TExp).
cat_nec([aa(U1,U2,Cat,Cat_Nec,Cats,Q,S,R,V)|A_Necs],IC,WPalDescs,Cs_Vazios,
        [aa(U1,U2,Cat,Cat_Nec,Cats,Q,S,R,V),NA|NPs],NTExp):-
        e_nt(Cat_Nec),
        propor(Cat_Nec,U2,Q,S,R,V,[],Ps),
        pos_combinar(Ps,IC,WPalDescs,Cs_Vazios,NA),!,
        cat_nec(A_Necs,IC,WPalDescs,Cs_Vazios,NPs,TExp),
        NTExp is TExp + 1.
cat_nec([A_Nec|A_Necs],IC,WPalDescs,Cs_Vazios,[A_Nec|Ps],TExp):-
        functor(A_Nec,ac,_),!,
        cat_nec(A_Necs,IC,WPalDescs,Cs_Vazios,Ps,TExp).
cat_nec([A_Nec|A_Necs],IC,WPalDescs,Cs_Vazios,Ps,TExp):-cat_nec(A_Necs,IC,WPalDescs,Cs_Vazios,Ps,TExp).
```

%%pos_combinar/5 - Tenta incorporar num arco activo da lista de pendentes (Ps) um arco completo inicial ou um arco **%%**completo correspondendo a uma palavra desconhecida ou um arco completo correspondendo a uma categoria **%%**derivada em vazio produzindo uma nova lista de pendentes ou falhando se não conseguir a incorporação

```
pos_combinar([aa(_,U2,_,Cat_Nec,Cats,Q,S,R,V)|Ps],IC,WPalDescs,Cs_Vazios,NA):-
        e_nt(Cat_Nec),!,
        propor(Cat_Nec,U2,Q,S,R,V,[],WPs),
        concatena(Ps,WPs,NPs),
        pos_combinar(NPs,IC,WPalDescs,Cs_Vazios,NA).
pos_combinar([Ac|Ps],IC,WPalDescs,Cs_Vazios,Ac):-functor(Ac,ac,_),!.
pos_combinar([Aa|Ps],IC,WPalDescs,Cs_Vazios,NA):-incorporar(Aa,IC,NA),!.
pos_combinar([A|Ps],IC,WPalDescs,Cs_Vazios,NA):-incorporacao(A,Cs_Vazios,NA),!.
pos_combinar([aa(U1,U2,Cat,Cat_Nec,Cats,Q,S,R,V)|Ps],IC,WPalDescs,Cs_Vazios,NA):-
        separa_pals(WPalDescs,U1,NWPalDescs),
        hipo_cats([aa(U1,U2,Cat,Cat_Nec,Cats,Q,S,R,V)],NWPalDescs,[NA]),!.
pos_combinar([A|Ps],IC,WPalDescs,Cs_Vazios,NA):-!,pos_combinar(Ps,IC,WPalDescs,Cs_Vazios,NA).
```

%%incorporacao/3 - Tenta incorpora num arco activo um arco completo correspondente a uma categoria derivada em **%%**vazio obtendo NA ou falhando se não conseguir efectuar a incorporação

```
incorporacao(A,[Vazio|Cs_Vazios],NA):-incorporar(A,Vazio,NA).
incorporacao(A,[Vazio|Cs_Vazios],NA):-!,incorporacao(A,Cs_Vazios,NA).
```

%%pal_oms/2 - Constrói uma lista com os arcos resultantes da incorporação de arcos completos correspondentes a **%%**palavras omissas nos arcos activos correspondentes à falha.

```
pal_oms([],[]):-!.
pal_oms([aa(U1,U2,Cat,Cat1,Cats,Q,S,R,V)|A_Necs],[NA|A_Oms]):-
        tira_ilha(Cat1,NCat1),
        functor(NCat1,NNCat1,Ari),
        functor(Erro,NNCat1,Ari),
        saca_pterminal(Erro,Erro ---> [[U2:pal_omi],{_}]),
        incorporar(aa(U1,U2,Cat,Cat1,Cats,Q,S,R,V),
        ac(U2,U2,omi:Erro,Q0-Q0,R0-R0,S0-S0,V0-V0),NA),!,
        Erro =.. LErro,
        constroi_pterminais(LErro,[NNErro]),
        pal_oms(A_Necs,A_Oms).
```

%%resolve_contr/3 - Elimina as contradições resultantes de considerar simultaneamente a mesma categoria **%%**correspondendo à hipótese de palavra omissa e à hipótese de palavra a mais

```
resolve_contr(_,[],[]):- !.
resolve_contr(A_Dups,[A_Om|A_Oms],NA_Oms):- igual_cat(A_Om,A_Dups),!,
                                                resolve_contr(A_Dups,A_Oms,NA_Oms).
resolve_contr(A_Dups,[A_Om|A_Oms],[A_Om|NA_Oms]):-!,resolve_contr(A_Dups,A_Oms,NA_Oms).
```

5. Detector e Corrector de erros

%%%alerta/4 - Predicado principal da detecção e correcção de erros

%ICats - Arcos completos correspondentes às categorias pré-terminais iniciais

%As/Cs - Grafo

%Uf - vértice onde terminou a análise por grafo

%PalDescs - Lista dos arcos completos correspondendo a palavras desconhecidas

%%

```
alerta(ICats,As/Cs,Uf,PalDescs):-
        (retira_acfs(Uf,Cs,WFinais);retira_acfs(_,Cs,WFinais)),
        WFinais \= [],!,
        constroi_final(WFinais,LPTerminais),
        constroi_lcats(ICats,LICats),
        erros_frase(LPTerminais,LICats,PalDescs,WErros),
        mel_erros(WErros,Erros),
        mel_finais(Erros,WErros,WFinais,Finais),
        repetidos(LICats,RepICats),
        concordancia(Finais,LICats,RepICats,PalDescs,Erros_Conc),
        nl,nl,write('Erros de Concordância: '),
        escreve(Erros_Conc,0),
        consult(sons),
        ortografia(Erros,NErros),
```

```
        nl,nl,write('Outros Erros:'),
        escreve(NErros,0).
alerta(ICats,As/Cs,U,PalDescs):-!, encontra_ultimo(As,U,[],Erros),
                                   write(' ---> Frase incompleta:'),
                                   nl,nl,write('****** Falta(m)'),nl,
                                   escreve(Erros).
```

%%encontra_ultimo/4 - Produz uma lista de arcos com vértices inicial e final iguais a um vértice Uf ou cujo vértice final **%%**seja Uf

```
encontra_ultimo([],_,Erros,Erros):-!.
encontra_ultimo([aa(U,Uf,_,Cat1,_,_,_,_,_)|As],Uf,Erros,NErros):-
        e_t(Cat1),
        tira_ilha(Cat1,NCat1),
        not membro([(Uf,Uf,NCat1)],Erros),!,
        concatena(Erros,[[(Uf,Uf,NCat1)]],WErros),
        encontra_ultimo(As,Uf,WErros,NErros).
encontra_ultimo([Aa|As],Uf,Erros,NErros):- !,encontra_ultimo(As,Uf,Erros,NErros).
```

%%retira_acfs/3 - Produz uma lista com todos os arcos completos do símbolo inicial cujo vértice final coincida com o **%%**vértice U onde terminou a análise por grafo

```
retira_acfs(_,[],[]):-!.
retira_acfs(U,[ac(1,U,Rg:Cat1,[]-[],[]-[],[]-[],[]-[])|Cs],[Frase|Finais]):- call(simbolo_inicial(Cat1)),
                                                                              Cat1 =.. [S|LCat1],
                                                                              membro(sint(S):Frase,LCat1),
                                                                              retira_acfs(U,Cs,Finais),
                                                                              not membro(LCat1,Finais).
retira_acfs(U,[Cs1|Cs],Finais):-!,retira_acfs(U,Cs,Finais).
```

%%constroi_lcats/2 - Dada uma lista de arcos completos constrói uma lista (LICats) formada por categorias e vértices **%%**respectivos do tipo do [[(U1,U2),Cat1,...,Catn],...]

