**CÉLIA YUMI OKANO TANIWAKI**

# FORMALISMOS ADAPTATIVOS NA ANÁLISE SINTÁTICA DE LINGUAGEM NATURAL

**Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Elétrica**

**SÃO PAULO**

**2001**

**CÉLIA YUMI OKANO TANIWAKI**

# FORMALISMOS ADAPTATIVOS NA ANÁLISE SINTÁTICA DE LINGUAGEM NATURAL

**Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Elétrica**

**Área de Concentração:**
**Sistemas Digitais**

**Orientador:**
**Prof. Dr. João José Neto**

**SÃO PAULO**

**2001**

FICHA CATALOGRÁFICA

**Taniwaki, Célia Yumi Okano**
**Formalismos adaptativos na análise sintática de linguagem natural. – São Paulo, 2001.**
**p.**

**Dissertação (Mestrado) -- Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.**

**1.Linguagem natural – Processamento 2.Teoria dos autômatos 3.Análise sintática I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.**

*Ao meu esposo, Paulo.*
*Aos meus filhos, Eliel e Samuel,*
*e à minha filha, que ainda está*
*para nascer.*
*Aos meus pais, Yukio e Paulina.*

*"Assim diz o SENHOR: Não se glorie o sábio na sua sabedoria, nem o forte, na sua força, nem o rico, nas suas riquezas; mas o que se gloriar, glorie-se nisto: em me conhecer e saber que eu sou o SENHOR e faço misericórdia, juízo e justiça na terra; porque destas cousas me agrado, diz o SENHOR."*

*- Jeremias 9:23,24 -*

# AGRADECIMENTOS

Agradeço, acima de tudo, a Deus, por ter me concedido a oportunidade de realizar este trabalho e por ter colocado em minha vida pessoas maravilhosas, sem as quais não teria tido condições de terminar esta dissertação.

Entre essas pessoas, primeiramente, o meu orientador, Prof. Dr. João José Neto, ao qual sou especialmente grata pela sua enorme paciência, pelo apoio constante, pelos conselhos imprescindíveis, pela valiosa amizade e pela confiança em mim depositada.

Sou grata, também, aos Profs. Drs. João Camargo Batista Júnior e Jorge Kinoshita, por participarem da banca examinadora de meu Exame de Qualificação, e por terem contribuído com opiniões e conselhos oportunos para o desenvolvimento da dissertação.

Ao colega Carlos Eduardo Dantas de Menezes, agradeço por ter disponibilizado o Analisador e Etiquetador Morfológico desenvolvido por ele em sua dissertação de mestrado.

E, finalizando, estou convicta de que sou privilegiada por pertencer a uma família muito especial, que sempre esteve me incentivando e apoiando em todos os sentidos, e também suportando a minha "ausência". Muitíssimo obrigada, de todo coração, ao meu marido Paulo, aos meus filhos Eliel e Samuel, aos meus pais Yukio e Paulina, à minha sogra Ai, às minhas cunhadas Noemi e Marta, à minha irmã Tiemi, e à minha avó Glorinha.

# RESUMO

Este trabalho tem como objetivo principal mostrar a viabilidade da utilização de Formalismos Adaptativos como modelo de representação de informações no processamento de linguagem natural.

Após apresentar uma visão geral dos formalismos mais utilizados na análise sintática de sentenças de linguagem natural, os seguintes formalismos: Gramática Livre de Contexo Estendida (*Augmented Context-Free Grammar)*, Rede de Transições Aumentadas (ATN – *Augmented Transition Network*), Gramática de Cláusulas Definidas (DCG – *Definite Clause Grammar*) e Gramática Baseada em Unificação ou Restrição (*Unification-Based or Constraint-Based Grammar*) são apresentados e comparados entre si e com os novos modelos propostos, genericamente denominados Formalismos Adaptativos.

Para verificar a viabilidade da utilização dos formalismos adaptativos, no procedimento de análise sintática de linguagem natural, esta dissertação apresenta propostas de algoritmos de mapeamento do formalismo ATN e da Gramática Baseada em Restrição para um Formalismo Adaptativo equivalente.

Finalizando, este trabalho também apresenta um caminho para a obtenção de uma gramática simplificada da estrutura superficial da língua portuguesa, expressa através de Formalismos Adaptativos.

# ABSTRACT

This work is intended to show that the Adaptive Formalisms may be used as a model of natural language processing knowledge representation.

After overviewing the most used formalisms for the syntactical analysis of sentences in natural language sentences, the following formalisms: *Augmented Context-Free Grammar*, *Augmented Transition Network (ATN), Definite Clause Grammar (DCG) and Unification-Based or Constraint-Based Grammar* are presented and compared to each other and to the new proposed models, generically known as Adaptive Formalisms.

In order to verify the factibility of Adaptive Formalisms in natural language syntactic analysis, this work proposes algorithms that map ATN and Constraint-Based Grammars into some equivalent Adaptive Formalism.

Finally, this work also suggests a way for obtaining a formal adaptive specification of the superficial structure of a simple subset of Portuguese language.

# SUMÁRIO

# Lista de Figuras

# 1. INTRODUÇÃO

Este capítulo apresenta de forma resumida a idéia proposta neste trabalho, assim como a sua motivação e a estrutura do texto desta dissertação.

Este trabalho tem como objetivo verificar a viabilidade do emprego dos formalismos adaptativos no processamento de linguagem natural, mais especificamente, na área de análise sintática e representação de informações da linguagem natural. Tem como principal motivação mostrar a viabilidade prática da utilização do formalismo adaptativo, como ferramenta de processamento de linguagens naturais, em particular, da língua portuguesa.

## 1.1. Apresentação

O processamento de linguagem natural é uma área complexa da Inteligência Artificial que tem sido objeto de pesquisas há várias décadas. É um campo que está intimamente ligado a algumas áreas fora do escopo da ciência da computação, tais como a lingüística e a psicologia.

Seu objetivo é conseguir produzir programas de computador capazes de "entender", ou seja, analisar e interpretar a língua humana, processando-a e gerando uma resposta, como ocorre, por exemplo, em sistemas de respostas a perguntas, sistemas de tradução automática ou sistemas de criação de sumários de texto. Atualmente, com a crescente expansão da Internet e da *World Wide Web*, uma aplicação interessante e de grande utilidade do processamento de linguagem natural é

a mineração de informações. Existem inúmeros mecanismos de busca disponíveis na Internet, que, na sua quase totalidade, restringem-se à busca através de palavras-chave e não pelo significado.

As pesquisas na área de processamento de linguagem natural levam em consideração tanto a linguagem falada quanto a escrita. O processamento de texto escrito envolve conhecimentos léxicos, sintáticos e semânticos da linguagem, como também informações necessárias sobre o mundo real [Rich-93]. O processamento da linguagem falada é mais complexo, pois além das informações necessárias ao processamento do texto escrito, são exigidos conhecimentos sobre fonologia e informações que permitam lidar com as ambigüidades que surgem na fala.

Para o processamento de uma linguagem natural, primeiramente é preciso que haja alguma forma de se processar a entrada e a saída das informações no computador, bem como um meio de codificá-la internamente ao processador.

Entre a entrada e a saída das informações, o processamento da sentença envolve as seguintes operações [Rich-93]: análise morfológica, análise sintática, análise semântica e análise pragmática. A análise morfológica procura atribuir uma classificação morfológica a cada palavra da sentença (como por exemplo, artigo, substantivo, verbo, etc.). A análise sintática procura identificar os relacionamentos sintáticos entre as seqüências lineares de palavras, produzindo, assim, a estrutura sintática da sentença (identifica, por exemplo, qual o sujeito, o predicado, o objeto direto, etc). A análise semântica procura mapear a estrutura sintática para o domínio da aplicação, fazendo com que a estrutura ganhe um significado. A análise pragmática procura reinterpretar a estrutura que representa o que foi dito, para determinar o que realmente se quis dizer.

Cada uma dessas operações é essencial e imprescindível para o bom processamento da linguagem natural, servindo de base para a operação seguinte. Este trabalho concentra-se principalmente na segunda operação: a análise sintática da linguagem natural.

Um dos principais formalismos de análise sintática e de representação das estruturas de linguagem natural é o modelo *Augmented Transition Network* (Rede de Transição Aumentada), desenvolvido por William Woods, em 1970 [Woods-70]. Pela sua simplicidade e flexibilidade, foi facilmente aceito, tornando-se, na década

de 70 e em grande parte da década de 80, o formalismo dominante na implementação de sistemas de linguagem natural [Bates-93].

Outro formalismo poderoso e claro para descrição de linguagens é a *Definite Clause Grammar* (Gramática de Cláusulas Definidas), originalmente descrito por Colmerauer, em 1975, baseado no método de expressar gramáticas através de lógica, desenvolvido por Colmerauer e Kowalski [Pereira-80].

Atualmente, os formalismos mais utilizados para a análise sintática e descrição de linguagens baseiam-se na unificação e no uso de restrições e são conhecidos como Gramáticas baseadas em unificação ou Gramáticas baseadas em restrições. Existem várias variantes e as mais conhecidas são PATR, PATR-II, LFG (*Lexical Functional Grammar* – Gramática Léxica Funcional), GPSG (*Generalized Phrase Structure Grammar* – Gramática de Estrutura de Frase Generalizada) e HPSG (*Head-driven Phrase Structure Grammar* – Gramática de Estrutura de Frase Direcionada Pelo Núcleo).

Neste trabalho, o formalismo dessas notações: *Augmented Transition Network*, *Definite Clause Grammar* e Gramáticas Baseadas em Unificação ou Gramáticas Baseadas em Restrições são apresentados e discutidos.

Além dos formalismos mencionados, esta dissertação também apresenta e discute os formalismos adaptativos, que foram criados para solucionar, principalmente, algumas deficiências existentes no projeto de reconhecedores sintáticos de linguagens de programação.

Primeiramente, os Autômatos Adaptativos surgiram a partir, entre outras, da necessidade de eliminar alguns desvios conceituais existentes na maioria dos compiladores de linguagens de programação dirigidos por sintaxe. Problemas como o de dependência de contexto, estruturas de blocos e escopo das variáveis, tratamento de macros, consistência de uso dos tipos das variáveis, entre outros, deveriam ser considerados no nível sintático e não no semântico.

Como tais compiladores se baseiam em máquina de estados e pilhas, surgiu então o conceito de Autômato Adaptativo [José-93] [José-94]. O Autômato Adaptativo é um modelo de máquina de estados que se caracteriza por sua adaptabilidade, isto é, sua capacidade de se auto-modificar à medida que vai reconhecendo a cadeia de entrada. Essa característica dinâmica o torna capaz de

reconhecer linguagens sensíveis ao contexto, podendo tratar os problemas acima mencionados no nível sintático, como é desejado.

Dando continuidade à linha de pesquisa dos autômatos adaptativos, [Iwai-00] introduz o conceito do formalismo das gramáticas adaptativas. A gramática adaptativa é um dispositivo gerador, do tipo sensível ao contexto, correspondente ao formalismo implementado pelos autômatos adaptativos e mais adequado para as etapas de projeto e estruturação de linguagens.

A gramática adaptativa, assim como o autômato adaptativo, caracteriza-se por permitir a sua auto-modificação durante a sua operação. À medida que uma sentença da linguagem vai sendo gerada pela gramática adaptativa, se a regra de produção que está sendo aplicada estiver associada a uma ação adaptativa, essa ação é acionada, resultando em alguma modificação na própria gramática. A execução da ação adaptativa permite consultar, remover e adicionar regras de produção, bem como alterar o conjunto de símbolos não-terminais.

Como os formalismos adaptativos podem potencialmente apresentar poder computacional equivalente ao da máquina de Turing [Iwai-00], podendo, portanto, representar linguagens de qualquer complexidade, intuitivamente, eles também podem ser empregados no processamento de linguagem natural.

## 1.2. Proposta

A proposta deste trabalho é verificar a viabilidade de os formalismos adaptativos serem utilizados como analisadores sintáticos e modelos de representação de informações no processamento de linguagem natural.

## 1.3. Motivação

Este trabalho tem, como principal motivação, mostrar a viabilidade prática da utilização do formalismo adaptativo na área de linguagens naturais, especificamente no que tange à fase da análise sintática, visto que tal formalismo é capaz de reconhecer e representar linguagens sensíveis ao contexto.

Como resultado desse trabalho, apresenta-se uma forma de mapeamento, para o formalismo adaptativo, dos formalismos *Augmented Transition Network* e Gramática Baseada em Unificação ou Restrições.

Como motivação secundária, tem-se a contribuição para o desenvolvimento das pesquisas na área de Processamento de Linguagem Natural, que passará a contar com mais uma ferramenta de apoio, baseada nos formalismos adaptativos.

## 1.4. Estrutura da Dissertação

O primeiro capítulo desta dissertação apresenta de maneira sucinta a proposta e as motivações da pesquisa e descreve a organização do seu texto.

O segundo capítulo apresenta uma visão geral dos formalismos de representação de linguagem natural. Primeiramente, discorre sobre os conceitos de representação de linguagens e tratamento sintático e expõe, em seguida, o histórico dos formalismos mais conhecidos utilizados na representação de linguagem natural, tecendo finalmente comentários comparativos entre esses formalismos. Apresenta, também, algumas considerações superficiais sobre o processamento computacional da língua portuguesa.

No terceiro capítulo, são introduzidos vários fundamentos conceituais sobre os quais esta dissertação foi desenvolvida. Primeiramente, alguns conceitos básicos acerca de analisadores morfológicos em geral e acerca do etiquetador morfológico utilizado na parte experimental desta dissertação. Em seguida, são apresentados e comparados os formalismos das principais notações relacionadas com os métodos apresentados nesta pesquisa: *Augmented Context-Free Grammar*, *Augmented*

*Transition Network*, *Definite Clause Grammar,* Gramática Baseada em Unificação ou Gramática Baseada em Restrições. Os formalismos adaptativos (Autômato Adaptativo e Gramática Adaptativa) são então detalhados e comparados com os demais modelos de representação de linguagem mencionados anteriormente.

No quarto capítulo, encontra-se o desenvolvimento da proposta desta dissertação. A verificação de que o formalismo adaptativo também pode ser utilizado como ferramenta de análise sintática e modelo de representação de informação da linguagem natural processa-se em duas etapas: primeiramente, mostra-se como é possível mapear qualquer *Augmented Transition Network*, formalismo tradicionalmente utilizado para esta finalidade, para um Autômato Adaptativo, e depois, verifica-se analogamente que é possível mapear a gramática baseada em unificação ou gramática baseada em restrições para um Formalismo Adaptativo. Dessa forma, constata-se que o Formalismo Adaptativo pode ser utilizado como ferramenta para análise sintática e representação das estruturas da linguagem natural.

O quinto capítulo expõe os resultados experimentais da dissertação, descrevendo os experimentos efetuados, e sua avaliação.

No sexto capítulo, encontra-se uma avaliação geral desta pesquisa, assim como considerações finais a respeito de suas contribuições e futuros trabalhos que possam dar continuidade a essa dissertação.

Após o sexto capítulo, são apresentadas as referências bibliográficas utilizadas nesta pesquisa e citadas no decorrer do texto.

Finalizando, o Apêndice A relaciona as regras da especificação GPSG da gramática da estrutura superficial da língua portuguesa, utilizadas nesta dissertação.

# 2. VISÃO GERAL DOS FORMALISMOS DE REPRESENTAÇÃO DE LINGUAGEM NATURAL

Este capítulo apresenta os conceitos de representação de linguagens e de tratamento sintático e, em seguida, um histórico e um comparativo dos formalismos mais utilizados na representação de linguagem natural. Apresenta, também, algumas considerações superficiais sobre o processamento computacional da língua portuguesa.

## 2.1. Representação de Linguagens

O estudo da representação de linguagens está intimamente ligado à teoria lingüística. A lingüística basicamente é o estudo da linguagem humana, preocupando-se com a maneira como a linguagem funciona e como é utilizada.

Uma *linguagem*, do ponto de vista formal, nada mais é do que um conjunto, em geral infinitamente grande, de sentenças [DeRoeck-83].

Uma *sentença* é uma *cadeia* formada a partir de um vocabulário finito de *símbolos*.

Em princípio, para se descrever uma determinada linguagem, bastaria listar todas as sentenças válidas da mesma (enumeração) [DeRoeck-83]. Como o número de sentenças em uma linguagem natural é infinitamente numeroso, a teoria

lingüística passou a se preocupar com as regras ou critérios que uma sentença deve obedecer, de modo que seja válida para a linguagem que estiver sendo descrita. Como esse conjunto de regras é finito, pode ser utilizado para descrever uma linguagem. O conjunto de tais regras forma uma gramática.

Dessa maneira, a cadeia de símbolos que forma uma sentença deve ser constituída em conformidade com as *regras de formação* definidas pela *gramática* [DeRoeck-83].

Os formalismos existentes de representação de linguagem baseiam-se, portanto, ou na gramática da linguagem, através da qual é possível gerar sentenças válidas da linguagem em questão, ou em dispositivos reconhecedores da linguagem, chamados de máquinas de estados ou autômatos, que contêm o conjunto de regras de aceitação de cadeias dessa linguagem, verificando assim, se uma determinada cadeia de símbolos é ou não uma sentença válida.

Antes de discorrer sobre o histórico dos formalismos de representação de linguagem mais utilizados, o próximo item apresenta o conceito de tratamento sintático de linguagem natural.

## 2.2. Tratamento Sintático

O processamento de uma sentença de linguagem natural compreende, usualmente, as seguintes operações básicas [Allen-94] [Rich-93]:

- análise morfológica – cada palavra da sentença é analisada isoladamente, identificando-se como ela é formada morfologicamente e atribuindo-se a ela uma categoria morfológica. Por exemplo: a palavra ‘superfaturamento’ é formada morfologicamente pela raiz ‘faturar’, pelo prefixo ‘super’ e pelo sufixo ‘mento’ e é classificada como sendo um substantivo masculino singular. O conceito de análise morfológica é melhor apresentado na seção 3.1 desta dissertação.
- análise sintática – seqüências lineares de palavras da sentença são analisadas procurando-se identificar os relacionamentos sintáticos existentes entre elas,

produzindo-se a estrutura sintática da sentença. Por exemplo, na sentença 'O menino comeu o chocolate.', pode-se identificar que o sujeito é 'O menino', o predicado é 'comeu o chocolate', o objeto direto é 'o chocolate'. A estrutura sintática hierárquica correspondente é ilustrada na Figura 1. Cadeias que não obedeçam às regras gramaticais da linguagem são consideradas sintaticamente inválidas. Um exemplo de cadeia sintaticamente inválida: 'Menino o chocolate comeu.'

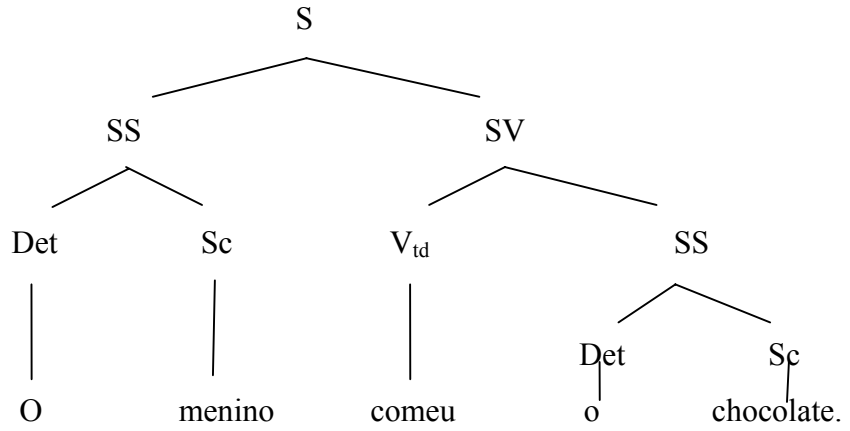



***Figura 1 - Estrutura sintática correspondente à sentença 'O menino comeu o chocolate.' (S – Sentença, SS – Sintagma Substantivo, SV – Sintagma Verbal, Det – Determinante, Sc – Substantivo comum, $V_{td}$ – Verbo transitivo direto).***

- análise semântica – as estruturas sintáticas criadas pelo analisador sintático são mapeadas para objetos no domínio da aplicação, fazendo, assim, com que as estruturas sintáticas ganhem um significado. As estruturas que não puderem ser mapeadas são rejeitadas como semanticamente inválidas.
- integração de discurso – relaciona o significado de uma sentença isolada com as anteriores e as posteriores. Essa fase é essencial para a interpretação de pronomes e de aspectos temporais. Seja, por exemplo: a sentença: 'Ele disse que viria mais tarde.' Sem a interpretação das sentenças anteriores, não é possível saber a quem se refere o pronome 'Ele', nem tampouco o que significa 'mais tarde'.
- análise pragmática – a representação resultante das atividades anteriores é reinterpretada, procurando-se determinar o que realmente se quis dizer.

O tratamento sintático, assim, é uma das etapas do processamento da linguagem natural, destinada a levantar a estrutura do relacionamento entre as partes da sentença, bem como classificá-las, disto resultando a obtenção da *"surface structure"*. A análise semântica, etapa posterior à análise sintática opera sobre os componentes da sentença, identificados previamente no tratamento sintático. Assim, a análise sintática prévia pode reduzir consideravelmente a complexidade global do sistema [Rich-93]. Da análise semântica, resulta o conhecimento da *"deep structure"* da sentença. Os conceitos de *"surface structure*" e *"deep structure"* são explicados no item 2.3 desta dissertação.

Os dois componentes principais utilizados na análise sintática, em grande parte dos sistemas, têm sido:

- a gramática – uma representação declarativa dos fatos sintáticos da linguagem.

  Um exemplo de conjunto de regras muitíssimo simplificado para definir um subconjunto trivial da língua portuguesa, e que pode ter gerado a sentença correspondente ao exemplo ilustrado anteriormente, na Figura 1, é:

  $S \rightarrow SS\ SV,$

  $SS \rightarrow Det\ Sc,$

  $SV \rightarrow V_{td}\ SS,$

  $Sc \rightarrow menino,$

  $Sc \rightarrow chocolate,$

  $V_{td} \rightarrow comeu,$

  $Det \rightarrow o$

  O conceito formal de gramática é apresentado na seção 3.2.1.

- reconhecedor – um procedimento que compara as regras de produção da gramática com as sentenças de entrada, verificando se são sintaticamente válidas. Acrescentando-se, a um reconhecedor, procedimentos de construção das árvores que representam as estruturas sintáticas correspondentes, obtém-se um analisador sintático ("*parser*").

A análise sintática de uma sentença pode ser efetuada de diversas maneiras, sendo as duas seguintes as mais utilizadas e referenciadas na literatura [Allen-94] [Rich-93]:

- análise *top-down* – parte-se do símbolo inicial da gramática e aplica-se uma regra de produção, reescrevendo-o como uma seqüência de símbolos. Aplica-se uma nova regra de produção sobre um desses símbolos, reescrevendo-o conforme indicado pela regra. Prossegue-se, assim, aplicando-se sucessivamente regras de produção da gramática até que a seqüência de símbolos resultantes coincida com a da sentença que está sendo analisada.
- análise *bottom-up* – parte-se da sentença em análise e aplicam-se as regras de produção no sentido inverso, procurando-se coincidências entre os símbolos da sentença e os símbolos das partes direitas das regras. Quando coincidirem, os símbolos são substituídos pela parte esquerda da regra e prossegue-se, assim, aplicando-se sucessivamente as regras de produção até que se obtenha o símbolo inicial da gramática.

Qualquer que seja a forma de análise escolhida (*top-down* ou *bottom-up*), a estrutura gramatical da sentença pode ir sendo produzida à medida que as regras são aplicadas.

Um problema que existe na análise sintática é quando há mais de uma regra aplicável em uma dada situação. Nesse caso, pode-se optar por 2 alternativas:

- aplicar em paralelo todas as regras possíveis, o que pode ser ineficiente já que se realizariam muitas tarefas desnecessárias.
- aplicar uma regra de cada vez, prosseguindo até finalizar a análise ou até detectar alguma falha, e então retroceder até o ponto em que foram detectadas várias possibilidades de continuação, para então escolher uma outra regra para prosseguir [Rich-93]. Esta técnica exaustiva é conhecida como mecanismo de retrocesso (*"backtracking"*).

Uma técnica para melhorar a eficiência do processo de análise sintática no caso de haver retrocessos durante a análise é a utilização de mapas (*charts*). A técnica consiste em se armazenar, em uma estrutura conhecida como mapa, os resultados parciais da análise durante o procedimento, de forma que tais informações possam ser reutilizadas mesmo que haja um retrocesso no processo de análise e se assuma um outro caminho alternativo [Allen-94]. No mapa também são armazenadas as regras que ainda foram aplicadas apenas parcialmente. Assim, há uma economia de trabalho, pois um constituinte, que é uma parte da sentença com significado sintático

(como, por exemplo, um sintagma substantivo), não precisa ser analisado mais de uma vez em caso de retrocesso.

Um outro problema grave que existe na análise sintática de linguagem natural é o tratamento da ambigüidade. A ambigüidade pode ocorrer nos diversos níveis do processamento da linguagem natural: morfológico, sintático e semântico.

No nível morfológico [Bick-00], tem-se:

- ambigüidade léxica:
  - formas básicas diferentes, categorias iguais. Exemplo: a palavra 'foi', que tanto pode ser o passado do verbo 'ir' como do verbo 'ser' (formas básicas: 'ir' e 'ser', categorias: verbo, passado, terceira pessoa do singular).
  - forma básica igual, diferença no inventário e no tipo da categoria. Exemplo: a palavra 'complementar', que pode ser um verbo ou um adjetivo (forma básica: 'complementar', categorias: verbo e adjetivo).
- ambigüidade flexiva:
  - forma básica igual, diferença em uma ou mais categorias da formação da palavra. Exemplo: a palavra 'amamos', que tanto pode ser presente como passado (forma básica: 'amar', categorias: presente e passado).
- ambigüidade léxica-flexiva:
  - formas básicas diferentes, categorias de lexemas e de formação de palavras diferentes. Exemplo: a palavra 'busca', que tanto pode ser um substantivo como um verbo (formas básicas: 'busca' e 'buscar', categorias: substantivo, feminino singular e verbo, presente, terceira pessoa do singular, respectivamente).

No nível sintático, [Bick-00] lista os tipos de ambigüidade:

- identidade do constituinte:
  - ambigüidade da forma sintática: não se sabe qual parte da sentença é modificada por um certo constituinte. Exemplo: 'O homem com a bicicleta da China.' (o homem ou a bicicleta é da China?)
  - ambigüidade da função sintática. Exemplo: 'um homem que ama toda mulher' ('homem' pode ser o sujeito do verbo amar ou o objeto direto, e assim, a função sintática de 'que' é ambígua).
- coesão:
  - anáfora, que é uma relação de co-referência, entre uma palavra e a palavra à qual ela se refere. No exemplo: 'Amava sua irmã.' (a irmã é de quem fala ou é do ouvinte?), não se pode afirmar com certeza a quem se refere o pronome 'sua'.
  - coordenação, que é uma relação entre duas palavras ou expressões. Surge uma ambigüidade quando a coordenação é seguida de um constituinte que pode se referir apenas à última palavra ou a todas as palavras relacionadas. – Exemplo: 'homens e mulheres no Brasil' (homens e mulheres no Brasil ou apenas as mulheres no Brasil?)

[Bick-00] também classifica os seguintes tipos de ambigüidade semântica:

- léxica: quando uma palavra apresenta as mesmas características morfológicas e sintáticas, mas apresenta diversidade de significados (polissemia). Exemplo: 'fato', que é classificada como substantivo, singular, masculino, mas possui três significados: acontecimento, terno ou rebanho de cabras.
- poliléxico: quando uma palavra tem significados diferentes, dependendo da sua flexão, como por exemplo: 'ter boas razões para' e 'ter razão'.
- escopo: quando a interpretação de uma palavra depende do escopo. Por exemplo: 'Não compre três garrafas de vinho, compre quatro!' ou 'Não compre três garrafas de vinho, compre licor!' (nessas sentenças, não é imediatamente claro se a negação se aplica ao número de garrafas ou ao tipo de bebida).

- papel temático: por exemplo, em ‘o sacrifício da moça’, não é claro se a moça é o agente do sacrifício ou se é o objeto.

Para tratar a ambigüidade, há duas técnicas principais:

- análise probabilística [Bick-00] [Allen-94] [Charniak-93] – baseia-se na análise estatística das regras da gramática ou de um corpus, que é um conjunto de documentos, dados e informações em um determinado idioma, utilizado justamente para ensaios lingüísticos e computacionais. Grande parte das implementações de modelos probabilísticos utiliza os Modelos Ocultos de Markov (HMM – *Hidden Markov Models*), autômatos finitos cujas transições estão associadas a probabilidades.
- análise baseada em regras [Bick-00] – uma técnica desse tipo para resolver os problemas de ambigüidade nos diversos níveis do processamento lingüístico é a Gramática de Restrição (CG – *Constraint Grammar*), desenvolvida por Fred Karlsson, da Universidade de Helsinki [Bick-00] [Bick-98] [Bick-96] [Leech-97]. Essa técnica apresenta uma boa robustez e eficiência no tratamento da ambigüidade e consiste em estabelecer regras que possibilitam a escolha, em um determinado contexto, de uma dada interpretação, descartando-se as demais.

Há também sistemas híbridos, que utilizam as duas técnicas mencionadas acima para tratar a ambigüidade. [Bick-00] e [Chanod-95] apresentam comparações entre os métodos estatísticos e a Gramática de Restrição no tratamento de ambigüidade morfológica, concluindo que a Gramática de Restrição se mostrou mais eficiente. O sistema ENGCG, baseado na Gramática de Restrição, e um sistema etiquetador estatístico treinado com o corpus Brown foram testados com textos de 50.000 palavras, contendo trechos jornalísticos, científicos e de manual técnico, e, para uma razão de 1,07 etiqueta por palavra, o sistema ENGCG apresentou uma taxa de erro de 0,1%, enquanto o sistema estatístico apresentou uma taxa de erro de 2,8% [Bick-00].

Para representar e analisar sintaticamente a linguagem natural, diversos formalismos foram criados ao longo das últimas décadas. O próximo item relata o histórico dos formalismos mais utilizados, e a seção 3.2 deste trabalho apresenta alguns formalismos com maiores detalhes.

## 2.3. Histórico dos Formalismos Mais Utilizados

Noam Chomsky, lingüista, formulou a partir de 1957, as primeiras gramáticas geradoras, a gramática linear, a gramática de estrutura de frase (*phrase structure grammar*), e em seguida a gramática geradora transformacional (*transformational generative grammar*) [Harris-85]. Tais gramáticas são capazes não só de gerar uma sentença como também permitem construir para ela uma descrição estrutural ou análise gramatical. Na gramática geradora transformacional, além das regras geradoras da sentença, existem as regras transformacionais, que permitem transformar, por exemplo, uma sentença da voz ativa para passiva ou uma sentença afirmativa em interrogativa. A estrutura representada apenas pelas regras de estrutura de frase é denominada *deep structure*, chamando-se *surface structure* a forma como será realmente utilizada na fala ou escrita.

O estudo da descrição de linguagens prosseguiu adicionando-se componentes semânticos à gramática de Chomsky, para que se pudesse interpretar o significado da sentença [Harris-85]. Novas teorias surgiram, tais como a Teoria Semântica de Jerrold J. Katz e Jerry A. Fodor, em 1964, e a Gramática de Casos (*Case Grammar*) de Charles J. Fillmore, em 1967. Chomsky também continuou a desenvolver o seu conceito de organização de gramática, absorvendo as idéias da interpretação semântica dessas novas teorias, procurando resolver as falhas existentes em suas primeiras definições de gramáticas.

Um caso particular da classe de gramáticas de estrutura de frases (*phrase structure grammars*), definido por Chomsky, é a gramática livre de contexto, introduzida por Chomsky em 1957 [Allen-94], largamente utilizada para representar linguagens de programação, devido à sua simplicidade e clareza. Para poder representar linguagens naturais, que normalmente apresentam problemas de concordância entre outros, o formalismo da gramática livre de contexto foi estendido para que aos seus constituintes possam ser associadas características (*features*). Tal formalismo é conhecido como gramática livre de contexto estendida, semelhante à gramática de atributos, introduzida por Knuth em 1968, para o tratamento de

linguagens de programação, e é apresentado com maiores detalhes na seção 3.2.1 desta dissertação.

Outra forma de representação das estruturas de linguagem natural é o modelo conhecido como redes de transição (*transition networks*), baseado na aplicação das noções matemáticas de teoria de grafos e autômatos finitos ao estudo das gramáticas. Dentro dessa classe, destaca-se o modelo *Augmented Transition Network* (Rede de Transições Aumentada), desenvolvido por William Woods, em 1970, embora seu artigo [Woods-70] referencie trabalhos anteriores na mesma linha de desenvolvimento por Conway (1963), Thorne, Bratley e Dewar (1968), e Bobrow e Fraser (1969). Pela sua simplicidade e flexibilidade, foi facilmente aceito, tornando-se, na década de 70 e em grande parte da década de 80, o formalismo dominante na implementação de sistemas de linguagem natural [Bates-93], e, segundo Pereira e Warren [Pereira-80], era considerado em 1980 o "estado da arte" no formalismo para análise de linguagem natural. Tal formalismo é estudado com maiores detalhes na seção 3.2.2 desta dissertação.

Outro formalismo poderoso e claro para descrição de linguagens é a *Definite Clause Grammar* (Gramática de Cláusulas Definidas), originalmente descrito por Colmerauer, em 1975, baseado no método de expressar gramáticas através de lógica, desenvolvido por Colmerauer e Kowalski [Pereira-80]. O formalismo DCG também é apresentado com maiores detalhes na seção 3.2.3 deste trabalho.

Na década de 80, surgiram vários formalismos novos de gramáticas para linguagem natural. O conceito de estruturas de características (*feature structures*) passou a ser generalizado de tal maneira que tornou desnecessária a gramática livre de contexto, passando a gramática a ser especificada como um conjunto de restrições entre as estruturas de características [Allen-94]. A maioria desses formalismos utiliza a unificação como principal operação para manipular essas estruturas de características, criando constituintes maiores a partir de constituintes menores [Wedekind-97] [Knight-89]. Por isso, tais formalismos são conhecidos como formalismos baseados em unificação.

No final da década de 80, surgiu o conceito de formalismo baseado em restrições (*constraint-based formalism*), referindo-se a uma classe de formalismos caracterizados pelo uso particular de construções declarativas, enfatizando a

modularização e a manipulação de informações e restrições para representar a linguagem natural [Shieber-92]. Como os formalismos classificados como baseados em unificação utilizavam informações e restrições como base para suas operações, eles passaram então a ser classificados também como formalismos baseados em restrições.

[Shieber-86] apresenta os principais formalismos que se baseiam na unificação, e [Shieber-92] cita os mesmos formalismos, classificando-os como baseados em restrições, incluindo o já citado DCG. De acordo com Shieber, inicialmente, Bresnan e Kaplan partiram do conceito ATN e desenvolveram o conceito de gramática léxica-funcional (LFG – *lexical-functional grammar*) [Arnold-00] em 1982. Simultaneamente, Kay desenvolveu o formalismo conhecido como gramática de unificação funcional (FUG – *functional unification grammar*) em 1983, utilizando estruturas funcionais (*functional structures*) [Shieber-86]. Independentemente, Pereira e Warren criaram a gramática de cláusulas definidas (DCG) já citada anteriormente, dando origem a outros formalismos gramaticais [Shieber-86], tais como: *extraposition*, *slot* e *gapping grammars*. Uma outra linha de pesquisa em paralelo cresceu do trabalho de análises lingüísticas não transfomacionais, a partir da qual Gazdar desenvolveu a gramática de estrutura de frase generalizada (GPSG – *Generalized Phrase Structure Grammar*) [Gazdar et al.-85], em 1982. A partir de desenvolvimento posterior da GPSG, Pollard e Sag, em 1987, desenvolveram o formalismo denominado gramática de estrutura de frase direcionada pelo núcleo (HPSG – *head-driven phrase-structure grammar*) [Shieber-86] [Ravera-90].

Baseado inicialmente em GPSG e recebendo influência de outros formalismos como o FUG e o DCG, Shieber e Pereira desenvolveram em 1984 o formalismo PATR-II [Shieber-86, Shieber-88], que pode ser considerado um dos mais simples formalismos baseados em restrições.

[Uszkoreit-95] também discorre sobre os formalismos das gramáticas baseadas em restrições e reconhece que é uma classe de formalismos bastante avançada e muito difundida. Segundo [Uszkoreit-95], entre os formalismos baseados em restrições, os mais utilizados são: gramática de unificação funcional (FUG), gramática de estrutura de frase direcionada pelo núcleo (HPSG), gramática léxica

funcional (LFG), gramática de unificação categorial (CUG – *Categorial Unification Grammar*) e gramática de adjunção de árvore (TAG – *Tree Adjunction Grammar*).

No final da década de 80 e início da década de 90, começaram a surgir os analisadores sintáticos probabilísticos [Leech-97], para tratar o problema da ambigüidade sintática. [Allen-94] e [Charniak-93] descrevem uma gramática livre de contexto probabilística, na qual, no caso de haver mais de uma regra referente a um determinado símbolo, probabilidades são associadas a elas, de forma que, quando da aplicação das regras, seja possível escolher a regra de maior probabilidade.

Paralelamente ao desenvolvimento do conceito de gramáticas de estrutura de frase, de Chomsky, desenvolveu-se o conceito da Gramática de Dependência (DG – *Dependency Grammar*) [Hudson-01] [Barbero-97] [Lombardo-91], na qual a relação básica é a dependência entre as palavras e não entre os constituintes (sintagmas) e suas partes. Existem diversas variantes da Gramática de Dependência [Bröker-99]: Descrição Geradora Funcional (*Functional Generative Description*), Gramática da Palavra (*Word Grammar*), Gramática Funcional (*Functional Grammar*) e Gramática de Ligação (*Link Grammar*). Maxwell propõe a Gramática de Unificação de Dependência (*Dependency Unification Grammar*), utilizando o conceito de dependência na Gramática de Unificação [Bröker-99].

Existem ainda vários formalismos de representação de linguagem natural que não estão citados neste histórico, pois o objetivo é apresentar as principais linhas de pensamento que norteiam os diversos formalismos existentes. O próximo item apresenta alguns comentários comparativos entre os formalismos mencionados neste histórico.

## 2.4. Comentários Comparativos sobre os Formalismos

São descritos, a seguir, alguns comentários comparativos sobre os vários formalismos analisados anteriormente quanto a aspectos gerais de suas características. Uma comparação mais detalhada entre os formalismos relacionados

com os métodos apresentados no desenvolvimento desta pesquisa encontra-se descrita no Capítulo 3 desta dissertação.

Pelo histórico apresentado no item anterior, percebe-se que as idéias de Chomsky tiveram grande influência no desenvolvimento de grande parte dos formalismos citados. A gramática de estrutura de frase de Chomsky, que especifica que as relações básicas numa sentença existem entre os seus constituintes ou sintagmas, é a precursora de diversos formalismos, tais como: gramática livre de contexto, DCG, GPSG, HPSG. A essa linha de raciocínio, opõe-se a linha das Gramáticas de Dependência, pois nestas as relações se referem às palavras individuais e não aos constituintes estruturais da frase.

A gramática livre de contexto e suas variantes (estendida [Allen-94], probabilística [Charniak-93], lexicalizada [Barbosa-00]) constituem uma classe de formalismos que têm sido largamente utilizados na lingüística computacional para a representação e a análise de linguagens naturais [Barbosa-00]. Apresenta a vantagem de ser um modelo tradicionalmente utilizado na ciência da computação, não apenas para linguagens naturais, como também para linguagens de programação e outras aplicações, justamente por ser um formalismo claro, simples e que conta com a existência de algoritmos eficientes para reconhecê-lo e analisá-lo [Gazdar-94]. Tem a desvantagem de não capturar todas as nuanças de linguagens mais complexas, como é o caso das linguagens naturais.

O formalismo ATN também sofreu a influência das idéias da gramática geradora transformacional de Chomsky, uma vez que foi projetado como uma linguagem de programação para a escrita de analisadores. Estes, ao contrário dos desenvolvidos a partir das gramáticas transformacionais, possibilitam desfazer as transformações da sentença durante o processamento [Gazdar-94]. No entanto, [Gazdar-94] relata que, apesar desta vantagem, o ATN não foi muito bem sucedido na tarefa de representar as transformações. Segundo [Gazdar-94], a principal contribuição dos ATNs foi introduzir o conceito de registradores, atribuições e testes nos modelos de redes, possibilitando, assim, que os ATNs representassem não só os aspectos livres de contexto da linguagem, como também os sensíveis ao contexto. Na literatura, há também vários textos ([Bates-78] [Allen-94] [Gazdar-94]) que atribuem ao ATN a vantagem de ser um formalismo simples e flexível. [Gazdar-94] alega que

o formalismo ATN começou a declinar, no final da década de 80, porque suas extensões prejudicam a natureza declarativa do formalismo e porque a análise sintática baseada em ATN é limitada nas estratégias de busca que pode empregar. [Bates-93] reconhece que houve uma tendência, a partir do final dos anos 80, para se adotar formalismos declarativos, em detrimento dos procedimentais, como o ATN.

No formalismo DCG, as informações são armazenadas em estruturas chamadas termos, podendo tal formalismo ser visto como um sistema baseado em restrições, já que utiliza informações do tipo característica-valor, sendo que as características correspondem a posições de argumentos [Shieber-86]. Apresenta a desvantagem de o número e a ordem dos argumentos serem importantes quando da utilização das informações. Sua vantagem é a de poder ser executado diretamente como um código em linguagem Prolog e, uma vez que existem compiladores Prolog eficientes, das gramáticas DCG podem resultar analisadores sintáticos rápidos. Além disso, é facilmente integrável com outros programas escritos em Prolog.

A gramática baseada em restrições refere-se a uma classe de formalismos que utilizam construções declarativas para representar as informações da linguagem natural e realizam a análise através da manipulação dessas informações por intermédio de alguma operação, como é o caso da unificação. Normalmente, as estruturas de informações relacionam pares do tipo atributo-valor, sendo que o atributo se refere em geral a alguma característica lingüística. Dentre os formalismos desta classe, citam-se a gramática léxica funcional (LFG – *lexical-functional grammar*), a gramática de unificação funcional (FUG – *functional unification grammar*), a gramática de estrutura de frase generalizada (GPSG – *generalized phrase structure grammar*), a gramática de estrutura de frase direcionada pelo núcleo (HPSG – *head-driven phrase-structure grammar*), que serão comparados a seguir.

No formalismo LFG, desenvolvido por Bresnan e Kaplan [Winograd-83], as estruturas de informações são chamadas *f-structures* ou estruturas funcionais (*functional structures*), e consistem de um sistema de característica-valor [Shieber-86]. Difere da gramática transformacional por apresentar um único nível de estrutura sintática [Arnold-00], representada pela *c-structure* ou estrutura constituinte (*constituent-structure*), que é semelhante a uma árvore de estrutura de frase livre de contexto. Além disso, há as regras (semelhantes às regras da gramática livre de

contexto), associadas a equações que denotam as informações a serem unificadas. A maioria das informações é codificada no léxico [Allen-94], que incorpora informações sobre a função gramatical do item léxico, bem como equações a serem satisfeitas para a devida utilização de tal item. A *f-structure* integra e unifica as informações contidas no léxico, na *c-structure* e nas regras, reunindo as informações em um sistema de matrizes. Apresenta as vantagens de tratar o problema da dependência de longa distância de maneira mais elegante do que a gramática transformacional, e de não impor uma ordem no processamento da análise sintática, como acontece no ATN [Winograd-83] . A dependência de longa distância refere-se ao caso em que um constituinte de uma sentença depende de outro constituinte que se encontra distante do mesmo. No exemplo: 'Qual livro o menino gostaria de ler?', 'livro' é o objeto direto do verbo 'ler', mas na sentença, essas duas palavras encontram-se distantes uma da outra, dificultando a análise de sua dependência. O LFG apresenta a desvantagem de não permitir mais do que duas características aninhadas nas regras [Shieber-86].

O formalismo FUG foi projetado por Martin Kay como uma ferramenta lingüística genérica baseada na unificação [Shieber-86]. As estruturas de informação são chamadas estruturas funcionais (*functional structures*) e constituem um sistema generalizado de característica-valor, que especificam uma combinação de características, funções gramaticais, itens léxicos e ordem dos constituintes. As regras são expressas por meio dessas estruturas funcionais, com valores compartilhados para os quais se deseja que haja unificação. Apresenta a vantagem de ser capaz de caracterizar qualquer linguagem recursivamente enumerável [Shieber-86]. Sua desvantagem é a de apresentar alguns problemas técnicos no desenvolvimento de uma gramática, relacionados às suas próprias características [Winograd-83].

No formalismo GPSG, o domínio de informações é um sistema restrito de característica-valor, dito sistema de categorias [Gazdar et al.-85]. As regras não estão associadas a equações, e a unificação é estabelecida através de princípios que governam a combinação das características. Uma árvore de estrutura de frase satisfaz as diversas regras da gramática GPSG se ela também satisfizer tais princípios. O item 3.2.4.2 desta dissertação apresenta uma descrição mais detalhada da GPSG. A GPSG

trata de forma elegante e bem sucedida os fenômenos da coordenação e da dependência de longa distância [Shieber-86]. No entanto, seu poder matemático expressivo é limitado [Gazdar et al.-85].

O formalismo HPSG foi desenvolvido por Pollard e Sag , a partir da GPSG, inspirado também no LFG e no FUG [Shieber-86]. Estende a GPSG, introduzindo as operações de cadeia chamadas *head-wrapping*, utilizadas para tratar a análise de constituintes descontínuos. Apresenta um léxico mais complexo do que a GPSG e a combinação das cadeias e das informações é estabelecida por um algoritmo de aplicação de regra de análise *bottom-up,* tendo, assim, um enfoque mais procedimental. O HPSG remove do sistema de características as restrições que limitavam o poder expressivo da GPSG [Shieber-86], e realiza, em seu léxico, o tratamento da subcategorização e da semântica dos itens léxicos. [Barbero-97], no entanto, relata a existência de problemas no tratamento da subcategorização no HPSG, devido ao fato de a estrutura de dados utilizada ser limitada para uma adequada representação da subcategorização dos itens.

A Gramática de Dependência opõe-se à linha das gramáticas de estruturas de frases, que estuda o relacionamento entre partes da sentença (sintagmas) [Bröker-99]. A Gramática de Dependência baseia-se no conceito de dependência existente entre as palavras da sentença, e não entre as partes da mesma. Assim, a estrutura da sentença é descrita em termos de relações de dependência, que são relações binárias entre as suas palavras [Barbero-97]. A relação de dependência se dá entre uma palavra denominada núcleo e uma outra, dita dependente. Tal relação deve estar associada a uma regra ou restrição que inter-relaciona tais palavras. As vantagens apresentadas pela Gramática de Dependência em relação às gramáticas baseadas em estrutura de frase são: uma descrição de linguagens mais elegante, em que a ordem de palavras não é tão rigorosa, descrições sintáticas mais simples e a possibilidade de se avaliar a dificuldade sintática de maneira simples e precisa. Um argumento desfavorável é o de que a Gramática de Dependência não é capaz de descrever fenômenos recursivos, nem de representar certos fenômenos próprios do paradigma baseado em sintagmas [Bröker-99].

Este item apresentou alguns comentários comparativos entre os diversos formalismos citados no histórico, para que se possa ter uma visão geral dos mesmos.

Os formalismos relacionados com os métodos apresentados nesta pesquisa (*Aumented Context-Free Grammar*, ATN, DCG e GPSG) são descritos com mais detalhes no Capítulo 3 desta dissertação. Maiores detalhes sobre os demais formalismos mencionados no histórico podem ser encontrados nas respectivas referências citadas.

O próximo item apresenta algumas considerações sobre o processamento computacional da língua portuguesa.

## 2.5. Considerações sobre o Processamento Computacional da Língua Portuguesa

Apesar de existirem muitos grupos trabalhando em pesquisas na área de processamento da língua portuguesa, ainda há uma quantidade pequena de trabalhos nesta área, comparando-se com o número de pesquisas referentes ao processamento de outras línguas, como o inglês, o japonês, o francês, o alemão e o espanhol.