```
constroi_lcats(ICats,LICats):- agrupa_pals(ICats,WCats),!,
                               lista_cats(WCats,LICats).
```

%%agrupa_pals/2 - Dada a lista de arcos completos correspondentes às categorias pré-terminais iniciais retorna uma lista **%%**constituída pelas categorias pré-terminais e respectivos vértices aí existentes

```
agrupa_pals([],[]):-!.
agrupa_pals([ac(U1,U2,pal:Pal,Q,R,S,V)|ICats],[NLICats|LICats]):-!,
        igual_arco([ac(U1,U2,pal:Pal,Q,R,S,V)|ICats],NICats,WLICats),
```

```
        concatena([(U1,U2)],WLICats,NLICats),
        agrupa_pals(NICats,LICats).
agrupa_pals([CI|ICats],LICats):-!,agrupa_pals(ICats,LICats).
```

%%igual_arco/3 - Retorna as categorias iniciais referentes aos mesmos vértices

```
igual_arco([ac(U1,U2,pal:Pal,Q,R,S,V),ac(U1,U2,pal:Pal1,Q1,R1,S1,V1)|ICats],NICats,[Pal|Lista_ICats]):-
        igual_arco([ac(U1,U2,pal:Pal1,Q1,R1,S1,V1)|ICats],NICats,Lista_ICats).
igual_arco([ac(_,_,pal:Pal,_,_,_,_)|ICats],ICats,[Pal]):-!.
```

%%lista_cats/2 - Retorna uma lista constituída por listas contendo a lista correspondente a cada categoria pré-terminal

```
lista_cats([[]],[[]]):-!.
lista_cats([[]|OutrasCats],[[]|NLICats]):-!,lista_cats(OutrasCats,NLICats).
lista_cats([[(U1,U2)|Catn]|OutrasCats],[[(U1,U2)|WLICats]|LICats]):-!,lista_cats([Catn|OutrasCats],[WLICats|LICats]).
lista_cats([[Cat1|Catn]|OutrasCats],[[LCat|WLICats]|LICats]):-!,
        Cat1 =.. WCat,
        constroi_pterminais(WCat,[LCat]),
        lista_cats([Catn|OutrasCats],[WLICats|LICats]).
```

%%constroi_final/2 - Retorna uma lista constituída pelas listas (Final) de listas correspondentes às categorias **%%**pré-terminais existentes nas categorias (Cat1/Cats)

```
constroi_final([],[]):-!.
constroi_final([Frase|Frases],[Final|Finais]):- constroi_pterminais(Frase,Final),
                                                constroi_final(Frases,Finais).
```

%%constroi_pterminais/2 - Dada uma categoria (Cat) retorna a lista de listas correspondentes às categorias pré-terminais **%%**aí existentes

```
constroi_pterminais([],[]):-!.
constroi_pterminais([sint(_):Resto1|Reston],Final):- constroi_pterminais(Resto1,Final1),
                                                     concatena(Final1,Finaln,Final),
                                                     constroi_pterminais(Reston,Finaln).
constroi_pterminais([Cat|Resto],[Cat|Final]):- call(pre_terminal(Cat)),!,
                                               constroi_pterminais(Resto,Final).
constroi_pterminais([Resto1|Resto],Final):-!,constroi_pterminais(Resto,Final).
```

%%mel_erros/2 - Dada uma lista de listas elemina as lista duplicadas, ordena-as restantes por menor nº de elementos e **%%**retorna as listas com menor nº de elementos

```
mel_erros(Erros,NErros):- elim_dups(Erros,Erros1),
                          ordena_tam(Erros1,WErros),
                          mel_erros1(WErros,NErros).
```

%%mel_erros1/2 - Retira duma lista de listas ordenadas por menor nº de elementos as listas com menor nº de elementos

```
mel_erros1([WErro],[WErro]):-!.
mel_erros1([WErro1,WErro2|WErros],[WErro1|NErros]):- length(WErro1,Tam),
                                                     length(WErro2,Tam),!,
                                                     mel_erros1([WErro2|WErros],NErros).
mel_erros1([WErro|_],[WErro]).
```

%%mel_finais/4 - Dada a lista de erros (Erros), a lista de listas com menor número de erros (WErros) e a lista (WFinais) **%%**de listas correspondentes aos arcos completos do simbolo inicial constrói a lista (Finais) correspondente a WErros

```
mel_finais([[]],[[]],Finais,Finais):-!.
mel_finais([],_,_,[]):-!.
mel_finais([Erro|Erros],WErros,WFinais,[Final|Finais]):- elemento_n(WErros,Elem,Erro),
                                                        n_elemento(WFinais,Elem,Final),
                                                        mel_finais(Erros,WErros,WFinais,Finais).
```

%%repetidos/2 - Constroi uma lista com os pré-terminais duplicados

```
repetidos([],[]):-!.
repetidos([[(U1,U2)|Cat]|LICats],[Cat|RepICats]):- membro([(U3,U4)|Cat],LICats),!,
                                                   repetidos(LICats,RepICats).
repetidos([LICat|LICats],RepICats):-!,repetidos(LICats,RepICats).
```

%%erros_frase/4 - Detecta os erros existentes em cada solução (LPTerminal) com base na lista (LICats) de listas das **%%**categorias pré-terminais iniciais e na lista (PalDescs) de arcos completos correspondentes a palavras desconhecidas

```
erros_frase([],_,_,[]):-!.
erros_frase([LPTerminal|LPTerminais],LICats,PalDescs,[WErros|WErros2]):-!,
        pertencem_lcats(LPTerminal,LICats,LPTerminal1,NLICats),
        pertencem_desc(LPTerminal1,PalDescs,NPalDescs,LPTerminal2,Erros_Descs),
        pertencem_om(LPTerminal2,Erros_Oms),
        concatena(Erros_Descs,Erros_Oms,WErros1),
        pertencem_amais(NLICats,NPalDescs,Erros_AMais),
        concatena(Erros_AMais,WErros1,WErros),
        erros_frase(LPTerminais,LICats,PalDescs,WErros2).
```

%%pertencem_lcats/4 - Dada a lista de listas correspondentes a cada solução (LPTerminal) e a lista de listas **%%**correspondentes às categorias pré-terminais iniciais (LICats) constrói a lista (NLPTerminais) de elementos de **%%**LPTerminal que não pertencem a LICats e a lista (NLICats) de elementos de LICats que não pertencem a LPTerminal

```
pertencem_lcats([],LICats,[],LICats):-!.
pertencem_lcats([LPTerminal|LPTerminais],LICats,NLPTerminais,NLICats):-
        pertence_lcats(LPTerminal,LICats,WLICats),
        WLICats\=LICats,!,
```

```
        pertencem_lcats(LPTerminais,WLICats,NLPTerminais,NLICats).
pertencem_lcats([LPTerminal|LPTerminais],LICats,[LPTerminal|NLPTerminais],NLICats):-!,
        pertencem_lcats(LPTerminais,LICats,NLPTerminais,NLICats).
```

%pertence_lcats/3