A comunidade de pesquisadores do processamento do português tem se organizado nos últimos anos de maneira tal que seus trabalhos possam ser divulgados e discutidos. Como resultado, há um Congresso anual chamado PROPOR (Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada) que se realiza desde 1996 e que reúne, alternadamente, em Portugal e no Brasil, os pesquisadores nessa área. Os anais desse congresso apresentam alguns trabalhos relacionados ao processamento do português.

Há também, em Portugal, o projeto Processamento Computacional do Português, existente desde 1998 e subsidiado pelo governo português, que apresenta como uma de suas prioridades a criação e manutenção de um site [PCP-01], no qual pode-se encontrar referências a endereços relacionados com o processamento do português. [Santos-00] descreve o balanço e as perspectivas desse projeto.

Serão considerados, a seguir, alguns trabalhos ligados ao processamento de língua portuguesa, apontando, em cada trabalho, o formalismo escolhido para representar o português.

[Bick-00] descreve o sistema "Palavras" de análise sintática automática. Tal sistema, juntamente com um analisador e um etiquetador morfológico da língua portuguesa, faz parte de um projeto maior de criação de um tradutor automático Português-Dinamarquês. O analisador sintático "Palavras" apresenta boa robustez e pode ser acessado em [VISL-01] Em termos de representação formal, o sistema utiliza a Gramática de Dependência e baseia-se no formalismo Gramática de Restrições (de Helsinki) para lidar com ambigüidades morfológicas e sintáticas. A análise sintática é realizada com a ajuda de etiquetas morfológicas e sintáticas aplicadas ao texto de entrada pelo analisador e etiquetador morfológico. A cada palavra do texto em análise é associada uma descrição sintática que inclui a classe morfológica (se é um substantivo, um advérbio, etc.) e a forma sintática, representada por marcadores de dependência, que indicam o núcleo da unidade sintática em questão.

[Barbosa-00] apresenta a utilização de uma Gramática Livre de Contexto Lexicalizada para a geração de consultas, em língua portuguesa, aplicável a uma interface em linguagem natural para banco de dados.

[Kinoshita-97] apresenta a implementação de um protótipo de tradutor inglês-português, no qual a análise sintática é realizada de acordo com a Gramática de Ligação (*Link Grammar*), um tipo de Gramática de Dependência, e o formalismo utilizado é o DCG.

[Chin-96] propõe uma especificação gramatical de análise do português baseado no formalismo Gramática de Estrutura de Frase Generalizada (GPSG – *Generalized Phrase-Structure Grammar*), aplicada no processamento da tradução português-inglês de textos da área médica.

[Nunes et al.-96] descreve o desenvolvimento de um revisor gramatical para o português brasileiro, utilizando o formalismo ATN (*Augmented Transition Network*). Esse revisor faz parte do produto REDATOR, um processador de texto comercializado pela Itautec-Philco S/A.

Pode-se constatar, assim, que os trabalhos existentes na área de processamento de língua portuguesa adotam formalismos diversos para a representação da linguagem. O presente trabalho pretende contribuir para as pesquisas da área,

mostrando a viabilidade e a conveniência da utilização do formalismo adaptativo como um veículo alternativo para essa finalidade.

# 3. FUNDAMENTAÇÃO CONCEITUAL

Este capítulo introduz vários fundamentos conceituais sobre os quais esta dissertação foi desenvolvida. Inicialmente, são apresentados alguns conceitos básicos referentes a analisadores morfológicos em geral e acerca do etiquetador morfológico cujas saídas são utilizadas na parte experimental deste trabalho.

Em seguida, são descritos os formalismos das principais notações relacionados aos métodos apresentados no desenvolvimento desta pesquisa: *Augmented Context-Free Grammar*, *Augmented Transition Network*, *Definite Clause Grammar*, Gramática Baseada em Unificação ou Gramática Baseada em Restrições e os formalismos adaptativos (Autômato Adaptativo e Gramática Adaptativa). Finalizando, o capítulo apresenta um comparativo entre esses formalismos.

## 3.1. Analisador Morfológico

Conforme apresentado no Capítulo 2, a análise sintática corresponde, normalmente, à segunda etapa do processamento de uma sentença de linguagem natural. O analisador ou etiquetador morfológico é responsável pelo processamento da primeira etapa, tendo como função extrair e analisar cada palavra da sentença, bem como associar a ela uma etiqueta, correspondente à sua categoria morfológica.

O analisador morfológico também é utilizado para se criar os chamados corpora lingüísticos, que constituem coleções de textos com anotações (como as anotações morfológicas e sintáticas, por exemplo), textos esses que devem ser

representativos da linguagem em questão, reunidos com a finalidade de servir como base para pesquisas lingüísticas [Leech-97]. Entre os corpora lingüísticos mais conhecidos, tem-se o Corpus anotado Penn Treebank, do Departamento de Ciências da Computação e Informação da Universidade da Pensilvânia (cerca de 4,5 milhões de palavras em inglês americano com anotações morfológicas); o Corpus anotado Penn-Helsinki de inglês medieval, do Departamento de Lingüística da Universidade da Pensilvânia (510.000 palavras com anotações morfológicas e sintáticas); Corpus anotado Tycho Brahe de português histórico, do Instituto de Estudos da Linguagem da Universidade de Campinas (pretende-se que tenha cerca de 1 milhão de palavras, com anotações morfológicas e sintáticas).

Assim como ocorre com a análise sintática, a ambigüidade é um grande problema que o analisador morfológico deve tratar. Como foi mencionado no Capítulo 2, existem diversos tipos de ambigüidades morfológicas. Um exemplo seria a palavra 'complementar', que tanto pode ser um verbo como um adjetivo. Para tratar a ambigüidade, o analisador morfológico, normalmente, deve analisar também as informações do contexto da sentença e não apenas cada palavra isoladamente.

Existem quatro métodos principais que podem ser utilizados na criação de um analisador morfológico de textos em linguagem natural [Menezes-00]:

- estatístico – baseia-se na análise estatística da ocorrência de etiquetas morfológicas em um corpus de treinamento, assumindo que a ocorrência de uma etiqueta esteja relacionada com as etiquetas vizinhas.
- baseado em regras escritas manualmente – utiliza regras que mapeiam a representação superficial da palavra (como ela é escrita e usada) para a sua representação léxica (sua forma canônica acrescida de suas flexões morfológicas)
- baseado em regras inferidas automaticamente – com base em treinamento através de um corpus pequeno, infere regras transformacionais que podem ser usadas posteriormente na análise morfológica de textos livres.
- baseado em exemplos memorizados – também consiste numa fase de treinamento, quando memoriza um conjunto de casos, e numa fase posterior de aplicação, quando infere a etiqueta de um novo item utilizando as informações memorizadas.

O analisador morfológico utilizado na parte experimental deste trabalho foi desenvolvido com base nos pontos fortes desses métodos e é descrito detalhadamente em [Menezes-00]. Trata-se de um etiquetador morfológico treinável através de informações extraídas de um corpus de treinamento. Requer para isto um corpus com anotações morfológicas, e que as palavras da linguagem em questão tenham uma estrutura da forma seguinte:

Palavra = Prefixos + Radical + Sufixos

Sua implementação baseia-se em autômatos adaptativos, e sua arquitetura compõe-se de três módulos: o primeiro realiza a etiquetação de palavras conhecidas, o segundo realiza a etiquetação de palavras desconhecidas e o último efetua um refinamento contextual.

Na parte experimental, utiliza-se esse analisador morfológico, que foi treinado com partes do corpus Tycho Brahe [TBCHP-98]. Assim, as etiquetas morfológicas empregadas são as mesmas que foram definidas para esse corpus. Recebendo como entrada um texto em língua portuguesa, o analisador morfológico gera como saída o mesmo texto, acrescido das correspondentes anotações morfológicas, isto é, para cada palavra do texto, são adicionados: o sinal '/' e sua anotação morfológica. Um exemplo de texto de saída desse analisador morfológico é (extraído de [Menezes-00]):

```
Teve/ET-D a/P mesma/ADJ-F sorte/ADV o/D padre/NPR Gregório/N
da/P+D-F Rocha/TR-D,/, natural/ADJ-G da/P+D-F capitania/N
de/P Pernambuco/NPR ./. Tinha/TR-D ,/, ao/P+D tempo/N em/P
que/WPRO Deus/NPR o/CL chamou/VB-D para/P si/PRO ,/,
trinta/D-F anos/N-P de/P idade/N /./ Entrou/VB-D no/P+D ano/N
de/P 1611/N ,/, de/P 15/NUM anos/N-P ,/, na/P+D-F
Companhia/NPR ,/, e/CONJ nela/P+PRO viveu/VB-D outros/ADJ-P
15/NUM com/P satisfação/N e/CONJ observância/N religiosa/ADJ-
F ./. Sabia/VB-D bem/ADV a/P língua/N da/P+D-F terra/P ;/.
e/CONJ melhor/ADJ-R-G a/P exercitava/VB-D nas/P+D-F-P
aldeias/N-P ,/, cultivando/CONJS os/D-P índios/N-P /./
```

que corresponde ao seguinte texto de entrada:

> Teve a mesma sorte o padre Gregório da Rocha, natural da capitania de Pernambuco. Tinha, ao tempo em que Deus o chamou para si, trinta anos de idade. Entrou no ano de 1611, de 15 anos, na Companhia, e nela viveu outros 15 com satisfação e observância religiosa. Sabia bem a língua da terra; e melhor a exercitava nas aldeias, cultivando os índios.

A saída desse analisador morfológico é utilizada como entrada do analisador sintático de fragmentos de uma sentença da língua portuguesa, simulado na parte experimental descrita no Capítulo 5.

O próximo item apresenta os formalismos de tratamento sintático que estão relacionados aos métodos apresentados no desenvolvimento da proposta deste trabalho.

## 3.2. Formalismos de Tratamento Sintático

São descritos nesta seção os formalismos de tratamento sintático, relacionados aos métodos apresentados no desenvolvimento desta pesquisa: *Augmented Context-Free Grammar*, *Augmented Transition Network*, *Definite Clause Grammar*, Gramática Baseada em Unificação ou Gramática Baseada em Restrições.

### 3.2.1. *Augmented Context-Free Grammar*

O formalismo chamado Gramática livre contexto estendida *(Augmented Context-Free Grammar)* não é diretamente empregado neste trabalho, mas é descrito a seguir por ser largamente utilizado na representação e tratamento sintático de linguagens naturais. Inicialmente, são apresentados os conceitos de gramática e de gramática livre de contexto.

### 3.2.1.1. Gramática

Uma gramática G é uma quádrupla ($V_N$, $V_T$, P, S), segundo [DeRoeck-83], sendo:

- $V_T$ é um conjunto finito de *símbolos terminais*. No caso de linguagem natural, estes símbolos correspondem às palavras da linguagem.
- $V_N$ é um conjunto finito de *símbolos não-terminais*. Em aplicações lingüísticas, correspondem às categorias.
- P é um conjunto finito de regras chamadas de *produções*. Elas são da forma "$\alpha \rightarrow \beta$" ("$\alpha$ pode ser reescrito como $\beta$"), sendo que tanto $\alpha$ como $\beta$ são cadeias de elementos de $V_N$ e $V_T$.
- S é o *símbolo inicial* ou *raiz*. S é um elemento de $V_N$ e deve ocorrer pelo menos uma vez no lado esquerdo das regras de produções.

Uma gramática G gera uma linguagem denotada por L(G).

Chomsky desenvolveu uma teoria formal de gramáticas, na qual as classificou de forma hierárquica de acordo com o seu *poder* [Winograd-83]. Segundo Chomsky, há quatro tipos de gramáticas (chamadas de *tipo 0, tipo 1, tipo 2, tipo 3*), cada uma definida pelo tipo de regras de substituição que contém. A de tipo 0 é a mais poderosa, e cada tipo com número consecutivo é mais restrito. Há linguagens que podem ser definidas pela gramática de tipo 0, mas não pela de tipo 1, e assim por diante.

A gramática mais simples, segundo a hierarquia de Chomsky, é a de *tipo 3* e é denominada de *gramática regular (ou linear)* [Winograd-83] [Allen-94]. A linguagem gerada por uma gramática regular é a *linguagem regular.* Nela, a lei de formação de sentenças é muito simples, baseando-se na concatenação de símbolos básicos ou de sentenças menores, também regulares, segundo os critérios de regras incondicionais e isentas de aninhamentos sintáticos [José-93].

A linguagem baseada apenas nos símbolos *a*, *b*, *c* e *d,* e que permite qualquer sentença em que tais símbolos apareçam em ordem alfabética, é um exemplo de linguagem regular. As sentenças *abd, ad, bcd, b* e *abcd,* por exemplo, são válidas segundo essa regra de formação. Para essa linguagem, uma possível gramática regular poderia ser definida pelas seguintes regras de produção:

| | | | |
|---|---|---|---|
| $S \rightarrow aS_1$ | $S_1 \rightarrow bS_2$ | $S_2 \rightarrow cS_3$ | $S_3 \rightarrow d$ |
| $S \rightarrow bS_2$ | $S_1 \rightarrow cS_3$ | $S_2 \rightarrow d$ | |
| $S \rightarrow cS_3$ | $S_1 \rightarrow d$ | | |
| $S \rightarrow d$ | | | |

Na gramática regular, as regras de produção de P são da forma $A \rightarrow aB$ ou $A \rightarrow a$, sendo que *A* e *B* são símbolos não-terminais e *a* é um símbolo terminal. [Allen-94].

### 3.2.1.2. Gramáticas Livres de Contexto

Um tipo particular de gramática muito utilizado na descrição de linguagens, devido à sua simplicidade e clareza é a Gramática Livre de Contexto.

Numa Gramática Livre de Contexto [DeRoeck-83], as produções devem ter a forma "$A \rightarrow \alpha$", sendo que 'A' é um símbolo pertencente a $V_N$ ( o conjunto dos símbolos não-terminais) e '$\alpha$' é uma cadeia de terminais e/ou não-terminais.

Esta gramática é um caso particular da classe de gramáticas de estrutura de frases (*phrase structure grammars*), definida por Chomsky.

A gramática livre de contexto corresponde à gramática de tipo 2 da hierarquia de Chomsky [Winograd-83] e gera a linguagem livre de contexto, que estende as linguagens regulares, permitindo os aninhamentos sintáticos.

Um exemplo de gramática livre de contexto simplificada, aplicada à definição de um conjunto muito restrito de sentenças da língua portuguesa, é:

$V_N$ = {S, SS, SV, Sc, Sp, Det, $V_{td}$ }
$V_T$ = {gato, rato, come, o }
P = { S $\rightarrow$ SS SV,
SS $\rightarrow$ Det $S_c$,
SV $\rightarrow$ $V_{td}$ SS,
$S_c$ $\rightarrow$ gato,
$S_c$ $\rightarrow$ rato,
$V_{td}$ $\rightarrow$ come,
Det $\rightarrow$ o }

Uma sentença gerada por essa gramática é "o gato come o rato.".

Os símbolos em $V_N$ seguem a nomenclatura utilizada por Savadovsky em [Savadovsky-88]:

S Sentença

SS Sintagma Substantivo: trecho da oração que define completamente uma entidade ou conjunto de entidades do mundo do falante

SV Sintagma Verbal: parte da oração que define uma atividade ou estado no tempo.

$S_c$ Substantivo comum

Det Determinante

$V_{td}$ Verbo transitivo direto

A estrutura gerada para a sentença "o gato come o rato" pode ser representada na forma de uma cadeia com parênteses (*bracketed string*) ou, equivalentemente, em forma de árvore, conforme ilustrado a seguir, na Figura 2.

Cadeia com parênteses:

$$(_S\ (_{SS}\ (_{Det}\ o)\ (_{Sc}\ gato)\ )\ (_{SV}\ (_{Vtd}\ come)\ (_{SS}\ (_{Det}\ o)\ (_{Sc}\ rato)\ )\ )\ )$$

Estrutura em árvore:

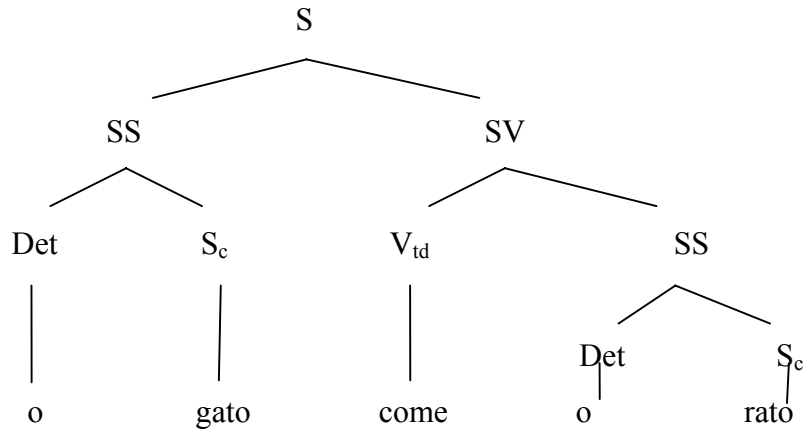



***Figura 2 - Representação em forma de árvore da estrutura sintática correspondente à sentença "o gato come o rato.".***

### 3.2.1.3. *Augmented Context-Free Grammar*

Na linguagem natural, é freqüente o problema de concordância entre as palavras que compõem uma sentença. Por exemplo, em português, o sintagma substantivo *o homens* é sintaticamente incorreto, pois não satisfaz à concordância de número exigida na língua portuguesa. Há outros tipos de concordância, como concordância de gênero, concordância de pessoa entre o sujeito e o verbo, entre outras [Allen-94].

Para representar este fenômeno, o formalismo gramatical livre de contexto foi estendido de forma a permitir que seus constituintes apresentem *características*.

Exemplificando, poderia ser definida a característica NÚMERO, que assumiria o valor *s* (para singular) ou *p* (para plural). Nesse caso, poder-se-ia ter a seguinte regra de gramática livre de contexto estendida:

$$SS \rightarrow DET\ S_c \text{ somente quando } NÚMERO_1 \text{ concorda com } NÚMERO_2.$$

Esta regra estabelece que um sintagma substantivo válido consiste de um determinante seguido de um substantivo comum, mas somente quando a característica NÚMERO da primeira palavra for a mesma da segunda. Esta regra única é equivalente a duas regras da gramática livre de contexto que usaria dois símbolos terminais diferentes para representar as formas singular e plural de todos os sintagmas substantivos, como por exemplo:

$$SS\text{-}SING \rightarrow DET\text{-}SING\ \ S_c\text{-}SING$$

$$SS\text{-}PLURAL \rightarrow DET\text{-}PLURAL\ \ S_c\text{-}PLURAL$$

Assim, todas as regras que envolvem o sintagma substantivo precisariam ser duplicadas para lidar com o singular e o plural. Se tal gramática também for projetada para considerar as demais concordâncias, como a de gênero, a do sujeito com o verbo, o número de regras cresceria sobremaneira.

Desse modo, a gramática, estendida com o uso de características, tem a vantagem de ter o seu tamanho mantido, mesmo lidando com o fenômeno da concordância.

Para isso, os constituintes da gramática estendida são definidos na forma de estruturas de características (*feature structure*), ou seja, de conjuntos de mapeamentos entre as características e os valores correspondentes que definem as

propriedades do constituinte. Por exemplo, uma estrutura de características para o constituinte DET1, que representa um uso particular da palavra *um* seria:

```
DET1:  (CAT    DET
        RAIZ    um
        NÚMERO  s)
```

que significa que é um constituinte de categoria morfológica DET (determinante), tem como raiz a palavra *um* e é singular. Normalmente, a palavra CAT é omitida da estrutura e assim, o constituinte DET1 poderia ser representado como

```
DET1:  (DET  RAIZ um  NÚMERO s)
```

No caso da representação do sintagma substantivo "um peixe", as estruturas de características de cada subconstituinte são usadas como valores. Os inteiros 1,2, etc, denotam a estrutura de característica do primeiro subconstituinte, do segundo, etc.

```
SS1:  (SS   NÚMERO s
            1 (DET   RAIZ um      NÚMERO s)
            2 (Sc    RAIZ peixe   NÚMERO s))
```

As estruturas de características são utilizadas também nas regras da gramática estendida, sendo que variáveis são aceitas como valores de características e desse modo, uma regra se aplica a várias situações. Por exemplo, uma regra para sintagma substantivo seria:

$$(\text{SS } \textbf{NÚMERO } ?n) \rightarrow (\text{DET } \textbf{NÚMERO } ?n)\ (S_c\ \textbf{NÚMERO } ?n)$$

que estabelece que um constituinte SS pode consistir de dois subconstituintes, sendo o primeiro um DET (determinante) e o segundo um $S_c$ (substantivo comum), desde que a característica NÚMERO dos três constituintes seja idêntica.

## 3.2.2. *Augmented Transition Network* (ATN)

O formalismo *Augmented Transition Network* teve um papel histórico muito importante na representação de linguagem natural, podendo ser considerado o formalismo dominante na implementação de sistemas de linguagem natural na década de 70 e em uma grande parte da década de 80. Por isso, ele foi escolhido neste trabalho como um dos formalismos a serem mapeados para um formalismo adaptativo equivalente, como parte da verificação de validade da proposta desta

dissertação. Primeiramente, são apresentados os conceitos de *Transition Network* e de *Recursive Transition Network*.

### 3.2.2.1. *Transition Network*

O modelo conhecido como *rede de transição (transition network)* surgiu como outra forma de representação das estruturas de linguagem natural, baseado na aplicação, ao estudo das gramáticas, das noções matemáticas de teoria de grafos e autômatos finitos [Harris-85].

Uma rede de transição consiste numa quádrupla (Q, A, I, F), sendo que:

Q é um conjunto finito de *estados*. Cada estado corresponde a um nó da rede e apresenta dois conjuntos:

- o conjunto de arcos que partem do estado: são arcos que têm este estado como estado de origem.
- o conjunto de arcos que chegam ao estado: são arcos que têm este estado como estado de destino.

A é um conjunto finito de *arcos*. Cada arco corresponde a uma transição da rede e é representado por uma tripla ($Q_o$, R, $Q_d$):

- $Q_o$ - estado de origem
- R - rótulo, que corresponde a uma palavra ou uma categoria léxica (denota uma classe de palavras, como por exemplo, o verbo visto no item 3.2.1.2.)
- $Q_d$ - estado de destino

Um arco é uma conexão orientada entre dois estados. Ele pode conectar um estado ao próprio estado, quando então $Q_o = Q_d$.

I é um subconjunto de Q, representando o conjunto de *estados iniciais*. Apenas um estado inicial pode não apresentar arcos que chegam.

F é um subconjunto de Q, representando o conjunto de *estados finais*. Apenas um estado final pode não apresentar arcos que partem.

Uma rede de transição pode ser determinística, quando, de cada estado, partir apenas um arco com um determinado rótulo, ou não-determinística, caso contrário.

Munido de uma base de conhecimento e de um dicionário, pode-se escrever um algoritmo que gera ou reconhece uma sentença da linguagem natural através de uma rede de transição. Para gerar uma sentença, deve-se escolher um estado inicial da rede e considerá-lo como sendo o estado corrente, e repetir o processo de escolha de um dos arcos que partem do estado corrente. Se o rótulo do arco corresponder a uma palavra, esta palavra é gerada, mas se o rótulo corresponder a uma categoria léxica, deve-se escolher uma palavra no dicionário que pertença a esta categoria, que será então a palavra gerada, e considerar o estado de destino deste arco o novo estado corrente. Este procedimento é repetido até que se chegue a um estado final da rede.

O reconhecimento de uma sentença é semelhante ao processo de geração, sendo que se deve verificar se cada palavra da sentença corresponde ao rótulo do arco corrente, conforme se caminha pela rede de transição, a partir de um estado inicial até um dos estados finais da rede.

As redes de transição podem ser representadas por grafos direcionados, como ilustrado no exemplo da Figura 3 a seguir:

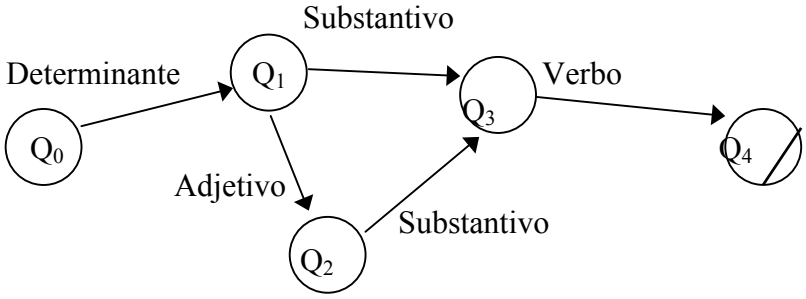



***Figura 3 - Exemplo de uma rede de transição representada através de um grafo direcionado.***

### 3.2.2.2. *Recursive Transition Network*

Em seguida, será apresentada a definição formal de um *Recursive Transition Network*, uma extensão do *Transition Network* que tem o poder de uma gramática tipo 2 (hierarquia de Chomsky) ou da gramática livre de contexto [Winograd-83].

Uma rede de transição recursiva consiste num conjunto de redes rotuladas. Essas redes são semelhantes às redes de transição, exceto pelo fato de que cada rede passa a ter um rótulo e cada arco é rotulado com uma palavra, uma categoria léxica, ou uma categoria sintática, que é o rótulo de alguma rede no conjunto de redes [Harris-85] [Winograd-83].

Na rede de transição (sem recursão), como cada arco corresponde a um símbolo terminal, com facilidade os grafos de representação de sentenças podem tornar-se complexos. O sintagma substantivo (correspondente ao *noun phrase*, em inglês), por exemplo, pode aparecer tanto no sujeito como no predicado da oração. O trecho da rede de transição que analisa o sintagma substantivo deve, então, ser repetido sempre que necessário. Na rede de transição recursiva, esse problema é solucionado, permitindo que o arco tenha também como rótulo, uma categoria sintática. Cada categoria sintática passa a ser analisada por uma rede de transição, que pode, por sua vez, acionar ou ser acionada por alguma rede de transição.

Uma rede de transição recursiva consiste numa quíntupla (R, Q, A, I, F), sendo que:

R é o rótulo da rede, que corresponde a uma categoria sintática (como por exemplo, o sintagma substantivo citado no item 3.2.1.2)

Q é um conjunto finito de *estados*. Cada estado corresponde a um nó da rede e apresenta dois conjuntos:

- o de arcos que partem do estado: são arcos que têm este estado como estado de origem.
- o de arcos que chegam ao estado: são arcos que têm este estado como estado de destino.

A é um conjunto finito de *arcos*. Cada arco corresponde a uma transição da rede e é representado por uma tripla ($Q_o$, R, $Q_d$):

- $Q_o$ - estado de origem

- R - rótulo, que corresponde a uma palavra, uma categoria léxica ou uma categoria sintática, ou o símbolo especial JUMP. Um arco JUMP pode ser transitado sem que corresponda a nenhum elemento na cadeia de entrada.
- $Q_d$ - estado de destino

Um arco é uma conexão orientada entre dois estados. Ele pode conectar um estado ao próprio estado, quando então $Q_o = Q_d$.

I é um subconjunto de Q, representando o conjunto de *estados iniciais* da rede. Apenas um estado inicial pode não apresentar arcos que chegam.

F é um subconjunto de Q, representando o conjunto de *estados finais* da rede. Apenas um estado final pode não apresentar arcos que partem.

Uma rede de transição recursiva é equivalente a um autômato de pilha, pois toda rede de transição recursiva é essencialmente um autômato de pilha cuja pilha de vocabulário é um subconjunto de seu conjunto de estados e o contrário também é verdadeiro, uma vez que todo autômato de pilha tem uma gramática livre de contexto equivalente, a qual por sua vez tem uma rede de transição recursiva equivalente [Winograd-83].

A Figura 4, a seguir, é uma ilustração da rede de transição recursiva, utilizando um exemplo simplificado aplicado à língua inglesa.

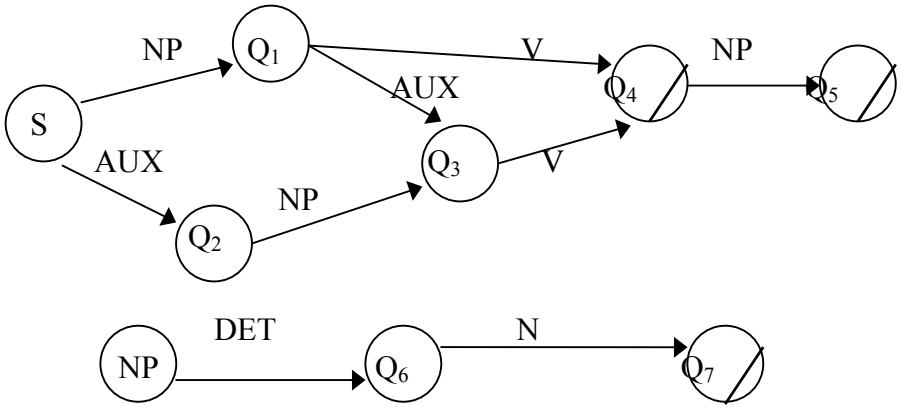



***Figura 4 - Exemplo simplificado de rede de transição recursiva aplicado à língua inglesa (adaptado de [Harris-85]).***

Na Figura 4, NP quer dizer *noun phrase* e corresponde ao sintagma substantivo mencionado no item 3.2.1.2, AUX é o verbo auxiliar (‘be’, ‘do’ ou ‘have’) e V o verbo, DET corresponde ao determinante e N ao substantivo.

Assim, no reconhecimento sintático de uma sentença, partindo-se do estado inicial S, a primeira parte pode ser um NP (*Noun Phrase*) ou um AUX (verbo auxiliar da língua inglesa: ‘be’, ‘do’ ou ‘have’).

O teste para verificar se é um NP deve ser feito salvando-se o contexto e transferindo o processamento para a rede de transição NP. Se realmente houver um NP no início da sentença, o processamento da rede NP terminará com sucesso, e o arco NP da rede S poderá ser transitado, prosseguindo no estado $Q_1$.

Se um NP não foi encontrado, então o teste para verificar se é um AUX deve ser realizado e caso seja bem sucedido, o processamento continua no estado $Q_2$.

Se, nem um NP, nem um AUX for encontrado, a cadeia de símbolos sendo testada não será aceita por esta rede de transição recursiva como uma sentença válida.

O modelo de Redes de Transição Recursivas, no entanto, não resolvia certos problemas encontrados no processamento de linguagem natural, tais como os relacionados à concordância, à incapacidade de mostrar a relação sistemática existente entre uma sentença afirmativa e sua interrogativa, etc. [Harris-85] [Winograd-83].

Para superar tais limitações, a funcionalidade do modelo de Redes de Transição Recursiva foi estendida, surgindo um novo modelo, chamado *Augmented Transition Network* (ATN) [Woods-70] [Bates-78], descrito a seguir.

### 3.2.2.3. *Augmented Transition Network*

O formalismo *Aumented Transition Network* (ATN) [Woods-70] [Bates-78] foi desenvolvido a partir do modelo de Redes de Transição Recursivas, estendendo-se sua funcionalidade para melhor representar e analisar a linguagem natural.

Muitas outras publicações existem sobre esse formalismo, como é o caso, por exemplo, de [Bolc-83], [Johnson-83], [Winograd-83], [Harris-85], [Rich-93], [Allen-94].

Em uma ATN, cada arco pode estar associado a uma condição, que deve ser satisfeita para que o arco seja transitado, e a um conjunto de ações de construção de estrutura que será executado caso o arco seja transitado.

Um modelo ATN consiste num conjunto de redes ATN, tendo cada rede um rótulo diferente.

Uma rede ATN consiste numa sêxtupla (R, Q, A, $Q_0$, CS, RG) [Bates-78] [Winograd-83], sendo que:

R é o rótulo da rede, que corresponde a uma categoria sintática (como por exemplo, o sintagma substantivo mencionado no item 3.2.1.2)

Q é um conjunto finito de *estados*. Cada estado corresponde a um nó da rede e apresenta dois conjuntos:

- o de arcos que partem do estado: são arcos que têm este estado como estado de origem
- o de arcos que chegam ao estado: são arcos que têm este estado como estado de destino.

Além disso, cada estado apresenta um rótulo, sendo que dois estados em uma rede não podem ter o mesmo rótulo.

A é um conjunto finito de *arcos*. Há 7 tipos de arcos:

- *CAT* – é comparado a uma palavra da cadeia de entrada, podendo ser transitado se essa palavra for da categoria léxica especificada. Seu rótulo é uma categoria léxica.
- *WRD* – pode ser transitado se a palavra da cadeia de entrada for a especificada neste arco.
- *MEM* – especifica uma lista de palavras, sendo que pode ser transitado se a palavra da cadeia de entrada pertencer a essa lista.
- *JUMP* – especifica o estado para o qual deverá haver uma transição sem que haja "consumo" da cadeia de entrada (transição incondicional em vazio).

- *PUSH* – especifica uma chamada recursiva à rede. Seu rótulo é o rótulo ou o estado inicial da rede que estiver sendo chamada.
- *POP* – especifica que o estado corrente é um estado final da rede, indicando qual a construção final a ser retornada como resultado da análise pela rede transitada. O efeito de se escolher um arco POP é o de completar o reconhecimento com sucesso através da rede.
- *VIR* – testa se um constituinte gramatical foi posicionado na lista *HOLD* por alguma ação *HOLD* executada previamente. A cada arco pode ser associada uma ou mais ações. A ação *HOLD* é uma delas e sua função é acrescentar uma informação a uma lista chamada *HOLD*, que depois pode ser verificada pelo arco *VIR*. O arco *VIR* e a ação *HOLD* constituem, assim, um mecanismo para salvar um constituinte, de forma que ele possa ser tratado posteriormente. Isso possibilita o melhor tratamento de certos fenômenos gramaticais, como por exemplo, a dependência de longa distância ou a formação de cláusulas relativas.

Cada arco corresponde a uma transição da rede, e é representado por uma quíntupla ($Q_o$, R, $Q_d$, C, Aç):

- $Q_o$ - estado de origem (estado do qual o arco parte)
- R - rótulo, que depende do tipo do arco. (pode ser *CAT*, *MEM*, *WRD*, *PUSH*, *POP*, *VIR* ou *JUMP*)
- $Q_d$ - estado de destino (estado ao qual o arco chega)
- C - conjunto de *condições* (ou *testes* – o arco poderá ser transitado se as condições forem satisfeitas ou se os testes forem verdadeiros)
- Aç - conjunto de *ações* (caso o arco seja transitado, as ações a ele associadas devem ser executadas)

$Q_0$ é o estado inicial da rede

CS é um conjunto de características sintáticas. Cada característica sintática apresenta:

- nome – seqüência de caracteres
- valores – conjunto de valores que a característica sintática pode assumir
- valor default – valor assumido pela característica sintática se nenhum valor for explicitamente especificado

Por exemplo, a característica sintática NÚMERO pode assumir dois valores: SINGULAR e PLURAL, sendo SINGULAR o seu valor default.

RG é um conjunto de nomes de registradores (*registers*). Os registradores são semelhantes a variáveis de uma linguagem de programação, apresentando um nome e armazenando alguma informação. São utilizados para armazenar características sintáticas e partes da árvore sintática construída durante a análise através da rede.

As condições e ações associadas aos arcos utilizam-se dos registradores (*registers*) para armazenar funções e características associadas aos nós da árvore sintática em construção. As condições fazem uso dos registradores para verificar propriedades, tanto de alguma palavra, como também da estrutura inteira, para ser comparada com o arco em questão. As ações utilizam os registradores para gerar as estruturas definitivas decorrentes da análise e para prover informações temporárias que podem ser usadas por condições em outros arcos.

Durante a operação da rede ATN, existe um certo número de registradores ativos. Ao transitar um arco PUSH, os valores desses registradores são empilhados e a rede passa a operar em um novo nível de recursão, com os registradores vazios. Ao transitar um arco POP, restauram-se os valores dos registradores a partir da pilha, cujo topo é desempilhado, fazendo com que fiquem iguais ao momento anterior à execução do PUSH.

A Figura 5 a seguir ilustra uma especificação de uma linguagem na qual uma ATN pode ser representada. A especificação, adaptada de [Woods-70] e de [Bates-78], é dada na forma de uma gramática livre de contexto estendida, na qual uma barra vertical separa formas alternativas de construção e o operador estrela de Kleene (*) indica repetições arbitrárias do elemento que a precede. A notação utilizada nessa especificação para representar as ações e as expressões é a "*Cambridge Polish*", uma notação na qual uma chamada de função é representada como uma lista entre parênteses, cujo primeiro elemento é o nome da função e os elementos restantes são os argumentos da função. Essa notação é aderente à da linguagem de programação LISP.

```
<rede-de-transição> → (<conjunto-de-arcos><conjunto-de-arcos>*)
<conjunto-de-arcos> → (<estado><arco>*)
<arco> → (CAT <categoria> <teste> <ação>* (TO <próx-estado>)) |
         (PUSH <estado> <teste> <ação>* (TO <próx-estado>)) |
         (WRD <palavra> <teste> <ação>* (TO <próx-estado>)) |
         (MEM <lista> <teste> <ação>* (TO <próx-estado>)) |
         (VIR <tipo-do-constituinte> <teste> <ação>* (TO <próx-
           estado>)) |
         (JUMP <estado> <teste> <ação>*) |
         (POP <expressão> <teste>)
<teste> → T | NIL |
         (NULLR <registrador>)
<ação> → (SETR <registrador> <expressão>) |
         (SENDR <registrador> <expressão>) |
         (LIFTR <registrador> <expressão>) |
         (HOLD <tipo-do-constituinte> <expressão>)
<expressão> → (GETR <registrador> ) |
         * |
         (GETF <característica-sintática> <palavra> ) |
         (APPEND <registrador> <expressão> ) |
         (BUILD <formato> <expressão>* )
```

***Figura 5 - Especificação de uma linguagem para representar as ATNs***

O arco pode ser transitado, dependendo de o resultado do teste associado ser verdadeiro ou não. Na especificação de <teste> acima, T representa a palavra *true*, correspondendo ao valor lógico verdadeiro, NIL é uma constante do LISP que representa falso, NULLR retorna verdadeiro se o registrador especificado ainda não recebeu nenhuma atribuição ou se seu valor for NIL. Além desses predicados, também podem ser utilizadas as funções booleanas AND, OR, NOT e EQUAL.

Os quatro tipos de ações SETR, SENDR, LIFTR, e HOLD são utilizados para se construir partes da sentença:

- SETR – atribui o valor de <expressão> ao registrador especificado.
- SENDR – ação utilizada nos arcos PUSH, atribuindo o valor de <expressão> ao registrador especificado, porém no nível inferior de recursão da rede. É uma forma de se passar um valor para o nível inferior de recursão, como se fosse a chamada de uma sub-rotina.

- LIFTR – é o inverso da ação SENDR, atribuindo o valor de <expressão> ao registrador especificado, porém no nível superior de recursão.
- HOLD – acrescenta o valor de <expressão> à lista HOLD, como um constituinte do tipo especificado por <tipo-do-constituinte>. A lista HOLD é acessível por qualquer nível de recursão da rede e é utilizada pelo arco VIR.

As expressões são utilizadas para descrever os dados que estiverem sendo manipulados. São funções aplicadas ao conteúdo do registrador, usualmente representadas em alguma linguagem funcional de especificação como LISP.

- GETR – retorna o valor contido no registrador especificado.
- * - palavra ou frase corrente na cadeia de entrada, isto é, o item que está sendo manipulado.
- GETF – retorna o valor da característica sintática especificada relativa à palavra especificada ou à palavra corrente na cadeia de entrada, caso não seja especificada a <palavra>.
- APPEND – utilizado para acrescentar o valor de <expressão> ao final do valor corrente no registrador especificado (por exemplo, acrescentar o sintagma verbal ao sintagma substantivo já formado e colocado no registrador SS)
- BUILD – constrói partes ou toda a árvore de análise sintática da sentença a partir do conteúdo das expressões especificadas. O <formato> representa como deve ser a lista a ser construída, e é constituído de constantes e de símbolos especiais. Cada símbolo especial em <formato> deve corresponder a uma <expressão>, na ordem em que elas ocorrem no comando e, na construção da lista, cada símbolo especial deve ser substituído pelo valor da <expressão> correspondente, sendo que o símbolo:
  - + - requer que a expressão correspondente seja um nome de registrador e faz com que seu conteúdo o substitua.
  - # - requer que a expressão correspondente seja avaliada e seu valor o substitua.

- * - requer que a palavra ou estrutura corrente o substitua, não havendo necessidade de que haja uma expressão correspondente.

Um exemplo de ATN, para a língua portuguesa, utilizando a linguagem descrita anteriormente :

```
((S/     (PUSH SS/ T
         (SETR SUJ *)
         (SETR TTIPO "DCL")
         (TO Q1))
   (CAT AUX T
         (SETR AUX *)
         (SETR TIPO "Q")
         (TO Q2))
(Q1      (CAT V T
         (SETR AUX NIL)
         (SETR V *)
         (TO Q4))
   (CAT AUX T
         (SETR AUX *)
         (TO Q3)))
(Q2      (PUSH SS/ T
         (SETR SUJ *)
         (TO Q3)))
(Q3      (CAT V T
         (SETR V *)
         (TO Q4)))
(Q4      (POP (BUILD (S +++ (SV +)) TIPO SUJ AUX V) T)
         (PUSH SS/ T
               (SETR SV (BUILD (SV (V +) *) V))
               (TO Q5)))
(Q5      (POP (BUILD (S ++++) TIPO SUJ AUX SV) T)))
```

que corresponde ao grafo da Figura 6:

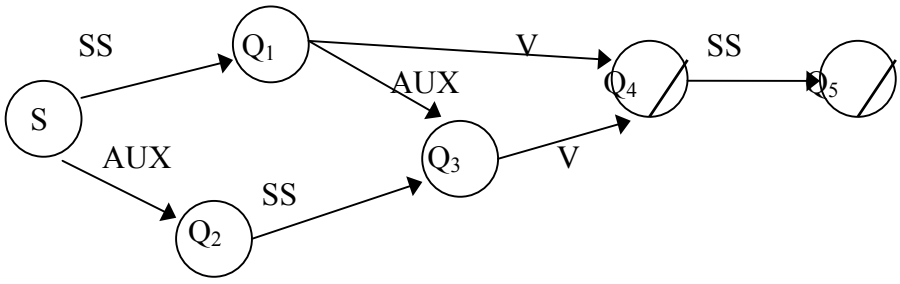



***Figura 6 - Grafo de representação da ATN correspondente ao exemplo***

sendo que SS quer dizer sintagma substantivo (mencionado no item 3.2.1.2), AUX é o verbo auxiliar e V o verbo.

Após o reconhecimento da sentença 'O gato come o rato', os registradores conteriam os seguintes valores:

```
TIPO        DCL
```

```
SUJ          O gato
V            come
```

e a estrutura construída seria:

```
(S   DCL   (SS   O gato)   (SV   come o rato)))
```

O formalismo ATN provê, dessa forma, os mecanismos importantes para a análise sintática da linguagem natural: construção da árvore sintática durante o reconhecimento da sentença, imposição de condições para que se aplique a transição e o tratamento de dependência de contexto.

O próximo item descreve um outro formalismo que também é muito utilizado na representação de linguagem natural: o *Definite Clause Grammar* (DCG).

### 3.2.3. *Definite Clause Grammar* (DCG)

Como o formalismo *Augmented Context-Free Grammar*, este formalismo não é diretamente empregado neste trabalho, mas é descrito a seguir por ser largamente utilizado na representação e tratamento sintático de linguagens naturais.

*Definite Clause Grammar* ou Gramática de Cláusulas Definidas nasceu da idéia de Colmerauer e Kowalski de traduzir o formalismo de gramáticas livres de contexto para lógica de predicados de primeira ordem [Pereira-80].

Assim, as regras livres de contexto passaram a ser expressas como declarações lógicas, conhecidas como *definite clauses* ou "*Horn clauses*". O problema de reconhecer ou analisar sintaticamente uma cadeia de uma linguagem se transformou então no problema de demonstrar que um certo teorema é válido, na hipótese de estarem corretos os axiomas das cláusulas definidas que descrevem a linguagem.

Paralelamente, Colmerauer e Kowalski [Pereira-80] tiveram ainda uma outra idéia: uma coleção de cláusulas definidas pode ser considerada como sendo um programa. Este conceito de programação em lógica foi desenvolvido então na forma da linguagem de programação Prolog.

Se uma gramática livre de contexto é expressa em cláusulas definidas, e executada como um programa escrito em Prolog, o programa se comporta como um

prático e intuitivo analisador sintático *top-down* para a linguagem que a gramática livre de contexto descreve [Pereira-80].

Para se entender o conceito do DCG, é importante assimilar a terminologia do subconjunto das cláusulas definidas, e a forma como se expressa uma gramática através da lógica, tópicos descritos nos itens que seguem.

### 3.2.3.1. Subconjunto das Cláusulas Definidas - Terminologia

Os elementos básicos que compõem programas escritos na linguagem Prolog são chamados *termos*, que podem ser uma *constante*, uma *variável* ou um *termo composto* [Pereira-80].

As constantes incluem *inteiros* e *átomos* (por exemplo: 'abc').

As variáveis devem começar com letra em maiúscula, e representam algum objeto particular não identificado, isto é, constituem um nome local para algum objeto de dado.

Um termo composto consiste de um *functor* e uma seqüência de um ou mais termos chamados de *argumentos*. Um *functor* é caracterizado pelo seu nome (que é um átomo) e pelo seu número de argumentos. Por exemplo, o functor cujo nome é 'soma' e cujo número de argumentos é 2 constitui um termo composto e é escrito: soma(X,Y). Um termo composto pode ser representado na forma de uma árvore.

As *listas* são uma importante classe de estrutura de dados. Uma lista pode ser vazia denotando-se como [ ], ou um termo composto, cujo *functor* é '.' e tem dois argumentos: o cabeçalho e o restante da lista.

*Gol* ou *chamada de procedimento* é um tipo especial de termo, distinguido apenas pelo contexto no qual ele aparece no programa. O *functor* de um *gol* é chamado *predicado*.

Uma seqüência de comandos chamados *cláusulas* constituem um *programa lógico*. Uma cláusula consiste de um *cabeçalho* e um *corpo*. O cabeçalho pode ser um simples *gol,* ou então ser vazio, enquanto o corpo consiste de uma seqüência de zero ou mais *gols*.

Quando nem o corpo nem o cabeçalho da cláusula forem vazios, a cláusula é escrita na forma:

```
M :- X, Y, Z.
```

sendo M o cabeçalho, X, Y e Z os *gols* que formam o corpo, e essa cláusula pode ser lida como "M é verdadeiro se X e Y e Z forem simultaneamente verdadeiros."

Quando o corpo da cláusula for vazio, ela é escrita na forma:

```
M.
```

sendo M o cabeçalho. A cláusula é então interpretada como "M é verdadeiro."

Quando o cabeçalho da cláusula for vazio, ela é chamada *questão,* e é escrita na forma:

```
?- X, Y.
```

sendo X e Y os *gols* do corpo. A cláusula é interpretada como "X e Y são verdadeiros?".

As cláusulas podem conter variáveis, mas o escopo das variáveis está sempre limitado a uma cláusula apenas.

Em um programa lógico, o *procedimento* para um particular predicado é a seqüência de cláusulas no programa cujos cabeçalhos tenham esse predicado como *functor* principal.

Instancia-se uma cláusula (ou termo) substituindo-se cada uma das suas variáveis por um novo termo, em todas as ocorrências da variável.

A *semântica declarativa* de cláusulas definidas indica quais *gols* podem ser considerados verdadeiros segundo um dado programa, e é definido recursivamente: Um *gol* é *verdadeiro* se for o cabeçalho de alguma instância de cláusula e se cada um dos *gols* do corpo dessa instância da cláusula for verdadeiro.

A execução de um gol baseia-se no processo de unificação de dois termos, que consiste em encontrar a instância comum mais genérica entre os dois termos.