```
pertence_lcats(_,[],[]):-!.
pertence_lcats(LPTerminal,[[(U1,U2)|Cat]|LICats],LICats):- equiv_lista(LPTerminal,Cat,_).
pertence_lcats(LPTerminal,[Cat|LICats],[Cat|NLICats]):-!, pertence_lcats(LPTerminal,LICats,NLICats).
```

%%pertencem_desc/5 - Dada uma lista de listas correspondente a cada solução (LPTerminal) e uma lista de arcos **%%**completos correspondentes a palavras desconhecidas (PalDescs) retorna os erros correspondentes a palavras **%%**desconhecidas, a lista que contém os LPTerminais restantes e a lista que contêm as palavras desconhecidas não **%%**pertencentes a LPTerminal

```
pertencem_desc([],PalDescs,PalDescs,[],[]):-!.
pertencem_desc([LPTerminal|LPTerminais],PalDescs,NPalDescs,NLPTerminais,[(U1,U2,desc:LPTerminal)|Erros]):-
        pertence_desc(LPTerminal,PalDescs,U1,WPalDescs),!,
        U2 is U1 + 1,
        pertencem_desc(LPTerminais,WPalDescs,NPalDescs,NLPTerminais,Erros).
pertencem_desc([LPTerminal|LPTerminais],PalDescs,NPalDescs,[LPTerminal|NLPTerminais],Erros):-!,
        pertencem_desc(LPTerminais,PalDescs,NPalDescs,NLPTerminais,Erros).
```

%pertence_desc/4

```
pertence_desc(LPTerminal,[ac(U1,U2,desc:pal_dsc(Pal),_,_,_,_)|PalDescs],U1,PalDescs):-
        LPTerminal =.. LLPTerminal,
        pertence(pal_dsc(Pal),LLPTerminal).
pertence_desc(LPTerminal,[PalDesc|PalDescs],Indice,[PalDesc|NPalDescs]):-!,
        pertence_desc(LPTerminal,PalDescs,Indice,NPalDescs).
```

%%pertencem_om/2 - Dada uma lista de listas correspondente a cada solução (LPTerminal) retorna os erros **%%**correspondentes a omissões

```
pertencem_om([],[]):-!.
pertencem_om([LPTerminal|LPTerminais],[(U3,U3,omissa:LPTerminal)|Erros]):-
        LPTerminal =.. LLPTerminal,
        membro(U3:pal_omi,LLPTerminal),!,
        substitui(U3:pal_omi,LLPTerminal,pal_omi,WLPTerminal),
        WWLPTerminal =.. WLPTerminal,
        pertencem_om(LPTerminais,Erros).
pertencem_om([LPTerminal|LPTerminais],[LPTerminal|NLPTerminais],Erros):-!,pertencem_om(LPTerminais,Erros).
```

%%pertencem_amais/3 - Dada uma lista de categorias LCat/LICat e uma lista de arcos completos correspondentes a **%%**palavras desconhecidas (PalDescs) retorna os erros correspondentes a palavras a mais

```
pertencem_amais([],[],[]):-!.
pertencem_amais([[(U1,U2)|LCat]|LICats],PalDescs,[(U1,U2,amais:LCat)|Erros]):-
        pertencem_amais(LICats,PalDescs,Erros).
pertencem_amais([],[ac(U1,U2,desc:pal_dsc(Pal),_,_,_,_)|PalDescs],[(U1,U2,amais:pal_dsc(Pal))|Erros]):-!,
        pertencem_amais([],PalDescs,Erros).
```

%%escreve/2 - Informa o utilizador sobre as diferentes hipóteses de correcções propostas para que uma sequência de **%%**palavras possa ser considerada uma frase

```
escreve([],0):-!,write(' não tem! ').
escreve([[]|Erros],0) :-!,write(' não tem! ').
escreve([],_):-!.
escreve(Erros,0) :-!,escreve(Erros,1).
escreve([Erros1|Erros],Hipotese) :-!,nl,write('****** Hipotese '),
                                        write(Hipotese),nl,
                                        escreve([Erros1]),
                                        NHipotese is Hipotese + 1,
                                        escreve(Erros,NHipotese).
```

%escreve/1

```
escreve([[]]):- !.
escreve([[(U1,U2,Erro1)|Erro]|Erros]):- !,write('De '),write(U1),
                                            write(' a '),write(U2),
                                            write(' '),write(Erro1),nl,
                                            escreve([Erro|Erros]).
escreve([[]|[Erros1|Erros]]):- !,write('- Ou '),nl,
                                escreve([Erros1|Erros]).
escreve([[Erro1|Erro]|Erros]):- !,write(Erro1),nl,
                                  escreve([Erro|Erros]).
```

5.1 Detecção e correcção de erros de concordância

%%%concordancia/5 - Predicado principal da detecção e correcção de erros de concordância

%Final/Finais - Listas de arcos completos correspondendo ao símbolo inicial

%LICats - Lista contendo listas correspondentes a cada categoria pré-terminal inicial

%RepICats - categorias pré-terminais atribuídas às palavras que se repetem na frase

%PalDescs - Lista dos arcos completos correspondentes a palavras desconhecidas

%Erros_Conc - Lista dos erros de concordância

%%

```
concordancia([],_,_,_,[]):-!.
concordancia([Final|Finais],LICats,RepICats,PalDescs,Erros_Conc):-
        detecta_erosc(Final,WErros_Conc),
        erros_conc(LICats,RepICats,PalDescs,WErros_Conc,Erros_Conc1),
        concordancia(Finais,LICats,RepICats,PalDescs,Erros_Conc2),
        concatena(Erros_Conc2,Erros_Conc1,Erros_Conc).
concordancia([Final|Finais],LICats,RepICats,PalDescs,WErros_Conc):-!,
        concordancia(Finais,LICats,RepICats,PalDescs,WErros_Conc).
```

%%detecta_erosc/2 - Retorna os arcos completos correspondentes a categorias não terminais existentes numa solução **%%**(LPTerminais) onde existem erros de concordância

```
detecta_erosc(Final,Erros_Conc):-
        sao_nt_err(Final,[Nao_Terminal|Nao_Terminais]),
        encontra_maiores(Nao_Terminais,Nao_Terminal,Erros_Conc).
```

%%sao_nt_err/2 - Detecta os não terminais com erro existentes na solucão

```
sao_nt_err([],[]):-!.
sao_nt_err([sint(NT):Resto|Reston],[sint(NT):Resto|Nao_Terminais]):-
        membro(conc:Param,Resto),
        encontra_err(Param),!,
        sao_nt_err(Resto,Nao_Terminais1),
        sao_nt_err(Reston,Nao_Terminais2),
        concatena(Nao_Terminais1,Nao_Terminais2,Nao_Terminais).
sao_nt_err([sint(NT):Resto|Reston],Nao_Terminais):-!,
        sao_nt_err(Resto,Nao_Terminais1),
        sao_nt_err(Reston,Nao_Terminais2),
        concatena(Nao_Terminais1,Nao_Terminais2,Nao_Terminais).
sao_nt_err([Resto1|Reston],Nao_Terminais):-!,sao_nt_err(Reston,Nao_Terminais).
```

%%encontra_maiores/3 - Retorna as categorias não terminais de maior amplitude que possuem parâmetros de **%%**concordância

```
encontra_maiores([],Lista,[Lista]):-!.
encontra_maiores([Lista2|Resto],Lista1,Maiores):-
        Lista1=sint(X):[conc:Param1|Resto1],
        Lista2=sint(Y):[conc:Param2|Resto2],
        param_err(Param1,Err_Param1),
        param_err(Param2,Err_Param2),
        subconj(Err_Param2,Err_Param1),
        e_parte(Lista2,Resto1),!,
        encontra_maiores(Resto,Lista1,Maiores).
encontra_maiores([Lista2|Resto],Lista1,[Lista1|Maiores]):-!,encontra_maiores(Resto,Lista2,Maiores).
```

%%e_parte/2 - Determina se uma dada estrutura sintáctica contém outra estrutura como os mesmos erros de concordância

```
e_parte(Lista1,Lista2):- membro(Lista1,Lista2),!.
e_parte(Lista1,Lista2):- contem_estrutura(Lista2,WLista2),
                         membro(Lista1,WLista2).
```

%%contem_estrutura/2 - Retorna os não terminais que contribuiram para a verificação da concordância num dado não **%%**terminal

```
contem_estrutura([],[]):-!.
contem_estrutura([sint(Y):[conc:Conc|Lista1]|Lista2],NLista):-
        contem_estrutura(Lista1,WLista1),
        contem_estrutura(Lista2,WLista2),!,
        concatena([sint(Y):[conc:Conc|Lista1]|WLista1],WLista2,NLista).
contem_estrutura([Lista|Lista2],NLista):-!,contem_estrutura(Lista2,NLista).
```

%%encontra_err/1 - Detecta a existência de um parâmetro igual a err

```
encontra_err([Param=X|Resto]):-X == err; X == emasc, X == esin.
encontra_err([Param|Resto]):-!,encontra_err(Resto).
```

%%erros_conc/5 - Dada a lista de listas correspondentes às categorias pré-terminais iniciais (LICats), os pré-terminais **%%**repetidos na sequência de palavras (RepICats), a lista de arcos completos correspondentes às palavras desconhecidas **%%**(PalDescs) e a lista de arcos completos correspondendo às categorias onde existem erros de concordância (Cats) **%%**retorna os erros de concordância

```
erros_conc(_,_,_,[],[[]]):-!.
erros_conc(LICats,RepICats,PalDescs,[sint(X):[conc:Param|Cat]|Cats],Erros_Conc):-
        param_err(Param,Param_Err),
        nterminais_errados(Cat,LCats),
        listas(LICats,RepICats,PalDescs,LCats,NLCats),
```

```
        corrige_conc(Param_Err,NLCats,Erros_Conc1),!,
        erros_conc(LICats,RepICats,PalDescs,Cats,Erros_Conc2),
        ad_listas(Erros_Conc2,Erros_Conc1,Erros_Conc).
```

%%param_err/2 - Produz uma lista constituída por todos os paramêtros que sejam iguais a *err*

```
param_err([],[]):-!.
param_err([Param=Erro|Params],[Param=err|Erros]):- Erro == err,!,
                                                   param_err(Params,Erros).
param_err([Param=Erro|Params],[Param=emasc|Erros]):- Erro == emasc,!,
                                                      param_err(Params,Erros).
param_err([Param=Erro|Params],[Param=esin|Erros]):- Erro == esin,!,
                                                    param_err(Params,Erros).
param_err([Param|Params],Erros):-!,param_err(Params,Erros).
```

%%nterminais_errados/2 - Detecta os não terminais que contêm parametros de concordancia os args como não têm são **%%**desprezados

```
nterminais_errados([],[]):-!.
nterminais_errados([sint(Cat):[conc:Param|Resto1]|Reston],Nao_Terminais):-
        nterminais_errados(Resto1,Nao_Terminais1),!,
        nterminais_errados(Reston,Nao_Terminais2),
        concatena(Nao_Terminais1,Nao_Terminais2,Nao_Terminais).
nterminais_errados([sint(Cat):[conc:Param|Resto1]],Nao_Terminais):-!,nterminais_errados(Resto1,Nao_Terminais).
nterminais_errados([Resto1|Reston],[Resto1|Nao_Terminais]):- call(pre_terminal(Resto1)),!,
                                                             nterminais_errados(Reston,Nao_Terminais).
nterminais_errados([Resto1|Reston],Nao_Terminais):-!,nterminais_errados(Reston,Nao_Terminais).
```

%%listas/5 - Constrói uma lista de listas correspondentes a cada categoria pré-terminal Cat1 que está contida num não **%%**terminal

```
listas(_,_,_,[],[]):-!.
listas(LICats,RepICats,[ac(U1,U2,desc:Pal,_,_,_,_)|PalDescs],[Cat1|Cats],[[(U1,U2)|LCat]|NLCats]):-
        Cat1 =.. LCat,
        membro(Pal,LCat),!,
        listas(LICats,RepICats,PalDescs,Cats,NLCats).
listas([[(U1,U2)|LICat]|LICats],RepICats,PalDescs,[Cat1|Cats],[[(U1,U2)|LCat]|NLCats]):-
        equiv_lista(Cat1,LICat,NCat),
        (not membro(LICat,RepICats);Cats=[]),
        NCat =.. LCat,
        listas(LICats,RepICats,PalDescs,Cats,NLCats).
listas([[(U1,U2)|LICat1],[(U3,U4)|LICat2]|LICats],RepICats,PalDescs,
         [Cat1,Cat2|Cats],[[(U1,U2)|LCat1],[(U3,U4)|LCat2]|NLCats]):-
        equiv_lista(Cat1,LICat1,NCat1),
```

```
        membro(LICat,RepICats),
        NCat1 =.. LCat1,
        equiv_lista(Cat2,LICat2,NCat2),!,
        NCat2 =.. LCat2,
        listas(LICats,RepiCats,PalDescs,Cats,NLCats).
listas([ICat|LICats],RepICats,PalDescs,Cats,NLCats):-listas(LICats,RepICats,PalDescs,Cats,NLCats).
```

%%ad_listas/3 - Dadas duas listas de listas retorna a lista de listas de todas as suas combinações possíveis

```
ad_listas([[]],L,L):-!.
ad_listas(L,[[]],L):-!.
ad_listas(_,[],[]):-!.
ad_listas([],_,[]):-!.
ad_listas([[Lista1|R]|Listan],[Erros1|Errosn],NListas):-!, ad_listas([Lista1|R],[Erros1|Errosn],WLista),
                                                          concatena(WLista,WListas,NListas),
                                                          ad_listas(Listan,[Erros1|Errosn],WListas).
ad_listas(Lista,[Erros1|Errosn],[WLista|NListas]):- concatena(Lista,Erros1,WLista),
                                                    ad_listas(Lista,Errosn,NListas).
```

%%corrige_conc/4 - Retorna a lista (Erros_Conc) contendo as correcções referentes aos erros de concordância dada uma **%%**lista de parâmetros (Params) correspondentes aos erros detectados, a categoria não terminal (Cat) onde existem os **%%**erros e a lista (NLCats) de categorias pré-terminais constituintes dessa categoria

```
corrige_conc([],_,[[]]):-!.
corrige_conc([Param=X|Params],NLCats,Erros_Conc):-
        encontra(NLCats,Param=X,[],Erros),!,
        erros_params(NLCats,NLCats,Erros,Erros_Conc1),
        corrige_conc(Params,NLCats,Erros_Conc2),!,
        ad_listas(Erros_Conc1,Erros_Conc2,Erros_Conc).
corrige_conc([Param=X|Params],NLCats,NErros_Conc):-corrige_conc(Params,NLCats,NErros_Conc).
```

%%encontra/4 - Produz uma lista de erros contendo os valores de Param encontrados em cada lista de uma categoria **%%**pré-terminal