Para se *executar* um *gol*, é preciso encontrar a primeira cláusula cujo cabeçalho corresponda ou *unifique* com aquele *gol*. Quando tal instância for encontrada, a cláusula correspondente é *ativada,* e cada um dos *gols* do seu corpo são executados, da esquerda para a direita. Quando não se encontrar uma instância que se equipare com um *gol*, realiza-se um retrocesso (*backtracking):* as cláusulas que foram mais recentemente ativadas são rejeitadas, e quaisquer substituições correspondentes a sua

execução são desfeitas. Reconsidera-se, então o *gol* inicial responsável pela ativação das cláusulas rejeitadas, e procura-se achar uma cláusula posterior que unifique com o *gol*.

### 3.2.3.2. Expressando a Gramática Através da Lógica

As gramáticas livres de contexto serão consideradas [Pereira-80] como sendo uma seqüência de regras da forma seguinte:

```
nt → corpo.
```

sendo que nt é um símbolo não-terminal e corpo é uma seqüência de um ou mais itens, separados por vírgulas. Cada item é um símbolo não-terminal ou uma seqüência de símbolos terminais.

Um símbolo não-terminal é escrito como um átomo do Prolog, enquanto uma seqüência de terminais é escrita como uma lista do Prolog, na qual um terminal pode ser qualquer termo Prolog. A cadeia vazia é denotada '[ ]'.

Assim, para o exemplo de gramática livre de contexto do item 3.2.1.2, teríamos as seguintes regras:

```
sentença → sintagma_substantivo, sintagma_verbal.
sintagma_substantivo → determinante, subst_comum.
sintagma_substantivo → substantivo_próprio.
sintagma_verbal → verbo_trans_direto, sintagma_substantivo.
subst_comum → [gato].
subst_comum → [rato].
verbo_trans_direto → [come].
determinante → [o].
```

Para se obter a tradução para cláusulas definidas de lógica, associa-se a cada não-terminal um predicado de dois argumentos. Tais argumentos representam os pontos de começo e final na cadeia da frase para aquele não-terminal. Assim, as regras acima ficariam:

```
sentença(S0,S):- sintagma_substantivo(S0,S1),
                 sintagma_verbal(S1,S).
sintagma_substantivo(S0,S):-determinante(S0,S1),
```

```
                                subst_comum(S1, S).
sintagma_substantivo (S0,S) :- substantivo_próprio(S0,S).
sintagma_verbal(S0,S):- verbo_trans_direto(S0,S1),
                        sintagma_substantivo(S1,S).
```

A primeira regra pode ser lida como "uma sentença se estende de S0 a S se há um sintagma substantivo de S0 a S1 e um sintagma verbal de S1 a S".

Para representar símbolos terminais em regras, utiliza-se um predicado de três argumentos, "connects", sendo que "connects(S1,T,S2)" significa "há um terminal T entre os pontos S1 e S2 na cadeia". As regras com terminais acima são traduzidas assim:

```
subst_comum(S0,S) :- connects (S0,gato,S).
subst_comum(S0,S) :- connects (S0,rato,S).
verbo_trans_direto (S0,S) :- connects (S0,come,S).
determinante (S0,S) :- connects (S0,o,S).
```

A sentença "O gato come o rato" seria codificada assim:

```
connects (1,o,2).
connects (2,gato,3).
connects (3,come,4).
connects (4,o,5).
connects (5,rato,6).
```

Reconhecer a sentença acima é o mesmo que tentar demonstrar o *gol*:

```
?- sentença (1,6).
```

### 3.2.3.3. Definite Clause Grammars (DCG)

A notação de DCGs estende a notação de gramáticas livres de contexto, expressa através de lógica (descrita no item anterior) da seguinte forma:

- os não-terminais podem também ser termos compostos, em lugar de simples átomos.
- do lado direito da regra, podem também existir *chamadas de procedimentos*, denotadas entre chaves. São utilizadas para expressar condições adicionais que devem ser satisfeitas para que a regra seja válida. Por exemplo:

  ```
  substantivo(N) → [N], {eh_substantivo(N)}.
  ```

  que corresponde à cláusula:

  ```
  substantivo(N,S0,S):-connects(S0,N,S),
                       eh_substantivo(N).
  ```

O formalismo da DCG provê assim, três mecanismos importantes para a análise de linguagens [Pereira-80]:

- construção de estruturas - como os não-terminais podem ser termos compostos, os quais por sua vez, podem ser expressos através de árvores, eles podem ser "expandidos", à medida que forem sendo equiparados às regras gramaticais, de maneira que a correspondente árvore sintática pode ir sendo gradativamente construída durante o reconhecimento da sentença.
- imposição de condições adicionais aos constituintes de uma frase, através das chamadas de procedimentos associadas às regras.
- tratamento geral de dependências de contexto - os argumentos dos não-terminais em uma DCG podem ser utilizados não só para a construção de estruturas, mas também para a representação e a verificação de informações contextuais. Como exemplo, a concordância de número (singular, plural) entre o sujeito e o verbo de uma frase pode ser facilmente expressa, fazendo-se que certos não-terminais apresentem um argumento extra, que possa assumir os valores 'singular' ou 'plural', em cada caso.

O próximo item descreve uma família de formalismos que tem sido recentemente muito utilizada para representar e analisar a linguagem natural: os formalismos baseados em restrições.

## 3.2.4. Gramáticas Baseadas em Restrições

Durante a década de 80, surgiram os formalismos conhecidos como gramáticas baseadas em restrições. Muitos dos sistemas de processamento de linguagem natural existentes atualmente baseiam-se nesses formalismos [Wedekind-97] [Uszkoreit-95]. Essa classe de formalismos caracteriza-se pelo uso particular de construções declarativas, contendo informações e restrições que representam a linguagem natural, dando ênfase à modularização e à manipulação dessas informações. Como esses formalismos utilizam a unificação como operação principal para manipulação das informações, tais formalismos são também conhecidos como gramáticas baseadas em unificação.

Alguns casos representativos dos formalismos baseados em restrições são: PATR [Shieber-88], PATR-II [Shieber-86], Gramática Léxica Funcional (LFG – *Lexical Functional Grammar*) [Shieber-86], Gramática de Unificação Funcional (FUG – *Functional Unification Grammar*) [Shieber-86], Gramática de Estrutura de Frase Generalizada (GPSG – *Generalized Phrase Structure Grammar*) [Gazdar et al.-85], HPSG (*Head-driven Phrase Structure Grammar* – Gramática de Estrutura de Frase Direcionada Pelo Núcleo) [Shieber-86], Gramática de Unificação Categorial (CUG – *Categorial Unification Grammar*) [Uszkoreit-95] e Gramática de Adjunção de Árvore (TAG – *Tree Adjunction Grammar*) [Uszkoreit-95].

A seguir, são apresentadas as principais características comuns aos formalismos baseados em restrições, sendo em seguida apresentado o formalismo GPSG (Gramática de Estrutura de Frase Generalizada), que foi escolhido no presente trabalho como representante dessa classe de formalismos para ser mapeado para Formalismo Adaptativo, no desenvolvimento da proposta desta pesquisa. Sua escolha se justifica, principalmente, por já existir uma especificação da gramática da língua portuguesa em GPSG [Chin-96], desenvolvida em uma tese de doutorado da área lingüística.

### 3.2.4.1. Características Comuns às Gramáticas Baseadas em Restrições

Um componente essencial dos formalismos de gramáticas baseadas em restrições é a descrição formal das informações lingüísticas através de conjuntos de pares da forma (atributo, valor), conhecidos como estruturas de características (*feature structures*). Trata-se do mesmo conceito apresentado anteriormente no item 3.2.1.3, utilizado na *Augmented Context-Free Grammar*.

A estrutura de características relaciona as características e os valores que definem as propriedades do constituinte gramatical de uma linguagem. Por exemplo, uma estrutura de características para o constituinte N1, que representa um uso particular da palavra *peixe* seria:

```
N1:   (CAT  N
       RAIZ peixe
       AGR  (NÚMERO   singular
             PESSOA   terceira))
```

sendo que CAT estabelece que a categoria morfológica do constituinte N1 é N (substantivo), RAIZ estabelece que a raiz léxica de N1 é a palavra *'peixe'*, e AGR representa a característica de concordância ("*agreement*") de N1 (terceira pessoa do singular). Informações contidas em partes da estrutura de característica podem ser referenciadas, usando-se para isso a notação de caminho (*path*), que é uma seqüência de um ou mais nomes de características, denotados entre os delimitadores < e >. No exemplo N1 citado, <CAT> refere-se ao valor da característica CAT, enquanto <AGR NÚMERO> se refere ao valor da característica NÚMERO, associada à característica AGR.

A nomenclatura e a notação da estrutura de características variam de formalismo para formalismo, mas, essencialmente, elas podem ser vistas como mapeamentos entre características e valores. O valor de uma característica também pode ser uma estrutura de característica, podendo, assim, existir estruturas de características aninhadas. Por exemplo, na estrutura de característica N1, definida

anteriormente, o valor da característica AGR é uma outra estrutura de característica. Alguns formalismos permitem a existência de valores disjuntos, como por exemplo, o valor {3s, 3p} para a característica AGR. Isso significa que a característica AGR pode assumir o valor 3s ou o valor 3p.

[Knight-89] observa que, embora as estruturas de características se assemelhem aos termos de primeira ordem, utilizados na lógica de predicados de primeira ordem, ou na programação em lógica (como no DCG, por exemplo), elas apresentam as seguintes vantagens sobre os termos lógicos: o número fixo de argumentos não é exigido, não há distinção entre função e argumento (todas as informações têm o mesmo status), e os valores das características são rotulados simbolicamente, em lugar de serem inferidos pela posição do argumento.

Outra característica comum aos formalismos baseados em restrições é o emprego da unificação como operação para manipular as informações lingüísticas contidas nas estruturas de características, tanto com a finalidade de mesclar, como de verificar as informações durante o processamento da análise sintática da linguagem representada.

O conceito de unificação baseia-se no conceito de extensão entre as estruturas de características. Uma estrutura de característica E2 é uma extensão de uma estrutura de característica E1 (escreve-se E1 •E2), ou seja, E1 é mais específica do que E2, se todo valor de característica em E1 estiver especificado em E2 [Allen-95]. Por exemplo, dadas as estruturas de características V1 e V2 a seguir:

```
V1:  (CAT   V                    V2:  (CAT V)
      RAIZ  escrever)
```

pode-se afirmar que V1 é uma extensão de V2 (V2 •V1).

Porém, dadas as estruturas de características V3 e V4:

```
V3:  (CAT  V                     V4:  (CAT V
      RAIZ escrever)                    VFORM pres)
```

pode-se afirmar que nem V3 estende V4, nem V4 estende V3.

Diz-se que duas estruturas de características E1 e E2 unificam se houver alguma estrutura de característica que seja extensão de ambas. A unificação das estruturas de características E1 e E2 (escreve-se (E1 •E2) é a estrutura de característica mínima que seja extensão de ambas.

Considerando-se as estruturas de características V3 e V4 citadas anteriormente, a unificação de V3 e V4 é V5 (V5 = V3 • V4):

```
V5:  (CAT   V
      RAIZ  escrever
      VFORM pres)
```

A estrutura de característica V5 estende ambas as estruturas de características V3 e V4, e não há estrutura de característica menor que seja extensão de ambas. Em contraste, as estruturas de característica:

```
V6:  (CAT V                    V7:  (CAT V
      AGR 3s)                        AGR 3p)
```

não unificam, pois como elas estabelecem valores contraditórios para a característica AGR, não há nenhuma estrutura de característica que possa ser extensão de ambas.

A estrutura de característica também pode ser representada como um grafo acíclico direcionado (DAG – *directed acyclic graph*), e, dessa maneira, utilizando-se a teoria dos grafos, as operações sobre as estruturas de características podem ser formalmente definidas [Shieber-86] [Allen-95]. No grafo que representa a estrutura de característica, as arestas (arcos) são rotuladas com os nomes das características, e cada uma das arestas aponta para um vértice, o qual representa o valor da característica correspondente, podendo ser um símbolo atômico ou um outro grafo acíclico direcionado.

Exemplificando, a seguinte estrutura de característica NP1:

```
NP1:  (CAT     NP
       SUJ:  (AGR:  ((NÚMERO  singular)
                     (PESSOA  terceira)))
```

que representa um sintagma substantivo na terceira pessoa do singular, corresponderia ao seguinte grafo acíclico direcionado:

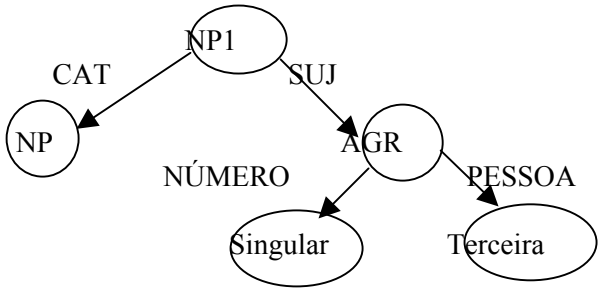



***Figura 7 - Grafo acíclico direcionado que representa a estrutura de característica NP1 (sintagma susbtantivo na terceira pessoa do singular)***

Uma vez definidas as estruturas de características que representam os constituintes e as unidades léxicas de uma determinada linguagem, resta definir as regras gramaticais que manipulam tais estruturas. Basicamente, a regra gramatical do formalismo baseado em restrições deve descrever a maneira como as unidades léxicas e os constituintes são concatenados para formar constituintes maiores, e também o modo como as estruturas de características correspondentes estão relacionadas. Dessa maneira torna-se possível ter, por exemplo, uma regra semelhante, à que seria utilizada em uma gramática livre de contexto, associada a expressões que relacionam as partes das estruturas de características correspondentes:

$X_0 \rightarrow X_1\ X_2$
$\quad\quad <X_0\ CAT> = S$
$\quad\quad <X_1\ CAT> = NP$
$\quad\quad <X_2\ CAT> = VP$
$\quad\quad <X_0\ NÚCLEO> = <X_2\ NÚCLEO>$
$\quad\quad <X_0\ NÚCLEO\ SUJEITO> = <X_1\ NÚCLEO>$

Esse formato de regra assemelha-se ao utilizado pela gramática de atributos, muito utilizada no desenvolvimento de compiladores de linguagens de programação para computador [Deransart-88].

Essa regra estabelece que o constituinte $X_0$ é formado pela concatenação dos constituintes $X_1$ e $X_2$. As três primeiras expressões estabelecem que a característica CAT dos constituintes $X_0$, $X_1$ e $X_2$ deve ser S, NP e VP, respectivamente. A quarta expressão exige que o valor da característica NÚCLEO associada com $X_0$ (ou seja, S) seja idêntico ao valor da característica NÚCLEO de $X_2$ (no caso, VP). A última expressão requer que o valor de SUJEITO em NÚCLEO de $X_0$ (que vale S) seja idêntico ao valor de NÚCLEO de $X_1$ (cujo valor é NP).

Exemplificando-se, as seguintes estruturas de características são aceitas por essa regra:

```
X0:  (CAT     S
      NÚCLEO: ((FORMA    finita)
               (SUJEITO: (AGR: ((NÚMERO singular)
                                (PESSOA terceira)))))
X1:  (CAT     NP
      RAIZ    Maria
      NÚCLEO: (AGR: ((NÚMERO singular)
                     (PESSOA terceira)))
X2:  (CAT     VP
      RAIZ    chorar
      NÚCLEO: ((FORMA    finita)
               (SUJEITO: (AGR: ((NÚMERO singular)
                                (PESSOA terceira)))))
```

As estruturas de características $X_1$ e $X_2$ correspondem, respectivamente, às palavras 'Maria' e 'chora'. A estrutura de característica $X_0$, corresponde, assim, de forma simplificada, à sentença 'Maria chora.'.

A seguir, são esquematizados os algoritmos para exemplificar como poderia ser realizada a unificação de duas estruturas de características, e de que forma a operação de unificação poderia ser utilizada na aplicação da regra gramatical.

A unificação de duas estruturas de características pode ser definida em termos de um algoritmo de combinação de grafos. Num grafo acíclico direcionado, define-se como fonte o vértice que não apresenta arestas convergentes (ou seja, arcos de entrada), e como sumidouro, o vértice que não apresenta arestas divergentes (ou seja, arcos de saída). O grafo acíclico direcionado que representa uma estrutura de característica possui apenas um vértice fonte, que é chamado raiz. Diz-se então que o grafo é enraizado nesse vértice. O rótulo de um vértice sumidouro tanto pode ser um valor atômico de característica como um conjunto de valores de características. O algoritmo de unificação de grafos recebe como entradas dois grafos, cada um apresentando apenas um vértice raiz, e retorna como saída um novo grafo, que é a unificação dos dois grafos de entrada recebidos.

**Algoritmo de unificação**: Para unificar um grafo acíclico direcionado de raiz $N_i$ com um grafo acíclico direcionado de raiz $N_j$, executa-se o seguinte algoritmo:

1. **Se** $N_i$ for igual a $N_j$,
   **Então** retorne $N_i$ e finalize este algoritmo com sucesso.

2. **Se** tanto $N_i$ como $N_j$ forem vértices sumidouros,
   **Então, se** os seus conjuntos de rótulos tiverem uma intersecção não nula,

**então** retorne um novo nó cujo rótulo seja essa intersecção
**senão** eles não unificam.

3. **Se** $N_i$ e $N_j$ não forem vértices sumidouros,
**Então** crie um novo nó N. Para cada aresta rotulada F partindo de $N_i$ para o nó $NF_i$, faça:

3a. **Se** houver uma aresta rotulada F partindo de $N_j$ para o nó $NF_j$,
**Então** unifique (recursivamente) $NF_i$ e $NF_j$ e crie uma aresta rotulada F a partir de N para o grafo resultante da chamada recursiva.

3b. **Se** não houver nenhuma aresta rotulada F a partir de $N_j$,
**Então** Crie uma aresta rotulada F de N para $NF_i$.

3c. Para cada aresta rotulada F de $N_j$ para $NF_j$,
**Se** não houver uma aresta F partindo de $N_i$,
**Então** crie uma nova aresta de rótulo F de N para $NF_j$.

A aplicação desse algoritmo pode ser ilustrada através das seguintes estruturas de características (supondo que a disjunção de valores seja permitida):

```
N2:  (CAT N                    N3:  (CAT N
      RAIZ ônibus                     AGR 3s)
      AGR  {3s, 3p})
```

A estrutura de característica N2 corresponde à palavra 'ônibus', um substantivo cuja concordância verbal é o da terceira pessoa, podendo estar sendo utilizado tanto no singular como no plural. A estrutura N3 corresponde a um substantivo cuja concordância verbal é o da terceira pessoa do singular. N2 e N3 podem ser representadas pelos grafos acíclicos direcionados mostrados na Figura 8:

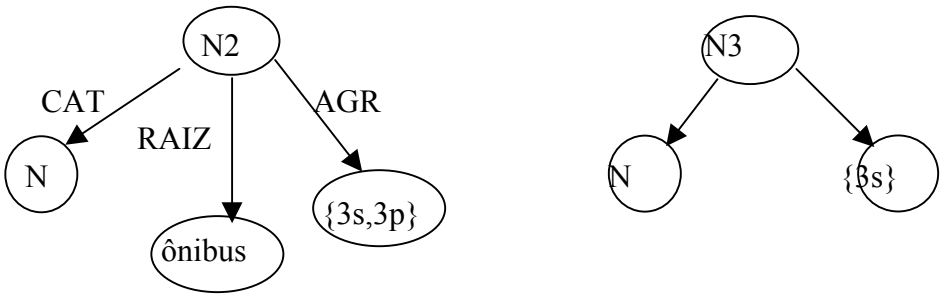



***Figura 8 - Grafos acíclicos direcionados que representam as estruturas de características N2 e N3.***

A unificação de N2 e N3 corresponde à estrutura de característica N4:

```
N4:  (CAT  N
      RAIZ ônibus
      AGR  3s)
```

representada pelo grafo acíclico direcionado da Figura 9.

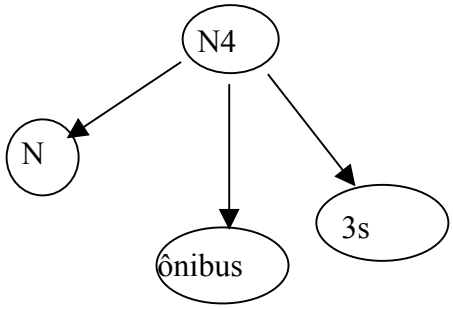



***Figura 9 - Grafo acíclico direcionado que representa a estrutura de característica N4, unificação de N2 e N3.***

Dada uma regra $X_0$ -> $X_1$ .... $X_n$ e um conjunto de expressões da forma <$X_i$ $F_i$> = V, sendo que $SC_1$, ... , $SC_n$ são os constituintes correspondentes a $X_1$, ..., $X_n$, e dados os grafos DAG que representam cada um dos constituintes, o algoritmo a seguir cria um DAG (grafo acíclico direcionado) que satisfaça a regra e todas as expressões relacionadas. Esse novo DAG representa o constituinte $SC_0$, correspondente ao $X_0$ da regra.

1. Crie um vértice $CC_0$ para ser a raiz do grafo correspondente à nova estrutura de característica.

2. Faça uma cópia de cada DAG que tenha $SC_i$ como raiz (chame a nova raiz de $CC_i$), e adicione uma aresta rotulada i de $CC_0$ para cada $CC_i$.

3. Para cada expressão da forma <$X_i$ $F_i$> = V, sendo que V é um valor, siga a aresta $F_i$ a partir do vértice $CC_i$ para um vértice $N_i$, e unifique $N_i$ com V.

4. Para cada expressão (da forma $F_i = G_j$)

   4a. **Se** houver uma aresta $F_i$ a partir de $CC_i$, e uma aresta $G_j$ a partir de $CC_j$, **então**
   - i. siga a aresta $F_i$ para o vértice $N_i$ e a aresta $G_j$ para o vértice $N_j$;
   - ii. unifique $N_i$ com $N_j$, usando o algoritmo de unificação de grafos para criar o novo vértice X.
   - iii. altere todos as arestas apontando tanto para $N_i$ como para $N_j$, para que apontem para X;

4b. **Se** não houver nenhuma aresta $F_i$ a partir de $CC_i$, mas houver uma aresta $G_j$ a partir de $CC_j$ para $N_j$, **então** crie um link $F_i$ de $CC_i$ para $N_j$;

4c. **Se** não houver nenhuma aresta $G_j$ a partir de $CC_j$, mas houver uma aresta $F_i$ a partir de $CC_i$ para $N_i$, **então** crie uma aresta $G_j$ de $CC_j$ para $N_j$.

A aplicação desse algoritmo é exemplificada através das estruturas de características $X_1$ e $X_2$ e da regra $X_0 \rightarrow X_1\ X_2$, mencionadas anteriormente. Os grafos acíclicos direcionados de X1 e X2 são ilustrados na Figura 10.

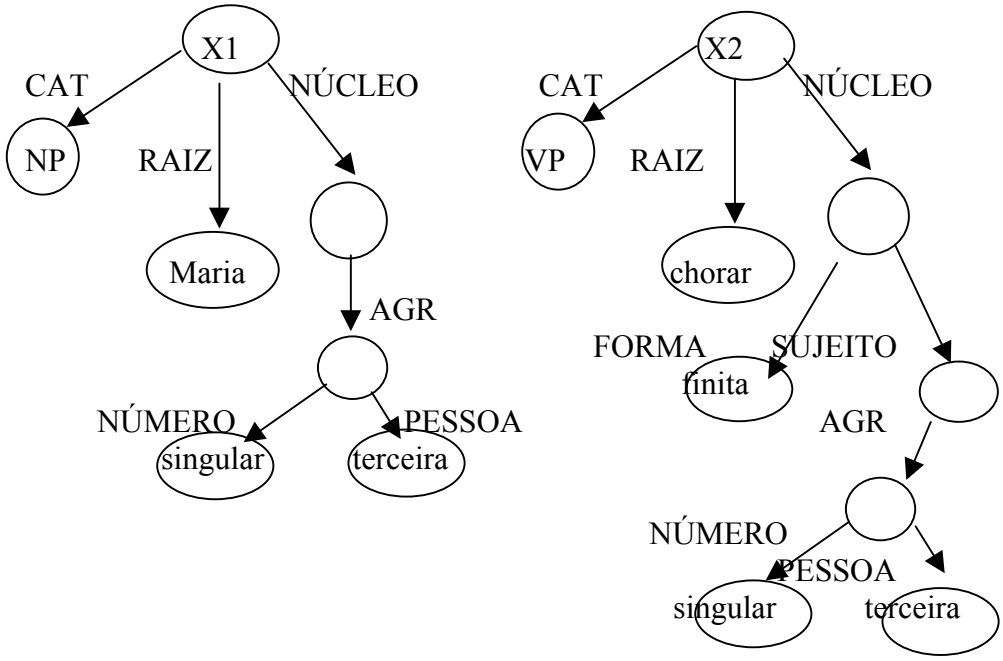



***Figura 10 - Grafos acíclicos direcionados que representam as estruturas de características $X_0$ e $X_1$.***

Após aplicado o algoritmo, é criado um novo grafo que satisfaz a regra gramatical e todas as expressões relacionadas, obtendo-se o grafo $X_0$, ilustrado na Figura 11.

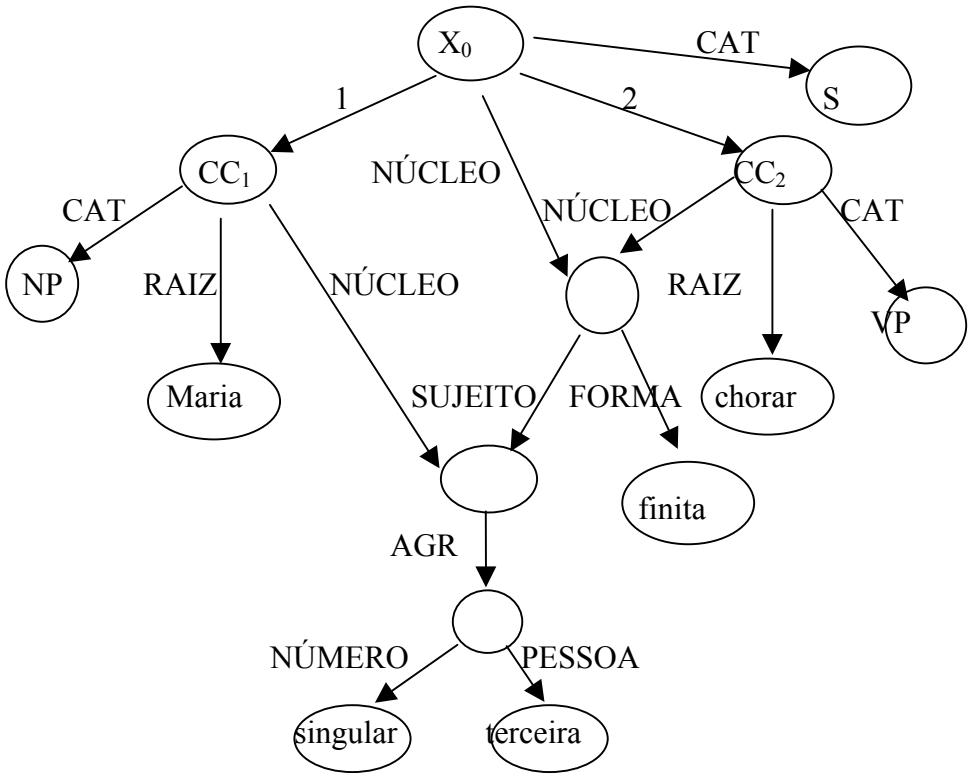



***Figura 11 - Grafo acíclico direcionado que representa a estrutura $X_0$, obtida da aplicação da regra gramatical $X_0 \rightarrow X_1 X_2$, juntamente com as expressões relacionadas.***

Apesar de esse item descrever as características comuns às gramáticas baseadas em restrições, a notação da regra gramatical aqui apresentada corresponde ao formalismo PATR-II, desenvolvido por Shieber, no início da década de 80, e tido como o formalismo mais simples da família de formalismos baseados em restrições.

O próximo item apresenta um outro formalismo, também baseado em restrições, a gramática de estrutura de frase generalizada (GPSG), que foi escolhido para ser utilizado no desenvolvimento da proposta desta dissertação. Como foi mencionado anteriormente, sua escolha se deve ao fato de existir publicada uma especificação da gramática da língua portuguesa nesta notação, o que permite que em futuros trabalhos de pesquisa nesta linha seja possível o aproveitamento dos esforços já despendidos, visando a geração de novos textos sobre o assunto, a respeito do qual pouco existe disponível na literatura.

### 3.2.4.2. Gramática de Estrutura de Frase Generalizada (GPSG)

A Gramática de Estrutura de Frase Generalizada foi formalmente especificada em 1985, por Gerald Gazdar, Ewan Klein, Geoffrey Pullum e Ivan Sag [Gazdar et al.-85]. Foi desenvolvido na tentativa de se obter um sistema que, apesar de ser formalmente restritivo, pudesse lidar com vários tipos de fenômenos sintáticos e semânticos do processamento da linguagem natural. Através da GPSG, por exemplo, pode-se tratar o caso de dependências irrestritas sem a necessidade de efetuar transformações de estruturas. Esse formalismo será utilizado no desenvolvimento da proposta desta dissertação, por já existir publicada uma especificação da língua portuguesa em GPSG [Chin-96].

Na GPSG, as estruturas de características são bastante restritas. Elas são denominadas categorias, sendo que cada uma delas representa um conjunto de especificações característica-valor. Por exemplo, a palavra ‘caneta’ estaria associada à categoria [N+, V-, BAR 0, PER 3, PLU-, MASC-], que estabelece que essa palavra é um substantivo, é um item léxico, cuja concordância verbal é o da terceira pessoa do singular e é do gênero feminino.

As principais características são [N] e [V], cujos valores são {+, -}, e são utilizados para definir as principais categorias morfossintáticas: os substantivos são representados por [N+, V-], os verbos por [N-, V+], os adjetivos por [N+ V+], e as preposições, [N-, V-]. Os advérbios são representados como os adjetivos, [N+ V+], com a especificação [ADV+].

A característica BAR pode assumir um dos três valores {0, 1, 2}, e indica o nível de projeção de uma categoria. Essa nomenclatura é proveniente da teoria X-barra [Schaüfele-99], que define que os sintagmas são formados a partir de um item léxico principal X (que pode ser um substantivo, um verbo, uma preposição ou um adjetivo), precedido por especificadores e seguido por complementos. Por exemplo, no sintagma substantivo ‘Um estudante de computação’, o item léxico principal é o substantivo ‘estudante’, que é precedido pelo especificador ‘um’ (no caso, um determinante), e é seguido pelo complemento ‘de computação’ (no caso, um sintagma preposicional).

Quando se adicionam complementos ao item léxico principal X, obtém-se uma projeção de X, denominada X' (X barra). No exemplo citado anteriormente, do sintagma substantivo 'Um estudante de computação', 'estudante de computação' é uma projeção de 'estudante'. Adicionando-se os especificadores a X', obtém-se o sintagma XP, ou X" (X-duas barras). No exemplo anterior, 'Um estudante de computação' corresponde ao segundo nível de projeção de 'estudante', e é o próprio sintagma substantivo.

As regras de formação de sintagma, segundo a teoria X-barra, são:

X' → <modificadores>, X, <complementos>

XP → <especificadores> X'

[SUBCAT] é uma outra característica que pode assumir um valor numérico (um número inteiro, de código n.n), uma letra ou um morfema. Para as principais categorias morfossintáticas (substantivo, verbo, adjetivo e preposição), ela indica qual a subcategorização. Por exemplo, os verbos que têm uma oração como complemento serão [SUBCAT 4.17]. As categorias que não incluem a característica [BAR] apresentam uma letra como valor de [SUBCAT], indicando um grupo de palavras, ou apresentam como valor de [SUBCAT] o próprio morfema.

As características de concordância dos verbos são: [PER], que indica a pessoa e assume um dos valores {1, 2, 3}, e [PLU], que indica o número (singular ou plural) e assume um dos valores {+, -}. [PLU] também é a característica de concordância dos adjetivos.

As características de flexão de substantivos são [PER] e [PLU], como nos verbos, e [MASC], que indica o gênero e assume um dos valores {+, -}.

[VFORM] indica a forma do verbo e assume como valores {FIN (finito), INF (infinitivo), BSE (subjuntivo), PAS (voz passiva), PSP (particípio passado), PRP (gerúndio)}.

Na GPSG, existem vários tipos de regras:

- **regras ID (*immediate dominance*)** – são semelhantes às da gramática livre de contexto, embora não especifiquem a ordem entre os vários constituintes. A regra ID é do tipo $C_0 \rightarrow C_1, C_2, ..., C_n$, sendo $C_0$ a categoria-mãe, que tem o domínio imediato sobre as categorias-filhas $C_1$, $C_2$, ... $C_n$. A categoria

nuclear, que é o item léxico principal que dá origem ao sintagma que estiver sendo formado, é representada pelo símbolo H, sendo o núcleo da categoria criada pela regra ID.

Exemplo de regra ID (extraído de [Chin-96]):

```
N1 → H[1.2], (P2[sobre])
```

Nesse exemplo, N1 é o mesmo que o conjunto de especificações [N+, V-, BAR 1], correspondendo a uma projeção de nível 1 de um substantivo e seu complemento, H[1.2] é a categoria nuclear da regra e representa [N+, V-, BAR 0, SUBCAT 1.2], correspondendo a um substantivo de subcategorização 1.2 (no exemplo, refere-se ao substantivo 'efeito'), e (P2[sobre]) representa [N-, V-, BAR 2, PFORM sobre], correspondendo a um sintagma preposicional cuja primeira palavra é 'sobre'.

- **regras LP (*linear precedence*)** – especificam a ordem linear de todos os constituintes que aparecem nas regras ID. A regra LP é do tipo $C_i < C_j < \ldots < C_y$, sendo que o símbolo < indica a relação de precedência entre os termos relacionados.

  Exemplo:

  ```
  S → SS, SV
  SS < SV
  ```

  Nesse exemplo, a primeira regra (ID) diz que uma sentença (S) é formada por um sintagma substantivo (SS) e por um sintagma verbal (SV), mas não estabelece a ordem entre eles. A segunda regra (LP) define que o sintagma substantivo deve sempre preceder o sintagma verbal.

- **meta-regras** – regras que definem novas regras, baseadas em regras ID existentes, e são utilizadas para expressar relações entre estruturas, como por exemplo, entre a voz ativa e a passiva.

  Exemplo de meta-regra de voz passiva:

  ```
  SV → W, SS
     ⇓
  SV[PAS] → W, (SP[por])
  ```

  De acordo com esse exemplo, se há uma regra ID que especifica que um sintagma verbal (SV) tem domínio imediato sobre um sintagma substantivo (SS) e mais outras categorias (representadas genericamente por W), então por

essa meta-regra, também deve haver uma regra ID que especifica que um sintagma verbal na forma passiva tenha domínio imediato sobre essas outras categorias (W), e sobre um sintagma preposicional (SP), iniciado pela palavra 'por'.

- **regras de especificação de defaults (FSD)** – definem os valores que certas características devem assumir quando não são especificados através de alguma regra.

  Exemplo de regra FSD (extraído de [Chin-96]):

  ```
  FSD: [ADV-]
  ```

  Na GPSG, tanto os adjetivos como os advérbios correspondem a [N+, V+], sendo que os advérbios são também [ADV+]. Assim, essa regra especifica que por default, caso não seja especificado um valor para [ADV], ele será – (menos), fazendo com que os advérbios devam ter explícito o valor [ADV+].

A GPSG também define algumas restrições:

- **restrições sobre a simultaneidade de ocorrência de características (FCR)** – determinam a simultaneidade de ocorrência de características. Por exemplo, toda oração com [COMP que] deve ser finita. Assim, tem-se

  ```
  FCR: [COMP que] ⊃ ([FIN] ∨ [BSE])
  ```

  eliminando-se a necessidade de se escrever S[FIN] ou S[BSE] quando [COMP que] estiver especificado nas categorias introduzidas numa regra.

Existem três classes de características:

- **HEAD ou nucleares**, que são {AGR, ADV, AUX, BAR, INV, LOC, N, PAST, PER, PFORM, PLU, PRD, SLASH, SUBCAT, SUBJ, V, VFORM}.
- **FOOT ou não-nucleares**, que consistem nas informações que devem passar de uma categoria não-nuclear para a categoria-mãe e são {RE, SLASH, WH}. Todas as características FOOT assumem como valor uma categoria. A característica RE refere-se aos pronomes recíprocos ou reflexivos, e assume como valor uma categoria contendo os detalhes de concordância do pronome. A característica SLASH refere-se a um constituinte da sentença que está ausente (implícito), e assume como valor uma categoria com os detalhes

desse constituinte. A característica WH refere-se aos pronomes interrogativos e relativos, e seu valor corresponde a uma categoria SS com as características de concordância do pronome.

- **de CONTROLE**, que são {SLASH, AGR} e são utilizadas para controlar a concordância entre os constituintes de uma frase.

Diz-se que uma regra ID $C_0 \rightarrow C_1, \ldots, C_n$ admite uma árvore local se e somente se a raiz dessa árvore tiver domínio imediato sobre os nós filhos, e se cada um dos nós filhos for uma extensão de cada uma das categorias filhas da regra.

Por exemplo, seja a regra ID:

(1) `S → SS, SV`

que especifica que uma sentença é formada por um sintagma substantivo e por um sintagma verbal.

E seja a árvore da Figura 12:

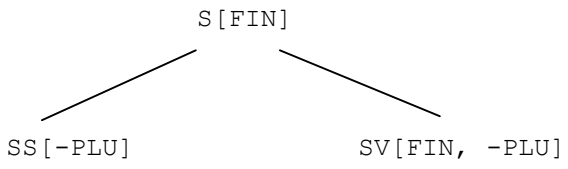



***Figura 12 - Árvore local admitida pela regra ID (1).***

A árvore local representada na Figura 12 é admitida pela regra ID (1), pois sua raiz tem domínio imediato sobre os nós filhos e cada um dos nós filhos é uma extensão de cada uma das categorias filhas da regra ID (1).

As características que estão presentes nas categorias da árvore podem ser herdadas ou instanciadas. As herdadas são aquelas determinadas pela própria regra ID (no exemplo, as categorias N e V). As instanciadas são as características presentes na árvore, porém não na regra (no exemplo, as categorias [VFORM FIN] e [PLU-]).

A regra ID, por si só, é bastante permissiva, pois a regra ID (1) do exemplo também admitiria perfeitamente uma árvore em que o sintagma substantivo tivesse concordância verbal de pessoa no singular [PLU-] e o sintagma verbal tivesse

concordância verbal de pessoa no plural [PLU+]. Para resolver problemas como esse, a GPSG determina que a árvore também deve satisfazer a determinados princípios de instanciação de características:

- **HFC (Head Feature Convention) – convenção das características nucleares** – exige que as características nucleares da categoria-mãe sejam idênticas às características nucleares da categoria-filha nuclear, desde que não seja uma das características nucleares da categoria-mãe já impostas pela regra ID ou por FCR. Por exemplo, considerando-se a regra:

  S → SS, SV

  na GPSG, as categorias S e SV tem as mesmas especificações para N e V [N-V+], com a diferença de que S tem a especificação [SUBJ+] e SV tem a especificação [SUBJ-]. Assim, nessa regra, a categoria-filha nuclear é o SV. O princípio HFC estabelece, assim, que as características nucleares de S, que não são impostas por regra ID ou por FCR, sejam idênticas às características nucleares de SV. SUBJ é uma característica nuclear, mas não pode ser idêntica nas duas categorias, pois é estabelecida por FCR. As demais características nucleares que forem instanciadas devem ser idênticas, como [PER] e [PLU], por exemplo. A funcionalidade desse princípio corresponde à unificação do formalismo PATR-II, estabelecida por:

  <S NÚCLEO> = <SV NÚCLEO>

- **CAP (Control Agreement Principle) – princípio de controle de concordância** – na GPSG, a concordância é analisada como função, que define o alvo e o controlador da concordância. A categoria que corresponde ao alvo da concordância deve ter a característica de controle AGR, cujo valor é a categoria controladora e as especificações de concordância que ela deve ter. O princípio CAP força a identidade entre as características de um controlador e do controlado, para que haja concordância sintática entre eles. Por exemplo, na regra:

  S → SS, SV

  define-se que o alvo da concordância é a categoria SV e o controlador, a categoria SS. A categoria SV deve ter, assim, a especificação da característica [AGR] cujo valor é SS (a categoria controladora), seguida das especificações

de [PER] e [PLU] que ela deve ter para que haja concordância sintática entre essas duas categorias.

- **FFP (Foot Feature Principle) – princípio de características não-nucleares** – estabelece que as características do grupo FOOT instanciadas na categoria-mãe sejam iguais à unificação das características FOOT instanciadas nas categorias-filhas. Dessa forma, as características não-nucleares das categorias-filhas podem ser passadas para a categoria-mãe. Assim, fenômenos como as dependências de longa distância e a formação de cláusulas relativas, com os quais as características FOOT estão relacionadas, podem ser devidamente tratados.

As diversas regras e princípios de uma gramática GPSG devem ser satisfeitas por uma árvore de estrutura de frase, que é um conjunto das diversas árvores locais admitidas e correspondentes a cada uma das regras, que também satisfazem todos os princípios de instanciação de características.

O próximo item descreve os Formalismos Adaptativos.

## 3.3. Formalismos Adaptativos

Embora o escopo dos chamados formalismos adaptativos seja conceitualmente muito mais amplo [José-01], no presente texto entende-se como formalismos adaptativos apenas os Autômatos Adaptativos [José-93] e as Gramáticas Adaptativas [Iwai-00]. Caracterizam-se por serem adaptativos, ou seja, pela sua capacidade de se auto-modificarem à medida que são utilizados. Essa característica dinâmica os torna capazes de representar e lidar com linguagens sensíveis ao contexto. São formalismos poderosos, apresentando poder computacional equivalente ao da máquina de Turing [Iwai-00]. Dessa maneira, podem também ser empregados no processamento de linguagem natural. As principais características desses dois formalismos adaptativos são apresentadas a seguir.

## 3.3.1. Autômatos Adaptativos

O Autômato Adaptativo consta de uma máquina de estados que se utiliza de uma memória organizada em pilha e tem características adaptáveis, pois permite que a configuração da máquina seja alterada dinamicamente, em função das transições efetuadas pelo autômato.

Seu conceito foi formalmente introduzido em [José-93] [José-94], e posteriormente aprimorado [Iwai-00], fruto de uma longa pesquisa que visava buscar uma nova solução para a automatização da elaboração de reconhecedores sintáticos para linguagens sensíveis ao contexto.

Antes que se apresente a definição formal do Autômato Adaptativo, é importante entender o relacionamento existente entre os conceitos de autômato finito, autômato de pilha estruturado, regras de transição, situação do autômato e ações adaptativas.

**Autômato Finito.** Um autômato finito, ou máquina de estados finitos, é um dispositivo utilizado para reconhecer e aceitar linguagens regulares (conceito apresentado no item 3.2.1.1). É definido formalmente como uma quíntupla M= (Q, Σ, R, $q_0$, F), sendo que:

- Q é o conjunto de estados de M.
- Σ é o alfabeto de entrada. As cadeias de entrada devem ser formadas por símbolos que pertençam a Σ.
- R é o conjunto de regras de transição, que são da forma *(e, sα):→ (e', s'α)*, na qual *e* e *e'* são estados pertencentes a Q, *s* e *s'* são símbolos pertencentes a Σ e $\alpha$ indica a parte da cadeia de entrada que ainda não foi reconhecida. A regra de transição especifica que, do estado *e*, pode-se transitar para o estado *e'*, com o consumo do símbolo *s* da cadeia de entrada e a inserção do símbolo *s'* na cadeia de entrada.
- $q_0$ é o estado inicial de M, devendo pertencer a Q.
- F é o conjunto de estados finais de M. ($F \subseteq Q$).

Diz-se que uma cadeia ω é aceita pelo autômato finito M, quando, partindo-se do estado inicial $q_0$ de M, houver em R regras de transição, tais que seja possível transitar sucessivamente de um estado para outro de M, consumindo os símbolos da cadeia de entrada ω, até que se atinja um dos estados finais de M (pertencente a F) e a cadeia de entrada esteja esgotada.

**Autômato de Pilha Estruturado.** O autômato finito, no entanto, não reconhece o fenômeno dos aninhamentos que ocorre nas linguagens livres de contexto, pois não possui, em sua estrutura, uma memória auxiliar que armazene, para uso do autômato, a informação correspondente.

Para tratar de forma eficiente tais fenômenos, característicos das linguagens livres de contexto, um dos formalismos sugeridos, compatível com o desenvolvimento aqui apresentado, é o autômato de pilha estruturado [José-93] [José-94].

O autômato de pilha estruturado é uma ênupla $M= (Q, A, \Sigma, \Gamma, R, Z_0, q_0, F)$, sendo que:

- Q é o conjunto de estados de M.
- A é a coleção de submáquinas (similares aos autômatos finitos) que implementam M. Cada uma das submáquinas $a_i$, sendo i =1,...,n, é da forma:

  $$a_i = ( Q_i, \Sigma, R_i, q_{0,i}, F_i )$$

  sendo que $Q_i$ é uma partição de Q, $R_i$ uma partição de R, $q_{0,i}$ é o estado inicial de $a_i$, e $F_i \subseteq Q_i$, é o conjunto de estados de retorno de $a_i$.
- Σ é o alfabeto de entrada. As cadeias de entrada devem ser formadas por símbolos que pertençam a Σ.
- Γ é o alfabeto de pilha.
- R é o conjunto de regras de transição do autômato, que são descritas adiante.
- $Z_0$ é o indicador de pilha vazia ($Z_0 \in \Gamma$).
- $q_0$ é o estado inicial de M, devendo pertencer a Q.
- F é o conjunto de estados finais de M. ($F \subseteq Q$).

O autômato de pilha estruturado, assim, é formado por uma coleção de submáquinas, cada qual responsável por reconhecer uma determinada classe de sub-cadeias que poderá existir na cadeia de entrada a ser analisada.

A operação interna de uma submáquina é idêntica à de um autômato finito, denominando-se transição interna a transição entre estados de uma mesma submáquina. Quando é executada uma chamada de submáquina, o estado para onde deverá ocorrer futuramente o retorno deve ser empilhado, e o próximo estado do autômato é feito igual ao estado inicial da submáquina que se chamou. Quando do retorno de uma submáquina ao estado de retorno da submáquina que a chamou, desempilha-se o estado de retorno que foi empilhado na ocasião da chamada, e que deve então se tornar o próximo estado do autômato.

As regras de transição de um autômato de pilha estruturado têm a seguinte forma geral:

$$(\gamma g,\ e,\ s\alpha) : \rightarrow (\gamma g',\ e',\ s'\alpha)$$

sendo que *g* e *g'* são símbolos pertencentes a Γ, *e* e *e'* são estados pertencentes a Q, *s* e *s'* são vazios ou então símbolos pertencentes a Σ, $\gamma$ indica a parte da pilha que é irrelevante a esta transição, e $\alpha$ indica a parte da cadeia de entrada da qual a presente transição não depende e que ainda não foi reconhecida. A regra de transição especifica que, do estado *e*, pode-se transitar para o estado *e'*, desempilhando-se *g* e empilhando-se *g'*, com o consumo do símbolo *s* da cadeia de entrada (caso não seja vazio) e a inserção do símbolo *s'* na cadeia de entrada (caso não seja vazio).

A situação do autômato de pilha estruturado, em um determinado instante do reconhecimento de uma cadeia, é definida por três informações: o conteúdo corrente da pilha, o estado corrente e a parte da cadeia de entrada a ser analisada: $(\gamma g,\ e,\ s\alpha)$.

Diz-se que uma cadeia ω é aceita pelo autômato de pilha estruturado M, quando, partindo-se da situação inicial de M (pilha vazia $Z_0$, estado inicial $q_0$, cadeia de entrada completa ω), houver regras de transições em R, tais que seja possível transitar sucessivamente em M, podendo ou não haver eventuais chamadas a submáquinas (sempre retornando devidamente à submáquina que efetuou a chamada), consumindo-se os símbolos da cadeia de entrada ω, até que se atinja uma

situação final de M (pilha vazia $Z_0$, um dos estados finais de M, cadeia de entrada esgotada).