```
encontra([],_,Erros,Erros):-!.
encontra([Cat|LCats],Param=emasc,Erros_Conc,NErros_Conc):-!,
        not membro(Param=masc,Erros_Conc),!,
        concatena(Erros_Conc,[Param=masc],WErros_Conc),
        encontra(LCats,Param=X,WErros_Conc,NErros_Conc).
encontra([Cat|LCats],Param=esin,Erros_Conc,NErros_Conc):-
        not membro(Param=sin,Erros_Conc),!,
        concatena(Erros_Conc,[Param=sin],WErros_Conc),
        encontra(LCats,Param=X,WErros_Conc,NErros_Conc).
```

```
encontra([Cat|LCats],Param=X,Erros_Conc,NErros_Conc):-
        membro(Param=N,Cat),
        not membro(Param=N,Erros_Conc),
        concatena(Erros_Conc,[Param=N],WErros_Conc),
        encontra(LCats,Param=X,WErros_Conc,NErros_Conc).
encontra([Cat|LCats],Param=X,WErros_Conc,NErro_Conc):-!,encontra(LCats,Param=X,WErros_Conc,NErro_Conc).
```

%%erros_params/4 - Constroi a lista de correcções de erros de concordância dada a lista de categorias pré-terminais **%%**(LCats) onde existe o erro referente ao parâmetro Param e a lista dos valores obtidos para esse parâmetro (Erros) **%%**obtendo a lista de correcções (NErros)

```
erros_params(_,_,[],[]):-!.
erros_params([],LCats,[Erro1|Erros],[[]|NErros]):-!,erros_params(LCats,LCats,Erros,NErros).
erros_params([LCat|LCats],WLCats,[Param=X|Erros],NErros):-
        membro(Param=N,LCat),
        (not atomic(N);N=X),!,
        erros_params(LCats,WLCats,[Param=X|Erros],NErros).
erros_params([LCat|LCats],WLCats,[Param=X|Erros],[[(U1,U2,Erro:Cat)|NErros1]|NErros]):-
        membro(Param=N,LCat),!,
        substitui(Param=N,LCat,Param=X,NLCat),
        [(U1,U2)|Resto]=NLCat,
        Cat =.. Resto,
        concat('conc_',Param,Erro),
        erros_params(LCats,WLCats,[Param=X|Erros],[NErros1|NErros]).
erros_params([LCat|LCats],WLCats,Params,NErros):-!,erros_params(LCats,WLCats,Params,NErros).
```

5.2 Corrector ortográfico

%%%ortografia/2 - Predicado principal da correcção ortográfica

%Erros - Lista de erros

%NErros - Nova lista de erros

%LPals - Lista contendo as correcções referentes a uma dada categoria Cat atribuída a uma dada palavra desconhecida (Pal)

%%

```
ortografia([[]],[]):-!.
ortografia([],[]):-!.
ortografia([[]|Errosn],[[]|NErros]):-!,ortografia(Errosn,NErros).
ortografia([[(U1,U2,desc:Cat)|Erros1]|Errosn],[[(U1,U2,desc(Pal):LPals)|LLPals]|NErros]):-
        Cat =..LCat,
        pesquisa_dsc(LCat,Ari,Pal),
        ortog(Cat,Pal,Ari,LPals),
        ortografia([Erros1|Errosn],[LLPals|NErros]).
ortografia([[Erro1|Erros1]|Errosn],[[Erro1|LLPals]|NErros]):-!,ortografia([Erros1|Errosn],[LLPals|NErros]).
```

%%pesquisa_desc/3 - Dada uma lista correspondendo aos argumentos duma categoria pré-terminal retorna a palavra
%%desconhecida e a sua posição de ordem na categoria

```
pesquisa_dsc([pal_dsc(Pal)|_],0,Pal):-!.
pesquisa_dsc([Elem|Resto],Ari,Pal):- pesquisa_dsc(Resto,Ari1,Pal),
                                     Ari is Ari1 + 1.
```

%%ortog/4 - Dada uma categoria, uma palavra desconhecida e a sua localização no functor que denota a categoria retorna
%%a lista de correcções (NLPals) para a palavra desconhecidas

```
ortog(Cat,PalDesc,Ari,NLPals):-
        functor(Cat,Nome,NAri),
        argrep(Cat,Ari,_,WCat),
        quem_subst(WCat,Ari,NAri,NWCat),
        encontra_lex(NWCat,PalDesc,Ascii_PalDesc,Ascii_Lex),
        findall(Tipo_err:NCat,(damerau(Ascii_PalDesc,Ascii_Lex,Pal,Tipo_err),argrep(WCat,Ari,Pal,NCat)),WCats),
        filtra_som(Ascii_PalDesc,Ascii_Lex,WCat,Ari,WCats,NCats),
        elim_dups(NCats,NNCats),
        ordena_hipos(NNCats,LPals),!,
        argrep(WCat,Ari,PalDesc,Neol_Cat),
        concatena(LPals,[neol:Neol_Cat],NLPals).
```

%%quem_subst/4 - Permite que, no caso de erro existente num verbo auxiliar, se pesquise o verbo adequado
%%Ex. A proposta de categoria para 'fi' poderia ser v(pes=P,num=N,gen=G,tempo=T,forma=F,modo=M,sc=SCAT,V,ser))

```
quem_subst(Cat,Ari,Ari,Cat):-!.
quem_subst(Cat,Ari,NAri,WCat):-!,argrep(Cat,NAri,Cat,WCat).
```

%%encontra_lex/4 - Dada uma categoria (Cat) e uma palavra desconhecida (PalDesc) retorna a lista dos caracteres
%%existentes na palavra desconhecida codificada em ASCII e a lista constituída pelas listas correspondentes à codificação
%%ASCII de todas as palavras existentes no léxico e pertencentes à categoria dada com uma variação absoluta de 2
%%caracteres em relação ao comprimento de PalDesc

```
encontra_lex(Cat,PalDesc,Ascii_PalDesc,Ascii_Lex):-!,
        findall(Pals,(saca_pterminal(Cat,WCat ---> [[Pals],{Dic}]),call(Dic)),Lex),
        comprimento(PalDesc,Tam,Ascii_PalDesc),
        Tam2 is Tam + 2,
        Tam1 is Tam - 2,
        condiciona_pals(Tam1,Tam2,Lex,Ascii_Lex).
```

%%condiciona_pals/4 - Dada uma lista de palavras retorna uma lista com as listas de caracteres ASCII que correspondem **%%**às palavras cujo comprimento seja superior ou igual a Tam1 e inferior ou igual a Tam2.

```
condiciona_pals(_,_,[],[]):-!.
condiciona_pals(Tam1,Tam2,[Pal1|Pals],[Ascii_Pal1|Ascii_Lex]):-
        comprimento(Pal1,Tam_Pal1,Ascii_Pal1),
        Tam_Pal1 >= Tam1,
        Tam_Pal1 =< Tam2,!,
        condiciona_pals(Tam1,Tam2,Pals,Ascii_Lex).
condiciona_pals(Tam1,Tam2,[Pal1|Pals],Ascii_Lex):-!,condiciona_pals(Tam1,Tam2,Pals,Ascii_Lex).
```

%%comprimento/3 - Dada uma palavra retorna o seu comprimento e a lista, correspondente, de caracteres ASCII

```
%%comprimento(Pal,Tam,Ascii_Pal):-!,string_term(Str_Pal,Pal),
                                        list_text(Ascii_Pal,Str_Pal),
                                        length(Ascii_Pal,Tam).
```

%%damerau/4 - Detecta os erros de damereau e retorna as palavras que correspondem às correcções bem como o tipo de **%%**erro encontrado (erros tipográficos ou resultantes de confusões sonoras)

```
damerau(Ascii_PalDesc,Lex,Pal,Tipo_err):-
        omissao(Ascii_PalDesc,Lex,Ascii_Pal),
        palavra(Ascii_Pal,Pal),
        sons(Ascii_PalDesc,Ascii_Pal,Tipo_err).
damerau(Ascii_PalDesc,Lex,Pal,Tipo_err):-
        insercao(Ascii_PalDesc,Lex,Ascii_Pal),
        palavra(Ascii_Pal,Pal),
        sons(Ascii_PalDesc,Ascii_Pal,Tipo_err).
damerau(Ascii_PalDesc,Lex,Pal,Tipo_err):-
        subst(Ascii_PalDesc,Lex,Ascii_Pal),
        palavra(Ascii_Pal,Pal),
        sons_ou_ac(Ascii_PalDesc,Ascii_Pal,Tipo_err).
damerau(Ascii_PalDesc,Lex,Pal,corr_tip):-
        transposicao(Ascii_PalDesc,Lex,Ascii_Pal),
        palavra(Ascii_Pal,Pal).
```

%%sons/3 - Detecta se o erro é referente a confusões sonoras e retorna o tipo de erro (corr_fon ou corr_tip)

```
sons(Ascii_PalDesc,Ascii_Pal,corr_fon):- pesquisa_cadeia(Ascii_PalDesc,Ascii_Pal,WAscii_PalDesc,WAscii_Pal),
                                        som(WAscii_PalDesc,WAscii_Pal),!.
sons(_,_,corr_tip):-!.
```

%%sons_ou_ac/3 - Detecta se o erro é referente a confusões sonoras ou a incorrecta acentuação e retorna o tipo de erro **%%** (corr_fon, corr_ac ou corr_tip)

```
sons_ou_ac(Ascii_PalDesc,Ascii_Pal,Tipo_err):-acentos(Ascii_PalDesc,Ascii_Pal,Tipo_err),!.
sons_ou_ac(Ascii_PalDesc,Ascii_Pal,Tipo_err):-!,sons(Ascii_PalDesc,Ascii_Pal,Tipo_err).
```

%%acentos/3 - Detecta se o erro é referente a incorrecta acentuação e retorna o tipo de erro (corr_ac)

```
acentos(Ascii_PalDesc,Ascii_Pal,corr_ac):- pesquisa_cadeia(Ascii_PalDesc,Ascii_Pal,WAscii_PalDesc,WAscii_Pal),
                                           acento(WAscii_PalDesc,WAscii_Pal),!.
```

%%filtra_som/6 - Detecta correcções fonéticas não coincidentes com erros de Damerau e adiciona-as à lista das possíveis **%%**correcções (WCats)

```
filtra_som(Ascii_PalDesc,Lex,WCat,Ari,WCats,NCats):- corrige_letras(Ascii_PalDesc,Ascii_Pal),
                                                     membro(Ascii_Pal,Lex),!,
                                                     palavra(Ascii_Pal,Pal),

                                                     argrep(WCat,Ari,Pal,NCat),
                                                     concatena([corr_fon->NCat],WCats,NCats).
filtra_som(_,_,_,_,NCats,NCats):-!.
```

%%corrige_letras/2 - Dada uma lista de caracteres ASCII pesquisa a tabela de correspondências de sons para os erros **%%**resultantes de confusões fonéticas e não coincidentes com erros de Damerau retornando a correcção (Ex: 'ss' em vez de **%%**'ç')

```
corrige_letras([Letra1,Letra2|Ascii_PalDesc],[Letra|Ascii_PalDesc]):-call(tab_esp([Letra],[Letra1,Letra2])).
corrige_letras([Letra|Ascii_PalDesc],NAscii_Pal):- call(tab_esp([Letra],Letras)),
                                                   concatena(Letras,Ascii_Pal,NAscii_Pal).
corrige_letras([Letra|Ascii_PalDesc],[Letra|Ascii_Pal]):-corrige_letras(Ascii_PalDesc,Ascii_Pal).
```

%%pesquisa_cad/4 - Produz duas listas contendo as diferenças encontradas em duas listas de caracteres ASCII **%%**correspondendo à existência de um erro de Damerau

```
pesquisa_cadeia([Letra1],[Letra2],[Letra1],[Letra2]):-!.
pesquisa_cadeia([Letra1,Letra2|Letras],[Letra1,Letra2|NLetras],WAscii_PalDesc,WAscii_Pal):-!,
        pesquisa_cadeia([Letra2|Letras],[Letra2|NLetras],WAscii_PalDesc,WAscii_Pal).
pesquisa_cadeia([Letra1|Letras],[Letra1,NLetra2|Letras],   [Letra1],[Letra1,NLetra2]):-!.
pesquisa_cadeia([Letra1,NLetra2|Letras],[Letra1|Letras],[Letra1,NLetra2],[Letra1]):-!.
pesquisa_cadeia([Letra1|Letras],[Letra|[Letra1|Letras]],[Letra1],[Letra,Letra1]):-!.
pesquisa_cadeia([Letra1,Letra2|Letras],[Letra2|Letras],[Letra1,Letra2],[Letra2]):-!.
pesquisa_cadeia([Letra1,Letra2|Letras],[Letra1,NLetra2|Letras],[Letra2],[NLetra2]):-Letra2 \=NLetra2,!.
pesquisa_cadeia([Letra,Letra2|Letras],[Letra1,Letra2|NLetras],[Letra],[Letra1]):-Letra \=Letra1,!.
```

%%som/2 - Dadas duas listas de caracteres ASCII verifica se existem na tabela de letras com o mesmo som (tab_som)

```
som([Letra],NLetras):- call(tab_som([Letra],LLetras)),!,
                        membro(NLetras,LLetras).
som(Letras,NLetras):- findall(Letra,(tab_som(Letra,LLetras),membro(Letras,LLetras)),Lista),
                      membro(NLetras,Lista).
```

%%acento/2 - Dadas duas listas de caracteres ASCII verifica se existem na tabela de acentos (tab_ac)

```
som([Letra],NLetras):- call(tab_som([Letra],LLetras)),!,
                        membro(NLetras,LLetras).
som(Letras,NLetras):- findall(Letra,(tab_som(Letra,LLetras),membro(Letras,LLetras)),Lista),
                      membro(NLetras,Lista).
```

%%palavra/2 - Transforma uma lista de caracteres ASCII numa palavra

```
palavra(Ascii_Pal,Pal):- list_text(Ascii_Pal,Str_Pal),
                         string_term(Str_Pal,Pal).
```

%%omissao/3 - Dada a lista (X) de caracteres ASCII correspondentes à palavra desconhecida e a lista (ASCII_Lex) das **%%**palavras existentes no léxico detecta erros de omissão retornando a lista (Y) de caracteres ASCII correspondentes à **%%**correcção

```
omissao(X,Ascii_Lex,Y):- ad_car(X,Y),
                         membro(Y,Ascii_Lex).
%
ad_car(X,[Y|X]).
ad_car([U|V],[U|W]):-ad_car(V,W).
```

%%insercao/3 - Dada a lista (X) de caracteres ASCII correspondentes à palavra desconhecida e a lista (ASCII_Lex) das **%%**palavras existentes no léxico detecta erros de inserção retornando a lista (Y) de caracteres ASCII correspondentes à **%%**correcção

```
insercao(X,Ascii_Lex,Y):- apaga_car(X,Y),
                          membro(Y,Ascii_Lex).
%
apaga_car([X|Y],Y).
apaga_car([X|U],[X|V]):-apaga_car(U,V).
```

%%subst/3 - Dada a lista (X) de caracteres ASCII correspondentes à palavra desconhecida e a lista (ASCII_Lex) das **%%**palavras existentes no léxico detecta erros de substituição retornando a lista (Y) de caracteres ASCII correspondentes **%%**à correcção

```
subst(X,Ascii_Lex,Y):- subst_car(X,Y),
                       membro(Y,Ascii_Lex).
%
```

```
subst_car([X|Y],[U|Y]).
subst_car([X|V],[X|W]):-subst_car(V,W).