**Autômato Adaptativo.** O Autômato Adaptativo tem sua estrutura baseada no autômato de pilha estruturado, acrescido de ações adaptativas, que podem estar associadas às regras de transição. Dessa forma, o autômato adaptativo ganha o poder de reconhecimento de dependências de contexto, fenômeno cujo tratamento não é possível no autômato de pilha estruturado.

No autômato adaptativo, as regras de transição têm, portanto, a seguinte forma:

$$(\gamma g, e, s\alpha)\text{: A, } \rightarrow (\gamma g', e', s'\alpha), \text{B}$$

sendo que:

- $\gamma, g, e, s, \alpha, g', e', s'$ são definidos como no caso dos autômatos de pilha estruturados.
- $(\gamma g, e, s\alpha)$ corresponde à situação do autômato antes da transição.
- $(\gamma g', e', s'\alpha)$ corresponde à situação do autômato após a transição.
- A denota a ação adaptativa que deve ser executada antes da execução da transição representada pela regra.
- B denota a ação adaptativa que deve ser executada após a execução da transição.

A definição formal do Autômato Adaptativo é apresentada a seguir [Iwai-00]:

Um Autômato Adaptativo $M=(E_0, A)$ é constituído por um par ordenado ($E_0$, $A$), sendo que:

- $E_0$ é a máquina de estados inicial (autômato de pilha estruturado) que implementa $M$ :

  $E_0 = (Q_0, SM_0, \Sigma, \Gamma, R_0, Z_0, q_0, F)$, sendo que:

  - $Q_0$ é um conjunto (inicial) de estados pré-definido, que contém o estado inicial de $M$, $q_0 \in Q_0$.
  - $SM_0$ é a coleção de submáquinas que implementa $E_0$.
  - $\Sigma$ e $\Gamma$ são os alfabetos de entrada e de pilha, respectivamente, e $Z_0 \in \Gamma$ é um símbolo que, quando no topo da pilha, indica que esta se encontra vazia.

- $R_0$ é o conjunto inicial de regras de transição da forma definida anteriormente.
- $F$ é o conjunto de estados finais, $F \subseteq Q_0$ .

- $A$ é um conjunto fixo das ações adaptativas, responsáveis pela dinâmica da topologia do autômato.

As ações adaptativas que estão associadas às transições correspondem a chamadas de funções adaptativas. As funções adaptativas, por sua vez, são compostas de ações adaptativas elementares (apresentadas a seguir), declarações de variáveis (recebem valores em decorrência da execução das ações adaptativas elementares e não sofrem mais alterações até o término da função) e de geradores (recebem um valor único no início da execução da função e não são mais alterados até o término da função), e eventuais chamadas de funções adaptativas anteriores e posteriores à execução de suas ações adaptativas elementares.

Existem três tipos de ações adaptativas elementares:

- Ação de inspeção, denotada por ?[($\gamma g$, *e*, *s*$\alpha$) : A, $\rightarrow$ ($\gamma g'$, *e'*, *s'*$\alpha$), B ]. Especifica que a regra de transição cujo formato está denotado entre colchetes deve ser pesquisada no conjunto de regras de transição do autômato. As variáveis que denotam os componentes da regra pesquisada (*g*, *e*, *s*, *g'*, *e'*, *s'*) podem estar indefinidas e têm seus valores preenchidos de acordo com a regra encontrada. Esta ação é útil, assim, para se obter informações do autômato. Por exemplo, partindo-se de um estado, pode-se querer saber em qual estado se chega ao ser consumido determinado símbolo de entrada, ou qual símbolo de entrada deve ser consumido para se transitar entre dois estados indicados, etc. A execução deste tipo de ação tem preferência sobre as demais ações elementares.
- Ação de eliminação, denotada por -[($\gamma g$, *e*, *s*$\alpha$) : A, $\rightarrow$ ($\gamma g'$, *e'*, *s'*$\alpha$), B ]. Especifica que a regra de transição cujo formato está denotado entre colchetes deve ser eliminada do conjunto de regras de transição do autômato. A execução deste tipo de ação tem preferência sobre as ações elementares de inserção.

- Ação de inserção, denotada por +[($\gamma g$, *e*, $s\alpha$) : A, $\rightarrow$ ($\gamma g'$, *e'*, $s'\alpha$), B ]. Especifica que a regra de transição cujo formato está denotado entre colchetes deve ser inserida no conjunto de regras de transição do autômato.

As funções adaptativas também podem receber argumentos em suas chamadas, sendo que cada um dos argumentos pode assumir a forma de qualquer elemento do autômato adaptativo: nome de estado, símbolo do alfabeto de entrada, símbolo de pilha, nome de função adaptativa, nome de parâmetro de função, nome de variável ou nome de gerador.

Submetendo-se uma cadeia de entrada ω a um autômato adaptativo, sua operação ocorre conforme esboçado no algoritmo a seguir, que utiliza uma variável chamada REJEITADA (cujos valores são verdadeiro ou falso) para indicar se a cadeia de entrada ω foi rejeitada ou não, e uma variável chamada FINAL (cujos valores também são verdadeiro ou falso) para indicar se o processamento do algoritmo chegou ao fim, ou seja, se algum estado final foi atingido ou caso não haja mais nenhuma regra aplicável, sem que tenha se chegado a um estado final.

1. Comece a partir da situação inicial do autômato adaptativo: ($Z_0$, $q_0$, ω).
2. Atribua à variável REJEITADA o valor FALSO.
3. Atribua à variável FINAL o valor FALSO.
4. Pesquise, no conjunto de regras de transição do autômato, alguma regra que seja aplicável à atual situação. [Diz-se que uma regra de transição é aplicável quando a situação corrente do autômato corresponder àquele que estiver sendo especificado pelas informações que compõem o lado esquerdo da regra de transição (conteúdo do topo da pilha, estado corrente e átomo corrente da cadeia de entrada)].
5. **Se** existirem regras aplicáveis,
   **Então Se** existir apenas uma regra aplicável
   **Então** Execute a transição correspondente.
   **Senão** Escolha uma das regras de transição aplicáveis, segundo o critério: as transições internas a uma sub-máquina têm precedência sobre as demais, e, entre elas, transições que efetuam consumo de átomos têm precedência

sobre as transições em vazio. **Se** ainda restar mais de uma regra de transição aplicável, **Então**, configura-se uma situação de não-determinismo na operação do autômato, devendo-se nesse caso executar em paralelo todas as transições aplicáveis correspondentes.

**Senão Se** isto ocorrer em um estado não-final

**Então** Atribua à variável REJEITADA o valor VERDADEIRO. A cadeia é rejeitada pelo autômato, devendo-se considerar que a cadeia não pertence à linguagem definida pelo autômato adaptativo (a não ser que haja, em paralelo, outras tentativas de reconhecimento em curso, sendo processadas no autômato, ativadas de forma não-determinística e, neste caso, a rejeição da cadeia ocorrerá somente se todas as tentativas fracassarem).

Atribua à variável FINAL o valor VERDADEIRO.

**Senão** Atribua à variável FINAL o valor VERDADEIRO.

**Se** a cadeia de entrada não estiver esgotada ou a pilha não estiver vazia,

**Então** Atribua à variável REJEITADA o valor VERDADEIRO.

6. **Se** FINAL for FALSO,

**Então** Volte ao passo 4 deste algoritmo.

**Senão** Encerre o processamento.

A seguir, descreve-se o algoritmo de execução de uma regra de transição da forma:

$$(\gamma g,\ e,\ s\alpha)\text{: A, } \rightarrow (\gamma g',\ e',\ s'\alpha),\ \text{B}$$

1. **Se** A estiver presente,

**Então** Execute inicialmente a função adaptativa correspondente.

**Se** a função adaptativa tiver como efeito de sua execução eliminado essa regra de transição,

**Então** Aborte a execução dessa regra, voltando a localizar nova regra para ser aplicada.

2. **Se** *g* estiver especificado,

**Então** Desempilhe-o.

3. **Se** *g'* estiver especificado,
   **Então** Empilhe-o.
4. **Se** *s* estiver especificado,
   **Então** Consuma-o da cadeia de entrada.
   (O átomo corrente passa a ser o próximo símbolo da cadeia de entrada, caso exista.)
5. **Se** *s'* estiver especificado,
   **Então** Insira-o à esquerda do átomo corrente da cadeia de entrada.
6. Atribua ao estado corrente o estado *e'*.
7. **Se** B estiver presente,
   **Então** Execute a função adaptativa correspondente.

Autômatos adaptativos podem ser representados pela notação gráfica apresentada a seguir, na Figura 13.

1. Estado inicial: $x_0$
2. Transição interna: $(\gamma, x, s\alpha):\rightarrow (\gamma, y, \alpha)$
3. Transição de chamada de submáquina: $(\gamma, x, \alpha):\rightarrow(y, S_0, \varepsilon)$, onde S0 é o estado inicial da submáquina S
4. Retorno de submáquina:
5. Empilhamento de informação contextual: $(\gamma, x, \alpha):\rightarrow(\gamma, y, s\alpha)$
6. Desempilhamento de informação contextual: $(\gamma, x, s\alpha):\rightarrow(\gamma, y, \alpha)$
7. Transição com ações adaptativas: $(\gamma, x, s\alpha)$:A, $\rightarrow (\gamma, y, \alpha)$, B
   A : ação adaptativa anterior
   B : ação adaptativa posterior

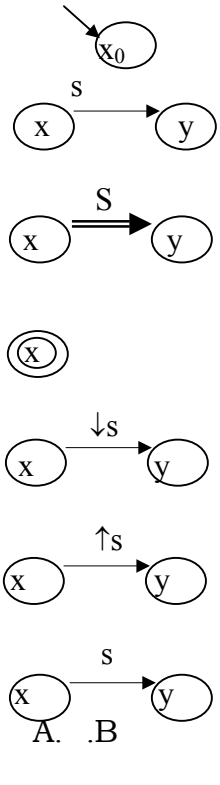



***Figura 13 - Notação gráfica utilizada para representar os autômatos adaptativos***

O seguinte exemplo de autômato adaptativo ilustra sua aplicação prática no reconhecimento de uma linguagem sensível ao contexto.

Seja a linguagem que aceita como sentenças válidas as expressões definidas a seguir, mas que só permite o uso das variáveis “a” e “b” se elas forem previamente declaradas.

PROG → (a|b|a,b|b,a) : $S_2$

$S_2$ → (a|b|<$S_2$>)((+|-|*|/)(a|b|<$S_2$>))*

Essa linguagem pode ser representada pelo autômato adaptativo ilustrado na Figura 14:

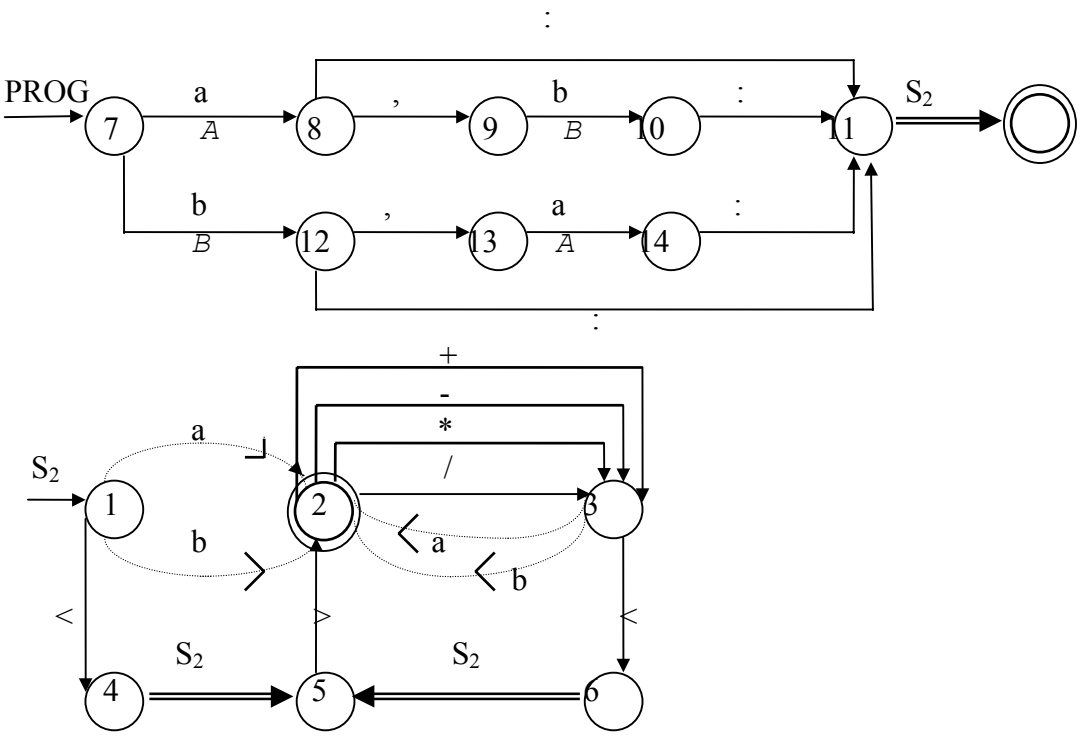



***Figura 14 - Exemplo de autômato adaptativo***

As transições representadas pelos arcos tracejados não existem na versão inicial do autômato adaptativo.

Quando a variável ‘a’ é encontrada na cadeia de entrada, a ação adaptativa *A*, executada na transição 7-8 ou 13-14 cria os arcos 1→2 e 3→2 com a variável ‘a’.

Da mesma forma, quando a variável ‘b’ é encontrada na cadeia de entrada, a ação adaptativa *B*, executada na transição 9-10 ou 7-12 cria os arcos 1→2 e 3→2 com a variável ‘b’.

Assim, os símbolos ‘a’, ‘b’ ou ambos, quando declarados à esquerda do sinal ‘:’ na sentença analisada, deverão ser aceitos nas expressões à direita de ‘:’, e rejeitados caso contrário, caracterizando a dependência de contexto da linguagem.

## 3.3.2. Gramáticas Adaptativas

O formalismo das gramáticas adaptativas foi apresentado pela primeira vez em [Iwai-00], dando continuidade à seqüência de atividades de pesquisa envolvendo formalismos adaptativos, desenvolvidos na Escola Politécnica da USP. Uma gramática adaptativa é um dispositivo generativo, do tipo sensível ao contexto, dual ao formalismo implementado pelos autômatos adaptativos, e mais adequado que estes para as etapas de projeto e estruturação de linguagens devido ao seu caráter gramatical.

Assim como o autômato adaptativo, a gramática adaptativa caracteriza-se por permitir a auto-modificação de seu conjunto de regras durante a sua utilização, ou seja, à medida que uma sentença da linguagem vai sendo por ela gerada. As modificações ocorrem quando da aplicação de regras de produção gramaticais que estejam associadas a ações adaptativas. A exemplo do que ocorre no caso dos autômatos adaptativos, a execução de uma ação adaptativa permite consultar, remover e adicionar regras de produção gramaticais, bem como alterar dinamicamente o conjunto dos símbolos não-terminais da gramática.

Para gerar uma sentença $\omega$ através de uma gramática adaptativa, parte-se de uma gramática inicial $G_0$, e, a cada ação adaptativa ativada durante a geração da sentença, cria-se uma gramática intermediária $G_i$, encerrando-se a geração da sentença em uma gramática final $G_n$.

Na gramática adaptativa, o conjunto de símbolos é constituído pelos símbolos não-terminais, pelos símbolos terminais e pelos símbolos de contexto.

As regras de produção dividem-se entre as que são aplicáveis às situações livres de contexto e as que são aplicáveis às situações dependentes de contexto e cada regra de produção pode estar associada a uma ação adaptativa.

As regras de produção aplicáveis às situações livres de contexto podem ter um dos seguintes formatos:

$$N \rightarrow \{A\}\ \alpha$$

$$N \rightarrow \phi$$

sendo N um símbolo não-terminal, $\alpha$ um símbolo não-terminal ou terminal, A uma ação adaptativa opcional associada à regra de produção e $\phi$ um meta-símbolo que indica o conjunto vazio. O primeiro formato de produção indica que o não-terminal N gera o símbolo $\alpha$ e o segundo formato de produção indica que não há nenhuma substituição prevista para o símbolo não-terminal N, sendo utilizada para os casos em que símbolos não-terminais referenciados pelas regras serão dinamicamente definidos, como resultado da execução de alguma ação adaptativa.

As regras de produção aplicáveis às situações dependentes de contexto podem ter um dos seguintes formatos:

$$\alpha N \leftarrow \{ A \}\ \beta M$$

$$\alpha N \rightarrow \{ A \}\ bM$$

sendo N e M símbolos não-terminais, $\alpha$ e $\beta$ símbolos de contexto, podendo $\alpha$ ser opcional, b um símbolo terminal opcional e A uma ação adaptativa opcional associada à regra de produção. O primeiro formato de produção indica que $\alpha$N deve ser trocado por $\beta$M na cadeia que estiver sendo gerada, inserindo assim, informações de contexto. O segundo formato de produção indica que $\alpha$N gera bM na cadeia de saída.

Uma gramática adaptativa pode ser formalmente definida como segue [Iwai-00]:

Uma gramática adaptativa G é constituída por uma tripla ordenada ($G^0$, T, $R^0$), sendo que

- T é um conjunto finito, eventualmente vazio, de funções adaptativas
- $R^0$ é uma relação que a cada regra de produção r, associa uma ação adaptativa A

  $R^0$ é, assim, uma relação (r,A ), na qual $r \in P^0=(P_L{}^0 \cup P_D{}^0)$ e $A \in T$, sendo:

  $P_L{}^0$ o conjunto de regras de produção aplicável às situações livres de contexto.

$P_D^0$ o conjunto de regras de produção aplicável às situações dependentes de contexto.

$R^0 \subseteq P^0 \times (T \cup \{\varepsilon\})$

- $G^0 = (V_N^0, V_T, V_C, P_L^0, P_D^0, S)$ é uma gramática inicial, sendo que
  - $V_N^0$ é um conjunto finito e não vazio de símbolos não-terminais,
  - $V_T$ é um conjunto finito e não vazio de símbolos terminais,

    $V_N^0 \cap V_T = \varnothing$
  - $V_C$ é um conjunto finito de símbolos de contexto.

    $V^0 = V_N^0 \cup V_T \cup V_C$

    $V_N^0$, $V_T$ e $V_C$ são conjuntos disjuntos dois a dois
  - $S \in V_N^0$ é o símbolo inicial da gramática.
  - $P_L^0$ é o conjunto de regras de produção aplicável às situações livres de contexto.
  - $P_D^0$ é o conjunto de regras de produção aplicável às situações dependentes de contexto.

As ações adaptativas, também ditas "chamadas de funções adaptativas" têm o formato de sua declaração semelhante ao das funções adaptativas definidas para os autômatos adaptativos:

Nome da ação (lista de parâmetros formais)
{ lista de variáveis, lista de geradores:
ação adaptativa opcional
ação elementar 1
ação elementar 2
...
ação elementar n
ação adaptativa opcional
}

As variáveis são elementos que são preenchidos durante a execução da função adaptativa, como resultado de ações adaptativas de consulta.

Os geradores são elementos que são automaticamente preenchidos no início da execução da função adaptativa, mantendo este valor durante toda a execução da função.

Como no caso dos autômatos adaptativos, estudado anteriormente, há três tipos de ações adaptativas elementares: inspeção, inserção e eliminação, representadas, respectivamente, das seguintes formas:

?[N → {ação_adaptativa_opcional} M]

+[N → {ação_adaptativa_opcional} M]

-[N → {ação_adaptativa_opcional} M]

sendo que N é um símbolo não-terminal e M é um símbolo.

A ação elementar de inspeção verifica a existência da regra de produção no formato indicado no conjunto corrente de produções da gramática. Caso exista, as variáveis serão preenchidas com os valores encontrados.

A ação elementar de inserção inclui na gramática uma nova regra de produção.

A ação elementar de eliminação consulta a existência de uma dada regra de produção e caso exista, as variáveis porventura utilizadas são preenchidas com os valores correspondentes. Além disso, nesse caso, efetua-se a exclusão da produção.

Quando houver mais de uma ação adaptativa a ser executada, as regras de inspeção têm precedência sobre as demais e as de eliminação têm precedência sobre as de inserção.

Como exemplo de gramática adaptativa, optou-se por um exemplo clássico de linguagem sensível ao contexto, representada pela expressão $a^n b^n c^n$. Cada sentença desta linguagem é formada por uma seqüência de terminais a's, seguida por uma seqüência de b's, e por fim uma seqüência de c's, sendo que a quantidade de símbolos de cada terminal é a mesma. Por exemplo, as sentenças são do tipo abc, aabbcc, aaabbbccc, etc.

A gramática adaptativa que representa esta linguagem é (extraído de [Iwai-00]):

$G=(G^0, T, F^0)$, sendo que

$G^0=(V_N^0, V_T, V_C, P_L^0, P_D^0, S)$

$V_N^0=\{S, E, A, A_1, B, B_1, C, C_1\}$

$V_T$={a,b,c}

$V_C$= ϕ

Os conjuntos de regras de produção são

$P_C^0$= ϕ

$P_L^0$= { S → EC
E → AB
A → { A (A, $A_1$, $B_1$, $C_1$)} a$A_1$
A → { B ($B_1$, $C_1$)} ε
B → $B_1$
C → $C_1$
$A_1$ → ϕ
$B_1$ → ϕ
$C_1$ → ϕ
}

O conjunto de ações adaptativas é

```
T = {
      A (t, x, y, z) = { x'*, y'*, z'* :
                          +[x → { A (x, x', y', z')} ax']
                          +[x → { B (y', z')} ε]
                          +[y → by']
                          +[z → cz']
                          +[x' → ϕ ]
                          +[y' → ϕ ]
                          +[z' → ϕ ]
                          -[t → { A (t, x, y, z)} ax]
                          +[t → ax]
                          -[t → { B (y', z')} ε ]
                         };
      B (x, y) = { +[x → ε ]
                   +[y → ε ]
                 }
    }
```

Em [Iwai-00] demonstra-se que o Autômato Adaptativo é equivalente à Gramática Adaptativa, constituindo dois formalismos duais, isto é, se uma linguagem é aceita por um Autômato Adaptativo, ela pode ser gerada por alguma Gramática Adaptativa, e fornece ainda algoritmos de conversão de autômatos adaptativos para gramáticas adaptativas e vice-e-versa.

O próximo item apresenta uma comparação entre os diversos formalismos apresentados neste capítulo.

# 3.4. Comparação entre os Formalismos Citados

Os formalismos citados neste capítulo, a saber: ATN, DCG, *Augmented Context-Free Grammar*, GPSG, Autômato Adaptativo e Gramática Adaptativa, podem ser empregados na análise de linguagem natural, pois todos possibilitam três mecanismos, que são essenciais para tal finalidade:

- construção da árvore sintática durante o reconhecimento da sentença.
- condições que devem ser satisfeitas para que se aplique a transição ou regra.
- tratamento de dependências de contexto.

No entanto, é interessante estabelecer algumas comparações entre esses formalismos. [Pereira-80] apresenta uma comparação entre os formalismos ATN e DCG, analisando as seguintes características: clareza, poder de geração, concisão, eficiência, flexibilidade e adequação ao trabalho teórico.

Com base nessas mesmas características, estabelece-se, a seguir, um comparativo entre os seis formalismos citados.

- *Clareza*.

  No artigo que introduz a ATN, [Woods-70] argumenta que a ATN provê o poder de uma gramática transformacional, porém mantendo a clareza do modelo da gramática livre de contexto.

  No entanto, segundo [Pereira-80], este é um ponto onde DCG é visivelmente melhor do que ATN, pois a DCG pode ser considerada como uma descrição de uma linguagem, sendo assim tão clara quanto as gramáticas livres de contexto. Observando-se uma regra, as suas conseqüências são imediatamente visíveis. Além disso, argumenta que DCG é mais modular, não apresenta atribuições e consiste de um formalismo uniforme de grande simplicidade, enquanto a ATN é um misto elaborado de dois formalismos (redes de transição e LISP), fazendo com que a DCG seja mais inteligível.

  Uma vez que [Woods-70] argumenta que ATN mantém a clareza do modelo de gramática livre de contexto, a *Augmented Context-Free Grammar,* a Gramática Adaptativa e a GPSG também mantêm tal clareza por poderem ser vistas como extensões da gramática livre de contexto.

  O Autômato Adaptativo também apresenta clareza por ser definido por transições, mas não é tão inteligível nos pontos em que referencia ações adaptativas.

- *Poder de geração.*

  Teoricamente, ATN e DCG têm o poder de uma máquina de Turing [Woods-70] [Bates-78] [Pereira-80], e assim, comportam a máxima generalidade possível.

  [Pereira-80] argumenta que a DCG tem essencialmente um mecanismo mais intuitivo e amigável para construção de estruturas do que a ATN.

  O Autômato Adaptativo, e, por equivalência, a Gramática Adaptativa, também têm o poder de uma máquina de Turing [Iwai-00], apresentando um grande poder de geração.

  O formalismo GPSG é o que apresenta o menor poder de geração, devido ao fato de permitir impor inúmeras restrições em seu formalismo [Gazdar et al.-85].

- *Concisão.*

  [Pereira-80] considera que a DCG é mais concisa que a ATN, pela mesma razão pela qual os programas lógicos são em geral mais concisos do que os programas escritos em linguagens convencionais.

  De todos os formalismos considerados neste item, DCG é o que parece ser mais conciso, e o Autômato Adaptativo e a Gramática Adaptativa parecem ser os menos concisos.

- *Eficiência.*

  DCGs podem ser expressas diretamente em uma linguagem de programação de propósito geral, o Prolog, não necessitando assim, de um interpretador ou compilador adicional. O formalismo ATN também é muito aderente à linguagem LISP, podendo ser eficientemente executado.

  O Autômato Adaptativo apresenta também uma boa eficiência por se basear em autômatos, pois, segundo [Menezes-00], um Autômato Adaptativo pode ser visto como um autômato finito por trechos e os autômatos finitos podem ser implementados de maneira muito eficiente. A Gramática Adaptativa, segundo [Iwai-00], no pior caso cresce em ordem quadrática, representando, em termos computacionais, um resultado polinomial de grau baixo, mesmo nos casos mais adversos.

- *Flexibilidade.*

ATN e DCG apresentam flexibilidade no sentido de que é possível alterar ou estender seus formalismos.

Tanto a *Augmented Context-Free Grammar* quanto o Autômato Adaptativo e a Gramática Adaptativa apresentam também essa característica. O Autômato Adaptativo e a Gramática Adaptativa, intuitivamente, parecem ser melhores, nesse aspecto, do que os demais formalismos em questão, pela sua capacidade de auto-modificação, e por permitirem, dessa maneira uma espécie de "aprendizado", processo em que o próprio autômato ou a própria gramática incorporam as mudanças necessárias à medida que vão operando, como se estivessem "aprendendo" pormenores acerca da linguagem analisada.

- *Adequação ao trabalho teórico.*

  ATN, DCG e GPSG provêm uma ponte, entre os lingüistas ou filósofos teóricos como Chomsky e Montague, e aqueles como Woods, preocupados com a implementação prática dos sistemas de linguagem natural.

  A *Augmented Context-Free Grammar,* o Autômato Adaptativo e a Gramática Adaptativa também compartilham essa característica, uma vez que se mostraram formalismos úteis para o desenvolvimento de pesquisas teóricas na área das linguagens.

Os formalismos de representação de linguagem considerados neste capítulo compartilham, no geral, as mesmas vantagens, e provêm os mesmos mecanismos essenciais à análise sintática de sentenças de linguagem natural.

Uma vez apresentados os fundamentos conceituais, descreve-se, no próximo capítulo, o desenvolvimento da proposta da dissertação.

# 4. DESENVOLVIMENTO DA PROPOSTA DA DISSERTAÇÃO

A verificação de que os Formalismos Adaptativos podem ser utilizados como forma de representação de informações no processamento de linguagem natural é feita por dois caminhos diferentes.

O primeiro consiste em mostrar que o Autômato Adaptativo pode representar a linguagem natural representada pelo *Augmented Transition Network*, sendo capaz de reconhecer a linguagem, gerar e armazenar a sua estrutura sintática. Para isto, mostra-se que é possível mapear qualquer *Augmented Transition Network* para um Autômato Adaptativo equivalente.

O segundo consiste em mostrar que o Formalismo Adaptativo pode representar a linguagem natural representada pela GPSG. Para isso, pretende-se mostrar que é possível mapear qualquer especificação GPSG de uma determinada linguagem natural para Formalismo Adaptativo.

Dessa forma, fica constatado que o Formalismo Adaptativo pode ser utilizado como ferramenta para análise sintática e para a representação das estruturas que compõem as sentenças da linguagem natural.

A escolha do formalismo ATN se deve ao fato dele ter sido, reconhecidamente, um padrão na área de representação de linguagens, sendo por isso, bastante empregado para essa finalidade, e também pelo fato de apresentar uma certa similaridade estrutural com o Autômato Adaptativo. Já o formalismo GPSG foi escolhido para representar a classe de formalismos baseados em restrições, que são os que têm sido mais empregados recentemente, e, apesar de constituir, dentre os

formalismos estudados, o mais restritivo e com menor poder de geração, sua adoção é fortemente motivada pelo fato de já existir disponível uma especificação da gramática da língua portuguesa em GPSG [Chin-96], desenvolvida por uma pessoa da área lingüística, o que não ocorre com os demais formalismos baseados em restrições.

São analisados também alguns exemplos de fragmentos da gramática da língua portuguesa representados em Autômato Adaptativo.

## 4.1. Mapeamento de Redes ATN para Autômato Adaptativo

Intuitivamente, pode-se perceber a existência de uma certa correspondência de estrutura entre ATN e Autômato Adaptativo, pois ambos se compõem de estados e transições, e possuem mecanismos para o tratamento sintático de casos de dependências de contexto.

Uma vez comprovada a existência de tal correspondência, pode-se concluir que o Autômato Adaptativo é capaz de representar a linguagem natural, à medida em que o ATN tem tal capacidade, reconhecendo a linguagem segundo as suas regras sintáticas e determinando a estrutura sintática correspondente.

Pretende-se, assim, indicar uma forma de mapeamento do ATN para Autômato Adaptativo, para através disto provar a correspondência entre esses dois formalismos.

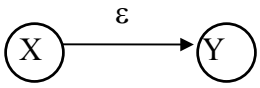
Analisando-se a notação ATN, buscou-se uma forma de representá-los através de um autômato adaptativo. Assim, para cada possível construção básica de ATN, estabelece-se uma correspondência com a construção equivalente do Autômato Adaptativo:

- No ATN, a notação de estado final:

corresponde à seguinte notação do Autômato Adaptativo:


- No ATN, a notação de arco do tipo JUMP:

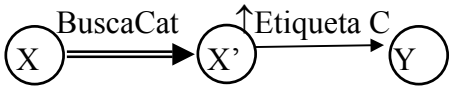

corresponde à notação de transição em vazio do Autômato Adaptativo:




- Transição não especificada no ATN corresponde à transição para um estado de erro no Autômato Adaptativo.

- A notação de arco do tipo CAT do ATN:

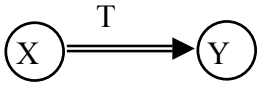

corresponde, no Autômato Adaptativo, ao seguinte diagrama:




sendo que BuscaCat é uma submáquina que verifica qual a categoria léxica da palavra da cadeia de entrada e insere na cadeia de entrada uma etiqueta correspondente a essa categoria léxica.

- A notação de arco do tipoPUSH do ATN:


corresponde, no Autômato Adaptativo, ao seguinte diagrama:



sendo que T é uma submáquina específica (ou seja, um autômato completo) capaz de reconhecer a classe sintática T.

Assim, considerando-se por exemplo, o seguinte ATN, que representa um analisador sintático da parte de uma sentença correspondente ao sintagma verbal, onde V representa a categoria léxica Verbo, SS representa outro ATN que analisa o sintagma substantivo (objeto direto do verbo), SP representa outra ATN que analisa o sintagma preposicional (objeto indireto do verbo):

```
(SV
  (0   (CAT V (TO 1)) )
  (1   (PUSH SS (TO 2))
       (JUMP 2)
       (POP) )
  (2   (PUSH SP (TO 2))
       (POP) ) )
```

que é assim representado:

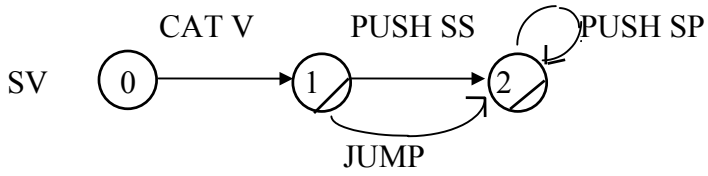



***Figura 15 - Rede ATN correspondente à Rede SV do exemplo.***

e que corresponde ao seguinte autômato adaptativo:

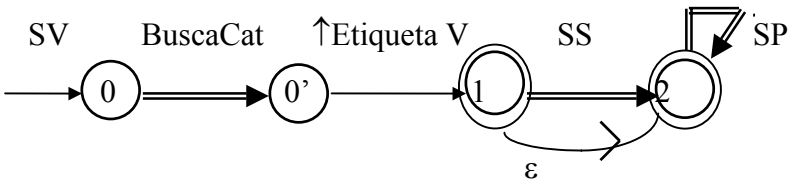



***Figura 16 - Autômato Adaptativo correspondente à Rede ATN SV.***

sendo SS e SP submáquinas correspondentes aos respectivos ATN's SS e SP. BuscaCat é a submáquina responsável por verificar qual a categoria léxica da palavra da cadeia de entrada, substituindo tal palavra por uma etiqueta que indica sua categoria léxica. Por exemplo, a palavra "escreve" seria reconhecida e substituída por uma etiqueta indicando que é um verbo (Etiqueta V).

Estendendo-se o raciocínio para os testes e ações contidos nos comandos ATN, tem-se o seguinte mapeamento:

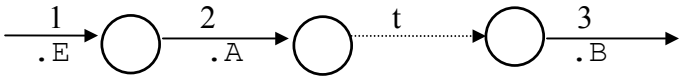
- A maioria dos arcos possuem o formato:

  `(ARCO <informação> <teste> <ação>* (TO <próx-estado>))`

  O tipo de arco e sua <informação> (que depende do tipo de arco) devem ser mapeados para uma transição do autômato adaptativo, conforme já foi descrito.

  A execução de <ação>* deve ser mapeada para uma função correspondente à ação adaptativa posterior associada a essa transição.

  Quando <teste> não for apenas T (true), a execução de <teste> deve ser mapeada para uma função correspondente à ação adaptativa anterior associada a essa transição. Nesse caso, devem ser inseridas algumas transições intermediárias que preparam o autômato para prosseguir somente quando <teste> for verdadeiro. Esse mapeamento é representado através do diagrama a seguir:

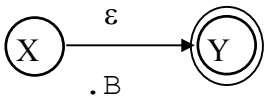



  Antes da transição que corresponde ao arco ATN (transição 2 do diagrama anterior), deve ser inserida uma transição em vazio (transição 1), associada a uma ação adaptativa posterior (`E`) que elimina a transição t, caso ela exista. Em seguida, deve ser criada a transição que corresponde ao arco ATN propriamente dito (transição 2), associada a uma ação posterior (`A`) que corresponde à execução de <teste>. Se teste for verdadeiro, deve inserir a transição t, como prosseguimento da transição que está sendo criada. No estado correspondente ao final da transição t, deve ser criada uma transição intermediária (transição 3), em vazio, associada a uma ação adaptativa posterior (`B`) que corresponde à execução de <ação>.

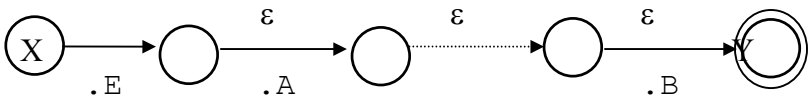
  Tanto em <teste> como em <ação>, podem existir chamadas de funções da linguagem LISP. Se existirem, essas funções também deverão ser mapeadas para Autômato Adaptativo. Como é possível escrever uma rede ATN sem a utilização de funções LISP, este trabalho concentrou-se apenas em mapear as

funções e arcos do ATN propriamente dito. Para que qualquer rede ATN que contenha funções LISP pudesse ser mapeada para Autômato Adaptativo, o ideal seria que houvesse um interpretador LISP, cuja implementação se baseasse em Autômato Adaptativo, isto é, um interpretador LISP que utilizasse o Autômato Adaptativo como linguagem de máquina.

Como exemplo do mapeamento de <ação> e <teste>, o arco POP, cujo formato é (POP <expressão> <teste>) deve ser mapeado para o seguinte autômato adaptativo:




quando <teste> for apenas T (true), sendo que B é a ação adaptativa posterior que corresponde à execução de <expressão>.
Quando <teste> não for apenas T (true), o arco POP deve ser mapeado para o seguinte autômato adaptativo:



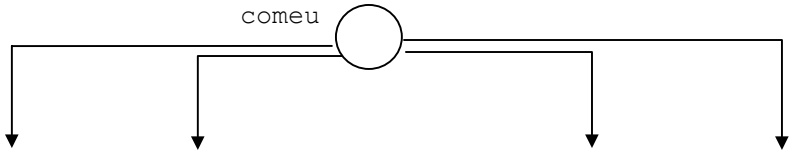

sendo que E é a ação adaptativa que elimina a transição representada através de linha tracejada, caso ela exista, A é a ação adaptativa posterior que corresponde à execução de <teste> e B é a ação adaptativa posterior que corresponde à execução de <expressão>.

Cada entrada léxica ATN também deve ser mapeada para um fragmento de autômato adaptativo, de forma que o item léxico corresponda a um estado inicial, a partir do qual partam transições correspondentes a cada uma de suas características para estados que correspondam aos respectivos valores das características.

Por exemplo, seja a seguinte entrada léxica:

```
comeu -   CAT=V   (Verbo),   VFORM=FIN   (Forma   Finita),
          RAIZ=COMER, PESSOA=3, NUMERO=SG (Singular)
```

Essa entrada léxica deve ser mapeada para o seguinte fragmento de autômato:



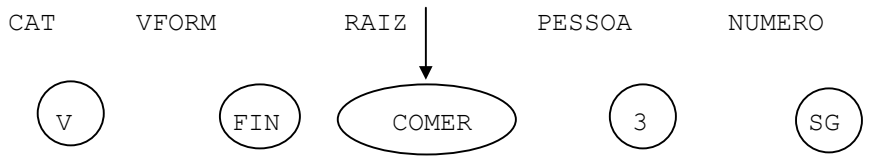



***Figura 17 - Autômato correspondente à entrada léxica 'comeu'.***

Utilizando-se o analisador e etiquetador morfológico desenvolvido em [Menezes-00], apresentado no item 3.1 desta dissertação, pode-se criar esses fragmentos de autômatos correspondentes a cada item léxico, à medida em que se lê o texto gerado por esse analisador morfológico.

Esse texto consiste em palavras seguidas de etiquetas morfológicas. Assim, o algoritmo de mapeamento léxico pode ler cada palavra, criar seu fragmento de autômato correspondente contendo suas informações léxicas, e substituir na cadeia de entrada essa palavra e suas etiquetas por uma etiqueta que representa a palavra. Essa etiqueta está associada, através de uma tabela e através de uma lista ligada implementada por intermédio de um autômato, à palavra que representa e ao início do fragmento do autômato que descreve suas informações léxicas.

Por exemplo, a sentença ‘O gato comeu o rato.’ é representada pela lista ligada:

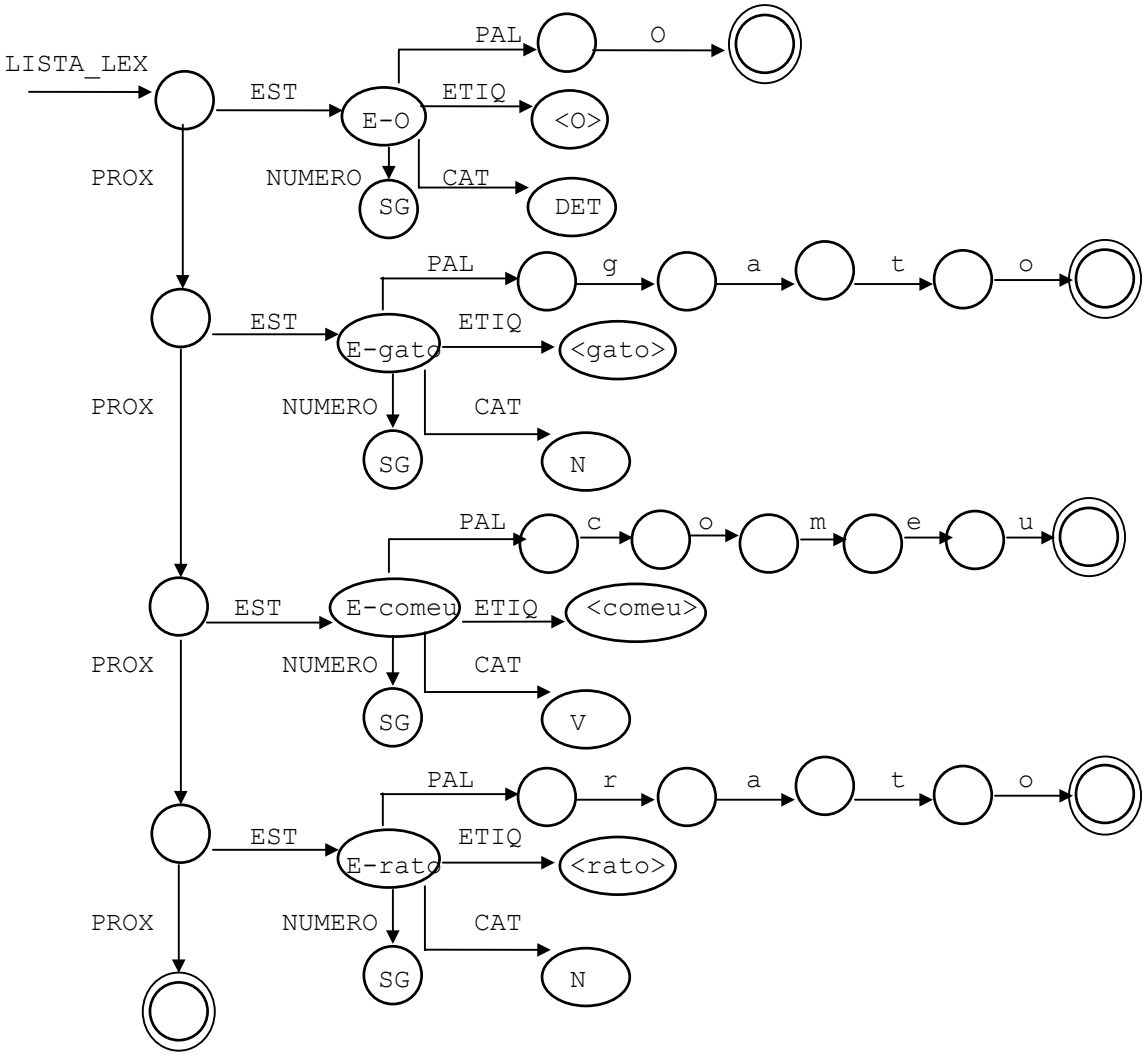



***Figura 18 - Autômato que representa a lista léxica correspondente à sentença 'O gato comeu o rato.'***

No exemplo ilustrado pela Figura 18, percebe-se que houve o reaproveitamento da entrada léxica da palavra ‘o’, que aparece duas vezes na sentença que está sendo representada. O início do autômato é apontado por LISTA_LEX. A transição ‘EST’ aponta para um estado ‘E-palavra’, correspondente ao estado que contém as informações sobre a palavra. A transição ‘PROX’ aponta para a próxima entrada da lista. A transição ‘PAL’ aponta para a representação da palavra em si, a transição ‘ETIQ’ aponta para o estado que contém a representação da etiqueta da palavra (<palavra>) e as demais transições correspondem aos valores léxicos e sintáticos da palavra. Por questões de simplicidade e clareza, na Figura 18, a ilustração do estado ‘N’ (nome) foi repetida para cada substantivo da sentença. O mesmo ocorreu para o

estado ‘SG’ (singular). Mas o autômato real terá apenas um estado ‘N’ e apenas um estado ‘SG’, que serão comuns a todas as palavras que os utilizam.

**Algoritmo de mapeamento das informações léxicas.**

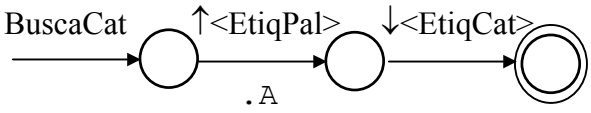
O algoritmo MapeamentoLexico, que processa a cadeia de entrada e cria o autômato correspondente à lista léxica é descrito a seguir:

1. Crie um estado inicial da lista, apontado por LISTA_LEX.
2. Atribua a PTR_LISTA o estado inicial da lista.
3. **Enquanto** houver palavras a serem lidas **faça**
   1a. Reconheça a palavra (finalizada por ‘/’).
   1b. **Se** essa palavra não existe na tabela ou **se** já existe, mas as características não são as mesmas
   **Então**
   - Insira-a na tabela.
   - Crie um ESTADO correspondente a essa palavra e insira esse estado na tabela.
   - Crie uma transição EST de PTR_LISTA para ESTADO.
   - Para cada etiqueta morfológica, crie um ESTADO_VALOR correspondendo ao valor da característica morfológica, caso ele já não exista, e crie uma transição de ESTADO para ESTADO_VALOR, cujo estímulo é a característica morfológica.
   - Crie uma etiqueta que representa essa palavra e insira essa etiqueta na tabela.
   - Crie um ESTADO_ETIQ correspondendo a essa etiqueta e crie uma transição ETIQ de ESTADO para ESTADO_ETIQ.
   - Crie um ESTADO_PAL correspondendo à palavra e cria uma transição PAL de ESTADO para ESTADO_PAL.
   - Crie a partir de ESTADO_PAL, transições e estados, de forma que cada transição corresponda a uma letra da palavra, finalizada por um estado final.
   - Crie um novo estado para a lista léxica e uma transição de PTR_LISTA para esse novo estado.
   - Atribua a PTR_LISTA esse novo estado.