%%transposicao/3 - Dada a lista (X) de caracteres ASCII correspondentes à palavra desconhecida e a lista (ASCII_Lex)
%%das palavras existentes no léxico detecta erros de transposição retornando a lista (Y) de caracteres ASCII
%%correspondentes à correcção
transposicao(X,Ascii_Lex,Y):- transpoe(X,Y),
                              membro(Y,Ascii_Lex).
%
transpoe([],[]).
transpoe([X|Y],[X|Z]):- transpoe(Y,Z).
transpoe([X|[W|Y]],[W|[X|Y]]).

%%elim_dups/2 - Elimina duplicados existentes numa lista. Este predicado é invocado por orto porque, por ex. em 'baro'
%%a omissão é considerada na posição 3 ou na posição 4 obtendo assim, duas correcções iguais a 'barro'
elim_dups([],[]):-!.
elim_dups([Elem|Resto],NLista):- membro(Elem,Resto),!,
                                 elim_dups(Resto,NLista).
elim_dups([Elem|Resto],[Elem|NLista]):- elim_dups(Resto,NLista).

%%ordena_hipos/2 - Ordena as hipóteses de correcção preterindo correcções tipográficas (corr_tip) em favor de
%%correcções fonéticas (corr_ac e corr_fon)
ordena_hipos([],[]):-!.
ordena_hipos([Tipo_err:Cat|Resto],[Tipo_err:Cat|NResto]):- (Tipo_err=corr_fon;Tipo_err=corr_ac),
                                                           ordena_hipos(Resto,NResto).
ordena_hipos([Cat|Resto],NLista):-!,
        ordena_hipos(Resto,Lista),
        concatena(Lista,[Cat],NLista).
```

Bibliografia

[Abn 90] ABNEY, Steven P., Rapid Incremental Parsing with Repair, *6th. Ann. Conf. of the UW Center for the New OED and Text Research*, pag. 1-9, 1990

[AD 89] ABRAMSON, Harvey, DAHL, Veronica, *Logic Grammars*, Springer-Verlag, 1989

[AgAAA 92] AGIRRE, E.; ALEGRIA, I..; ARREGI, X.; ARTOLA, X.; DÍAZ de ILARRAZA, A.; MARITXALAR, M.; SARASOLA, K.; URKIA, M.; XUXEN: A Spelling Checker/Corrector for Basque Based on Two-Level Morphology, *3ª Conf. of Applied NLP*, pag. 119-125, 1992

[Al 87] ALLEN, James, *Natural Language Understanding*, Benjamin/Cummings Company, Inc., 1987

[AFW 83] ANGELL, R. C.; FREUND, G. E.; WILLETT, P., Automatic Spelling Correction using a Trigram Similarity Measure, *Inf. Process. Manage.* 19, pag. 255-261, 1983

[AtE 87] ATWELL, E.; ELLIOTT, S., Dealing with Ill-Formed English Text, In *Computational Analysis of English: A Corpus-Based Approach*, R. Garside, G. Leach, G. Sampson, Ed. Longman, Inc. New York, 1987

[BDS 88] BERKEL, Briggite van; SMEDT, Koenraad De,Triphone Analysis: a Combined Method for the Correction of Ortographical and Typographical Errors, *2th. Conf. on Applied NLP*, pag. 77-83, 1988

[Ber 87] BERGHEL, H. L., A Logical Framework for the Correction of Spelling Errors in Electronic Documents**,** *Information Processing & Manag., vol. 23, No 5*, pag. 477-494, 1987

[CBMA 83] CARBONELL, Jaime G.; BOGS, W. Mark; MAULDIN, Michael L.; ANICK, Peter G., First Steps Towards an Integrated Natural Language Interface, *XCALIBUR PROJECT REPORT 1*, 1983

[CBMA 83a] CARBONELL, Jaime G.; BOGS, W. Mark; MAULDIN, Michael L.; ANICK, Peter G., A Natural Language Interface to Expert Systems, *Proceed. 8th. IJCAI*, pag. 653-656, 1983

[Car 84] CARBERRY, S., Understanding Pragmatically Ill-Formed Input, *Proc. of the 10th International Conference on Computational Linguistics*, ACL, pag. 100-206, 1984

[CDL 87] CASTRO, Ivo; DUARTE, Inês; LEIRIA, Isabel, *A Demanda da Ortografia Portuguesa*, Ed. João Sá da Costa, 1987

[CaH 83] CARBONELL, J. G.; HAYES, P. J., Recovery Strategies for Parsing Extragrammatical Language, *Am. J. Comput. Ling. 9*, 3-4, pag. 123-146, 1983

[CG 91] CHURCH, K. W.; GALE W. A., Probability Scoring for Spelling Correction, *Stat. Comput. 1*, pag. 93-103, 1991

[CL 71] CUESTA, Pilar Vázquez; LUZ, Maria Albertina Mendes da, *Gramática da Língua Portuguesa*, Edições 70, 1971

[CuC 85] CUNHA, Celso; CINTRA, Lindley, *Breve Gramática do Português Contemporâneo*, Edições João Sá da Costa, 1985

[Dam 64] DAMERAU, F. J., A Technique for Computer Detection and Correction of Spelling Erros, *Communications of ACM 7, 3*, pag. 171-176, 1964

[DLS 83] DURHAM, Ivor; LAMB, David A.; SAXE, James B., Spelling Correction in User Interfaces, *Communications of ACM 26, 10*, pag. 764-773, 1983

[Eu 91] EUROTRE 6-1, Rule Formalism and Virtual Machine Design Study, *Tec. Report EUROTRE 6-1*, Commission of E. Communities, DG XIII B-5, pag. 44-69, 1991

[FW 83] FASS, D.; WILKS, Y., Preference Semantics, Ill-Formedness, and Metaphor, *Amer. J. Comput. Ling. 9*, 3-4, pag. 178-189, 1983

[GC 90] GALE, W. A.; CHURCH, K. W., Estimation Procedures for Language Context: Poor Estimates Are Worse than None, *Proc. of Compstat-90* (Dubrovnick, Yugoslavia), Springer-Verlag, New York, pag. 69-74, 1990

[GM 89] GAZDAR, Gerald; MELLISH, Chris, *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*, Addison-Wesley Pub. Comp., 1989

[GP 85] GAZDAR, Gerald, PULLUM, Geoffrey K., Computationally Relevant Properties of Natural Language and Their Grammars, *New Generation Computing, 3*, pag 273-306, 1985

[Goe 92] GOESER, Sebastian, Chart Parsing of Robust Grammars, *COLING*, pag. 120-126, 1992

[GP 87] GRISHMAN, Ralph; PENG, Ping, Responding to Semantically Ill-Formed Input, *Proteus Project Memorandum #7*, 1987

[Hau 92] HAUGENEDER, Hans, A Computational Model for Processing Coordinate Structures: Parsing Coordination without Grammatical Specification, *ECAI*, pag. 513-517, 1992