   1c. Substitua na cadeia de entrada a palavra e suas etiquetas morfológicas pela etiqueta que representa essa palavra.
4. Faça PTR_LISTA ser o estado final do autômato.

```
5. Encerre o processamento.
```

A submáquina BuscaCat é responsável por reconhecer a etiqueta da palavra na cadeia de entrada e procurá-la na lista léxica, inserindo na cadeia de entrada a etiqueta correspondente à sua categoria léxica. O diagrama a seguir ilustra o funcionamento da submáquina BuscaCat:




**Algoritmo da ação adaptativa de BuscaCat, que percorre a lista léxica procurando a etiqueta da palavra e a etiqueta da categoria.**

Esse algoritmo, além de buscar a etiqueta da palavra e a etiqueta da categoria na lista léxica, devolve o estado ESTR_CORR (estrutura corrente) apontando para o estado correspondente à palavra que está sendo reconhecida.

```
1. Atribua a PTR_LISTA o estado inicial de LISTA_LEX.
2. Enquanto PTR_LISTA não for o fim do autômato faça
   2a. Atribua a ESTADO o estado apontado por PTR_LISTA,
       através da transição EST.
   2b. Se ESTADO aponta para a etiqueta da palavra que está
       sendo reconhecida, através da transição ETIQ
       Então Inclua uma transição de ESTR_CORR para ESTADO.
             Atribua a CATEG a etiqueta correspondente a categoria
             da palavra (estado apontado pela transição CAT).
             Encerre o processamento.
       Senão Atribua a PTR_LISTA o estado apontado por PTR_LISTA,
             através da transição PROX.
```

**Algoritmo de mapeamento de rede ATN para Autômato Adaptativo.**

Seja uma rede ATN no seguinte formato:

```
(Nome-da-rede
 (Estado1   (Arco1)
            ....
            (ArcoM))
.....
 (EstadoN   (Arco1)
            ....
            (ArcoX)))
```

Essa rede pode ser mapeada para autômato adaptativo através do seguinte algoritmo (Rotina MapeamentoRedeATN-AA):

1. Inicialize a lista de estados finais (deve conter o estado de erro).
2. Crie um EST_HOLD (estado HOLD), correspondente à lista HOLD desta rede.
3. **Se** houver declaração do tipo (REGS <reg>*)
   **Então Enquanto** houver <$reg_i$> **faça**
   Crie um estado $R_i$ e associe-o em uma tabela ao nome <$reg_i$>.
   Faça $R_i$ apontar para o estado Falso.
4. **Enquanto** houver estados a serem processados **faça**:
   4a. **Se** já existe um estado correspondente ao estado sendo processado
   **Então** Atribua a ESTADO_CORRENTE esse estado.
   **Senão** Crie um ESTADO_CORRENTE correspondente ao estado sendo processado.
   4b. **Se** for o primeiro estado da rede
   **Então** Faça ESTADO_CORRENTE ser o estado inicial dessa rede.
   4c. Atribua 'Falso' a FLG_BUSCACAT.
   4d. **Enquanto** houver arcos ATN a serem processados **faça**:
   I. **Se** o arco for do tipo (POP <expressão> <teste>)
   **Então** Crie um ESTADO_CORRENTE'.
   Acrescente ESTADO_CORRENTE' à lista de estados finais.
   **Se** <teste> for <T>
   **Então** Crie uma transição em vazio do ESTADO_CORRENTE para ESTADO_CORRENTE', cuja ação posterior seja a execução de <expressão>, tratada pela rotina TrataExpressão.Ação posterior também deve atribuir a ESTR_CORR para o estado apontado por RetornoExpr.
   **Senão** Chama a rotina PreparaTeste (ESTADO_CORRENTE, ESTADO_CORRENTE', ε, 'F').
   **Volte** ao passo 4d.
   II. **Se** o arco for do tipo (JUMP <estado> <teste> <ação>*)
   **Então Se** <teste> for <T>

**Então** Crie uma transição em vazio do ESTADO_CORRENTE para <estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.

**Senão** Chama a rotina PreparaTeste (ESTADO_CORRENTE, <estado>, ε, 'F').

**Volte** ao passo 4d.

III. **Se** o arco for do tipo (CAT <categoria> <teste> <ação>* (TO próx-estado))

**Então Se** já existe uma transição com a chamada da submáquina BuscaCat, a partir de ESTADO_CORRENTE

**Então** Atribua a ESTADO_CORRENTE' o estado destino dessa transição

Elimine a transição com o estímulo <categoria>, de ESTADO_CORRENTE' para estado de erro.

**Senão** Crie um ESTADO_CORRENTE'.

Crie uma transição com a chamada da submáquina BuscaCat, do ESTADO_CORRENTE para o ESTADO_CORRENTE'.

**Se** <teste> for <T>

**Então** Crie uma transição com o estímulo <categoria>, com o consumo da cadeia, de ESTADO_CORRENTE' para <próx-estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.

**Senão** Chama a rotina PreparaTeste (ESTADO_CORRENTE', <próx-estado>, <categoria>, 'F').

**Se** FLG_BUSCACAT for 'Falso'

**Então** Crie transições com estímulos diferentes de <categoria> do ESTADO_CORRENTE' para um estado final de erro.

Atribua 'Verdadeiro' a FLG_BUSCACAT.

**Volte** ao passo 4d.

IV. **Se** o arco for do tipo (PUSH <estado> <teste> <ação>* (TO <próx-estado>))

**Então Se** <teste> for <T>

**Então** Crie uma transição com a chamada da submáquina <estado>, do ESTADO_CORRENTE

para <próx-estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.

**Senão** Chama a rotina PreparaTeste (ESTADO_CORRENTE, <próx-estado>, <estado>, 'V').

**Volte** ao passo 4d.

V. **Se** o arco for do tipo (WRD <palavra> <teste> <ação>* (TO próx-estado))

**Então** Procure qual a etiqueta correspondente a <palavra>.

**Se** <teste> for <T>

**Então** Crie uma transição com o estímulo <etiqueta da palavra>, com o consumo da cadeia, de ESTADO_CORRENTE para <próx-estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.

**Senão** Chama a rotina PreparaTeste (ESTADO_CORRENTE, <próx-estado>, <etiqueta da palavra>, 'F').

Crie transições com estímulos diferentes de <etiqueta da palavra> do ESTADO_CORRENTE para um estado final de erro.

**Volte** ao passo 4d.

VI. **Se** o arco for do tipo (MEM <lista de palavras> <teste> <ação>* (TO próx-estado))

**Então** Crie EST_INTERM1, EST_INTERM2, EST_INTERM3.

Crie uma transição em vazio, de ESTADO_CORRENTE para EST_INTERM1, cuja ação posterior é eliminar a transição entre EST_INTERM2 e EST_INTERM3.

Crie uma transição em vazio, de EST_INTERM3 para <próx-estado>, cuja ação posterior seja a execução de <ação>, tratada pela rotina TrataAção.

**Enquanto** houver palavras em <lista de palavras> **faça**

- Procure qual a etiqueta correspondente a <palavra>.
- Crie uma transição com o estímulo <etiqueta da palavra>, com o consumo da cadeia, de

EST_INTERM1 para EST_INTERM2, cuja ação posterior seja a execução de <teste>, tratada pela rotina TrataTeste (se <teste> for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3).

Crie transições com estímulos diferentes de etiquetas de palavras de <lista de palavras>, do ESTADO_CORRENTE para um estado final de erro.

**Volte** ao passo 4d.

VII. **Se** o arco for do tipo (VIR <tipo-do-constituinte> <teste> <ação>* (TO próx-estado))

**Então** Crie EST_INTERM1, EST_INTERM2, EST_INTERM3, EST_INTERM4.

Crie uma transição em vazio, de ESTADO_CORRENTE para EST_INTERM1, cuja ação posterior é eliminar a transição entre EST_INTERM2 e EST_INTERM3 e a transição entre EST_INTERM3 e EST_INTERM4.

Crie uma transição em vazio, de EST_INTERM4 para <próx-estado>, cuja ação posterior seja a execução de <ação>, tratada pela rotina TrataAção.

Crie uma transição em vazio de EST_INTERM1 para EST_INTERM2, cuja ação anterior seja a execução de <teste <tipo-do-constituinte>>, tratada pela rotina TrataTesteTipoConstituinte. (se esse teste for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3) e cuja ação posterior seja a execução de <teste>, tratada pela rotina TrataTeste (se <teste> for Verdadeiro, então cria transição em vazio, de EST_INTERM3 para EST_INTERM4).

**Volte** ao passo 4d.

VIII. **Se** o arco não for nenhum dos tipos acima

**Então Imprima** "Arco ATN inválido: ", <arco>

**Volte** ao passo 4d.

5. Encerre o processamento.

**Algoritmo que prepara o autômato para a execução de <teste>.**

O algoritmo que mapeia uma rede ATN para Autômato Adaptativo chama este algoritmo (Rotina PreparaTeste) para preparar o autômato para a execução de <teste>. Recebe como argumentos:

EST_ORIG – estado origem do trecho do autômato que contém o <teste>

EST_DEST – estado destino do trecho do autômato que contém o <teste>

ESTÍMULO – estímulo da transição a ser criada, se FLAG_SUBM for 'F' ou estado inicial da submáquina a ser chamada, se FLAG_SUBM for 'V'.

FLAG_SUBM – flag que indica se é ou não chamada de submáquina

1. Crie EST_INTERM1, EST_INTERM2, EST_INTERM3.
2. Crie uma transição em vazio, de EST_ORIG para EST_INTERM1, cuja ação posterior seja a eliminação da transição entre EST_INTERM2 e EST_INTERM3.
3. **Se** FLAG_SUBM for igual a 'F'
   **Então** Crie uma transição de EST_INTERM1 para EST_INTERM2, com estímulo ESTÍMULO, cuja ação posterior seja a execução de <teste>, tratada pela rotina TrataTeste. (Se <teste> for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3).
   **Senão** Crie uma transição de EST_INTERM1 para EST_INTERM2, com a chamada da submáquina ESTÍMULO, cuja ação posterior seja a execução de <teste>, tratada pela rotina TrataTeste (Se <teste> for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3).
4. Crie uma transição em vazio, de EST_INTERM3 para EST_DEST, cuja ação posterior seja a execução de <ação>, tratada pela rotina TrataAção.
5. Encerre o processamento.

**Algoritmo que trata <ação> e cria a ação adaptativa posterior.**

O algoritmo MapeamentoRedeATN-AA, que mapeia uma rede ATN para Autômato Adaptativo chama este algoritmo (Rotina TrataAção) para tratar <ação>.

1. **Enquanto** houver <ação> a ser processada **faça:**
   /* a <ação> pode ser do tipo (SETR <reg> <expressão>), (SENDR <reg> <expressão>), (LIFTR <reg> <expressão>) ou (HOLD <tipo-do-constituinte> <expressão>) */
   1a. Chame a rotina TrataExpressão que procede o tratamento de <expressão> e retorna RetornoExpr apontando para o estado que corresponde ao novo valor de <reg>.
   1b. **Se** <ação> é HOLD
      **Então** Atribua a Ri o estado que corresponde a HOLD.
            Crie uma transição de $R_i$ para o estado apontado por RetornoExpr, com estímulo <tipo-do-constituinte>.
      **Senão** Ache o $R_i$ que corresponde a <reg>. Se não houver tal $R_i$, então Imprima "Registrador <reg> não foi definido".
            **Se** $R_i$ e RetornoExpr não apontam para o mesmo estado
            **Então** Elimine a transição de $R_i$ para o estado para o qual aponta atualmente.
                  Crie uma transição de $R_i$ para o estado apontado por RetornoExpr.
2. Encerre o processamento.

**Algoritmo que trata <expressão> contida na <ação>.**

O algoritmo que trata <ação> utiliza a rotina TrataExpressão que trata a parte <expressão> contida na sintaxe da ação. Essa rotina é descrita a seguir:

1. **Se** a <expressão> for do tipo *
   **Então Retorne** o estado para o qual o ESTR_CORR aponta.
2. **Se** a <expressão> for do tipo 'palavra
   **Então** Crie um estado correspondente a palavra.
         **Retorne** esse estado.
3. **Se** a <expressão> for do tipo (GETR <reg>)
   **Então** Ache $R_i$ correspondente a <reg>.
         **Retorne** o estado para o qual o $R_i$ aponta.

4. **Se** a <expressão> for do tipo (GETF <característica-sintática> <palavra>)
   **Então Se** <palavra> estiver especificada
   **Então** Ache $R_i$ correspondente a <palavra>.
   **Senão** Atribua a $R_i$ o estado apontado por ESTR_CORR.
   **Retorne** o estado para o qual o $R_i$ aponta com estímulo <característica-sintática>.
5. **Se** a <expressão> for do tipo (APPEND <reg> <expressão>)
   **Então** Ache $R_i$ correspondente a <reg>.
   Chame a rotina TrataExpressão que procede o tratamento de <expressão> e retorna RetornoExpr apontando para o estado que corresponde à <expressão>.
   Crie uma transição de $R_i$ para o estado apontado por RetornoExpr.
   **Retorne** o estado $R_i$.
6. **Se** a <expressão> for do tipo (BUILD <formato> <expressão>*)
   **Então** Crie um EST_BUILD_INICIAL.
   Atribua a EST_BUILD_PTR o EST_BUILD_INICIAL.
   **Enquanto** houver símbolos em <formato> **faça**
   **Se** o símbolo for uma palavra
   **Então** Crie uma transição de EST_BUILD_PTR para um estado novo, com estímulo igual a essa palavra.
   Atribua a EST_BUILD_PTR esse estado novo.
   **Senão Se** o símbolo for igual a '+'
   **Então** Ache $R_i$ correspondente a $<reg_i>$.
   Crie uma transição de EST_BUILD_PTR para $R_i$.
   **Senão Se** o símbolo for igual a '#'
   **Então** Chame a rotina TrataExpressão que procede o tratamento da <expressão> correspondente e retorna RetornoExpr apontando para o estado que corresponde ao resultado de <expressão>.
   Crie uma transição de EST_BUILD_PTR para o estado apontado por RetornoExpr.
   **Senão** Crie uma transição de EST_BUILD_PTR para o estado apontado por ESTR_CORR.

**Retorne** EST_BUILD_INICIAL.

**Algoritmo que trata <teste> e cria a ação adaptativa correspondente.**

O algoritmo MapeamentoRedeATN-AA, que mapeia uma rede ATN para Autômato Adaptativo e o algoritmo PreparaTeste, que prepara o autômato para a execução de <teste> chamam este algoritmo (Rotina TrataTeste) para tratar <teste>.

1. **Se** o <teste> é do tipo T
   **Então Retorne** o estado Verdadeiro.
2. **Se** o <teste> é do tipo (NULLR <reg>)
   **Então** Ache o estado $R_i$ correspondente a <reg>.
   **Retorne** o estado para o qual o $R_i$ aponta.
3. **Se** o <teste> é do tipo (AGREE $<expressão_1>$ $<expressão_2>$)
   **Então** Chame a rotina TrataExpressão que procede o tratamento de $<expressão_1>$ e retorna $RETORNO\_EXPR_1$ apontando para o estado que corresponde à $<expressão_1>$.
   Chame a rotina TrataExpressão que procede o tratamento de $<expressão_2>$ e retorna $RETORNO\_EXPR_2$ apontando para o estado que corresponde à $<expressão_2>$.
   **Se** $RETORNO\_EXPR_1$ e $RETORNO\_EXPR_2$ apontam para o mesmo estado
   **Então Retorne** o estado Verdadeiro.
   **Senão Retorne** o estado Falso.
4. **Se** o <teste> é do tipo (GETR <reg>)
   **Então** Ache o estado $R_i$ correspondente a <reg>.
   **Se** $R_i$ aponta para o estado Falso
   **Então Retorne** o estado Falso.
   **Senão Retorne** o estado Verdadeiro.
5. **Se** o <teste> é do tipo (AND $<teste_1>$ $<teste_2>$)
   **Então** Chame a rotina TrataTeste que procede o tratamento de $<teste_1>$ e retorna $RETORNO\_TESTE_1$ apontando para o estado que corresponde à $<teste_1>$.
   Chame a rotina TrataTeste que procede o tratamento de $<teste_2>$ e retorna $RETORNO\_TESTE_2$ apontando para o estado que corresponde à $<teste_2>$.
   **Se** $RETORNO\_TESTE_1$ e $RETORNO\_TESTE_2$ apontam para o estado Verdadeiro
   **Então Retorne** o estado Verdadeiro.
   **Senão Retorne** o estado Falso.
6. **Se** o <teste> é do tipo (OR $<teste_1>$ $<teste_2>$)

**Então** Chame a rotina TrataTeste que procede o tratamento de <teste$_1$> e retorna RETORNO_TESTE$_1$ apontando para o estado que corresponde à <teste$_1$>.
Chame a rotina TrataTeste que procede o tratamento de <teste$_2$> e retorna RETORNO_TESTE$_2$ apontando para o estado que corresponde à <teste$_2$>.
**Se** RETORNO_TESTE$_1$ e RETORNO_TESTE$_2$ apontam para o estado Falso
**Então Retorne** o estado Falso.
**Senão Retorne** o estado Verdadeiro.

7. **Se** o <teste> é do tipo (NOT <teste>)
**Então** Chame a rotina TrataTeste que procede o tratamento de <teste> e retorna RETORNO_TESTE apontando para o estado que corresponde à <teste>.
**Se** RETORNO_TESTE aponta para o estado Falso
**Então Retorne** o estado Verdadeiro.
**Senão Retorne** o estado Falso.

**Algoritmo que trata <teste <tipo-do-constituinte>> e cria a ação adaptativa correspondente.**

O algoritmo MapeamentoRedeATN-AA, que mapeia uma rede ATN para Autômato Adaptativo chama este algoritmo (Rotina TrataTesteTipoConstituinte) para tratar <teste <tipo-do-constituinte>>.

1. **Se** existe transição com estímulo <tipo-do-constituinte> a partir de EST_HOLD (estado correspondente à lista HOLD)
**Então Retorne** o estado Verdadeiro.
**Senão Retorne** o estado Falso.

Como exemplo da aplicação dos algoritmos propostos, seja a seguinte Rede ATN, que define um sintagma substantivo da gramática da língua portuguesa. Essa rede foi obtida adaptando-se um exemplo de Rede ATN, que define o sintagma substantivo da gramática da língua inglesa, extraído de [Bates-78].

```
(NP/
 (REGS DET NUMDET GENDET N NUM GEN PP)
 (NP/
   (CAT DET T
       (SETR DET *)
       (SETR NUMDET (GETF NUMERO))
       (SETR GENDET (GETF GENERO))
```

```
        (TO NP/DET))
    (CAT N T
        (SETR N *)
        (SETR NUM (GETF NUMERO))
        (SETR GEN (GETF GENERO))
        (TO NP/N3))
 )
 (NP/DET
    (CAT N T
        (SETR N *)
        (SETR NUM (GETF NUMERO))
        (SETR GEN (GETF GENERO))
        (TO NP/N1))
 )
 (NP/N1
    (JUMP NP/N2 (AND  (AGREE (GETR NUMDET) (GETR NUM))
                      (AGREE (GETR GENDET) (GETR GEN)))
 )
 (NP/N2
    (PUSH PP/ T (SETR PP *) (TO NP/PP1))
    (POP (BUILD (+ + + +) DET NUM GEN N) T)
 )
 (NP/PP1
    (PUSH PP/ T (SETR PP (APPEND PP *)) (TO NP/PP1))
    (POP T (BUILD (+ + + + +) DET NUM GEN N PP))
 )
(NP/N3
    (PUSH PP/ T (SETR PP *) (TO NP/PP2))
    (POP (BUILD (+ + +) NUM GEN N) T)
 )
 (NP/PP2
    (PUSH PP/ T (SETR PP (APPEND PP *)) (TO NP/PP2))
    (POP T (BUILD (+ + + +) NUM GEN N PP))
 )
)
```

Essa rede pode ser representada pelo seguinte diagrama:

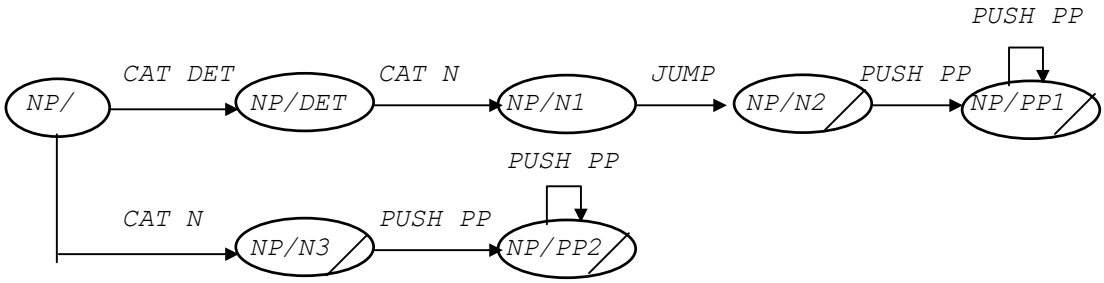
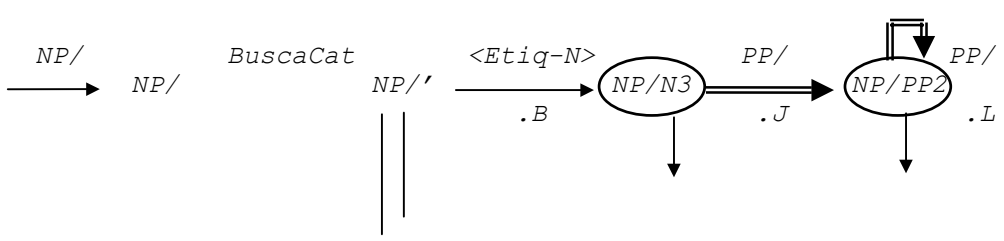



***Figura 19 - Rede ATN correspondente à Rede NP/.***

Submetendo-se essa rede ao algoritmo de mapeamento de Rede ATN para Autômato Adaptativo, obtém-se o autômato adaptativo ilustrado a seguir. A simulação passo a passo desse mapeamento é descrita no item 5.1 desta dissertação.



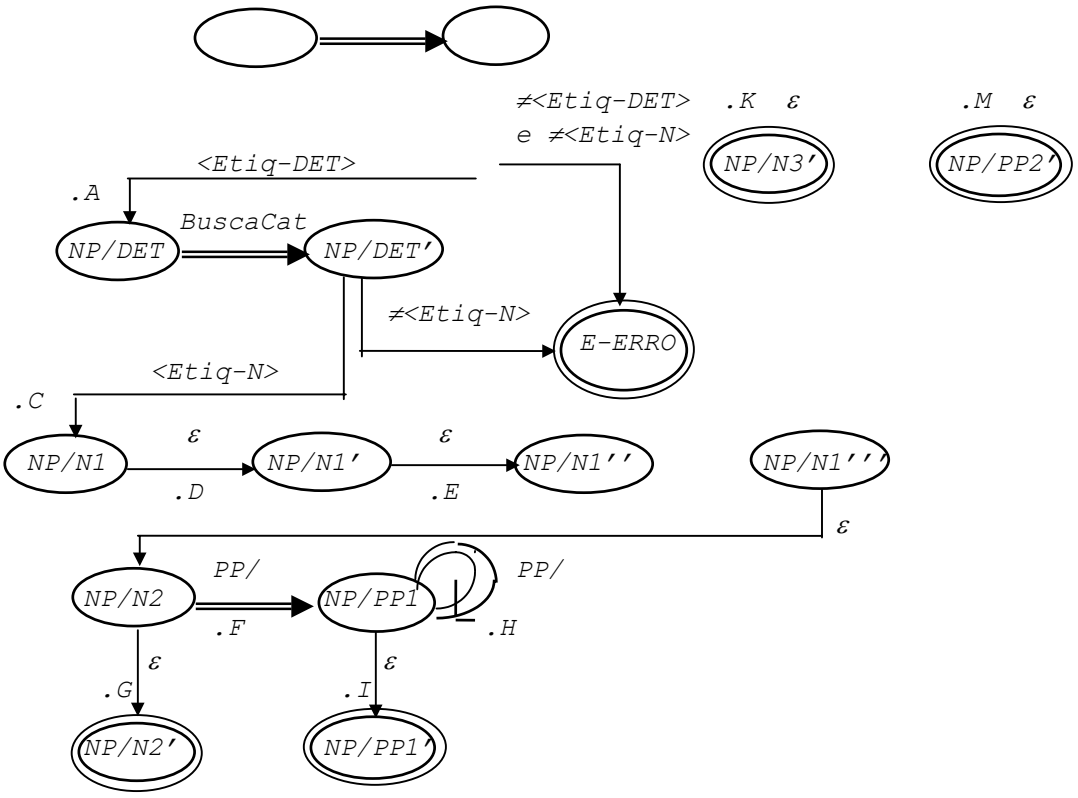



***Figura 20 - Autômato adaptativo correspondente ao mapeamento da Rede NP/.***

As ações adaptativas do autômato obtido pelo mapeamento estão descritas no item 5.1.1 deste trabalho.

Através dos algoritmos aqui propostos, é possível, assim, mapear uma rede ATN para um Autômato Adaptativo e, dessa forma, atinge-se o objetivo proposto de se verificar que o Autômato Adaptativo pode ser usado na representação e análise sintática do processamento de linguagem natural.

No item seguinte, propõe-se o mapeamento do formalismo GPSG para o Formalismo Adaptativo.

## 4.2. Mapeamento de GPSG para Formalismo Adaptativo

Como já foi mencionado anteriormente, o GPSG foi o formalismo escolhido para representar a classe de formalismos baseados em restrições, que correspondem aos formalismos mais empregados recentemente para representar a linguagem

natural. Apesar das suas limitações, o GPSG foi eleito para ser utilizado no desenvolvimento da proposta devido, principalmente, ao fato de já existir publicada uma especificação da gramática da língua portuguesa nessa notação.

Assim como foi feito para o ATN e o Autômato Adaptativo, pretende-se indicar uma forma de mapeamento da GPSG para um Formalismo Adaptativo, no caso, o Autômato Adaptativo, e, assim, provar que qualquer especificação GPSG de uma determinada linguagem natural pode ser mapeada para uma representação em Autômato Adaptativo. Uma vez comprovada a existência desse mapeamento, pode-se concluir que o Autômato Adaptativo é capaz de representar a linguagem natural, à medida em que a GPSG tem tal capacidade, reconhecendo a linguagem segundo as suas regras sintáticas e determinando a estrutura sintática correspondente.

Primeiramente, cada entrada léxica GPSG deve ser mapeada para um fragmento de autômato adaptativo, de forma que o item léxico corresponda a um estado inicial, a partir do qual partam transições correspondentes a cada uma de suas características para estados que correspondam aos respectivos valores das características.

Por exemplo, seja a seguinte entrada léxica:

```
preferiu [N-, V+, BAR 0, SUBCAT n, PAST+,
          VFORM FIN, AGR NP[PER 3, PLU-]]
```

Essa entrada léxica deve ser mapeada para o seguinte fragmento de autômato:

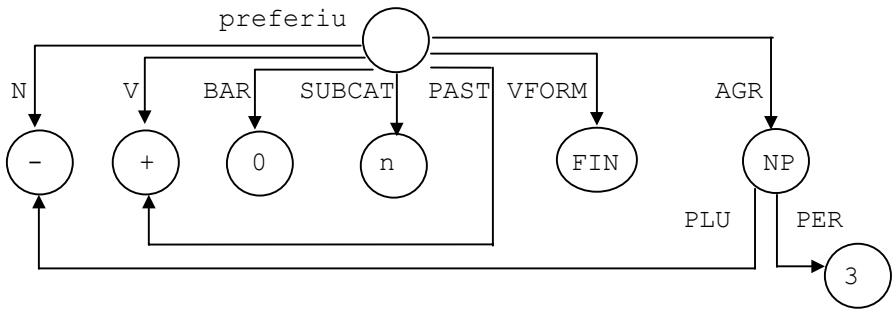



***Figura 21 - Autômato correspondente à entrada léxica 'preferiu'.***

Observa-se, no exemplo, que no mapeamento léxico, há o aproveitamento de estados que correspondem a valores repetidos de características, como é o caso do estado que representa o valor '-' e é compartilhado pelas transições 'N' e 'PLU'.

Como foi explicado no mapeamento de Rede ATN para Autômato Adaptativo, utilizando-se o analisador e etiquetador morfológico desenvolvido em [Menezes-00], apresentado no item 3.1 desta dissertação, pode-se criar esses fragmentos de

autômatos correspondentes a cada item léxico conforme se lê o texto gerado por esse analisador morfológico.

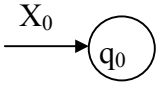
Esse texto consiste em palavras seguidas de etiquetas morfológicas. Assim, o algoritmo de mapeamento léxico pode ler cada palavra, criar seu fragmento de autômato correspondente contendo suas informações léxicas, e substituir na cadeia de entrada essa palavra e suas etiquetas por uma etiqueta que representa a palavra. Essa etiqueta está associada, através de uma tabela e através de uma lista ligada implementada por intermédio de um autômato, à palavra que representa e ao início do fragmento do autômato que descreve suas informações léxicas.

Como exemplo, a sentença 'O gato comeu o rato.' é representada pela lista ligada ilustrada na Figura 18 do item 4.1 desta dissertação.

Pode-se, assim, utilizar o mesmo algoritmo de mapeamento léxico (Rotina MapeamentoLexico), descrito no item 4.1, para realizar o mapeamento léxico da sentença que será submetida ao autômato adaptativo resultante do mapeamento da especificação GPSG.

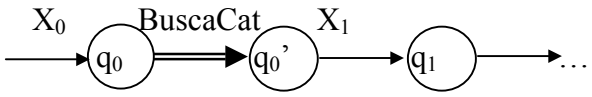
Uma regra ID $X_0 \rightarrow X_1, \ldots, X_n$ (básica ou derivada de uma meta-regra) de uma especificação GPSG pode ser mapeada para Autômato Adaptativo da seguinte forma:

- a categoria-mãe $X_0$ corresponderá a um autômato adaptativo, com um estado inicial $q_0$.

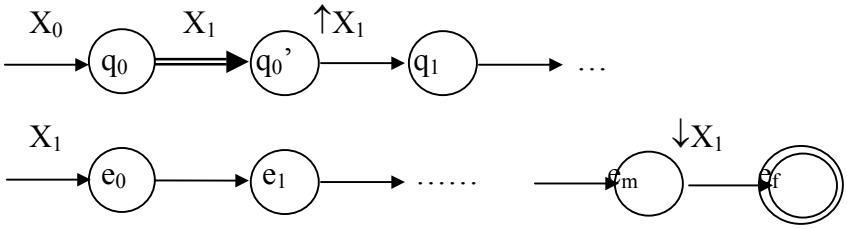



- cada categoria-filha $X_i$ corresponderá a duas transições desse autômato.
  - se $X_i$ corresponder a um item léxico, então ela será mapeada para duas transições: a primeira será uma chamada da submáquina BuscaCat e a segunda tem como estímulo a etiqueta correspondente à categoria léxica de $X_i$. A submáquina BuscaCat é a mesma que é utilizada no mapeamento de Rede ATN para Autômato Adaptativo e é descrita no item 4.1 desta dissertação. Ela é responsável por reconhecer a etiqueta da palavra na cadeia de entrada e procurá-la na lista léxica, inserindo na cadeia de entrada a etiqueta correspondente à sua categoria léxica e

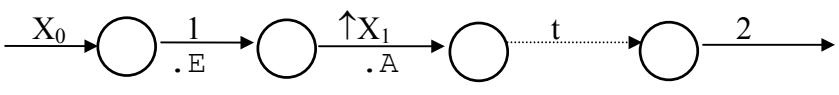
devolvendo o estado ESTR_CORR apontando para o estado que corresponde ao item léxico na lista léxica.




- se $X_i$ não corresponder a um item léxico, então ela será mapeada para duas transições: a primeira será uma chamada de uma sub-máquina do autômato adaptativo que trata essa categoria e que, em caso de sucesso, insere na cadeia de entrada uma etiqueta correspondente à categoria tratada; e a segunda transição terá como estímulo a etiqueta da cadeia de entrada correspondente a essa categoria, desempilhando-o da cadeia de entrada.

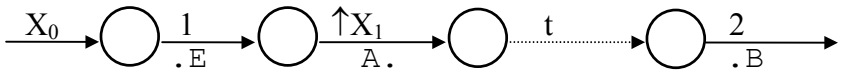



- cada transição com estímulo corresponderá a uma ação adaptativa anterior, que processará um teste para verificar se realmente a transição poderá ser processada. Esse teste consiste em conferir se as características do item léxico ou da categoria são compatíveis com a categoria-filha $X_i$ correspondente. Se tais características forem compatíveis, então essa transição poderá ser processada. O processamento dessa transição representa o reconhecimento da categoria-filha $X_i$ pelo autômato. Para que esse trecho do algoritmo possa ser executado várias vezes, sempre que houver teste a ser executado, devem ser inseridas algumas transições intermediárias que preparam o autômato para prosseguir somente quando o resultado do teste for verdadeiro. Nesse caso, esse mapeamento é representado através do diagrama a seguir:

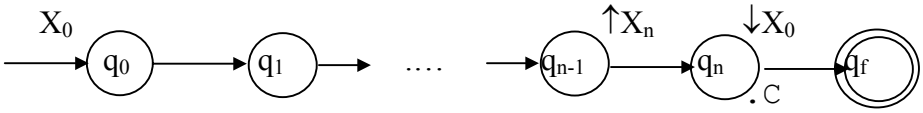


Antes da transição que corresponde à categoria-filha $X_1$, deve ser inserida uma transição em vazio (transição 1), associada a uma ação adaptativa posterior (E) que elimina a transição t, caso ela exista. Em seguida, deve ser criada a transição que corresponde à categoria-filha $X_1$ propriamente dita, associada a uma ação posterior (A) que corresponde à execução do teste. Se teste for verdadeiro, deve inserir a transição t, como prosseguimento da transição que está sendo criada. No estado correspondente ao final da transição t, deve ser criada uma transição intermediária (transição 2).

- Cada transição com estímulo corresponderá a uma ação adaptativa posterior, que criará um estado correspondente à categoria reconhecida e uma transição partindo desse estado para o estado inicial do item sendo tratado. Além disso, verificará se existe alguma característica não-nuclear instanciada para a categoria correspondente. Em caso afirmativo, cria uma transição, a partir do estado correspondente a essa categoria, para o estado correspondente a essa característica. No diagrama, essa ação posterior é associada à última transição em vazio (transição 2), que é executada após a verificação de que o resultado do teste é verdadeiro.




- Após o reconhecimento da última categoria-filha $X_n$ da regra, cria-se mais uma transição em vazio para um estado final, cuja ação posterior consistirá em:




  - Criar um estado que corresponda à categoria $X_0$, e criar transições entre $X_0$ e $X_1$, $X_0$ e $X_2$, e assim sucessivamente, até $X_0$ e $X_n$.
  - Verificar se existe concordância sintática entre as categorias da regra, ou seja, verificar se a regra satisfaz o princípio CAP: para cada par de categorias-filhas $X_i$ e $X_j$, tais que $X_i$ controla $X_j$, unificar as características de concordância de $X_i$ e $X_j$.

- Verificar se a regra satisfaz o princípio FFP: para cada característica não-nuclear instanciada de $X_i$ e não definida de $X_0$, acrescentar essa característica a $X_0$, e seu valor será a unificação dos valores dessa característica para todo $X_i$ que tiver essa característica instanciada.
- Verifica se a regra satisfaz o princípio HFC: unificar cada característica nuclear da categoria-filha nuclear $X_h$, com as características nucleares da categoria-mãe $X_0$.

A aplicação dos princípios CAP, FFP e HFC, assim como a aplicação do FSD no mapeamento sugerido por esta dissertação baseia-se no mapeamento do formalismo GPSG para o formalismo PATR, desenvolvido em [Shieber-88].

No mapeamento sugerido neste trabalho, por questões de simplicidade, supõe-se que as categorias-filhas da regra ID devem ocorrer na ordem em que estão dispostas na regra. Não é difícil alterar posteriormente o mapeamento para que aceite que as categorias-filhas ocorram em qualquer ordem.

Também não estão sendo consideradas, neste mapeamento, as restrições FCR (restrições sobre a simultaneidade de ocorrência de características), uma vez que é possível realizar um pré-processamento das regras GPSG para que essas restrições sejam impostas. Por exemplo, a seguinte FCR:

```
FCR: [V+, N-, BAR 2,FIN] ⊃ [CASO NOM]
```

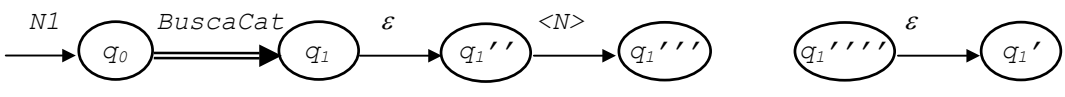
determina que toda regra que tiver alguma categoria com as especificações `[V+, N-, BAR 2,FIN]` também deverá ter a especificação `[CASO NOM]`.

Haverá apenas um autômato adaptativo para cada categoria-mãe $X_0$, ou seja, se houver mais de uma regra ID correspondente à mesma categoria-mãe $X_0$, então todas essas regras serão mapeadas para um único autômato adaptativo.

Por exemplo, sejam as seguintes regras ID:

```
(1)a.  N1 → H[1.0]
   b.  N1 → H[1.1], P2[de]
```

1a e 1b seriam mapeadas para o seguinte autômato adaptativo (a simulação passo a passo deste mapeamento encontra-se descrita no item 5.1 desta dissertação):



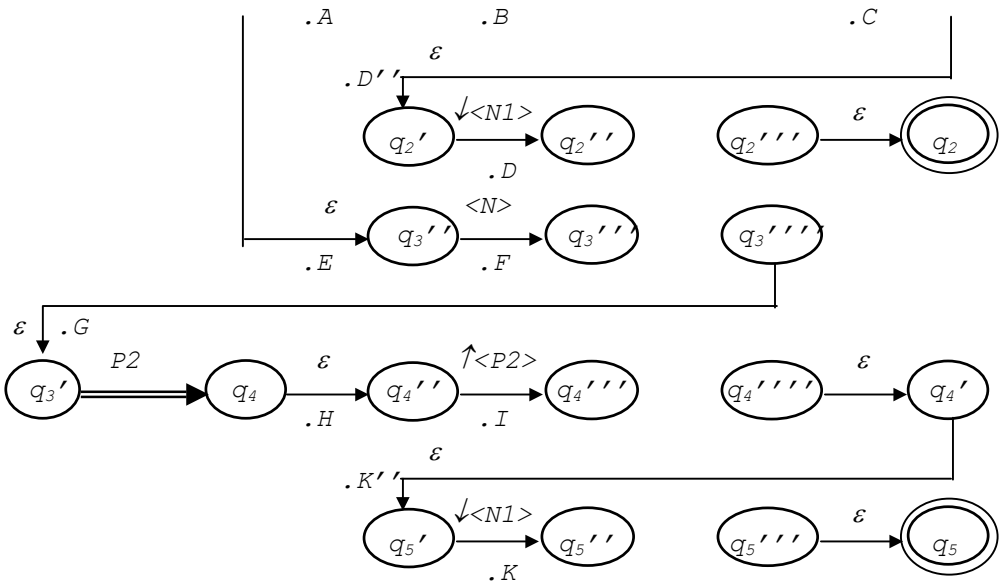



***Figura 22 - Autômato adaptativo correspondente ao mapeamento das regras 1a e 1b.***

A ação adaptativa A elimina a transição entre os estados $q_1$''' e $q_1$'''', caso tal transição exista.

A ação adaptativa B deve testar se as características de N[1.0] são válidas, ou seja, se o item léxico lido da cadeia de entrada tem as seguintes características e valores: N+, V-, BAR 0, SUBCAT 1.0. Esse teste pode ser processado, conferindo-se as transições e estados do fragmento de autômato correspondente ao item léxico lido. Se o resultado de execução do teste for verdadeiro, cria a transição entre os estados $q_1$''' e $q_1$''''

A ação adaptativa C cria um estado correspondente a N[1.0] e uma transição desse estado para o estado correspondente ao item léxico reconhecido. Verifica também se há alguma característica não-nuclear instanciada nessa categoria. Se houver, cria uma transição a partir do estado correspondente a essa categoria para o estado correspondente a essa característica.

A ação adaptativa D'' elimina a transição entre os estados $q_2$'' e $q_2$''', caso tal transição exista.

A ação adaptativa D cria um estado correspondente à categoria N1 e uma transição entre esse estado e o estado correspondente a N[1.0]. Além disso, deve verificar e garantir que a regra satisfaz aos princípios CAP, FFP e HFC. Se a regra satisfizer a todos esses princípios, cria a transição entre os estados $q_2$'' e $q_2$'''.

A ação adaptativa E elimina a transição entre os estados $q_3$''' e $q_3$'''', caso tal transição exista.

A ação adaptativa F deve testar se as características de N[1.1] são válidas, ou seja, se o item léxico lido da cadeia de entrada tem as seguintes características e valores: N+, V-, BAR 0, SUBCAT 1.1. Se o resultado de execução do teste for verdadeiro, cria a transição entre os estados $q_3$''' e $q_3$''''.

A ação adaptativa G cria um estado correspondente a N[1.1] e uma transição desse estado para o estado correspondente ao item léxico reconhecido. Verifica também se há alguma característica não-nuclear instanciada nessa categoria. Se houver, cria uma transição a partir do estado correspondente a essa categoria para o estado correspondente a essa característica.

A ação adaptativa H elimina a transição entre os estados $q_4$''' e $q_4$'''', caso tal transição exista.

A ação adaptativa I deve testar se as características de P2[de] são válidas, ou seja, se o item reconhecido na cadeia de entrada tem as seguintes características e valores: N-, V-, BAR 2, PFORM de. Se o resultado de execução do teste for verdadeiro, cria a transição entre os estados $q_4$''' e $q_4$''''.

A ação adaptativa K'' elimina a transição entre os estados $q_5$'' e $q_5$''', caso tal transição exista.

A ação adaptativa K cria um estado correspondente à categoria N1, uma transição entre esse estado e o estado correspondente a N[1.1], e uma transição entre esse estado e o correspondente a P2[de]. Além disso, deve verificar e garantir que a regra satisfaz aos princípios CAP, FFP e HFC. Se a regra satisfizer a todos esses princípios, cria a transição entre os estados $q_5$'' e $q_5$'''.

**Algoritmo de mapeamento de uma regra ID para autômato adaptativo.**

Seja uma regra ID $X_0 \rightarrow X_1, \ldots, X_n$ (básica ou derivada de uma meta-regra) de uma especificação GPSG, sendo que uma categoria $X_i$ ($1 \leq i \leq n$) denotada entre parênteses significa que essa categoria é opcional, e uma categoria $X_i$ ($1 \leq i \leq n$) denotada entre chaves significa que essa categoria pode ser repetida uma ou mais vezes. Essa regra pode ser mapeada para autômato adaptativo através do algoritmo (Rotina MapeamentoRegraID-AA) a seguir, que utiliza as seguintes variáveis:

TRANSICAO_CORR – armazena a transição corrente

ESTADO_CORR – armazena o estado corrente

ESTADO_ORIGEM_TRANS – estado de origem da transição a ser criada

ESTADO_DESTINO_TRANS – estado destino da transição a ser criada

ESTADO_FINAL_XI – estado final de reconhecimento da categoria $X_i$

PROXIMO_ESTADO – armazena o valor de próximo estado

1. Atribua a $X_i$ a categoria $X_1$.
2. **Se** já existe um autômato adaptativo correspondente a $X_0$,
   **Então** Atribua a ESTADO_CORR o estado inicial do autômato correspondente a $X_0$.
   Atribua a i o valor 1.
   Atribua a CATEG a categoria $X_i$.
   Inicialize CONT com 1.
   **Enquanto** CONT ≤ X0_CONT_CAT **faça**
   Atribua a IND o valor de CONT.
   Atribua a ACHOU o valor FALSO.
   **Enquanto** IND ≤ X0_CONT_CAT e não ACHOU **faça**
   **Se** na lista referente a X0, ESTADO_ORIGEM_TRANS correspondente ao índice IND for igual a ESTADO_CORR
   **Então** ACHOU= VERDADEIRO.
   **Senão** Incremente o valor de IND.
   **Se** ACHOU
   **Então Se** na lista referente a X0, CATEGORIA correspondente ao índice IND for igual a CATEG
   **Então** Atribua a ESTADO_CORR o ESTADO_FINAL_XI correspondente ao índice IND, na lista referente a X0.
   Incremente i.
   Atribua a CATEG o valor de $X_i$.
   Atribua a CONT o valor de IND.
   Atribua a ACHOU= FALSO.
   **Senão** Incremente CONT.
   **Senão** Atribua a CONT o valor de X0_CONT_CAT+1.
   Atribua a ESTADO_ORIGEM_TRANS o ESTADO_CORR.
   **Senão** Inicialize o contador de categorias de $X_0$: X0_CONT_CAT.
   Crie um estado inicial para $X_0$.
   Atribua a ESTADO_ORIGEM_TRANS o estado inicial.
3. Crie um estado para $X_i$.

4. Atribua a ESTADO_DESTINO_TRANS o estado criado para $X_i$.
5. **Se** $X_i$ corresponder a um item léxico
   **Então Se** a partir de ESTADO_ORIGEM_TRANS não existe uma chamada de submáquina BuscaCat
      **Então** Crie uma transição de ESTADO_ORIGEM_TRANS para ESTADO_DESTINO_TRANS, que é uma chamada à submáquina BuscaCat.
      **Senão** Atribua a ESTADO_DESTINO_TRANS o estado para o qual aponta a chamada da submáquina BuscaCat.
      Crie um estado e atribua seu valor a ESTADO_FINAL_XI.
      Chame a rotina PreparaTesteGPSG (ESTADO_DESTINO_TRANS, ESTADO_FINAL_XI, $X_i$, 'V', 'F'). Essa transição deve ser associada a uma ação adaptativa posterior: <ação>, que criará um estado correspondente à categoria reconhecida e uma transição partindo desse estado para o estado inicial do item sendo tratado, e verificará a existência de alguma característica não-nuclear instanciada para a categoria correspondente.
   **Senão** Crie uma transição de ESTADO_ORIGEM_TRANS para ESTADO_DESTINO_TRANS, que é uma chamada à submáquina $X_i$.
      Crie um estado e atribua seu valor a ESTADO_FINAL_XI.
      Chame a rotina PreparaTesteGPSG (ESTADO_DESTINO_TRANS, ESTADO_FINAL_XI, $X_i$, 'F', 'F'). Essa transição não é associada a nenhuma ação adaptativa posterior.
6. Crie um estado e atribua esse valor a PROXIMO_ESTADO.
7. **Se** $X_i$ está entre parênteses
   **Então** Crie uma transição em vazio de ESTADO_ORIGEM_TRANS para ESTADO_FINAL_XI.
8. **Se** $X_i$ está entre chaves
   **Então** Crie uma transição em vazio de ESTADO_FINAL_XI para ESTADO_ORIGEM_TRANS.
9. Insira em uma lista referente ao autômato $X_0$, o ESTADO_ORIGEM_TRANS, o ESTADO_FINAL_XI e a categoria-filha correspondente $X_i$ e atualiza o contador de categorias de $X_0$: X0_CONT_CAT.
10. Atribua a ESTADO_ORIGEM_TRANS o ESTADO_FINAL_XI.
11. Atribua a ESTADO_DESTINO_TRANS o PROXIMO_ESTADO.
12. **Se** $X_i$ for diferente de $X_n$,
   **Então** Atribua a $X_i$ a categoria $X_{i+1}$.
      **Volta** para Passo 5.

13. Chame a rotina PreparaTesteGPSG (ESTADO_ORIGEM_TRANS, ESTADO_DESTINO_TRANS, $X_0$, 'F', 'V'), que prepara o autômato para o teste de unificação e cria uma transição com inserção, na cadeia de entrada, de etiqueta correspondente a $X_0$, e com ação adaptativa posterior (tratada pela rotina ReconheceRegra) que monta a estrutura referente a $X_0$, devolvendo ESTR_CORR apontando para essa estrutura, e verifica se a regra satisfaz os princípios CAP, FFP e HFC.
14. Acrescente ESTADO_DESTINO_TRANS à lista de estados finais do autômato adaptativo correspondente a $X_0$.
15. Encerre o processamento.

**Algoritmo que prepara o autômato para a execução do teste.**

O algoritmo MapeamentoRegraID-ATN, que mapeia uma regra ID para Autômato Adaptativo, chama este algoritmo (Rotina PreparaTesteGPSG) para preparar o autômato para a execução do teste. Recebe como argumentos:

EST_ORIG – estado origem do trecho do autômato que contém o <teste>

EST_DEST – estado destino do trecho do autômato que contém o <teste>

ESTÍMULO – estímulo da transição a ser criada.

FLG_AÇÃO – flag que indica se deve ou não criar ação posterior final.

FLG_UNIF – flag que indica se a transição a ser criada é para testar unificação (reconhecimento de regra) ou não.

1. Crie EST_INTERM1, EST_INTERM2, EST_INTERM3.
2. Crie uma transição em vazio, de EST_ORIG para EST_INTERM1, cuja ação posterior seja a eliminação da transição entre EST_INTERM2 e EST_INTERM3.
3. **Se** FLG_UNIF = 'F'

   **Então** Crie uma transição de EST_INTERM1 para EST_INTERM2, com estímulo ESTÍMULO, cuja ação posterior seja a execução de <teste>, tratada pela rotina TrataTesteGPSG. (Se <teste> for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3).

   **Senão** Crie uma transição de EST_INTERM1 para EST_INTERM2, com inserção na cadeia de entrada, do estímulo ESTÍMULO, cuja ação posterior (tratada pela rotina ReconheceRegra) monta a estrutura referente a $X_0$, devolvendo ESTR_CORR apontando para essa estrutura, e verifica se a regra satisfaz os princípios CAP, FFP e

HFC. (Se o Retorno de ReconheceRegra for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3).

4. **Se** FLG_AÇÃO = 'V'
   **Então** Crie uma transição em vazio, de EST_INTERM3 para EST_DEST, cuja ação posterior seja a execução de <ação>, tratada pela rotina TrataAçãoGPSG.
   **Senão** Crie uma transição em vazio, de EST_INTERM3 para EST_DEST.
5. Encerre o processamento.

**Algoritmo que cria a ação adaptativa correspondente ao teste das características da categoria-filha $X_i$ de uma regra ID.**

O passo 5 do algoritmo de mapeamento de uma regra ID para autômato adaptativo chama a rotina PreparaTesteGPSG, que cria uma transição que corresponde ao reconhecimento da categoria-filha $X_i$ da regra. Essa transição está associada a uma ação adaptativa posterior que processa o teste das características de $X_i$. O algoritmo (Rotina TrataTesteGPSG), que cria essa ação adaptativa, é descrito a seguir.

1. Crie uma ação adaptativa.
2. Acrescente à ação adaptativa:
   - inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.
   - **Se** o valor de BAR é zero
     **Então Se** houver alguma FSD léxica especificando um valor default para alguma característica p,
       **Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p
         **Se** não houver tal transição,
         **Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.
     **Senão Se** houver alguma FSD não léxica especificando um valor default para alguma característica p,
       **Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p
         **Se** não houver tal transição,

**Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.

3. Para cada uma das características de $X_i$, acrescente à ação adaptativa:
   - inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo que representa a característica de $X_i$.
   - **Se** não houver tal transição,
     **Então Retorne** estado Falso.
   - **Se** o estado que representa o valor corresponde a essa característica de $X_i$,
     **Então Retorne** estado Falso.
4. **Retorne** estado Verdadeiro.

**Algoritmo que cria a ação adaptativa posterior da transição que corresponde ao reconhecimento da categoria-filha $X_i$ de uma regra ID.**

O passo 5 do algoritmo de mapeamento de uma regra ID para autômato adaptativo chama a Rotina PreparaTesteGPSG, que cria uma transição que corresponde ao reconhecimento da categoria-filha $X_i$ da regra. Essa transição está associada a uma ação adaptativa posterior que monta uma pequena estrutura contendo as informações da categoria-filha $X_i$ que está sendo reconhecida (cria um estado correspondente à categoria reconhecida e uma transição partindo desse estado para o estado inicial do item sendo tratado). Além disso, verifica se há alguma característica não-nuclear instanciada para a categoria correspondente. Se houver, cria uma transição a partir do estado correspondente a essa categoria para o estado correspondente a essa característica. O algoritmo TrataAçãoGPSG, que cria essa ação adaptativa, é descrito a seguir.

1. Crie uma ação adaptativa.
2. Crie um ESTADO_XI correspondente a $X_i$.
3. Acrescente à ação adaptativa:
   - inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.
   - inserção da transição ESTADO_XI para ESTADO_INFO_XI, com estímulo igual ao nome da categoria referente a $X_i$.
   - inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR.

- **Se** o valor de BAR é zero,
  **Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo igual a alguma característica não-nuclear
  **enquanto** houver alguma transição desse tipo e corresponder a alguma característica que não pertence a $X_i$ (característica instanciada) **faça**
  inspeção para verificar se de ESTADO_XI parte alguma transição com estímulo FOOT_INST.
  **Se** não houver tal transição,
  **Então** inserção da transição de ESTADO_XI para um estado novo com estímulo FOOT_INST.
  copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.

4. **Retorne.**

**Algoritmo que cria a ação adaptativa posterior da transição que corresponde ao final do reconhecimento de uma regra ID.**

O passo 13 do algoritmo de mapeamento de uma regra ID para autômato adaptativo cria uma transição que corresponde ao final do reconhecimento de uma regra ID, ou seja, ao reconhecimento da categoria-mãe $X_0$. Essa transição está associada a uma ação adaptativa posterior que monta a estrutura final contendo as informações da categoria-mãe $X_0$ que está sendo reconhecida (cria um estado correspondente à categoria reconhecida e uma transição partindo desse estado para o estado correspondente a $X_1$, uma transição do estado correspondente a $X_0$ para o estado correspondente a $X_2$, e assim por diante). Além disso, verifica se a regra satisfaz aos princípios CAP, FFP e HFC. O algoritmo (Rotina ReconheceRegra) que cria essa ação adaptativa é descrito a seguir.

1. Crie uma ação adaptativa.
2. Crie um ESTADO_X0 correspondente a $X_0$.
3. Atribua a $X_i$ o valor de $X_1$.
4. Acrescente à ação adaptativa:
   - Inserção de uma transição de ESTADO_CORR para ESTADO_X0.
   - Acrescente a ESTADO_X0 as características herdadas de $X_0$.
   - **Enquanto** $X_i$ for diferente de $X_n$ **faça**

- inserção da transição de ESTADO_X0 para estado correspondente a $X_i$, com estímulo igual ao nome da categoria referente a $X_i$.
- atribua a $X_i$ o valor de $X_{i+1}$.

5. **(Princípio CAP)**

**Se** n ≥ 2

**Então** Atribua a i o valor 1.

**Enquanto** i < n **faça**

Atribua a j o valor i+1.

**Enquanto** j ≤ n **faça**

**Se** $X_j$ controla $X_i$

**Então** Atribua a CONTROLE a característica AGR.

**Se** SLASH estiver definida para $X_j$

**Então** Atribua a CONTROLE a característica SLASH.

Acrescente à ação adaptativa:

- UnifiqueConcordancia ($X_i$, $X_j$, CONTROLE).

**se** Retorno aponta para estado Falso

**então** Retorne.

Incremente o valor de j.

Incremente o valor de i.

6. **(Princípio FFP)**
   Atribua a i o valor 1.
   **Enquanto** i ≤ n **faça**
     Acrescente à ação adaptativa:
     - **Se** há alguma CARAC=característica não-nuclear de $X_i$ não definida para $X_0$
       **Então** Unifique ($X_0$, $X_i$, CARAC).
             **se** Retorno aponta para estado Falso
             **então Retorne.**
     Incremente o valor de i.
7. **(Princípio HFC)**
   Atribua a $X_h$ o valor de $X_i$ nuclear.
   Acrescente à ação adaptativa:
   - Para cada característica nuclear CARAC, instanciada em $X_h$, Unifique ($X_0$, $X_h$, CARAC).
     **se** Retorno aponta para estado Falso
     **então Retorne.**
8. **Retorne** o estado Verdadeiro.

**Algoritmo de unificação de características de concordância.**

O algoritmo que verifica se a regra ID satisfaz o princípio CAP chama esta rotina de unificação de características (Rotina UnifiqueConcordancia), que recebe como argumentos:

$X_i$ – estado que contém a característica a ser unificada e que deverá conter o resultado da unificação.

$X_j$ – estado que contém a característica a ser unificada

CARAC – característica a ser unificada

Essa rotina é descrita a seguir:

1. Inspeção para verificar se $X_j$ corresponde à categoria sintática que a característica CONTROLE de $X_i$ tem como valor.
2. **Se** não corresponder
   **Então Retorne** o estado Falso.
3. Inspeção para atribuir a ESTADO_CONTROLE o estado apontado pela transição CONTROLE de $X_i$ (corresponde ao valor de CONTROLE de $X_i$).
4. Unifique (ESTADO_CONTROLE, $X_i$, PER).
   **Se** Retorno = Falso

**Então Retorne** o estado Falso.

5. Unifique (ESTADO_CONTROLE, $X_i$, PLU).
   **Se** Retorno = Falso
   **Então Retorne** o estado Falso.
6. Unifique (ESTADO_CONTROLE, $X_i$, MASC).
   **Se** Retorno = Falso
   **Então Retorne** o estado Falso.
   **Senão Retorne** o estado Verdadeiro.

**Algoritmo de unificação de características.**

Esse algoritmo chama-se Unifique, é chamado pelos algoritmos que verificam se a regra ID satisfaz os princípios FFP e HFC, e também pelo algoritmo UnifiqueConcordância. Recebe como argumentos:

ESTADO_ALVO – estado que contém a característica a ser unificada e que deverá conter o resultado da unificação.

ESTADO_BASE – estado que contém a característica a ser unificada

CARAC – característica a ser unificada

1. Inspeção para verificar se de ESTADO_ALVO parte alguma transição com estímulo CARAC.
2. **Se** houver tal transição
   **Então** Inspeção para verificar se de ESTADO_BASE também parte alguma transição com estímulo CARAC.
   **Se** houver tal transição
   **Então** UnifiqueValor (valor de CARAC de ESTADO_ALVO, valor de CARAC de ESTADO_BASE).
   **Se** Retorno = Falso
   **Então Retorne** o estado Falso.
   **Senão** Inspeção para verificar se de ESTADO_BASE também parte alguma transição com estímulo CARAC
   **Se** houver tal transição
   **Então** Inserção da transição com estímulo igual a característica CARAC, de ESTADO_ALVO para o valor de CARAC de ESTADO_BASE.
3. **Retorne** o estado Verdadeiro.

**Algoritmo de unificação de valores.**

Esse algoritmo chama-se UnifiqueValor, é chamado pelo algoritmo Unifique e tem como função unificar dois valores e, caso esses valores apresentem características, unificar também todas as suas características. Recebe como argumentos:

$X_i$ – estado que contém o valor a ser unificado e que deverá conter o resultado da unificação.

$X_j$ – estado que contém o valor a ser unificado

```
1. Se Xi e Xj forem diferentes
   Então Retorne o estado Falso.
   Senão Se Xi e Xj apresentam transições
         Então Para cada transição de Xi para VALOR_XI faça
                  Se houver transição correspondente de Xj para
                  VALOR_XJ
                  Então UnifiqueValor (VALOR_XI, VALOR_XJ).
                        Se Retorno = Falso
                        Então Retorne o estado Falso.
               Para cada transição de Xj para VALOR_XJ faça
                  Se não houver transição correspondente de Xi
                  para VALOR_XI
                  Então Inserção dessa transição a partir de Xi
                        com VALOR_XJ
         Retorne o estado Verdadeiro.
```

Mediante os algoritmos descritos, pode-se converter, assim, uma especificação GPSG para Autômato Adaptativo.

Segundo o algoritmo proposto em [Iwai-00], pode-se mapear qualquer Autômato Adaptativo para uma Gramática Adaptativa equivalente.

Sejam, por exemplo, as seguintes regras ID, já mencionadas anteriormente e que foram utilizadas também na simulação passo a passo dos algoritmos de mapeamento GPSG para Autômato Adaptativo, descrita no item 5.1 desta dissertação:

```
(1)a. N1 → H[1.0]
  b.  N1 → H[1.1], P2[de]
```

O autômato adaptativo resultante do mapeamento dessas regras é:

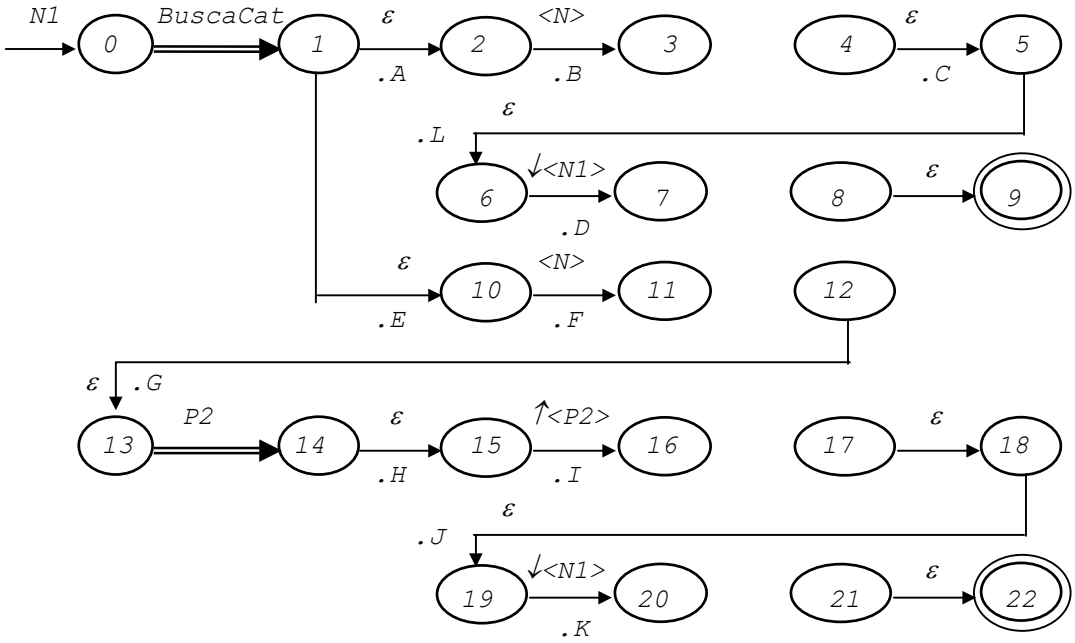



***Figura 23 - Autômato Adaptativo correspondente ao mapeamento das regras ID 1a e 1b.***

A ilustração representa o autômato adaptativo $M=(E_0, A)$, sendo que:

- $E_0=(Q_0, SM_0, \Sigma, \Gamma, R_0, Z_0, q_0, F)$ é a máquina de estados inicial (autômato de pilha estruturado) que implementa $M$ :
  - $Q_0$={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21, 22} é o conjunto (inicial) de estados pré-definido.
  - $SM_0$={N1} é a coleção de submáquinas que implementa $E_0$.
  - $\Sigma$={<N>,<N1>,<P2>}
  - $\Gamma$={1,14}
  - $R_0$ = { ( ε, 0, ε):→ ( 1, BC, ε)
    ( ε, 1, ε):→ ( ε, 2, ε), A
    ( ε, 2, <N>):→ ( ε, 3, ε), B
    ( ε, 4, ε):→ ( ε, 5, ε), C
    ( ε, 5, ε):→ ( ε, 6, ε), L
    ( ε, 6, ε):→ ( ε, 7, <N1>), D
    ( ε, 8, ε):→ ( ε, 9, ε)
    ( HALT, 9, ε):→ ( ε, HALT, ε)
    ( ε, 1, ε):→ ( ε, 10, ε), E
    ( ε, 10, <N>):→ ( ε, 11, ε), F
    ( ε, 12, ε):→ ( ε, 13, ε), G
    ( ε, 13, ε):→ ( 14, P2, ε)

( ε, 14, ε):→ ( ε, 15, ε), H

( ε, 15, <P2>):→ ( ε, 16, ε), I

( ε, 17, ε):→ ( ε, 18, ε)

( ε, 18, ε):→ ( ε, 19, ε), J

( ε, 19, ε):→ ( ε, 20, <N1>), K

( ε, 21, ε):→ ( ε, 22, ε)

( HALT, 22, ε):→ ( ε, HALT, ε)

}

- *F* = { 9, 22 } é o conjunto de estados finais.

- *A* é o conjunto fixo das ações adaptativas, responsáveis pela dinâmica da topologia do autômato.

*A* = { A = { -[( ε, 3, ε):→ ( ε, 4, ε)]}

L = { -[( ε, 7, ε):→ ( ε, 8, ε)]}

H = { -[( ε, 16, ε):→ ( ε, 17, ε)]}

L = { -[( ε, 20, ε):→ ( ε, 21, ε)]}

B(...) = { ... }

C(...) = { ... }

...

K(...) = { ... }

}

Com exceção das ações adaptativas A, L, H e J, as demais ações adaptativas deste autômato foram omitidas por serem mais extensas.

Aplicando-se a esse autômato adaptativo o algoritmo de mapeamento de Autômato Adaptativo para Gramática Adaptativa, proposto por [Iwai-00], obtém-se a seguinte gramática adaptativa equivalente $G = (G^0, T, R^0)$.

- $G^0$= ($V_N^0$, $V_T$, $V_C$, $P_L^0$, $P_D^0$, S) é a gramática inicial, sendo que:
  - $V_N^0$={$V_1$,$V_2$,$V_3$,$V_4$,$V_5$,$V_6$,$V_7$,$V_8$,$V_9$,$V_{10}$,$V_{11}$,$V_{12}$,$V_{13}$,$V_{14}$,$V_{15}$,$V_{16}$,$V_{17}$,$V_{18}$, $V_{19}$,$V_{20}$,$V_{21}$,$V_{22}$,$V_6'$,$V_{19}'$} é o conjunto de símbolos não-terminais,
  - $V_T$ = {<N>} é o conjunto de símbolos terminais,
  - $V_C$ = {<N1>,<P2>} é o conjunto de símbolos de contexto.
  - S = $V_0$.
  - $P_L^0 \cup P_D^0$ é o conjunto de regras de produção =

{ $V_0 \rightarrow V_{BC}\ V_1$

$V_1 \rightarrow$ { A′ } $V_2$

$V_2 \rightarrow$ { B′ } <N> $V_3$

$V_4 \rightarrow \{ C' \} V_5$

$V_5 \rightarrow \{ L' \} V_6$

$V_6 \rightarrow \varepsilon V_6'$

$V_6' \leftarrow \{ D' \} <N1> V_7$

$V_8 \rightarrow \varepsilon V_9$

$V_9 \rightarrow \varepsilon$

$V_1 \rightarrow \{ E' \} V_{10}$

$V_{10} \rightarrow \{ F' \} <N> V_{11}$

$V_{12} \rightarrow \{ G' \} V_{13}$

$V_{13} \rightarrow V_{P2} V_{14}$

$V_{14} \rightarrow \{ H' \} V_{15}$

$<P2> V_{15} \rightarrow \{ I' \} V_{16}$

$V_{17} \rightarrow \varepsilon V_{18}$

$V_{18} \rightarrow \{ J' \} V_{19}$

$V_{19} \rightarrow \varepsilon V_{19}'$

$V_{19}' \leftarrow \{ K' \} <N1> V_{20}$

$V_{21} \rightarrow \varepsilon V_{22}$

$V_{22} \rightarrow \varepsilon$

}

sendo que $V_{BC}$ e $V_{P2}$ correspondem aos símbolos não terminais iniciais das gramáticas adaptativas equivalentes aos autômatos adaptativos BuscaCat e P2, respectivamente.

- T é o conjunto de ações adaptativas

T = { A' = { -[$V_3 \rightarrow \varepsilon V_4$] }

L' = { -[$V_7 \rightarrow \varepsilon V_8$] }

H' = { -[$V_{16} \rightarrow \varepsilon V_{17}$] }

J' = { -[$V_{20} \rightarrow \varepsilon V_{21}$] }

B'( ... ) = { ... }

…

K'( ... ) = { ... }

}

- $R^0$ é a relação que a cada regra de produção r, associa uma ação adaptativa A

Com exceção das ações adaptativas A', L', H' e J', as demais não estão especificadas. No entanto, procedendo-se de maneira semelhante às ações que foram mapeadas, não é difícil converter as demais ações adaptativas do autômato adaptativo para gramática adaptativa, seguindo o algoritmo de mapeamento de

Autômato Adaptativo para Gramática Adaptativa equivalente, especificado em [Iwai-00].

Dessa forma, conclui-se que é possível utilizar a Gramática Adaptativa na representação e análise sintática do processamento de linguagem natural, à medida em que é possível utilizar o Autômato Adaptativo para essa mesma finalidade. A notação da Gramática Adaptativa tem a vantagem de ser mais familiar aos lingüistas.

## 4.3. Gramática da Estrutura Superficial da Língua Portuguesa em Autômato Adaptativo

Uma vez definidos os algoritmos de mapeamento de redes ATN e de especificações GPSG para Autômato Adaptativo, pode-se escrever uma gramática de estrutura superficial da língua portuguesa, tanto em rede ATN como em GPSG, e submeter tal especificação a esses algoritmos. O resultado seria não apenas uma especificação da gramática da estrutura superficial da língua portuguesa em Autômato Adaptativo, como também um analisador sintático da língua portuguesa implementado através de Autômato Adaptativo.

Nesta dissertação, adotou-se a escolha da gramática portuguesa especificada em GPSG, dado que já existe um trabalho [Chin-96], desenvolvido por uma especialista da área lingüística, que apresenta uma especificação da gramática da estrutura superficial da língua portuguesa em GPSG, com a finalidade de se indicar como seria uma tradução automática de textos da área médica, do português para o inglês.

Exemplificando, as regras para os sintagmas substantivos da língua portuguesa, em GPSG, segundo definidas em [Chin-96], são:

```
N1 → H[1.0]
N1 → H[1.1], (P2[de])
N1 → H[1.2], (P2[sobre])
N1 → H[1.3], (P2[de]), (P2[com])
N1 → H[1.4], (P2[de]), (P2[a])
N1 → H[1.5], (P2[entre])
N1 → H[1.6], (P2[de]), (P2[em])
N1 → H[1.7], (P2[de]), (P2[para])
P1 → H[3.0], N2
P1 → H[3.1], V2[INF]
P1 → H[3.2], S[FIN]
P2 → H1
N2 [DEF-] → {[SUBCAT A]}, H1
N2 [DEF+] → {[SUBCAT B]}, H1
```

```
N2 [DEF+] → {[SUBCAT C]}, H1
N2 [QUANT+] → {[SUBCAT D]}, H1
N2 [QUANT+] → {[SUBCAT D]}, H2[DEF+]
N2 → {[SUBCAT E]}, H2[QUANT+]
N2 [DEF+] → {[SUBCAT G]},{[SUBCAT F]},H1
N2 → H1
N1 [SUBST +] → H[1.0, SUBST +], (X2)
N1 [SUBST +] → H1[1.0, SUBST +], (A2)
N1 [SUBST +] → H1[1.0, SUBST +], (P2)
N2 → H1[SUBST +]
N2[Q+] → {[SUBCAT D]}, H2[DET +, Q -]
N2 → {[SUBCAT E]}, H2[Q+]
FCR: [N+, V-, BAR 2] ⊃ ~[SUBST]
N1[PRON+] → H[1.0, PRON+]
N2 → H1[PRON+]
N2 → {[SUBCAT E]}, H1[PRON+, Q+]
FCR: [V+, N-, BAR 2, FIN] ⊃ [CASO NOM]
FCR: [N-, V+, BAR 0, SUBCAT α]⊃[CASO ACC]
FCR: [N-, V-, BAR 0] ⊃ [CASO OBL]
TYP(N2[AGR V2])=<TYP(V2[FIN]), TYP(S)>
TYP(N2[AGR  V0])  =  <TYP(V0[SUBCAT  α]),
                      TYP(V1)>
   α ∈ {verbos que têm Caso acusativo a
       atribuir}
TYP(N2[AGR P0]) = <TYP(P0), TYP(P1)>
Q1 → H[n]
Q2 → {[SUBCAT G]}, H1
Q2 → {[SUBCAT H]}, H2
N2 → Q2, H1
N2 → H2, N2
```

As regras para os demais sintagmas e para as orações da língua portuguesa em GPSG, segundo definidas em [Chin-96], estão listadas no Apêndice A deste trabalho.

Utilizando-se os algoritmos de mapeamento de GPSG para Autômato Adaptativo, descritos no item anterior, essas regras podem ser convertidas para Autômato Adaptativo.

Sejam por exemplo, as regras ID:

```
(1)a.  N1 → H[1.0]
   b.  N1 → H[1.1], P2[de]
   c.  N2 → H1
   d.  N2 → [SUBCAT D], H2
   e.  P1 → H[3.0], n2
   f.  P2 → H1
```

Aplicando-se os algoritmos de mapeamento de regras GPSG para Autômato Adaptativo, obtém-se os seguintes autômatos adaptativos (a simulação passo a passo desse mapeamento é descrita no item 5.1 deste trabalho):

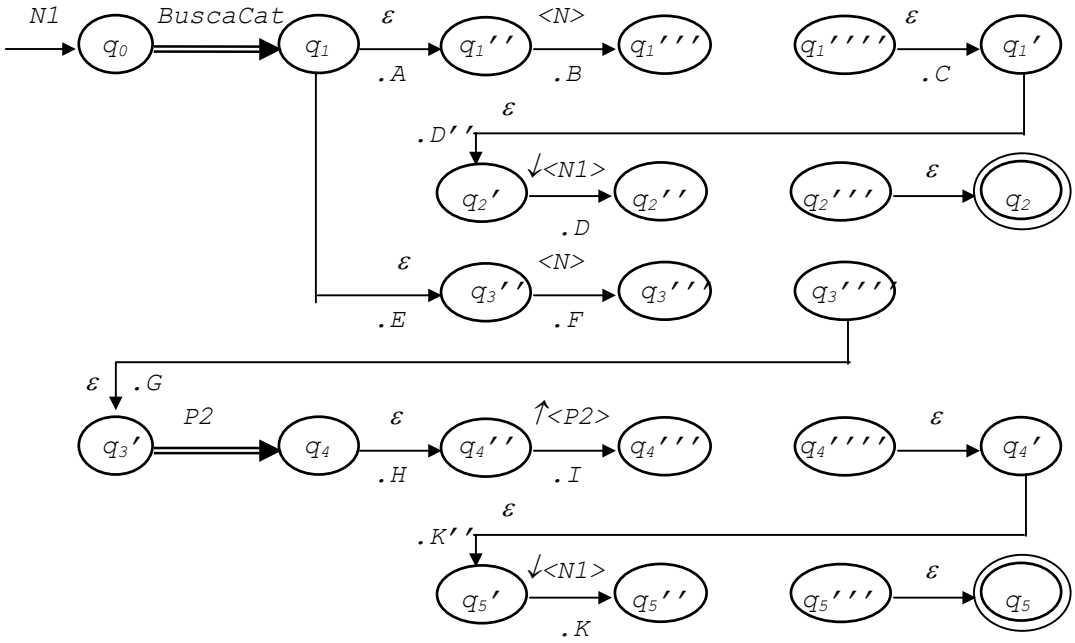



***Figura 24 - Autômato adaptativo correspondente ao mapeamento das regras 1a e 1b.***

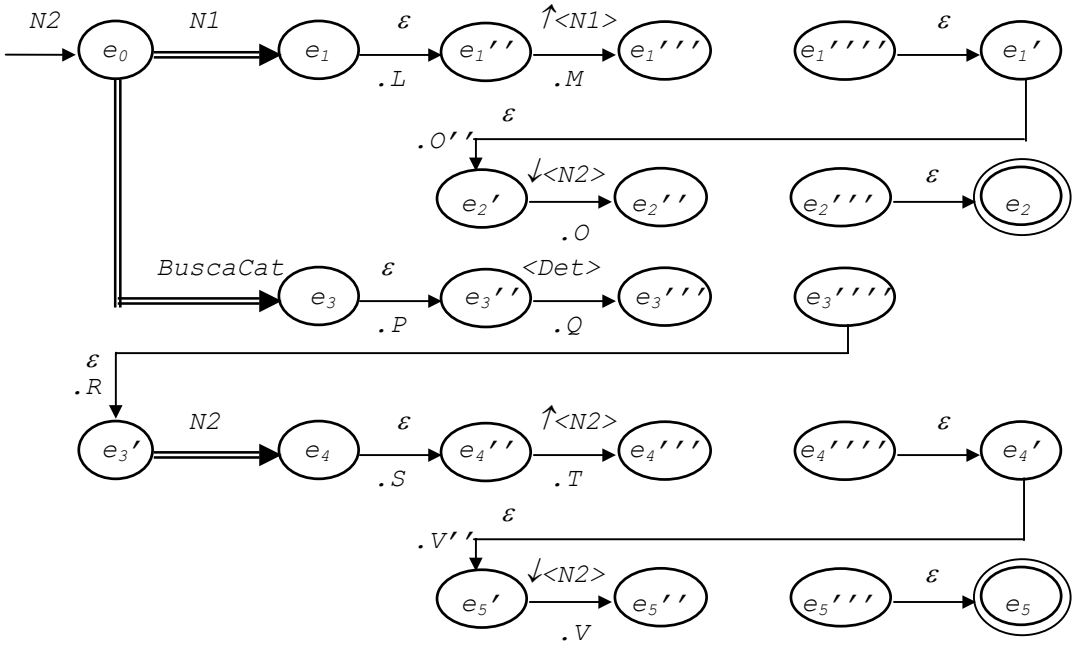



***Figura 25 - Autômato adaptativo correspondente ao mapeamento das regras 1c e 1d.***

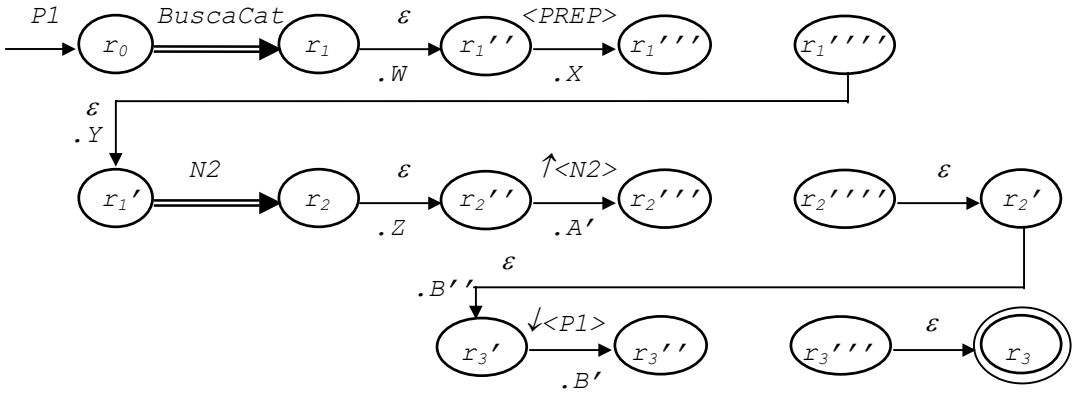



***Figura 26 - Autômato adaptativo correspondente ao mapeamento da regra 1e.***

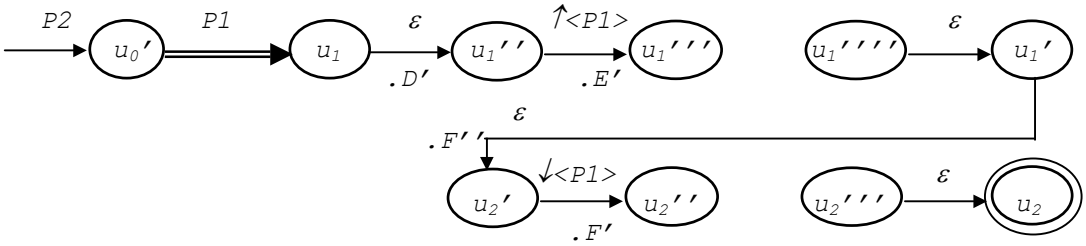



***Figura 27 - Autômato adaptativo correspondente ao mapeamento da regra 1f.***

As ações adaptativas dos autômatos representados nas Figuras acima estão descritas no item 5.1.2 desta dissertação.

Da mesma forma como se mapeou as regras 1a a 1f, pode-se mapear as demais regras da especificação da gramática da língua portuguesa em GPSG, definidas em [Chin-96]. O mapeamento completo de todas as regras resultará numa especificação da gramática superficial da língua portuguesa em Autômato Adaptativo e num analisador sintático da língua portuguesa implementado através de Autômato Adaptativo.

Como foi indicado no item 4.2 deste trabalho, pode-se também mapear o Autômato Adaptativo para Gramática Adaptativa, utilizando-se o algoritmo especificado em [Iwai-00] para essa finalidade. Disso resultaria uma especificação da gramática superficial da língua portuguesa em Gramática Adaptativa, cuja notação certamente é mais familiar para os lingüistas.

O próximo capítulo descreve os resultados experimentais obtidos neste trabalho.

# 5. RESULTADOS EXPERIMENTAIS

## 5.1. Experimentos Realizados

Para se verificar, na prática, o desenvolvimento da proposta deste trabalho, os algoritmos de mapeamento indicados no Capítulo 4, tanto de redes ATN, como o de GPSG, para Autômato Adaptativo, foram simulados passo a passo.

Na simulação realizada, foram submetidas algumas redes ATN definidas para a gramática da língua portuguesa ao algoritmo de mapeamento de Redes ATN para Autômato Adaptativo. Foram submetidas também algumas regras GPSG definidas para a gramática da estrutura superficial da língua portuguesa (conforme [Chin-96]) ao algoritmo de mapeamento de regra GPSG para Autômato Adaptativo. Isso resultou num analisador sintático de fragmentos de sentenças da língua portuguesa, especificado através de Autômato Adaptativo.

Para que se possa submeter sentenças em língua portuguesa a esse analisador sintático especificado através de Autômato Adaptativo, pode-se utilizar o analisador e etiquetador morfológico desenvolvido em [Menezes-00].

A sentença resultante desse analisador e etiquetador morfológico é, então, submetida ao algoritmo de mapeamento léxico, descrito no item 4.1 desta dissertação. Esse algoritmo cria um autômato adaptativo equivalente a uma lista léxica, que contém cada palavra da sentença, associada às suas características léxicas e sintáticas. A sentença submetida como entrada a esse algoritmo é substituída por uma cadeia de etiquetas, sendo que cada etiqueta representa uma palavra da sentença original. Dessa forma, os autômatos adaptativos resultantes dos mapeamentos podem

receber fragmentos de sentenças escritos em língua portuguesa, previamente etiquetados pelo analisador morfológico de [Menezes-00].

A Figura 28, a seguir, esquematiza a parte experimental deste trabalho.

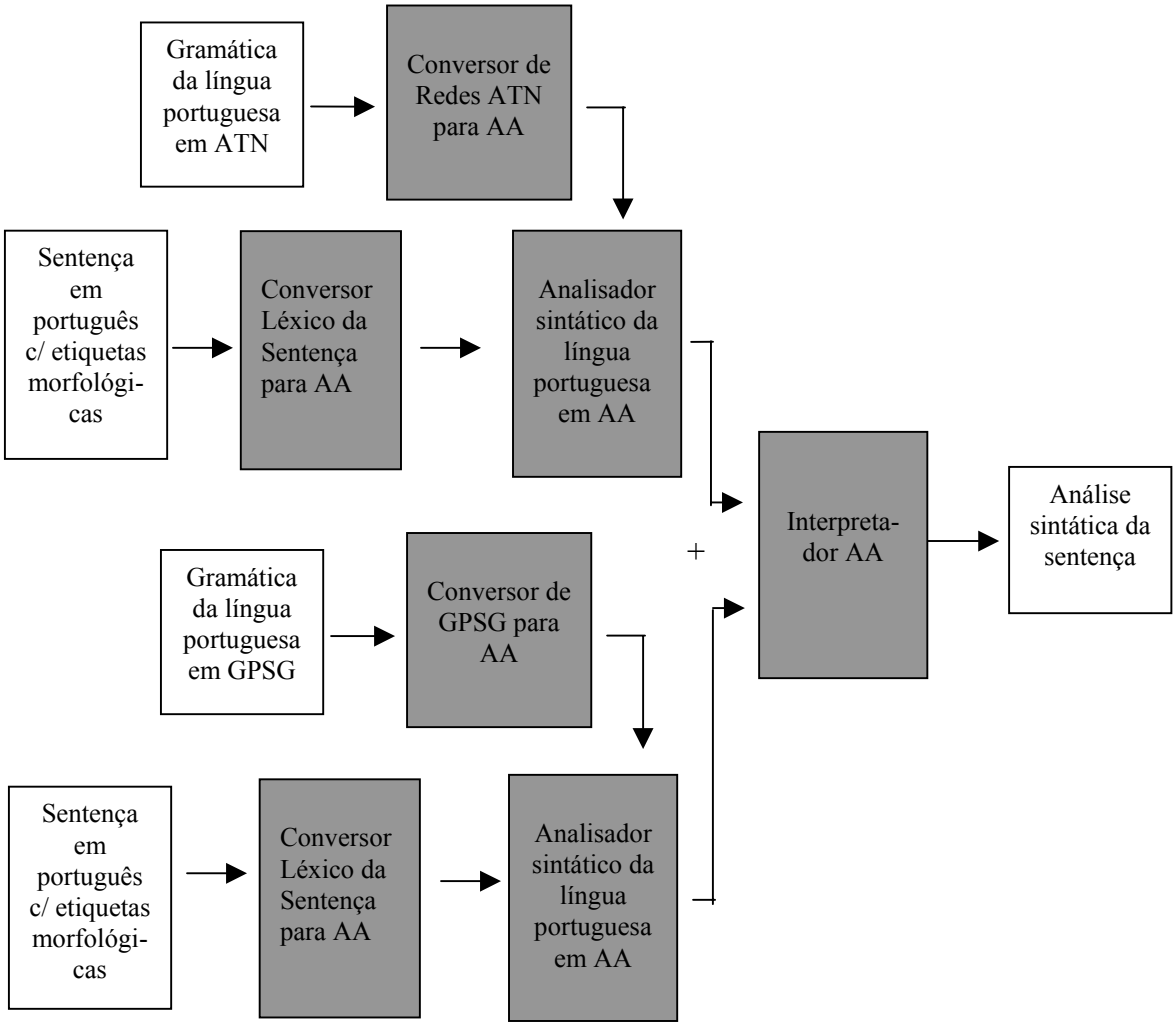



***Figura 28 - Esquema da parte experimental***

O próximo item descreve as simulações realizadas do algoritmo de mapeamento de Redes ATN para Autômato Adaptativo.

### 5.1.1. Experimentos Relativos ao Mapeamento de Redes ATN para Autômato Adaptativo

Seja a seguinte Rede ATN, que define um sintagma substantivo da gramática da língua portuguesa. Essa rede foi obtida adaptando-se um exemplo de Rede ATN, que define o sintagma substantivo da gramática da língua inglesa, extraído de [Bates-78].

```
(NP/
 (REGS DET NUMDET GENDET N NUM GEN PP)
 (NP/
    (CAT DET T
        (SETR DET *)
        (SETR NUMDET (GETF NUMERO))
        (SETR GENDET (GETF GENERO))
        (TO NP/DET))
    (CAT N T
        (SETR N *)
        (SETR NUM (GETF NUMERO))
        (SETR GEN (GETF GENERO))
        (TO NP/N3))
 )
 (NP/DET
    (CAT N T
        (SETR N *)
        (SETR NUM (GETF NUMERO))
        (SETR GEN (GETF GENERO))
        (TO NP/N1))
 )
 (NP/N1
    (JUMP NP/N2 (AND  (AGREE (GETR NUMDET) (GETR NUM))
                      (AGREE (GETR GENDET) (GETR GEN)))
 )
 (NP/N2
    (PUSH PP/ T (SETR PP *) (TO NP/PP1))
    (POP (BUILD (+ + + +) DET NUM GEN N) T)
 )
 (NP/PP1
    (PUSH PP/ T (SETR PP (APPEND PP *)) (TO NP/PP1))
    (POP T (BUILD (+ + + + +) DET NUM GEN N PP))
 )
(NP/N3
    (PUSH PP/ T (SETR PP *) (TO NP/PP2))
    (POP (BUILD (+ + +) NUM GEN N) T)
 )
 (NP/PP2
    (PUSH PP/ T (SETR PP (APPEND PP *)) (TO NP/PP2))
    (POP T (BUILD (+ + + +) NUM GEN N PP))
 )
)
```

Essa rede pode ser representada pelo seguinte diagrama:

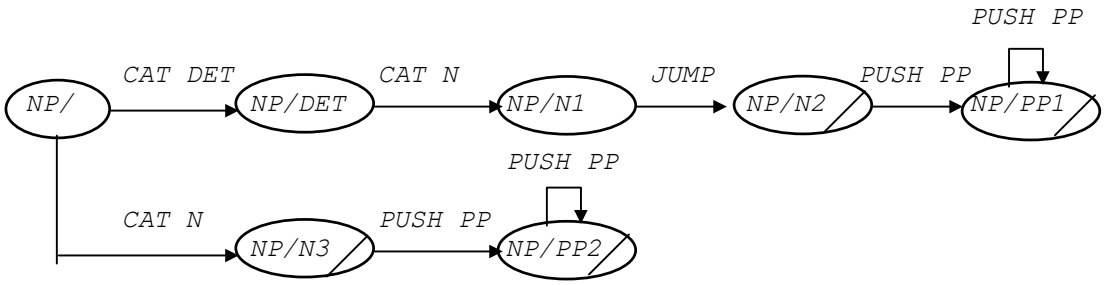



***Figura 29 - Rede ATN correspondente à Rede NP/ .***

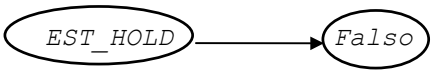
Submetendo-se a rede NP/ ao algoritmo de mapeamento de rede ATN para Autômato Adaptativo (MapeamentoRedeATN-AA), tem-se a simulação de execução passo a passo, descrita a seguir. As partes dos algoritmos que estão sendo executadas encontram-se alternadas com a descrição da simulação de sua respectiva execução. A simulação da execução está representada em letras em itálico.

***Rotina MapeamentoRedeATN-AA.***

1. Inicialize a lista de estados finais (deve conter o estado de erro).

*NP_LISTA_EST_FINAL= {E-ERRO}.*

2. Crie um EST_HOLD correspondente à lista HOLD desta rede.




3. **Se** houver declaração do tipo (REGS <reg>*)

*Há a declaração (REGS DET NUMDET GENDET N NUM GEN PP)*

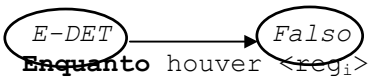
**Então Enquanto** houver $<reg_i>$ **faça**

*$<reg_i>$ = DET*

Crie um estado $R_i$ e associe-o em uma tabela ao nome $<reg_i>$.


| *REGISTRADOR* | *ESTADO* |
|---|---|
| *DET* | *E-DET* |

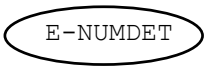
Faça $R_i$ apontar para o estado Falso.




**Enquanto** houver $<reg_i>$ **faça**

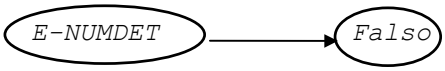
*<reg$_2$> = NUMDET*

Crie um estado $R_i$ e associe-o em uma tabela ao nome <reg$_i$>.




| *REGISTRADOR* | *ESTADO* |
|---|---|
| *DET* | *E-DET* |
| *NUMDET* | *E-NUMDET* |

Faça $R_i$ apontar para o estado Falso.



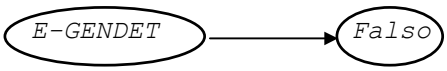

**Enquanto** houver <reg$_i$> **faça**

*<reg$_3$> = GENDET*

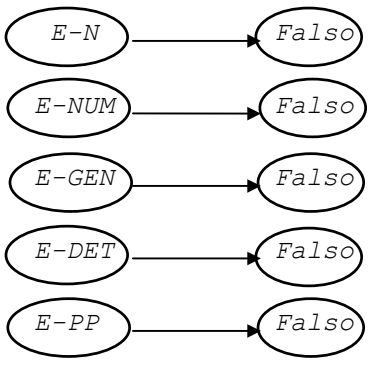
Crie um estado $R_i$ e associe-o em uma tabela ao nome <reg$_i$>.


| *REGISTRADOR* | *ESTADO* |
|---|---|
| *DET* | *E-DET* |
| *NUMDET* | *E-NUMDET* |
| *GENDET* | *E-GENDET* |

Faça $R_i$ apontar para o estado Falso.




*... e assim por diante, até que todos os <reg$_i$> tenham sido tratados, obtendo-se a seguinte configuração:*




| *REGISTRADOR* | *ESTADO* |
|---|---|

| | |
|---|---|
| *DET* | *E-DET* |
| *NUMDET* | *E-NUMDET* |
| *GENDET* | *E-GENDET* |
| *N* | *E-N* |
| *NUM* | *E-NUM* |
| *GEN* | *E-GEN* |
| *PP* | *E-PP* |

***(Mapeamento do primeiro estado)***

4. **Enquanto** houver estados a serem processados **faça**:

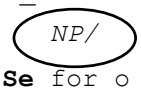
*Há o estado: NP/*

4a.**Se** já existe um estado correspondente ao estado sendo processado

*Não existe um estado correspondente ao estado NP/*

**Senão** Crie um ESTADO_CORRENTE correspondente ao estado sendo processado.

*ESTADO_CORRENTE:*

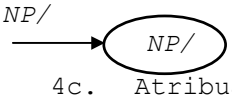



4b.**Se** for o primeiro estado da rede

*É o primeiro estado.*

**Então** Faça ESTADO_CORRENTE ser o estado inicial dessa rede.

*NP/*




4c. Atribua 'Falso' a FLG_BUSCACAT.

*FLG_BUSCACAT= 'Falso'.*

***(Mapeamento do primeiro arco de NP/)***

4d. **Enquanto** houver arcos ATN a serem processados **faça**:

*Há o arco:*

```
(CAT DET T
    (SETR DET *)
    (SETR NUMDET (GETF NUMERO))
    (SETR GENDET (GETF GENERO))
    (TO NP/DET))
```

I. **Se** o arco for do tipo (POP <expressão> <teste>)

*Não é arco POP.*

II. **Se** o arco for do tipo (JUMP <estado> <teste> <ação>*)

*Não é arco JUMP*

III. **Se** o arco for do tipo (CAT <categoria> <teste> <ação>* (TO próx-estado))

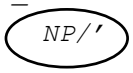
*É arco CAT*

**Então Se** já existe uma transição com a chamada da submáquina BuscaCat, a partir de ESTADO_CORRENTE

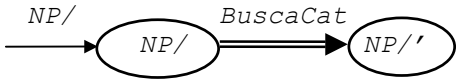
*Não existe tal transição a partir de ESTADO_CORRENTE*

**Senão** Crie um ESTADO_CORRENTE'.

*ESTADO_CORRENTE':*



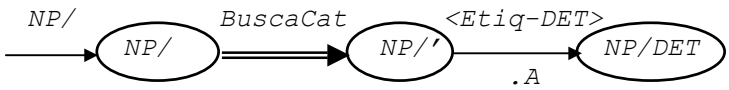

Crie uma transição com a chamada da submáquina BuscaCat, do ESTADO_CORRENTE para o ESTADO_CORRENTE'.




**Se** <teste> for <T>

*<teste> é <T>*

**Então** Crie uma transição com o estímulo <categoria>, com o consumo da cadeia, de ESTADO_CORRENTE' para <próx-estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.




*Nesse instante, a rotina TrataAção é chamada para preparar o autômato para a execução de <ação>.*

***Rotina TrataAção.***

1. **Enquanto** houver <ação> a ser processada **faça**:

*Há a ação(SETR DET *)*

1a. Chame a rotina TrataExpressão que procede o tratamento de <expressão> e retorna RetornoExpr apontando para o estado que corresponde ao novo valor de <reg>.

*Nesse instante, a rotina TrataExpressão é chamada para proceder o tratamento de <expressão>.*

***Rotina TrataExpressão.***

1. **Se** a <expressão> for do tipo *

*<expressão> é do tipo **

**Então Retorne** o estado para o qual o ESTR_CORR aponta.

*Ação adaptativa A deve executar:*

- *inspeção para descobrir para qual estado o ESTR_CORR aponta.*
- *inserção da transição do estado RetornoExpr para esse estado.*

*Retorna à rotina TrataAção.*

***Rotina TrataAção.***

1b.**Se** <ação> é HOLD

*<ação> não é HOLD.*

**Senão** Ache o $R_i$ que corresponde a <reg>. Se não houver tal $R_i$, então Imprima "Registrador <reg> não foi definido".

*$R_i$ = estado E-DET*

**Se** $R_i$ e RetornoExpr não apontam para o mesmo estado

**Então** Elimine a transição de $R_i$ para o estado para o qual aponta atualmente.

Crie uma transição de $R_i$ para o estado apontado por RetornoExpr.

*Ação adaptativa A deve executar:*

- *se $R_i$ e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-DET para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-DET para o estado apontado por RetornoExpr.*

1. **Enquanto** houver <ação> a ser processada **faça**:

*Há a ação (SETR NUMDET (GETF NUMERO))*

1a. Chame a rotina TrataExpressão que procede o tratamento de <expressão> e retorna RetornoExpr apontando para o estado que corresponde ao novo valor de <reg>.