[HJMB 82] HEIDORN, G. E.; JENSEN, K.; MILLER, L. A.; BYRD, R. J.; CHODOROW, M. S., The EPISTLE Text-Critiquing System, *IBM Syst. J. 21, 3*, pag. 305-326, 1982

[KV 90] KEMPEN, G.; VOSSE, T., A Language Sensitive Text Editor for Dutch, *Proceedings of the Computers and Writing III Conference*, Edinburgh, 1990

[KT 93] KIYONO, Masaki; TSUJII, Jun-ichi, Linguistic Knowledge Acquisition from Parsing Failures, *EACL*, pag. 222-231, 1993

[Ku 92] KUKICH, Karen, Techniques for Automatically Correcting Words in Text, *ACM Computing Surveys 24 (4)*, pag. 377-439, 1992

[Lop 92] LOPES, José G. P., *Apontamentos da cadeira de PLN*, UNL, 1992

[LMR 94] LOPES, José G. P.; MARQUES, Nuno; ROCIO, Victor, POLARIS: a Portuguese Lexicon Acquisition and Retrieval Interactive System, *Proc. of the Int. Conf. in Pratical Applications of Prolog*, 1994

[Lut 93] LUTZ, Rudi, Chart Parsing of Attributed Structure-Sharing Flowgraphs with Tie-Point Relationships, *International Workshop in Parsing Technologies, 10-13*, *pag. 145-168,* 1993

[MBDF 89] MATEUS, Maria Helena Mira; BRITO, Ana Maria; DUARTE, Inês; FARIA, Isabel Hub, *Gramática da Língua Portuguesa*, ED. Caminho, 1989

[Mc 89] McCOY, K. F., Generating Context-Sensitive Responses to Object-Related Misconceptions, *Artif. Intell. 41*, pag. 157-195, 1989

[MDM 91] MAYS, E., DAMERAU, F. J., MERCER, R. L., Context Based Spelling Correction, *Inf. Process. Manage. 27*, 5, pag. 517-522, 1991

[Mea 88] MEANS, Linda G., Cn Yur Cmputr raed Ths?, *2th. Conf. on Applied NLP*, pag. 93-100, 1988

[Mel 89] MELLISH, Chris S., Some Chart-Based Techniques for Parsing Ill-Formed Input, *27th Ann. Meet. of ACL*, pag. 102-109, 1989

[MHF 85] MINTON, S.; HAYES, P. J.; FAIN, J., Controlling Search in Flexible Parsing, *Proc. of the International Joint Conference on Artificial Intelligence,* Morgan Kaufman, San Mateo, Calif., pag. 786-787, 1985

[PaMW 90] PARTEE, Barbara H., MEULEN, Alice ter, WALL, Robert E. Wall, *Mathematical Methods in Linguistics*, Kluwer Academic Pub., 1990

[Pau 93] PAULO, Alexandre P. G., Analisador por Grafo Extendido a Gramáticas de Movimentação e Ancoragem, *Trabalho de Sintaxe da cadeira de PLN do Mestrado em Eng. Inf. da UNL*, 1993

[Per 81] PEREIRA, Fernando C. N., Extraposition Grammars, *American Journal of Computational Linguistics 7(4),* pag. 243-255, 1981

[PW 80] PEREIRA, Fernando C. N.; WARREN, David H. D., Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artificial Intelligence 13*, pag. 231-278, 1980

[PW 83] PEREIRA, Fernando C. N.; WARREN, David H. D., Parsing as Deduction, *21 Annual Meeting of ACL*, pag. 137-144, 1983

[Ram 89] RAMSHAW, L. A., Pragmatic Knowledge for Resolving Ill-Formedness, *Tecn. Rep. 89-18*, BBN, Cambridge, Mass., 1989

[RB 88] RICHARDSON, Stephen D.; BRADEN-HARDER, Lisa C., The Experience of Developing a Large-Scale Natural Language Text Processing System: Critique, *2th. Conf. on Applied NLP*, pag. 195-202, 1988

[Ros 89] ROSADO, Paulo, Perspectivas Sequencial e Concorrente da Análise por Grafo em Ambientes de Programação em Lógica, *Relatório Técnico do CRIA - UNL*

[Ros 89a] ROSADO, Paulo, Extraposição e Análise por Grafo, *Relatório Técnico do CRIA - UNL*, 1989

[Rose 91] ROSENBLUETH, David A., Chart Parsers as Proof Procedures for Fixed-Mode Logic Programs, *Instituto de Investigaciones en Matematicas Aplicadas e en Sistemas - Unam*, 1991

[Sau 78] SAUSSURE, Ferdinand de, *Curso de Linguística Geral*, Pub. D. Quixote, 1978

[SchLB 80] SCHANK, R. C.; LEBOWITZ, M.; BIRNBAUM, L., An Integrated Understander, *Am. J. Comput. Ling. 6*, 1, pag. 13-30, 1980

[SikA 93] SIKKEL, Klaas; AKKER, Rieks op den, Predictive Head-Corner Chart-Parsing, *International Workshop in Parsing Technologies* 10-13, pag. 267-276, 1993

[SimH 90] SIMPKINS, Neil K.; HANCOX, Peter, Chart Parsing in Prolog, *New Generation Computing 8*, pag. 113-138, 1990

[Su 91] SURI, L. Z., Language Transfer: A Foundation for Correcting the Written English of ASL Signers, *Tecn. Rep. Nº 91-19*, Dept. of Computer and Information Sciences, Univ. of Delware, Newark, Del., 1991

[SuMC 91] SURI, L. Z.;McCoy, K. F., Language Transfer in Deaf Writing: A Correction Methodology for an Instructional System, *Tecn. Rep. Nº 91-20*, Dept. of Computer and Information Sciences, Univ. of Delware, Newark, Del., 1991

[Tom 93] TOMABECHI, Hideto, A Soft Unification Method for Robust Parsing, *International Workshop in Parsing Technologies 10-13*, pag. 1-8, 1993

[Tomi 86] TOMITA, Masaru, *Efficient Parsing for Natural Language: a fast algoritm for practical systems*, Dordecht, Kluwer, 1986

[Tra 83] TRAWICK, D. J., Robust Sentence Analysis and Habitability, *Ph. D. dissertation*, California, Inst. of Tecnology, Pasadena, Calif., 1983

[Ver 88] VERONIS, Jean, Morphosyntactic Correction in Natural Language Interfaces, *COLING*, Budapest, pag. 708-713, 1988

[Vos 92] VOSSE, Theo, Detecting and Correcting Morpho-Syntatic Errors in Real Texts, *Proc. of the 3rd. Conf. on Applied NLP*, ACL, pag. 111-118, 1992

[Wal 78] WALTZ, D. , L., An English Language Question Answering System for a Large Relational Database, *Commun. ACM 21*, 7, pag. 526-539, 1978

[WS 83] WEISCHEDEL, R. M.; SONDHEIMER, N. K., Meta-Rules as a Basis for Processing Ill-formed Input, *Amer. J. Comput. Ling. 9*, 3-4, pag. 161-177, 1983

[Yag 91] YAGUELLO, Marina, *Alice no País da Linguagem - para compreender a linguística*, Editorial Estampa, 1991