*Nesse instante, a rotina TrataExpressão é chamada para proceder o tratamento de <expressão>.*

***Rotina TrataExpressão.***

1. **Se** a <expressão> for do tipo *

*<expressão> não é do tipo **

2. **Se** a <expressão> for do tipo 'palavra

*<expressão> não é do tipo 'palavra*

3. **Se** a <expressão> for do tipo (GETR <reg>)

*<expressão> não é do tipo (GETR <reg>)*

4. **Se** a <expressão> for do tipo (GETF <característica-sintática> <palavra>)

*<expressão> é do tipo (GETF <característica-sintática>)*

**Então Se** <palavra> estiver especificada

*<palavra> não está especificada*

**Senão** Atribua a $R_i$ o estado apontado por ESTR_CORR.

*Ação adaptativa A deve executar:*

- *inspeção para descobrir $R_i$ = estado para o qual o ESTR_CORR aponta.*

**Retorne** o estado para o qual o $R_i$ aponta com estímulo <característica-sintática>.

*Ação adaptativa A deve executar:*

- *inspeção para descobrir para qual estado o $R_i$ aponta com estímulo NUMERO.*
- *inserção da transição do estado RetornoExpr para esse estado.*

*Retorna à rotina TrataAção.*

***Rotina TrataAção.***

1b.**Se** <ação> é HOLD

*<ação> não é HOLD.*

**Senão** Ache o $R_i$ que corresponde a <reg>. Se não houver tal $R_i$, então Imprima "Registrador <reg> não foi definido".

*$R_i$ = estado E-NUMDET*

**Se** $R_i$ e RetornoExpr não apontam para o mesmo estado

**Então** Elimine a transição de $R_i$ para o estado para o qual aponta atualmente.

Crie uma transição de $R_i$ para o estado apontado por RetornoExpr.

*Ação adaptativa A deve executar:*

- *se $R_i$ e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-NUMDET para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-NUMDET para o estado apontado por RetornoExpr.*

1. **Enquanto** houver <ação> a ser processada **faça**:

*Há a ação (SETR GENDET (GETF GENERO))*

1a.Chame a rotina TrataExpressão que procede o tratamento de <expressão> e retorna RetornoExpr apontando para o estado que corresponde ao novo valor de <reg>.

*Nesse instante, a rotina TrataExpressão é chamada para proceder o tratamento de <expressão>.*

***Rotina TrataExpressão.***

1. **Se** a <expressão> for do tipo *

*<expressão> não é do tipo **

2. **Se** a <expressão> for do tipo 'palavra

*<expressão> não é do tipo 'palavra*

3. **Se** a <expressão> for do tipo (GETR <reg>)

*<expressão> não é do tipo (GETR <reg>)*

4. **Se** a <expressão> for do tipo (GETF <característica-sintática> <palavra>)

*<expressão> é do tipo (GETF <característica-sintática>)*

**Então Se** <palavra> estiver especificada

*<palavra> não está especificada*

**Senão** Atribua a $R_i$ o estado apontado por ESTR_CORR.

*Ação adaptativa A deve executar:*

- *inspeção para descobrir $R_i$ = estado para o qual o ESTR_CORR aponta.*

**Retorne** o estado para o qual o $R_i$ aponta com estímulo <característica-sintática>.

*Ação adaptativa A deve executar:*

- *inspeção para descobrir para qual estado o $R_i$ aponta com estímulo GENERO.*
- *inserção da transição do estado RetornoExpr para esse estado.*

*Retorna à rotina TrataAção.*

***Rotina TrataAção.***

1b.**Se** <ação> é HOLD

*<ação> não é HOLD.*

**Senão** Ache o $R_i$ que corresponde a <reg>. Se não houver tal $R_i$, então Imprima "Registrador <reg> não foi definido".

*$R_i$ = estado E-GENDET*

**Se** $R_i$ e RetornoExpr não apontam para o mesmo estado

**Então** Elimine a transição de $R_i$ para o estado para o qual aponta atualmente.

Crie uma transição de $R_i$ para o estado apontado por RetornoExpr.

*Ação adaptativa A deve executar:*

- *se $R_i$ e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-GENDET para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-GENDET para o estado apontado por RetornoExpr.*

1. **Enquanto** houver <ação> a ser processada **faça**:

*Não há mais <ação> a ser processada.*

2. Encerre o processamento.

*Fim da rotina TrataAção.*

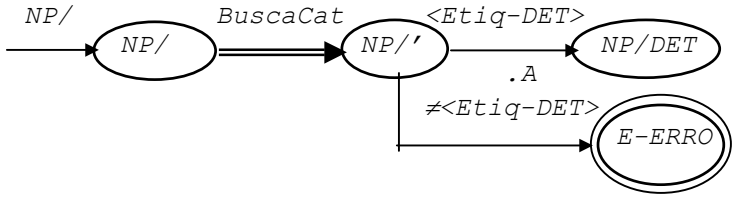
*Retorna à rotina MapeamentoRedeATN-AA*

***Rotina MapeamentoRedeATN-AA.***

**Se** FLG_BUSCACAT for 'Falso'

*FLG_BUSCACAT é 'Falso'*

**Então** Crie transições com estímulos diferentes de <categoria> do ESTADO_CORRENTE' para um estado final de erro.




Atribua 'Verdadeiro' a FLG_BUSCACAT.

*FLG_BUSCACAT= 'Verdadeiro'.*

**Volte** ao passo 4d.

***(Mapeamento do segundo arco de NP/)***

4d. **Enquanto** houver arcos ATN a serem processados **faça:**

*Há o arco:*

*(CAT N T*
*(SETR N *)*
*(SETR NUM (GETF NUMERO))*
*(SETR GEN (GETF GENERO))*
*(TO NP/N3))*

I. **Se** o arco for do tipo (POP <expressão> <teste>)

*Não é arco POP.*

II. **Se** o arco for do tipo (JUMP <estado> <teste> <ação>*)

*Não é arco JUMP*

III. **Se** o arco for do tipo (CAT <categoria> <teste> <ação>* (TO próx-estado))

*É arco CAT*

**Então Se** já existe uma transição com a chamada da submáquina BuscaCat, a partir de ESTADO_CORRENTE

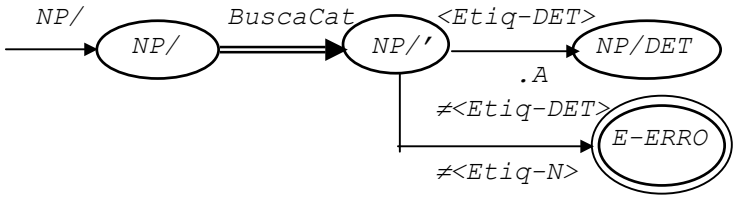
*Já existe tal transição a partir de ESTADO_CORRENTE*

**Então** Atribua a ESTADO_CORRENTE' o estado destino dessa transição

*ESTADO_CORRENTE':*

*NP/'*

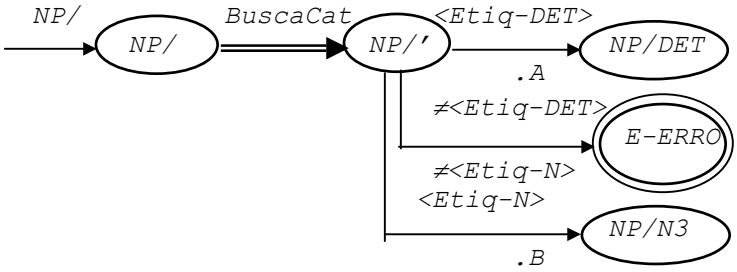
Elimine a transição com o estímulo <categoria>, de ESTADO_CORRENTE' para estado de erro.




**Se** <teste> for <T>

*<teste> é <T>*

**Então** Crie uma transição com o estímulo <categoria>, com o consumo da cadeia, de ESTADO_CORRENTE' para <próx-estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.




*Nesse instante, a rotina TrataAção é chamada para preparar o autômato para a execução de <ação>.*

*... prossegue-se de maneira semelhante ao tratamento de <ação> do arco anterior, obtendo-se:*

*Ação adaptativa B deve executar:*

- *inspeção para descobrir para qual estado o ESTR_CORR aponta.*
- *inserção da transição do estado RetornoExpr para esse estado.*
- *se $R_i$=E-N e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-N para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-N para o estado apontado por RetornoExpr.*
- *inspeção para descobrir $R_i$= estado para o qual o ESTR_CORR aponta.*
- *inspeção para descobrir para qual estado o $R_i$ aponta com estímulo NUMERO.*

- *inserção da transição do estado RetornoExpr para esse estado.*
- *se $R_i$=E-NUM e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-NUM para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-NUM para o estado apontado por RetornoExpr.*
- *inspeção para descobrir $R_i$= estado para o qual o ESTR_CORR aponta.*
- *inspeção para descobrir para qual estado o $R_i$ aponta com estímulo GENERO.*
- *inserção da transição do estado RetornoExpr para esse estado.*
- *se $R_i$=E-GEN e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-GEN para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-GEN para o estado apontado por RetornoExpr.*

1. **Enquanto** houver <ação> a ser processada **faça**:

*Não há mais <ação> a ser processada.*

2. Encerre o processamento.

*Fim da rotina TrataAção.*

*Retorna à rotina MapeamentoRedeATN-AA*

***Rotina MapeamentoRedeATN-AA.***

**Se** FLG_BUSCACAT for 'Falso'

*FLG_BUSCACAT é 'Verdadeiro'*

**Volte** ao passo 4d.

4d.**Enquanto** houver arcos ATN a serem processados **faça**:

*Não há mais arcos referentes ao estado NP/*

***(Mapeamento do segundo estado)***

4. **Enquanto** houver estados a serem processados **faça**:

*Há o estado: (NP/DET*

4a.**Se** já existe um estado correspondente ao estado sendo processado

*Já existe o estado NP/DET.*

**Então** Atribua a ESTADO_CORRENTE esse estado.

*ESTADO_CORRENTE:*

*NP/DET*

4b.**Se** for o primeiro estado da rede

*Não é o primeiro estado da rede.*

4c.Atribua 'Falso' a FLG_BUSCACAT.

*FLG_BUSCACAT= 'Falso'.*

***(Mapeamento do primeiro arco de NP/DET)***

4d.**Enquanto** houver arcos ATN a serem processados **faça:**

*Há o arco:*

*(CAT N T*

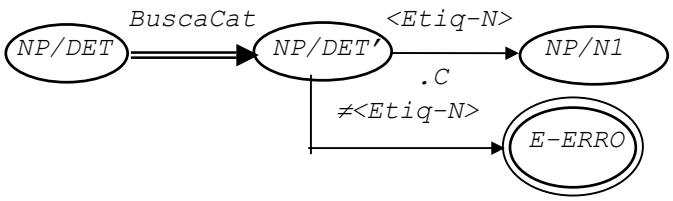
*(SETR N *)*

*(SETR NUM (GETF NUMERO))*

*(SETR GEN (GETF GENERO))*

*(TO NP/N1))*

*...prossegue-se de maneira semelhante ao mapeamento dos arcos anteriores, obtendo-se:*




*Ação adaptativa C deve executar:*

- *inspeção para descobrir para qual estado o ESTR_CORR aponta.*
- *inserção da transição do estado RetornoExpr para esse estado.*
- *se $R_i$=E-N e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-N para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-N para o estado apontado por RetornoExpr.*
- *inspeção para descobrir $R_i$= estado para o qual o ESTR_CORR aponta.*
- *inspeção para descobrir para qual estado o $R_i$ aponta com estímulo NUMERO.*
- *inserção da transição do estado RetornoExpr para esse estado.*
- *se $R_i$=E-NUM e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-NUM para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-NUM para o estado apontado por RetornoExpr.*

- *inspeção para descobrir $R_i$= estado para o qual o ESTR_CORR aponta.*
- *inspeção para descobrir para qual estado o $R_i$ aponta com estímulo GENERO.*
- *inserção da transição do estado RetornoExpr para esse estado.*
- *se $R_i$=E-GEN e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-GEN para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-GEN para o estado apontado por RetornoExpr.*

***(Mapeamento do terceiro estado)***

4. **Enquanto** houver estados a serem processados **faça**:

*Há o estado: (NP/N1*

4a.**Se** já existe um estado correspondente ao estado sendo processado

*Já existe o estado NP/N1.*

**Então** Atribua a ESTADO_CORRENTE esse estado.

*ESTADO_CORRENTE:*


4b.**Se** for o primeiro estado da rede

*Não é o primeiro estado da rede.*

4c.Atribua 'Falso' a FLG_BUSCACAT.

*FLG_BUSCACAT= 'Falso'.*

***(Mapeamento do primeiro arco de NP/N1)***

4d.**Enquanto** houver arcos ATN a serem processados **faça**:

*Há o arco:*

*(JUMP NP/N2 (AND (AGREE (GETR NUMDET) (GETR NUM))*
*(AGREE (GETR GENDET) (GETR GEN)))*

I. **Se** o arco for do tipo (POP <expressão> <teste>)

*Arco não é do tipo POP.*

II. **Se** o arco for do tipo (JUMP <estado> <teste> <ação>*)

*Arco é do tipo JUMP.*

**Então Se** <teste> for <T>

*<teste> não é <T>.*

**Senão** Chama a rotina PreparaTeste (ESTADO_CORRENTE, <estado>, ε, 'F').

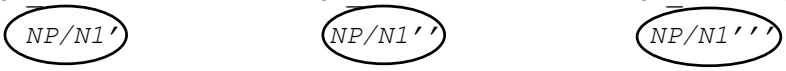
*Nesse instante, a rotina PreparaTeste é chamada para preparar o autômato para a execução de <teste>.*

***Rotina PreparaTeste.***

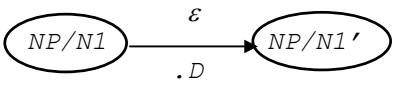
*EST_ORIG=NP/N1, EST_DEST=NP/N2, ESTÍMULO=ε, FLG_SUBM='F'*

1. Crie EST_INTERM1, EST_INTERM2, EST_INTERM3.

*EST_INTERM1* *EST_INTERM2* *EST_INTERM3*




2. Crie uma transição em vazio, de EST_ORIG para EST_INTERM1, cuja ação posterior seja a eliminação da transição entre EST_INTERM2 e EST_INTERM3.



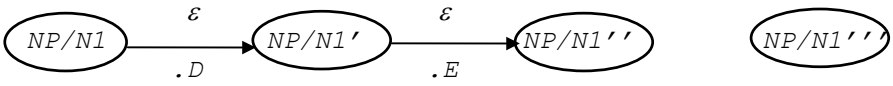

*Ação adaptativa D deve executar:*

- *eliminação da transição de NP/N1'' para NP/N1'''.*

3. **Se** FLAG_SUBM for igual a 'F'

*FLG_SUBM= 'F'*

**Então** Crie uma transição de EST_INTERM1 para EST_INTERM2, com estímulo ESTÍMULO, cuja ação posterior seja a execução de <teste>, tratada pela rotina TrataTeste. (Se <teste> for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3).




*Nesse instante, a rotina TrataTeste é chamada para tratar <teste>*

***Rotina TrataTeste.***

*<teste> = (AND (AGREE (GETR NUMDET) (GETR NUM))*
*(AGREE (GETR GENDET) (GETR GEN)))*

1. **Se** o <teste> é do tipo T

*<teste> não é do tipo T.*

2. **Se** o <teste> é do tipo (NULLR <reg>)

*<teste> não é do tipo (NULLR <reg>).*

3. **Se** o <teste> é do tipo (AGREE <expressão$_1$> <expressão$_2$>)

*<teste> não é do tipo (AGREE ... ).*

4. **Se** o <teste> é do tipo (GETR <reg>)

*<teste> não é do tipo (GETR <reg>).*

5. **Se** o <teste> é do tipo (AND $<teste_1>$ $<teste_2>$)

*<teste> é do tipo (AND ... ).*

**Então** Chame a rotina TrataTeste que procede o tratamento de $<teste_1>$ e retorna $RETORNO\_TESTE_1$ apontando para o estado que corresponde à $<teste_1>$.

*Nesse instante, a rotina TrataTeste é chamada para tratar $<teste_1>$.*

***Rotina TrataTeste – 2ª. instância.***

*<teste> =(AGREE (GETR NUMDET) (GETR NUM))*

1. **Se** o <teste> é do tipo T

*<teste> não é do tipo T.*

2. **Se** o <teste> é do tipo (NULLR <reg>)

*<teste> não é do tipo (NULLR <reg>).*

3. **Se** o <teste> é do tipo (AGREE $<expressão_1>$ $<expressão_2>$)

*<teste> é do tipo (AGREE ... ).*

**Então** Chame a rotina TrataExpressão que procede o tratamento de $<expressão_1>$ e retorna $RETORNO\_EXPR_1$ apontando para o estado que corresponde à $<expressão_1>$.

*Nesse instante, a rotina TrataExpressão é chamada para tratar $<expressão_1>$.*

***Rotina TrataExpressão.***

*<expressão> = GETR NUMDET*

1. **Se** a <expressão> for do tipo *

*<expressão> não é do tipo **

5. **Se** a <expressão> for do tipo 'palavra

*<expressão> não é do tipo 'palavra*

6. **Se** a <expressão> for do tipo (GETR <reg>)

*<expressão> é do tipo (GETR <reg>)*

**Então** Ache $R_i$ correspondente a <reg>.

*$R_i$= E-NUMDET.*

**Retorne** o estado para o qual o $R_i$ aponta.

*Ação adaptativa E deve executar:*

- *inspeção para descobrir qual o estado apontado por E-NUMDET.*
- *inserção da transição de $RETORNO\_EXPR_1$ para esse estado.*

*Retorna à rotina TrataTeste – 2ª. instância.*

***Rotina TrataTeste – 2ª. instância.***

Chame a rotina TrataExpressão que procede o tratamento de $<expressão_2>$ e retorna $RETORNO\_EXPR_2$ apontando para o estado que corresponde à $<expressão_2>$.

*... Prossegue-se de maneira semelhante, obtendo-se:*

*Ação adaptativa E deve executar:*

- *inspeção para descobrir qual o estado apontado por E-NUM.*
- *inserção da transição de RETORNO_EXPR$_2$ para esse estado.*

**Se** RETORNO_EXPR$_1$ e RETORNO_EXPR$_2$ apontam para o mesmo estado

**Então Retorne** o estado Verdadeiro.

**Senão Retorne** o estado Falso.

*Ação adaptativa E deve executar:*

- *se RETORNO_EXPR$_1$ e RETORNO_EXPR$_2$ apontam para o mesmo estado então inserção da transição de RETORNO_TESTE$_1$ para Verdadeiro.*
  *senão inserção da transição de RETORNO_TESTE$_1$ para Falso.*

*Retorna à rotina TrataTeste – 1ª. instância.*

***Rotina TrataTeste – 1ª. instância.***

Chame a rotina TrataTeste que procede o tratamento de <teste$_2$> e retorna RETORNO_TESTE$_2$ apontando para o estado que corresponde à <teste$_2$>.

*... Prossegue-se de maneira semelhante, obtendo-se:*

*Ação adaptativa E deve executar:*

- *inspeção para descobrir qual o estado apontado por E-GEN.*
- *inserção da transição de RETORNO_EXPR$_2$ para esse estado.*
- *inspeção para descobrir qual o estado apontado por E-GENDET.*
- *inserção da transição de RETORNO_EXPR$_1$ para esse estado.*
- *se RETORNO_EXPR$_1$ e RETORNO_EXPR$_2$ apontam para o mesmo estado então inserção da transição de RETORNO_TESTE$_2$ para Verdadeiro.*
  *senão inserção da transição de RETORNO_TESTE$_2$ para Falso.*

**Se** RETORNO_TESTE$_1$ e RETORNO_TESTE$_2$ apontam para o estado Verdadeiro

**Então Retorne** o estado Verdadeiro.

**Senão Retorne** o estado Falso.

*Ação adaptativa E deve executar:*

- *se RETORNO_TESTE$_1$ e RETORNO_TESTE$_2$ apontam para Verdadeiro*
  *então inserção da transição de RETORNO_TESTE para Verdadeiro.*
  *senão inserção da transição de RETORNO_TESTE para Falso.*

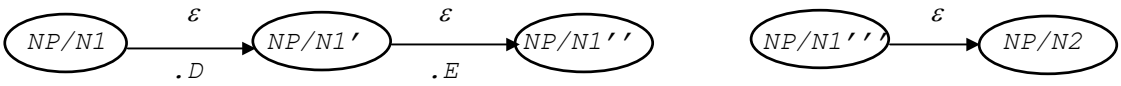
*Retorna à rotina PreparaTeste.*

***Rotina PreparaTeste.***

*Ação adaptativa E deve executar:*

- *se RETORNO_TESTE aponta para Verdadeiro*
  *então inserção da transição de NP/N1'' para NP/N1'''.*

4. Crie uma transição em vazio, de EST_INTERM3 para EST_DEST, cuja ação posterior seja a execução de <ação>, tratada pela rotina TrataAção.




*No arco sendo processado, não há <ação>.*

5. Encerre o processamento.

*Fim da rotina PreparaTeste..*

*Retorna à rotina MapeamentoRedeATN-AA*

***Rotina MapeamentoRedeATN-AA.***

**Volte** ao passo 4d.

4d.**Enquanto** houver arcos ATN a serem processados **faça**:

*Não há mais arcos referentes ao estado NP/N1*

***(Mapeamento do quarto estado)***

4. **Enquanto** houver estados a serem processados **faça**:

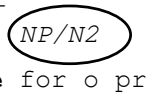
*Há o estado: (NP/N2*

4a.**Se** já existe um estado correspondente ao estado sendo processado

*Já existe o estado NP/N2.*

**Então** Atribua a ESTADO_CORRENTE esse estado.

*ESTADO_CORRENTE:*




4b.**Se** for o primeiro estado da rede

*Não é o primeiro estado da rede.*

4c.Atribua 'Falso' a FLG_BUSCACAT.

*FLG_BUSCACAT= 'Falso'.*

***(Mapeamento do primeiro arco de NP/N2)***

4d.**Enquanto** houver arcos ATN a serem processados **faça**:

*Há o arco:*

*(PUSH PP/ T (SETR PP *)) (TO NP/PP1))*

I. **Se** o arco for do tipo (POP <expressão> <teste>)

*Arco não é do tipo POP.*

II. **Se** o arco for do tipo (JUMP <estado> <teste> <ação>*)

*Arco não é do tipo JUMP.*

III. **Se** o arco for do tipo (CAT <categoria> <teste> <ação>* (TO próx-estado))

*Arco não é do tipo CAT.*

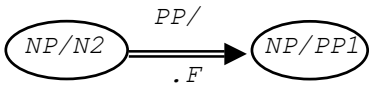
IV. **Se** o arco for do tipo (PUSH <estado> <teste> <ação>* (TO <próx-estado>))

*Arco é do tipo PUSH*

**Então Se** <teste> for <T>

*<teste> é T*

**Então** Crie uma transição com a chamada da submáquina <estado>, do ESTADO_CORRENTE para <próx-estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.




*Nesse instante, a rotina TrataAção é chamada para tratar <ação>.*

*... prosseguindo-se de maneira semelhante aos tratamentos de <ação> anteriores, obtém-se:*

*Ação adaptativa F deve executar:*

- *inspeção para descobrir qual o estado apontado por ESTR_CORR.*
- *inserção da transição de RetornoExpr para esse estado.*
- *se $R_i$=E-PP e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-PP para o estado para o qual aponta atualmente.*
  - *inserção da transição de E-PP para o estado apontado por RetornoExpr.*

**Volte** ao passo 4d.

***(Mapeamento do segundo arco de NP/N2)***

4d.**Enquanto** houver arcos ATN a serem processados **faça**:

*Há o arco:*

*(POP (BUILD (+ + + +) DET NUM GEN N) T)*

I. **Se** o arco for do tipo (POP <expressão> <teste>)

*Arco é do tipo POP.*

**Então** Crie um ESTADO_CORRENTE'.

*ESTADO_CORRENTE':*

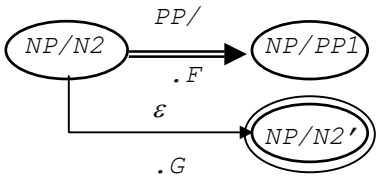

Acrescente ESTADO_CORRENTE' à lista de estados finais.

*NP_LISTA_EST_FINAL= {E-ERRO, NP/N2'}.*

**Se** <teste> for <T>

*<teste> é T.*

**Então** Crie uma transição em vazio do ESTADO_CORRENTE para ESTADO_CORRENTE', cuja ação posterior seja a execução de <expressão>, tratada pela rotina TrataExpressão.




*Nesse instante, a rotina TrataExpressão é chamada para tratar <expressão>.*

***Rotina TrataExpressão.***

1. **Se** a <expressão> for do tipo *

*<expressão> não é do tipo **

2. **Se** a <expressão> for do tipo 'palavra

*<expressão> não é do tipo 'palavra*

3. **Se** a <expressão> for do tipo (GETR <reg>)

*<expressão> não é do tipo GETR*

4. **Se** a <expressão> for do tipo (GETF <característica-sintática> <palavra>)

*<expressão> não é do tipo GETF*

5. **Se** a <expressão> for do tipo (APPEND <reg> <expressão>)

*<expressão> não é do tipo APPEND*

6. **Se** a <expressão> for do tipo (BUILD <formato> <expressão>*)

*<expressão> é do tipo (BUILD (+ + + +) DET NUM GEN N)*

**Então** Crie um EST_BUILD_INICIAL e um EST_BUILD_PTR.

*EST_BUILD_INICIAL:*


Atribua a EST_BUILD_PTR o EST_BUILD_INICIAL.

*EST_BUILD_PTR= EBI.*

**Enquanto** houver símbolos em <formato> **faça**

*Há o primeiro símbolo '+'.*

**Se** o símbolo for uma palavra

*Símbolo não é uma palavra.*

**Senão Se** o símbolo for igual a '+'

*Símbolo é '+'.*

**Então** Ache $R_i$ correspondente a $<reg_i>$.

*$R_i$= E-DET.*

Crie uma transição de EST_BUILD_PTR para $R_i$.

*Ação adaptativa G deve executar:*

- *inserção da transição de EBI para E-DET.*

**Enquanto** houver símbolos em <formato> **faça**

*Há o segundo símbolo '+'.*

**Se** o símbolo for uma palavra

*Símbolo não é uma palavra.*

**Senão Se** o símbolo for igual a '+'

*Símbolo é '+'.*

**Então** Ache $R_i$ correspondente a <$reg_i$>.

*$R_i$= E-NUM.*

Crie uma transição de EST_BUILD_PTR para $R_i$.

*Ação adaptativa G deve executar:*

- *inserção da transição de EBI para E-NUM.*

**Enquanto** houver símbolos em <formato> **faça**

*... prossegue-se de maneira semelhante para o terceiro e o quarto '+', obtendo-se:*

*Ação adaptativa G deve executar:*

- *inserção da transição de EBI para esse E-GEN.*
- *inserção da transição de EBI para E-N.*

**Retorne** EST_BUILD_INICIAL.

*Ação adaptativa G deve executar:*

- *inserção da transição de RetornoExpr para estado apontado por EBI.*

Ação posterior também deve atribuir a ESTR_CORR para o estado apontado por RetornoExpr.

*Ação adaptativa G deve executar:*

- *inserção da transição de ESTR_CORR para estado apontado por RetornoExpr.*

**Volte** ao passo 4d.

4d.**Enquanto** houver arcos ATN a serem processados **faça**:

*Não há mais arcos referentes ao estado NP'N2.*

### *(Mapeamento do quinto estado)*

4. **Enquanto** houver estados a serem processados **faça**:

*Há o estado: NP/PP1*

4a.**Se** já existe um estado correspondente ao estado sendo processado

*Já existe o estado NP/PP1.*

**Então** Atribua a ESTADO_CORRENTE esse estado.

*ESTADO_CORRENTE:*

*NP/PP1*

4b.**Se** for o primeiro estado da rede

*Não é o primeiro estado da rede.*

4c.Atribua 'Falso' a FLG_BUSCACAT.

*FLG_BUSCACAT= 'Falso'.*

***(Mapeamento do primeiro arco de NP/PP1)***

4d.**Enquanto** houver arcos ATN a serem processados **faça:**

*Há o arco:*

*(PUSH PP/ T (SETR PP (APPEND PP *)) (TO NP/PP1))*

I. **Se** o arco for do tipo (POP <expressão> <teste>)

*Arco não é do tipo POP.*

II. **Se** o arco for do tipo (JUMP <estado> <teste> <ação>*)

*Arco não é do tipo JUMP.*

III. **Se** o arco for do tipo (CAT <categoria> <teste> <ação>* (TO próx-estado))

*Arco não é do tipo CAT.*

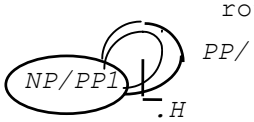
IV. **Se** o arco for do tipo (PUSH <estado> <teste> <ação>* (TO <próx-estado>))

*Arco é do tipo PUSH*

**Então Se** <teste> for <T>

*<teste> é T*

**Então** Crie uma transição com a chamada da submáquina <estado>, do ESTADO_CORRENTE para <próx-estado>, cuja ação posterior seja a execução de <ação>*, tratada pela rotina TrataAção.




*Nesse instante, a rotina TrataAção é chamada para tratar <ação>.*

***Rotina TrataAção.***

1. **Enquanto** houver <ação> a ser processada **faça:**

*Há a <ação> = (SETR PP (APPEND PP *))*

1c.Chame a rotina TrataExpressão que procede o tratamento de <expressão> e retorna RetornoExpr apontando para o estado que corresponde ao novo valor de <reg>.

*Nesse instante, a rotina TrataExpressão é chamada para tratar <expressão>.*

***Rotina TrataExpressão.***

1. **Se** a <expressão> for do tipo *

*<expressão> não é do tipo **

2. **Se** a <expressão> for do tipo 'palavra

*<expressão> não é do tipo 'palavra**

3. **Se** a <expressão> for do tipo (GETR <reg>)

*<expressão> não é do tipo GETR*

4. **Se** a <expressão> for do tipo (GETF <característica-sintática> <palavra>)

*<expressão> não é do tipo GETF*

5. **Se** a <expressão> for do tipo (APPEND <reg> <expressão>)

*<expressão> é do tipo APPEND*

**Então** Ache $R_i$ correspondente a <reg>

*$R_i$= E-PP.*

Chame a rotina TrataExpressão que procede o tratamento de <expressão> e retorna RetornoExpr apontando para o estado que corresponde à <expressão>.

*... Prossegue-se de maneira semelhante ao tratamento de expressões anteriores, obtendo-se:*

*Ação adaptativa H deve executar:*

- *inspeção para descobrir qual o estado apontado por ESTR_CORR.*
- *inserção da transição de RetornoExpr para esse estado.*

Crie uma transição de $R_i$ para o estado apontado por RetornoExpr.

*Ação adaptativa H deve executar:*

- *inserção da transição de E-PP para estado apontado por RetornoExpr.*

**Retorne** o estado $R_i$.

*Ação adaptativa F deve executar:*

- *inserção da transição de RetornoExpr para E-PP.*

*Retorna à rotina TrataAção.*

***Rotina TrataAção.***

1b.**Se** <ação> é HOLD

*<ação> não é HOLD.*

**Senão** Ache o $R_i$ que corresponde a <reg>. Se não houver tal $R_i$, então Imprima "Registrador <reg> não foi definido".

*$R_i$= E-PP.*

**Se** $R_i$ e RetornoExpr não apontam para o mesmo estado

**Então** Elimine a transição de $R_i$ para o estado para o qual aponta atualmente.
Crie uma transição de $R_i$ para o estado apontado por RetornoExpr.

*Ação adaptativa H deve executar:*

- *se $R_i$ e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-PP para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-PP para o estado apontado por RetornoExpr.*

2. Encerre o processamento.

*Fim da TrataAção.*

***Rotina MapeamentoRedeATN-AA.***

**Volte** ao passo 4d.

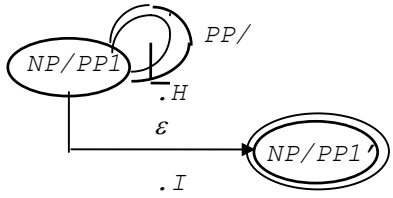
***(Mapeamento do segundo arco de NP/PP1)***

4d.**Enquanto** houver arcos ATN a serem processados **faça:**

*Há o arco:*

*(POP T (BUILD (+ + + + +) DET NUM GEN N PP))*

*... prossegue-se de maneira semelhante ao tratamento do arco POP anterior, obtendo-se:*




*NP_LISTA_EST_FINAL= {E-ERRO, NP/N2', NP/PP1'}.*

*Ação adaptativa I deve executar:*

- *inserção da transição de EBI para E-DET.*
- *inserção da transição de EBI para E-NUM.*
- *inserção da transição de EBI para esse E-GEN.*
- *inserção da transição de EBI para E-N.*
- *inserção da transição de EBI para E-PP.*
- *inserção da transição de RetornoExpr para estado apontado por EBI.*
- *inserção da transição de ESTR_CORR para estado apontado por RetornoExpr.*

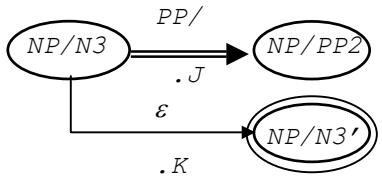
***(Mapeamento do sexto estado)***

*(NP/N3*

*(PUSH PP/ T (SETR PP *) (TO NP/PP2))*

```
    (POP (BUILD (+ + +) NUM GEN N) T)
)
```

*... prossegue-se de maneira semelhante ao tratamento dos estados anteriores, obtendo-se:*




*NP_LISTA_EST_FINAL= {E-ERRO, NP/N2',NP/PP1', NP/N3'}.*

*Ação adaptativa J deve executar:*

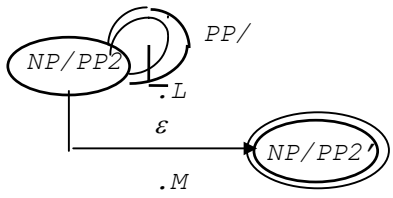
- *inspeção para descobrir qual o estado apontado por ESTR_CORR.*
- *inserção da transição de RetornoExpr para esse estado.*
- *se $R_i$=E-PP e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-PP para o estado para o qual aponta atualmente.*
  - *inserção da transição de E-PP para o estado apontado por RetornoExpr.*

*Ação adaptativa K deve executar:*

- *inserção da transição de EBI para E-NUM.*
- *inserção da transição de EBI para esse E-GEN.*
- *inserção da transição de EBI para E-N.*
- *inserção da transição de RetornoExpr para estado apontado por EBI.*
- *inserção da transição de ESTR_CORR para estado apontado por RetornoExpr.*

***(Mapeamento do sétimo e último estado)***

```
(NP/PP2
   (PUSH PP/ T (SETR PP (APPEND PP *)) (TO NP/PP2))
   (POP T (BUILD (+ + + +) NUM GEN N PP))
)
```




*NP_LISTA_EST_FINAL= {E-ERRO, NP/N2', NP/PP1', NP/PP2'}.*

*Ação adaptativa L deve executar:*

- *inspeção para descobrir qual o estado apontado por ESTR_CORR.*
- *inserção da transição de RetornoExpr para esse estado.*

- *inserção da transição de E-PP para estado apontado por RetornoExpr.*
- *inserção da transição de RetornoExpr para E-PP.*
- *se $R_i$=E-PP e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-PP para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-PP para o estado apontado por RetornoExpr.*

*Ação adaptativa M deve executar:*

- *inserção da transição de EBI para E-NUM.*
- *inserção da transição de EBI para esse E-GEN.*
- *inserção da transição de EBI para E-N.*
- *inserção da transição de EBI para E-PP.*
- *inserção da transição de RetornoExpr para estado apontado por EBI.*
- *inserção da transição de ESTR_CORR para estado apontado por RetornoExpr.*

*Como não há mais estados a serem processados na rede NP/, encerra-se o processamento da rotina MapeamentoRedeATN-AA, obtendo-se, assim, o autômato adaptativo correspondente à rede NP/.*

Reunindo-se todos os fragmentos de autômatos obtidos durante a simulação do mapeamento da Rede NP/ para Autômato Adaptativo, obtém-se o autômato, ilustrado na Figura 30, a seguir:

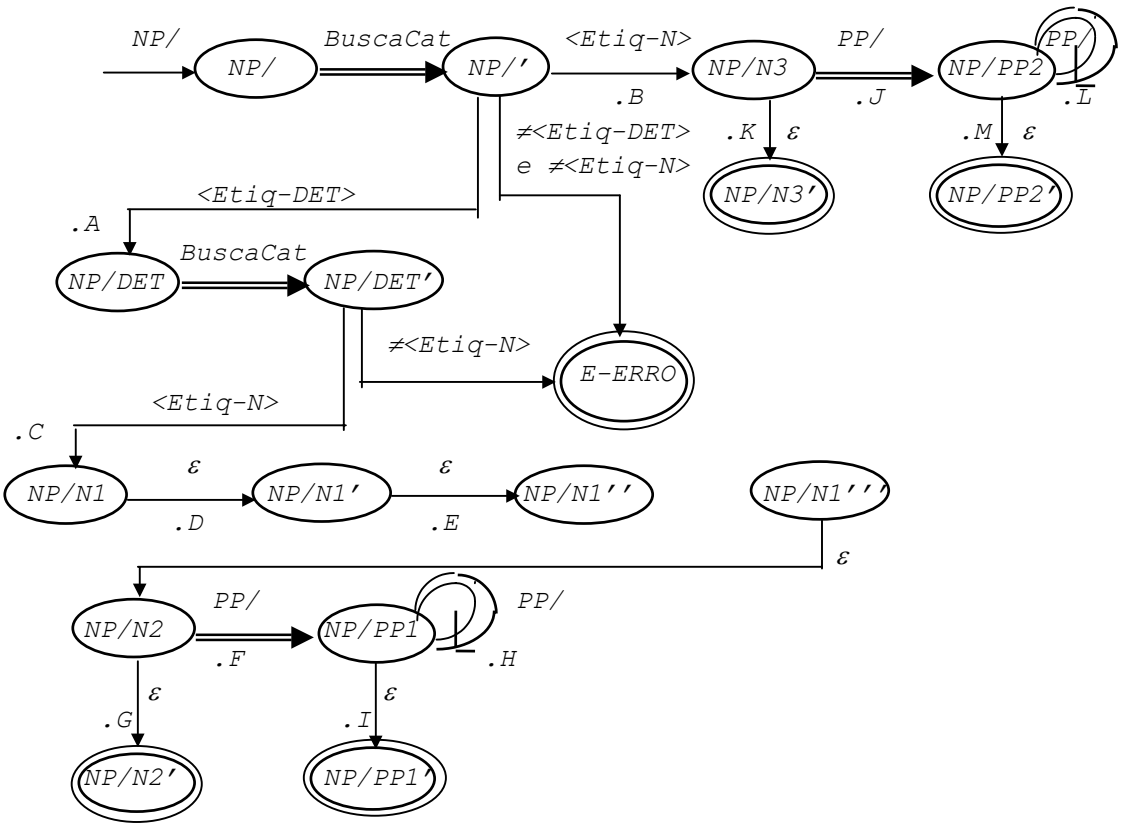



***Figura 30 - Autômato adaptativo correspondente ao mapeamento da Rede NP/.***

Seja a seguinte Rede ATN, que especifica um sintagma preposicional da gramática da língua portuguesa. Essa rede também foi obtida adaptando-se um exemplo equivalente à língua inglesa, extraído de [Bates-78].

```
(PP/
 (REGS PREP NP)
 (PP/
    (CAT PREP T (SETR PREP *) (TO PP/PREP))
 )
 (PP/PREP
    (PUSH NP/ T (SETR NP *) (TO PP/NP))
 )
 (PP/NP
    (POP (BUILD (+ +) PREP NP) T)
 )
)
```

Essa rede pode ser representada pelo seguinte diagrama:


***Figura 31 - Rede ATN correspondente à Rede PP/.***

Submetendo-se a rede PP/ ao algoritmo MapeamentoRedeATN-AA, obtém-se:

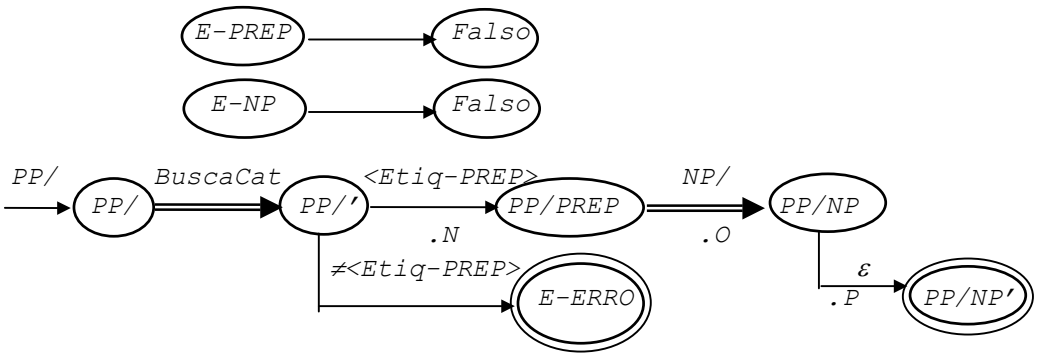



***Figura 32 - Autômato adaptativo correspondente ao mapeamento da Rede PP/.***

*NP_LISTA_EST_FINAL= {E-ERRO, PP/NP'}.*

*Ação adaptativa N deve executar:*

- *inspeção para descobrir para qual estado o ESTR_CORR aponta.*
- *inserção da transição do estado RetornoExpr para esse estado.*
- *se $R_i$=E-PREP e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-PREP para o estado para o qual aponta atualmente.*
  - *inserção da transição do estado E-PREP para o estado apontado por RetornoExpr.*

*Ação adaptativa O deve executar:*

- *inspeção para descobrir qual o estado apontado por ESTR_CORR.*
- *inserção da transição de RetornoExpr para esse estado.*
- *se $R_i$=E-NP e RetornoExpr não apontam para o mesmo estado, então:*
  - *eliminação da transição de E-NP para o estado para o qual aponta atualmente.*
  - *inserção da transição de E-NP para o estado apontado por RetornoExpr.*

*Ação adaptativa P deve executar:*

- *inserção da transição de EBI para E-PREP.*
- *inserção da transição de EBI para esse E-NP.*
- *inserção da transição de RetornoExpr para estado apontado por EBI.*
- *inserção da transição de ESTR_CORR para estado apontado por RetornoExpr.*

Submetendo-se o sintagma substantivo ‘a destruição da cidade’ ao analisador e etiquetador morfológico de [Menezes-00], obtém-se:

```
a /D-F destruição /N da /P+D-F cidade/N
```

Submetendo-se a saída do analisador e etiquetador morfológico referente ao sintagma substantivo ‘a destruição da cidade’ à rotina MapeamentoLexico, descrita no item 4.1 desta dissertação, obtém-se:

```
Cadeia de entrada:
<a> <destruição> <de> <a> <cidade>
```

sendo que <palavra> refere-se a uma etiqueta associada à palavra.

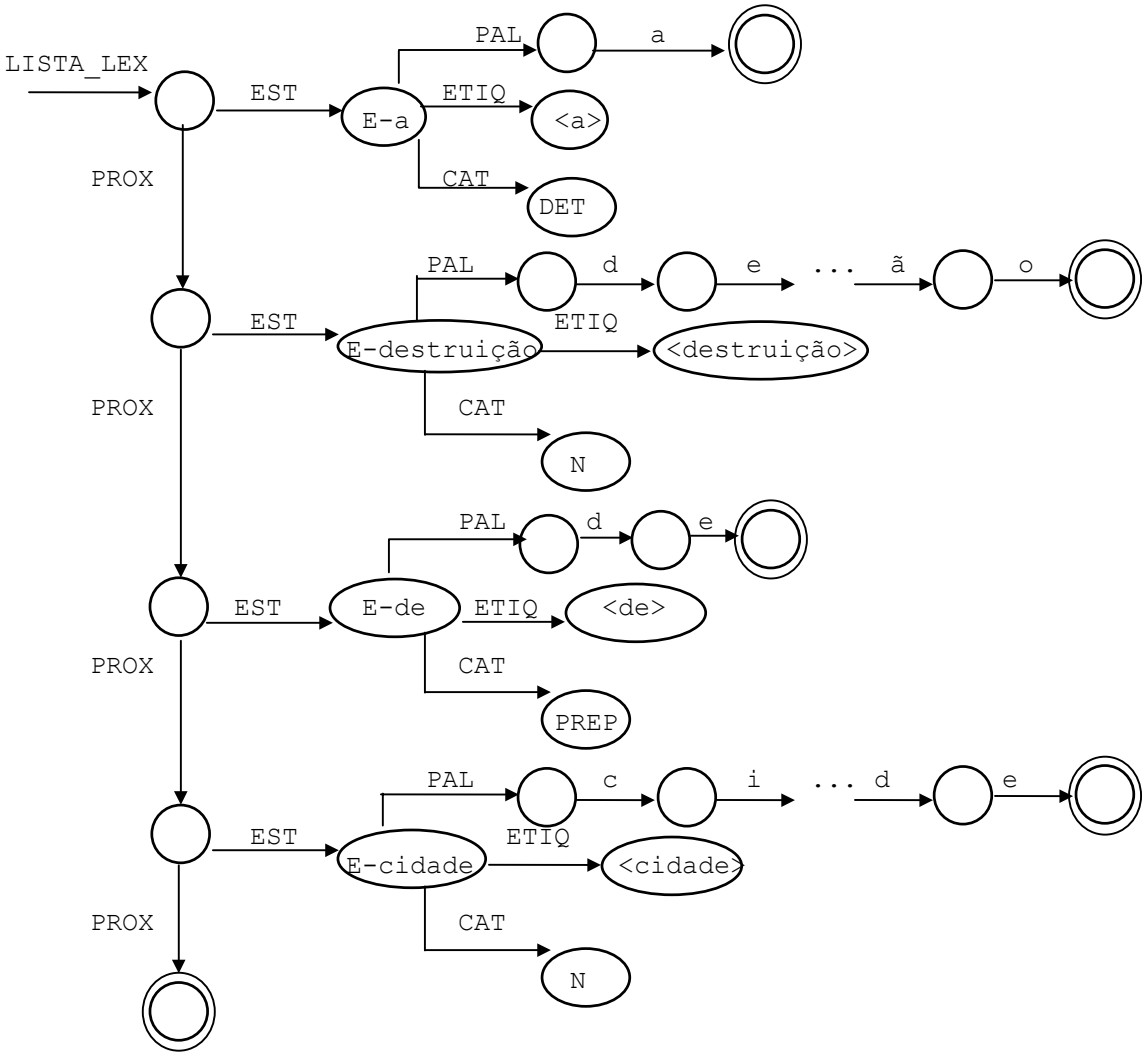



***Figura 33 - Autômato que representa a lista léxica correspondente ao sintagma substantivo 'a destruição da cidade'.***

Na Figura 33, por questões de simplicidade, foram omitidas as ilustrações referentes às demais características de cada palavra. Assim, do estado ‘E-a’ devem

partir transições correspondentes às características de ‘a’, para os estados que representam os respectivos valores: NUMERO= SG, GENERO= FEM.

Do estado ‘E-destruição’, omitiu-se as transições e estados referentes às características e valores: NUMERO= SG, GENERO= FEM, PESSOA= 3.

Do estado ‘E-cidade’, omitiu-se as transições e estados referentes às características e valores: NUMERO= SG, GENERO= FEM, PESSOA= 3.

Submetendo-se a saída da rotina MapeamentoLéxico referente ao sintagma substantivo ‘a destruição da cidade’ aos autômatos adaptativos gerados pelo mapeamento das redes ATN NP/ e PP/, obtém-se a seguinte estrutura, correspondente à análise sintática do sintagma. (A simulação passo a passo foi omitida, para que a dissertação não ficasse muito extensa).

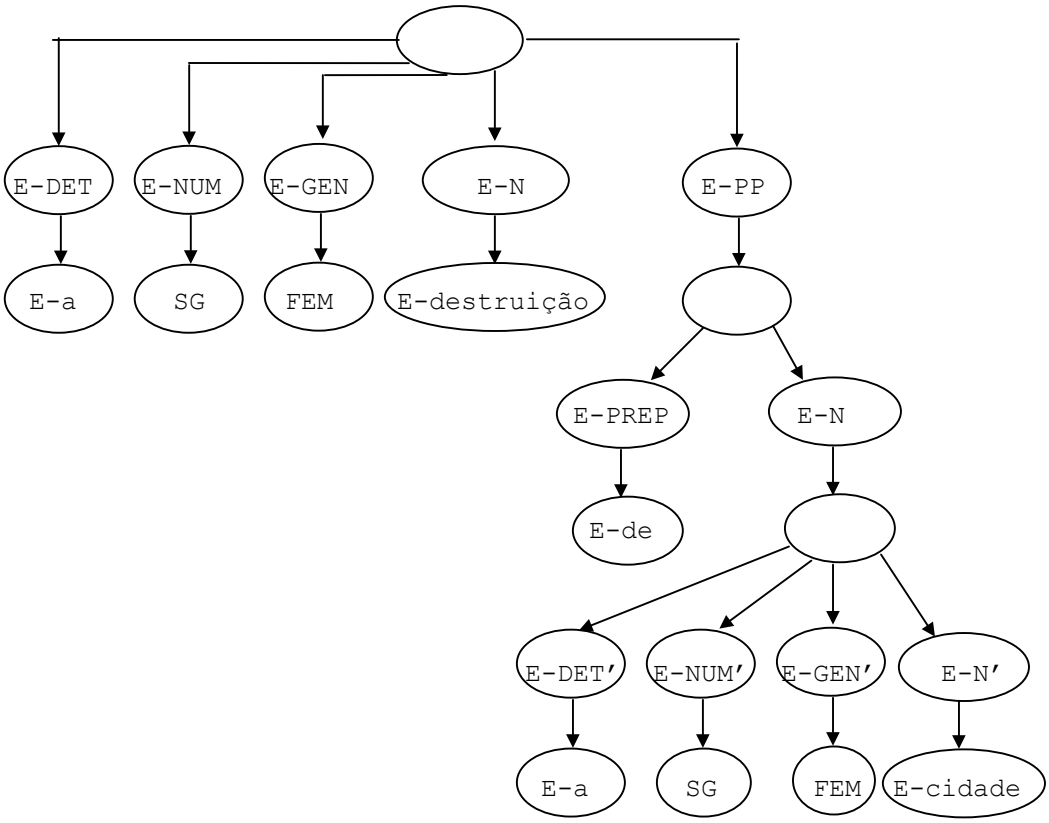



***Figura 34 - Estrutura obtida como resultado do reconhecimento do sintagma substantivo 'a destruição da cidade', pelos autômatos adaptativos correspondentes às Redes NP/ e PP/.***

Observa-se neste exemplo, que houve a verificação da concordância entre o determinante e o substantivo em ‘a destruição’ e em ‘a cidade’.

Comprova-se, assim, que os autômatos adaptativos resultantes do mapeamento das Redes ATN NP/ e PP/ para Autômato Adaptativo são capazes de analisar sintaticamente um sintagma substantivo e criar uma estrutura sintática em decorrência dessa análise.

Novos experimentos podem ser realizados, mapeando-se os demais sintagmas de uma sentença, e assim, obter um analisador sintático representado através de Autômato Adaptativo, capaz de reconhecer sentenças da língua portuguesa.

O próximo item descreve as simulações realizadas do mapeamento de GPSG para Autômato Adaptativo.

### 5.1.2. Experimentos Relativos ao Mapeamento de GPSG para Autômato Adaptativo

Sejam as seguintes regras ID:

```
(1)a.  N1 → H[1.0]
   b.  N1 → H[1.1], P2[de]
   c.  N2 → H1
   d.  N2 → [SUBCAT D], H2
   e.  P1 → H[3.0], n2
   f.  P2 → H1
```

Submetendo-se a regra 1a ao algoritmo de mapeamento de regra ID para Autômato Adaptativo (MapeamentoRegraID-AA), tem-se a simulação de execução passo a passo, descrita a seguir. As partes dos algoritmos que estão sendo executadas encontram-se alternadas com a descrição da simulação de sua respectiva execução. A simulação da execução está representada em letras em itálico.

***Rotina MapeamentoRegraID-AA.***

*N1 → H[1.0]*

1. Atribua a $X_i$ a categoria $X_1$.

*$X_i$= H[1.0]*

2. **Se** já existe um autômato adaptativo correspondente a $X_0$,

*Não existe autômato adaptativo correspondente a N1.*

**Senão** Inicialize o contador de categorias de $X_0$: X0_CONT_CAT.

*N1_CONT_CAT= 0.*

Crie um estado inicial para $X_0$.

*$q_0$ corresponde a N1.*

Atribua a ESTADO_ORIGEM_TRANS o estado inicial.

*ESTADO_ORIGEM_TRANS= $q_0$*

3. Crie um estado para $X_i$.

*$q_1$ corresponde a H[1.0].*

4. Atribua a ESTADO_DESTINO_TRANS o estado criado para $X_i$.

*ESTADO_DESTINO_TRANS= $q_1$.*

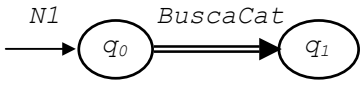
5. **Se** $X_i$ corresponder a um item léxico

*H[1.0] corresponde a um item léxico.*

**Então Se** a partir de ESTADO_ORIGEM_TRANS não existe uma chamada de submáquina BuscaCat

*Não existe chamada de submáquina BuscaCat a partir de $q_0$.*

**Então** Crie uma transição de ESTADO_ORIGEM_TRANS para ESTADO_DESTINO_TRANS, que é uma chamada à submáquina BuscaCat.




Crie um estado e atribua seu valor a ESTADO_FINAL_XI.

*ESTADO_FINAL_XI= $q_1$'*

Chame a rotina PreparaTesteGPSG (ESTADO_DESTINO_TRANS, ESTADO_FINAL_XI, $X_i$, 'V', 'F'). Essa transição deve ser associada a uma ação adaptativa posterior: <ação>, que criará um estado correspondente à categoria reconhecida e uma transição partindo desse estado para o estado inicial do item sendo tratado, e verificará a existência de alguma característica não-nuclear instanciada para a categoria correspondente.

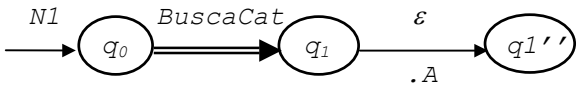
*Nesse instante, a Rotina PreparaTesteGPSG é chamada.*

***Rotina PreparaTesteGPSG.***

1. Crie EST_INTERM1, EST_INTERM2, EST_INTERM3.

*EST_INTERM1= $q_1$'' EST_INTERM2= $q_1$''' EST_INTERM3= $q_1$''''.*

2. Crie uma transição em vazio, de EST_ORIG para EST_INTERM1, cuja ação posterior seja a eliminação da transição entre EST_INTERM2 e EST_INTERM3.



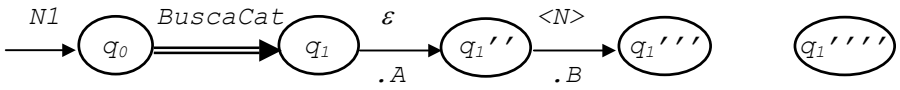

*Ação adaptativa A deve executar:*

- *eliminação da transição de $q_1$''' para $q_1$''''.*

3. **Se** FLG_UNIF = 'F'

*FLG_UNIF= 'F'.*

**Então** Crie uma transição de EST_INTERM1 para EST_INTERM2, com estímulo ESTÍMULO, cuja ação posterior seja a execução de <teste>, tratada pela rotina TrataTesteGPSG. (Se <teste> for Verdadeiro, então cria transição em vazio, de EST_INTERM2 para EST_INTERM3).




*Nesse instante, a Rotina TrataTesteGPSG é chamada.*

***Rotina TrataTesteGPSG.***

1. Crie uma ação adaptativa.

*Ação adaptativa B.*

2. Acrescente à ação adaptativa:

*Ação adaptativa B deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*

3. Para cada uma das características de $X_i$, acrescente à ação adaptativa:
   - inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo que representa a característica de $X_i$.
   - **Se** não houver tal transição,
     **Então Retorne** estado Falso.
   - **Se** o estado que representa o valor corresponde a essa característica de $X_i$,
     **Então Retorne** estado Falso.

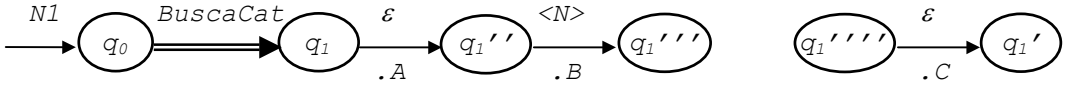
*Ação adaptativa B deve executar:*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '+'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '0'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo SUBCAT para o estado '1.0'.*
- *se alguma dessas inspeções falharem,*
  *então inserção da transição de RetornoTeste para estado Falso.*
  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*

*Fim da Rotina TrataTesteGPSG.*

***Rotina PreparaTesteGPSG.***

*Ação adaptativa B deve executar:*
- *se RetornoTeste aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $q_1'''$ para $q_1''''$*

4. Se FLG_AÇÃO = 'V'
   Então Crie uma transição em vazio, de EST_INTERM3 para EST_DEST, cuja ação posterior seja a execução de <ação>, tratada pela rotina TrataAçãoGPSG.




*Nesse instante, a rotina TrataAçãoGPSG é chamada.*

***Rotina TrataAçãoGPSG.***

1. Crie uma ação adaptativa.

*Ação adaptativa C.*

2. Crie um ESTADO_XI correspondente a $X_i$.

*ESTADO_XI = EST-N.*

3. Acrescente à ação adaptativa:

*Ação adaptativa C deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- *inserção da transição ESTADO_XI para ESTADO_INFO_XI, com estímulo igual a 'N0'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR.*
- ***Se** o valor de BAR é zero,*

  ***Então** Se de ESTADO_INFO_XI parte alguma transição com estímulo SLASH*

  *Então inspeção para verificar se de ESTADO_XI parte alguma transição com estímulo FOOT_INST.*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição de ESTADO_XI para um estado novo com estímulo FOOT_INST.*

  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*

  *Se de ESTADO_INFO_XI parte alguma transição com estímulo WH*

  *Então inspeção para verificar se de ESTADO_XI parte alguma transição com estímulo FOOT_INST.*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição de ESTADO_XI para um estado novo com estímulo FOOT_INST.*

  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*

  *Se de ESTADO_INFO_XI parte alguma transição com estímulo RE*

  *Então inspeção para verificar se de ESTADO_XI parte alguma transição com estímulo FOOT_INST.*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição de ESTADO_XI para um estado novo com estímulo FOOT_INST.*

  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*

4. **Retorne.**

*Fim da Rotina TrataAçãoGPSG.*

***Rotina PreparaTesteGPSG.***

5. Encerre o processamento.

*Fim da Rotina PreparaTesteGPSG.*

***Rotina MapeamentoRegraID-AA.***

6. Crie um estado e atribua esse valor a PROXIMO_ESTADO.

*PROXIMO_ESTADO= $q_2$.*

7. **Se** $X_i$ está entre parênteses

*$X_1$ não está entre parênteses.*

8. **Se** $X_i$ está entre chaves

*$X_1$ não está entre chaves.*

9. Insira em uma lista referente ao autômato $X_0$, o ESTADO_ORIGEM_TRANS, o ESTADO_FINAL_XI e a categoria-filha correspondente $X_i$.

| *ESTADO_ORIGEM_TRANS* | *ESTADO_FINAL_XI* | *XI* |
|---|---|---|
| $q_0$ | $q_1'$ | *H[1.0]* |

10. Atribua a ESTADO_ORIGEM_TRANS o ESTADO_FINAL_XI.

*ESTADO_ORIGEM_TRANS= $q_1'$.*

11. Atribua a ESTADO_DESTINO_TRANS o PROXIMO_ESTADO.

*ESTADO_DESTINO_TRANS= $q_2$.*

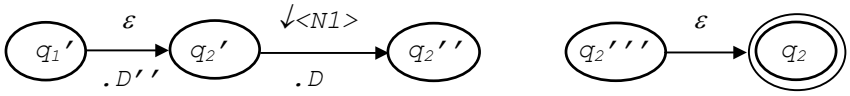
12. **Se** $X_i$ for diferente de $X_n$,

*$X_1$ é igual a $X_n$*

13. Chame a rotina PreparaTesteGPSG (ESTADO_ORIGEM_TRANS, ESTADO_DESTINO_TRANS, $X_0$, 'F', 'V'), que prepara o autômato para o teste de unificação e cria uma transição com inserção, na cadeia de entrada, de etiqueta correspondente a $X_0$, e com ação adaptativa posterior (tratada pela rotina ReconheceRegra) que monta a estrutura referente a $X_0$, devolvendo ESTR_CORR apontando para essa estrutura, e verifica se a regra satisfaz os princípios CAP, FFP e HFC.

*Nesse instante, a rotina PreparaTesteGPSG é chamada.*

*... prosseguindo-se como no tratamento anterior de PreparaTesteGPSG, obtém-se:*




*Ação adaptativa D'' deve executar:*

- *eliminação da transição de $q_2''$ para $q_2'''$.*

*Nesse instante, a rotina ReconheceRegra é chamada.*

***Rotina ReconheceRegra.***

1. Crie uma ação adaptativa.

*Ação adaptativa D.*

2. Crie um ESTADO_X0 correspondente a $X_0$.

*ESTADO_X0= EST-N1.*

3. Atribua a $X_i$ o valor de $X_1$.

*$X_i$ = $X_1$ = H[1.0].*

4. Acrescente à ação adaptativa:

- Acrescente a ESTADO_X0 as características herdadas de $X_0$.

*Ação adaptativa D deve executar:*

- *inserção de uma transição de ESTADO_CORR para ESTADO_X0.*
- *inserção da transição de ESTADO_X0 para estado '+', com estímulo N.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo V.*
- *inserção da transição de ESTADO_X0 para estado '1', com estímulo BAR.*
- **enquanto** $X_i$ for diferente de $X_n$ **faça**
  - inserção da transição de ESTADO_X0 para estado correspondente a $X_i$, com estímulo igual ao nome da categoria referente a $X_i$.
  - atribua a $X_i$ o valor de $X_{i+1}$.

*Ação adaptativa D deve executar:*

- *inserção da transição de ESTADO_X0 para EST-N, com estímulo N0.*

5. **(Princípio CAP)**

**Se** n ≥ 2

*n = 1.*

6. **(Princípio FFP)**

Atribua a i o valor de 1.

*i=1.*

**Enquanto** i ≤ n **faça**

Acrescente à ação adaptativa:

- **Se** há alguma CARAC=característica não-nuclear de $X_i$ não definida para $X_0$

  **Então** Unifique ($X_0$, $X_i$, CARAC).

  **se** Retorno aponta para estado Falso

  **então Retorne** o estado Falso.

Incremente o valor de i.

*Acrescente à ação adaptativa D:*

- *inspeção para descobrir o estado F para o qual o EST-N aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*

  *então se de EST-N1 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-N1, F, SLASH).*
  *Se Retorno aponta para o estado Falso*
  *Então Retorne.*
- *se F existe e se de F parte alguma transição com estímulo WH*

  *então se de EST-N1 não parte nenhuma transição com estímulo WH*
  *Unifique (EST-N1, F, WH).*
  *Se Retorno aponta para o estado Falso*
  *Então Retorne.*
- *se F existe e se de F parte alguma transição com estímulo RE*

  *então se de EST-N1 não parte nenhuma transição com estímulo RE*
  *Unifique (EST-N1, F, RE).*
  *Se Retorno aponta para o estado Falso*
  *Então Retorne.*

7. **(Princípio HFC)**

Atribua a $X_h$ o valor de $X_i$ nuclear.

*$X_h = X_1 = H[1.0]$.*

Acrescente à ação adaptativa:

- Para cada característica nuclear CARAC, instanciada em $X_h$, Unifique ($X_0$, $X_h$, CARAC).
  se Retorno aponta para estado Falso
  então Retorne.

*Acrescente à ação adaptativa D:*

- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo AGR*
  *então Unifique (EST-N1, ESTADO_INFO_XI, AGR).*
  *Se Retorno aponta para o estado Falso,*
  *Então Retorne.*
- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo PER*
  *então Unifique (EST-N1, ESTADO_INFO_XI, PER).*
  *Se Retorno aponta para o estado Falso,*

*Então Retorne.*

- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo PLU*
  *então Unifique (EST-N1, ESTADO_INFO_XI, PLU).*
  *Se Retorno aponta para o estado Falso,*
  *Então Retorne.*
- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo MASC*
  *Se Retorno aponta para o estado Falso,*
  *Então Retorne.*
  *então Unifique (EST-N1, ESTADO_INFO_XI, MASC).*

8. **Retorne** o estado Verdadeiro.

*Fim da rotina ReconheceRegra.*

*Acrescente à ação adaptativa D:*

- *se Retorno aponta para o estado Verdadeiro*
  *então inserção da transição em vazio, de $q_2''$ para $q_2'''$.*

***Rotina MapeamentoRegraID-AA.***

14. Acrescente ESTADO_DESTINO_TRANS à lista de estados finais do autômato adaptativo correspondente a $X_0$.

*N1_LISTA_ESTADO_FINAL= {$q_2$}.*

15. Encerre o processamento.

*Fim da rotina MapeamentoRegraID-AA.*

Submetendo-se a regra 1b ao algoritmo de mapeamento de regra ID para Autômato Adaptativo (MapeamentoRegraID-AA), tem-se a simulação de execução passo a passo, descrita a seguir.

***Rotina MapeamentoRegraID-AA.***

*N1 → H[1.1], P2[de]*

1. Atribua a $X_i$ a categoria $X_1$.

*$X_i$= H[1.1]*

2. **Se** já existe um autômato adaptativo correspondente a $X_0$,

*Já existe autômato adaptativo correspondente a N1.*

**Então** Atribua a ESTADO_CORR o estado inicial do autômato correspondente a $X_0$.

*ESTADO_CORR= $q_0$.*

Atribua a i o valor 1.

*i=1.*

Atribua a CATEG a categoria $X_i$.

*CATEG= H[1.1]*

Inicialize CONT com 1.

*CONT= 1.*

**Enquanto** CONT ≤ X0_CONT_CAT **faça**

Atribua a IND o valor de CONT.

*IND= 1.*

Atribua a ACHOU o valor FALSO.

*ACHOU= FALSO.*

**Enquanto** IND ≤ X0_CONT_CAT e não ACHOU **faça**

**Se** na lista referente a X0, ESTADO_ORIGEM_TRANS correspondente ao índice IND for igual a ESTADO_CORR

*ESTADO_ORIGEM_TRANS correspondente a IND=1 é $q_0$.*

**Então** ACHOU= VERDADEIRO.

*ACHOU= VERDADEIRO.*

**Se** ACHOU

**Então Se** na lista referente a X0, CATEGORIA correspondente ao índice IND for igual a CATEG

*CATEGORIA correspondente ao IND=1 é H[1.0], que é diferente de CATEG.*

**Senão** Incremente CONT.

*CONT= 2.*

Atribua a ESTADO_ORIGEM_TRANS o ESTADO_CORR.

*ESTADO_ORIGEM_TRANS= $q_0$.*

3. Crie um estado para $X_i$.

*$q_3$ corresponde a H[1.1].*

4. Atribua a ESTADO_DESTINO_TRANS o estado criado para $X_i$.

*ESTADO_DESTINO_TRANS= $q_3$.*

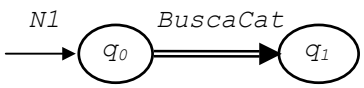
5. **Se** $X_i$ corresponder a um item léxico

*H[1.1] corresponde a um item léxico.*

**Então Se** a partir de ESTADO_ORIGEM_TRANS não existe uma chamada de submáquina BuscaCat

*Existe chamada de submáquina BuscaCat a partir de $q_0$.*

**Senão** Atribua a ESTADO_DESTINO_TRANS o estado para o qual aponta a chamada da submáquina BuscaCat.




*ESTADO_DESTINO_TRANS= $q_1$.*

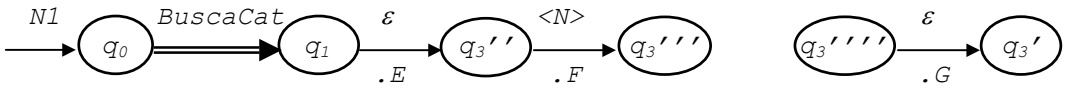
Crie um estado e atribua seu valor a ESTADO_FINAL_XI.

*ESTADO_FINAL_XI= $q_3$'*

Chame a rotina PreparaTesteGPSG (ESTADO_DESTINO_TRANS, ESTADO_FINAL_XI, $X_i$, 'V', 'F'). Essa transição deve ser associada a uma ação adaptativa posterior: <ação>, que criará um estado correspondente à categoria reconhecida e uma transição partindo desse estado para o estado inicial do item sendo tratado, e verificará a existência de alguma característica não-nuclear instanciada para a categoria correspondente.

*Nesse instante, a Rotina PreparaTesteGPSG é chamada.*

*...prosseguindo-se como no tratamento da regra anterior, obtém-se:*




*Ação adaptativa E deve executar:*

- *eliminação da transição de $q_3'''$ para $q_3''''$.*

*Ação adaptativa F deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '+'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '0'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo SUBCAT para o estado '1.1'.*
- *se alguma dessas inspeções falharem,*

  *então inserção da transição de RetornoTeste para estado Falso.*

  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*

- *se RetornoTeste aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $q_3'''$ para $q_3''''$*

*Ação adaptativa G deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- *inserção da transição de EST-N para ESTADO_INFO_XI, com estímulo igual a N0.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR.*
- ***Se** o valor de BAR é zero,*
  ***Então** Se de ESTADO_INFO_XI parte alguma transição com estímulo SLASH*
  *Então inspeção para verificar se de EST-N parte alguma transição com estímulo FOOT_INST.*
  ***Se** não houver tal transição,*
  ***Então** inserção da transição de EST-N para um estado novo com estímulo FOOT_INST.*
  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*
  *Se de ESTADO_INFO_XI parte alguma transição com estímulo WH*
  *Então inspeção para verificar se de EST-N parte alguma transição com estímulo FOOT_INST.*
  ***Se** não houver tal transição,*
  ***Então** inserção da transição de EST-N para um estado novo com estímulo FOOT_INST.*
  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*
  *Se de ESTADO_INFO_XI parte alguma transição com estímulo RE*
  *Então inspeção para verificar se de EST-N parte alguma transição com estímulo FOOT_INST.*
  ***Se** não houver tal transição,*
  ***Então** inserção da transição de EST-N para um estado novo com estímulo FOOT_INST.*
  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*

6. Crie um estado e atribua esse valor a PROXIMO_ESTADO.
*PROXIMO_ESTADO= $q_4$.*
7. **Se** $X_i$ está entre parênteses

*$X_1$ não está entre parênteses.*

8. **Se** $X_i$ está entre chaves

*$X_1$ não está entre chaves.*

9. Insira em uma lista referente ao autômato $X_0$, o ESTADO_ORIGEM_TRANS, o ESTADO_FINAL_XI e a categoria-filha correspondente $X_i$.

| *ESTADO ORIGEM TRANS* | *ESTADO FINAL XI* | *XI* |
|---|---|---|
| $q_0$ | $q_1'$ | *H[1.0]* |
| $q_0$ | $q_3'$ | *H[1.1]* |

10. Atribua a ESTADO_ORIGEM_TRANS o ESTADO_FINAL_XI.

*ESTADO_ORIGEM_TRANS= $q_3'$.*

11. Atribua a ESTADO_DESTINO_TRANS o PROXIMO_ESTADO.

*ESTADO_DESTINO_TRANS= $q_4$.*

12. **Se** $X_i$ for diferente de $X_n$,

*$X_1$ é diferente de $X_n$.*

**Então** Atribua a $X_i$ a categoria $X_{i+1}$.

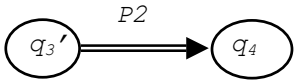
*$X_i$ = P2[de].*

**Volta** para Passo 5.

5. **Se** $X_i$ corresponder a um item léxico

*$X_i$ não corresponde a um item léxico.*

**Senão** Crie uma transição de ESTADO_ORIGEM_TRANS para ESTADO_DESTINO_TRANS, que é uma chamada à submáquina $X_i$.



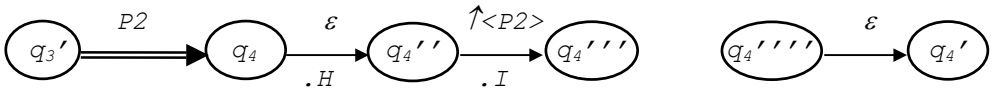

Crie um estado e atribua seu valor a ESTADO_FINAL_XI.

*ESTADO_FINAL_XI= $q_4'$.*

Chame a rotina PreparaTesteGPSG (ESTADO_DESTINO_TRANS, ESTADO_FINAL_XI, $X_i$, 'F', 'F'). Essa transição deve ser associada a uma ação adaptativa posterior, que criará um estado correspondente à categoria reconhecida e uma transição partindo desse estado para o estado inicial do item sendo tratado, e verificará a existência de alguma característica não-nuclear instanciada para a categoria correspondente.

*Nesse instante, a Rotina PreparaTesteGPSG é chamada.*

*...prosseguindo-se como no tratamento da regra anterior, obtém-se:*



*Ação adaptativa H deve executar:*

- *eliminação da transição de $q_4'''$ para $q_4''''$.*

*Ação adaptativa I deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD não léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '2'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo PFORM para o estado 'de'.*
- *se alguma dessas inspeções falharem,*

  *então inserção da transição de RetornoTeste para estado Falso.*

  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*
- *se RetornoTeste aponta para estado Verdadeiro*

  *então inserção da transição em vazio, de $q_4'''$ para $q_4''''$*

6. Crie um estado e atribua esse valor a PROXIMO_ESTADO.

*PROXIMO_ESTADO= $q_5$.*

7. **Se** $X_i$ está entre parênteses

*$X_i$ não está entre parênteses.*

8. **Se** $X_i$ está entre chaves

*$X_i$ não está entre chaves.*

9. Insira em uma lista referente ao autômato $X_0$, o ESTADO_ORIGEM_TRANS, o ESTADO_FINAL_XI e a categoria-filha correspondente $X_i$ e atualiza o contador de categorias de $X_0$: X0_CONT_CAT.

| *ESTADO ORIGEM TRANS* | *ESTADO FINAL XI* | *XI* |
|---|---|---|
| *$q_0$* | *$q_1'$* | *H[1.0]* |

| | | |
|---|---|---|
| $q_0$ | $q_3'$ | *H[1.1]* |
| $q_3'$ | $q_4'$ | *P2[de]* |

10. Atribua a ESTADO_ORIGEM_TRANS o ESTADO_FINAL_XI.

*ESTADO_ORIGEM_TRANS= $q_4'$.*

11. Atribua a ESTADO_DESTINO_TRANS o PROXIMO_ESTADO.

*ESTADO_DESTINO_TRANS= $q_5$.*

12. **Se** $X_i$ for diferente de $X_n$,

*$X_i$ é igual a $X_n$.*

13. Chame a rotina PreparaTesteGPSG (ESTADO_ORIGEM_TRANS, ESTADO_DESTINO_TRANS, $X_0$, 'F', 'V'), que prepara o autômato para o teste de unificação e cria uma transição com inserção, na cadeia de entrada, de etiqueta correspondente a $X_0$, e com ação adaptativa posterior (tratada pela rotina ReconheceRegra) que monta a estrutura referente a $X_0$, devolvendo ESTR_CORR apontando para essa estrutura, e verifica se a regra satisfaz os princípios CAP, FFP e HFC.

*Nesse instante, a rotina PreparaTesteGPSG é chamada.*

*...prosseguindo-se como no tratamento da regra anterior, obtém-se:*


*Ação adaptativa K'' deve executar:*

- *eliminação da transição de $q_5$'' para $q_5$'''.*

*Ação adaptativa K deve executar:*

- *criação de um estado EST-N1 (ESTADO_X0).*
- *inserção de uma transição de ESTADO_CORR para ESTADO_X0.*
- *inserção da transição de ESTADO_X0 para estado '+', com estímulo N.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo V.*
- *inserção da transição de ESTADO_X0 para estado '1', com estímulo BAR.*
- *inserção da transição de ESTADO_X0 para EST-N, com estímulo N0.*
- *inserção da transição de ESTADO_X0 para EST-P, com estímulo P2.*
- *inspeção para descobrir o estado F para o qual o EST-N aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-N1 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-N1, F, SLASH).*
  *Se Retorno aponta para estado Falso*
  *Então Retorne.*
- *(idem ação anterior, para estímulo= WH)*
- *(idem ação anterior, para estímulo= RE)*
- *inspeção para descobrir o estado F para o qual o EST-P aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-N1 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-N1, F, SLASH).*
- *(idem ação anterior, para estímulo= WH)*
- *(idem ação anterior, para estímulo= RE)*
- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo AGR*
  *então Unifique (EST-N1, ESTADO_INFO_XI, AGR).*

*Se Retorno aponta para estado Falso*
*Então Retorne.*

- *(idem ação anterior, para estímulo PER)*
- *(idem ação anterior, para estímulo PLU)*
- *(idem ação anterior, para estímulo MASC)*
- *se Retorno aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $q_5''$ para $q_5'''$.*

14. Acrescente ESTADO_DESTINO_TRANS à lista de estados finais do autômato adaptativo correspondente a $X_0$.

*N1_LISTA_ESTADO_FINAL= {$q_2$, $q_5$}.*

15. Encerre o processamento.

*Fim da rotina MapeamentoRegraID-AA.*

Assim, reunindo-se os diversos fragmentos de autômatos gerados pelo mapeamento das regras 1a e 1b, obtém-se o seguinte autômato, responsável pelo reconhecimento da característica sintática N1:

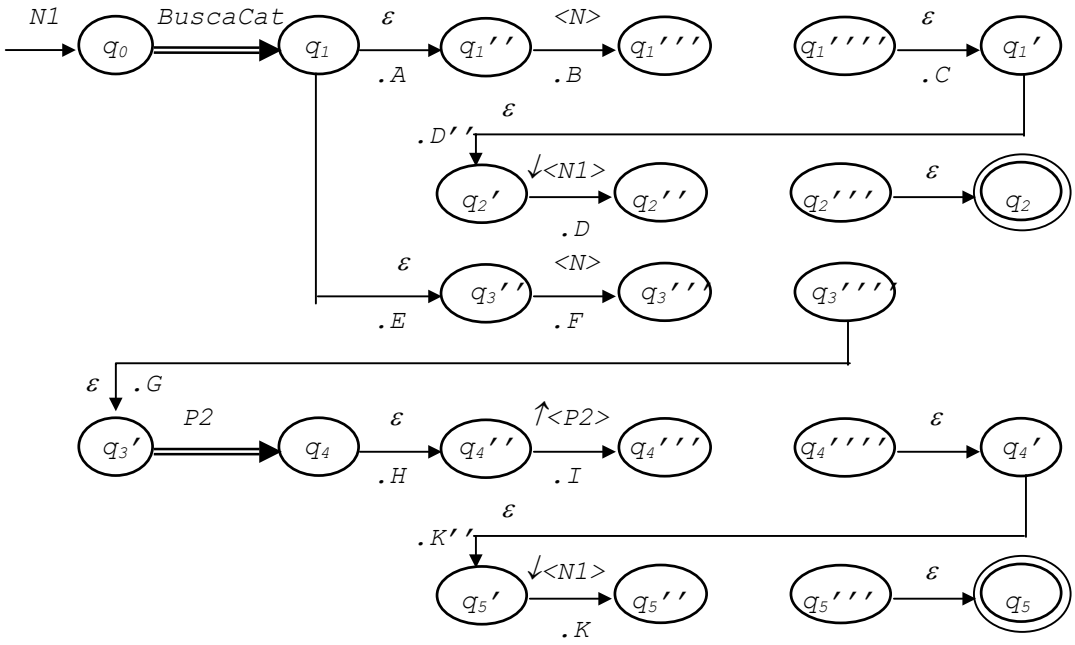



***Figura 35 - Autômato adaptativo correspondente ao mapeamento das regras ID 1a e 1b.***

Submetendo-se a regra 1c ao algoritmo de mapeamento de regra ID para Autômato Adaptativo (MapeamentoRegraID-AA), pode-se proceder de maneira semelhante ao realizado para as regras anteriores, obtendo-se:

*N2 → H1*

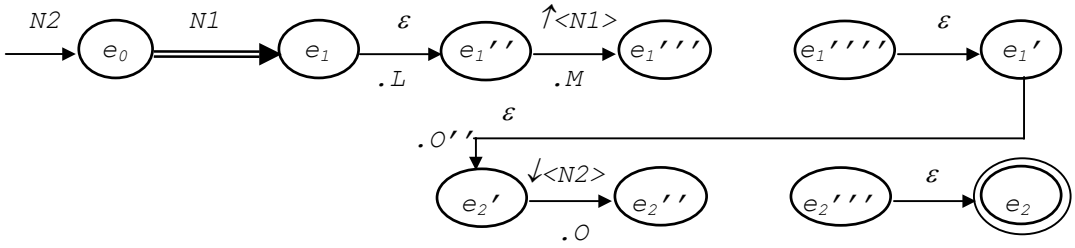



***Figura 36 - Autômato adaptativo correspondente ao mapeamento da regra ID 1c.***

*Ação adaptativa L deve executar:*

- *eliminação da transição de $e_1'''$ para $e_1''''$.*

*Ação adaptativa M deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD não léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '+'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '1'.*
- *se alguma dessas inspeções falharem,*

  *então inserção da transição de RetornoTeste para estado Falso.*

  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*
- *se RetornoTeste aponta para estado Verdadeiro*

  *então inserção da transição em vazio, de $e_1'''$ para $e_1''''$*

| ESTADO ORIGEM TRANS | ESTADO FINAL XI | XI |
|---|---|---|
| $e_0$ | $e_1'$ | H1 |

*Ação adaptativa O'' deve executar:*

- *eliminação da transição de $e_2''$ para $e_2'''$.*

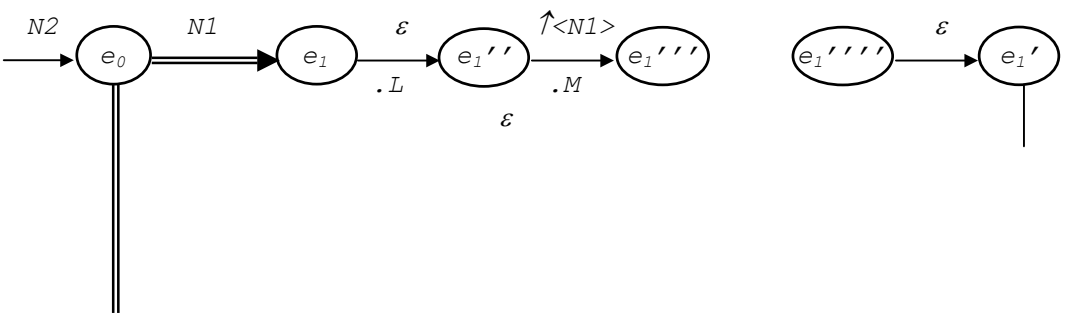
*Ação adaptativa O deve executar:*

- *criação de um estado EST-N2 (ESTADO_X0).*
- *inserção de uma transição de ESTADO_CORR para ESTADO_X0.*
- *inserção da transição de ESTADO_X0 para estado '+', com estímulo N.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo V.*
- *inserção da transição de ESTADO_X0 para estado '2', com estímulo BAR.*
- *inserção da transição de ESTADO_X0 para EST-N1 com estímulo N1.*
- *inspeção para descobrir o estado F para o qual o EST-N1 aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-N2 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-N2, F, SLASH).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *(idem ação anterior, com estímulo WH)*
- *(idem ação anterior, com estímulo RE)*
- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo AGR*
  *então Unifique (EST-N2, ESTADO_INFO_XI, AGR).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *(idem ação anterior, com estímulo PER)*
- *(idem ação anterior, com estímulo PLU)*
- *(idem ação anterior, com estímulo MASC)*
- *se Retorno aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $e_2''$ para $e_2'''$.*

*N2_LISTA_ESTADO_FINAL= {$e_2$}.*

Submetendo-se a regra 1d ao algoritmo de mapeamento de regra ID para Autômato Adaptativo (MapeamentoRegraID-AA), obtém-se:

*N2 → [SUBCAT D], H2*



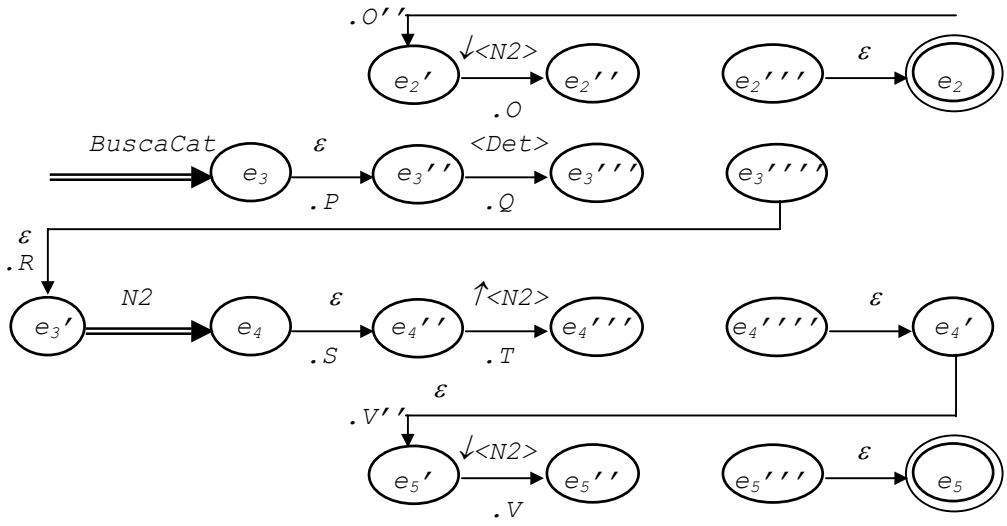



***Figura 37 - Autômato adaptativo correspondente ao mapeamento das Regras ID 1c e 1d.***

*Ação adaptativa P deve executar:*

- *eliminação da transição de $e_3'''$ para $e_3''''$.*

*Ação adaptativa Q deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo SUBCAT para o estado 'D'.*
- *se alguma dessas inspeções falharem,*

  *então inserção da transição de RetornoTeste para estado Falso.*

  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*
- *se RetornoTeste aponta para estado Verdadeiro*

  *então inserção da transição em vazio, de $e_3'''$ para $e_3''''$*

*Ação adaptativa R deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*

- *inserção da transição de EST-D para ESTADO_INFO_XI, com estímulo igual a DET.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR.*
- ***Se** o valor de BAR é zero,*

  ***Então** Se de ESTADO_INFO_XI parte alguma transição com estímulo SLASH*

  *Então inspeção para verificar se de EST-D parte alguma transição com estímulo FOOT_INST.*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição de EST-D para um estado novo com estímulo FOOT_INST.*

  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*

  *(idem ação anterior, com estímulo WH)*

  *(idem ação anterior, com estímulo RE)*

*Ação adaptativa S deve executar:*

- *eliminação da transição de $e_4'''$ para $e_4''''$.*

*Ação adaptativa T deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD não léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '+'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '2'.*
- *se alguma dessas inspeções falharem,*

  *então inserção da transição de RetornoTeste para estado Falso.*

  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*
- *se RetornoTeste aponta para estado Verdadeiro*

*então inserção da transição em vazio, de $e_4'''$ para $e_4''''$*

| *ESTADO ORIGEM TRANS* | *ESTADO FINAL XI* | *XI* |
|---|---|---|
| $e_0$ | $e_1'$ | *H1* |
| $e_0$ | $e_3'$ | *[SUBCAT D]* |
| $e_3'$ | $e_4'$ | *H2* |

*Ação adaptativa V'' deve executar:*

- *eliminação da transição de $e_5''$ para $e_5'''$.*

*Ação adaptativa V deve executar:*

- *criação de um estado EST-N2 (ESTADO_X0).*
- *inserção de uma transição de ESTADO_CORR para ESTADO_X0.*
- *inserção da transição de ESTADO_X0 para estado '+', com estímulo N.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo V.*
- *inserção da transição de ESTADO_X0 para estado '2', com estímulo BAR.*
- *inserção da transição de ESTADO_X0 para EST-D, com estímulo DET.*
- *inserção da transição de ESTADO_X0 para EST-N, com estímulo N2.*
- *inspeção para descobrir o estado F para o qual o EST-D aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-N2 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-N2, F, SLASH).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *(idem ação anterior, com estímulo WH)*
- *(idem ação anterior, com estímulo RE)*
- *inspeção para descobrir o estado F para o qual o EST-N aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-N2 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-N2, F, SLASH).*
  *se Retorno aponta para estado Falso*

*então Retorne.*

- *(idem ação anterior, com estímulo WH)*
- *(idem ação anterior, com estímulo RE)*
- *UnifiqueConcordancia (EST-D, EST-N, AGR).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *se de ESTADO_INFO_XI correspondente a $X_2$ parte alguma transição com estímulo AGR*
  *então Unifique (EST-N2, ESTADO_INFO_XI, AGR).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *(idem ação anterior, com estímulo PER)*
- *(idem ação anterior, com estímulo PLU)*
- *(idem ação anterior, com estímulo MASC)*

*N2_LISTA_ESTADO_FINAL= {$e_2$, $e_5$}.*

Submetendo-se a regra 1e ao algoritmo de mapeamento de regra ID para Autômato Adaptativo (MapeamentoRegraID-AA), obtém-se:

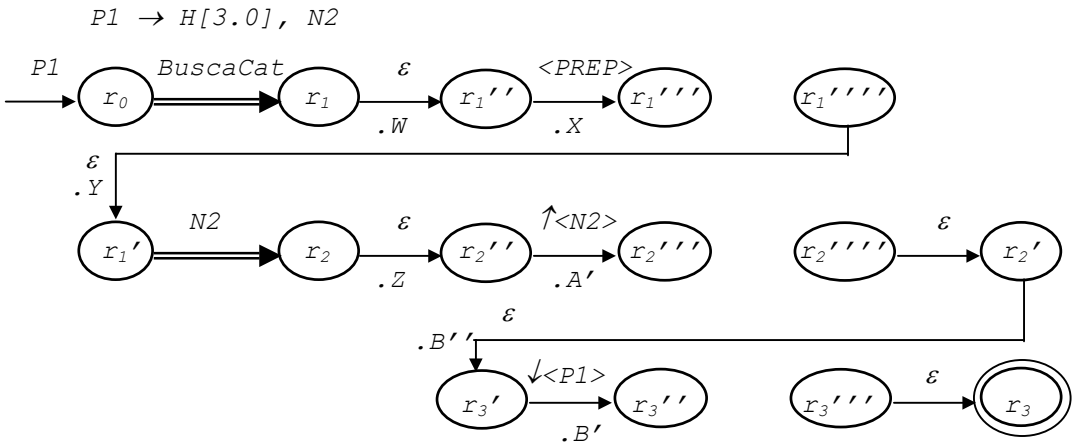



***Figura 38 - Autômato adaptativo correspondente ao mapeamento da Regra ID 1e.***

*Ação adaptativa W deve executar:*

- *eliminação da transição de $r_1'''$ para $r_1''''$.*

*Ação adaptativa X deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '0'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo SUBCAT para o estado '3.0'.*
- *se alguma dessas inspeções falharem,*

  *então inserção da transição de RetornoTeste para estado Falso.*

  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*
- *se RetornoTeste aponta para estado Verdadeiro*

  *então inserção da transição em vazio, de $r_1'''$ para $r_1''''$*

*Ação adaptativa Y deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- *inserção da transição de EST-P para ESTADO_INFO_XI, com estímulo igual a P0.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR.*
- ***Se** o valor de BAR é zero,*

  ***Então** Se de ESTADO_INFO_XI parte alguma transição com estímulo SLASH*

  *Então inspeção para verificar se de EST-P parte alguma transição com estímulo FOOT_INST.*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição de EST-P para um estado novo com estímulo FOOT_INST.*

  *copia a estrutura dessa característica a partir do estado destino da transição FOOT_INST.*

  *(idem ação anterior, com estímulo WH)*

  *(idem ação anterior, com estímulo RE)*

*Ação adaptativa Z deve executar:*

- *eliminação da transição de $r_2'''$ para $r_2''''$.*

*Ação adaptativa A' deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD não léxica especificando um valor default para alguma característica p,*

  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*

  ***Se** não houver tal transição,*

  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '+'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '2'.*
- *se alguma dessas inspeções falharem,*

  *então inserção da transição de RetornoTeste para estado Falso.*

*senão inserção da transição de RetornoTeste para estado Verdadeiro.*

- *se RetornoTeste aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $r_2'''$ para $r_2''''$*

| *ESTADO ORIGEM TRANS* | *ESTADO FINAL XI* | *XI* |
|---|---|---|
| $r_0$ | $r_1'$ | *H[3.0]* |
| $r_1'$ | $r_2'$ | *N2* |

*Ação adaptativa B'' deve executar:*

- *eliminação da transição de $r_3''$ para $r_3'''$.*

*Ação adaptativa B' deve executar:*

- *criação de um estado EST-P1 (ESTADO_X0).*
- *inserção de uma transição de ESTADO_CORR para ESTADO_X0.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo N.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo V.*
- *inserção da transição de ESTADO_X0 para estado '1', com estímulo BAR.*
- *inserção da transição de ESTADO_X0 para EST-P, com estímulo P0.*
- *inserção da transição de ESTADO_X0 para EST-N, com estímulo N2.*
- *inspeção para descobrir o estado F para o qual o EST-P aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-P1 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-P1, F, SLASH).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *(idem ação anterior, com estímulo WH)*
- *(idem ação anterior, com estímulo RE)*
- *inspeção para descobrir o estado F para o qual o EST-N aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-P1 não parte nenhuma transição com estímulo SLASH*

*Unifique (EST-P1, F, SLASH).*
*se Retorno aponta para estado Falso*
*então Retorne.*

- *(idem ação anterior, com estímulo WH)*
- *(idem ação anterior, com estímulo RE)*
- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo PFORM*
  *então Unifique (EST-P1, ESTADO_INFO_XI, PFORM).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *se Retorno aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $r_3''$ para $r_3'''$.*

*P1_LISTA_ESTADO_FINAL= {$r_3$}.*

Submetendo-se a regra 1e ao algoritmo de mapeamento de regra ID para Autômato Adaptativo (MapeamentoRegraID-AA), obtém-se:

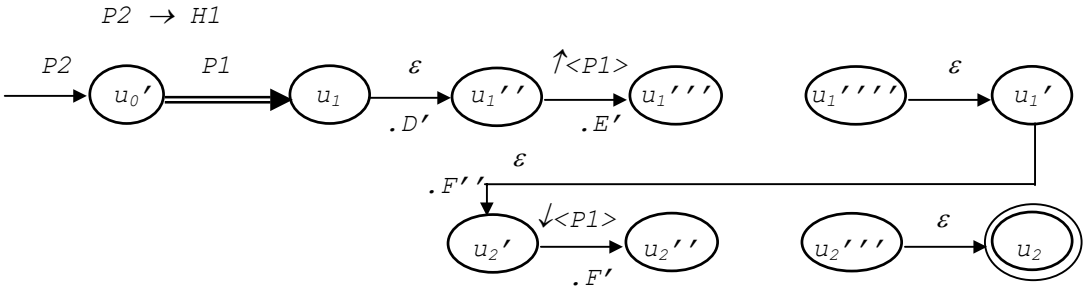



***Figura 39 - Autômato adaptativo correspondente ao mapeamento da Regra ID 1f.***

*Ação adaptativa D' deve executar:*

- *eliminação da transição de $u_1'''$ para $u_1''''$.*

*Ação adaptativa E' deve executar:*

- *inspeção para atribuir a ESTADO_INFO_XI o estado apontado por ESTR_CORR.*
- ***Se** houver alguma FSD não léxica especificando um valor default para alguma característica p,*
  ***Então** inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo p*
  ***Se** não houver tal transição,*
  ***Então** inserção da transição, com estímulo p, de ESTADO_INFO_XI para o estado que representa o seu valor default.*

- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo N para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo V para o estado '-'.*
- *inspeção para descobrir se de ESTADO_INFO_XI parte alguma transição com estímulo BAR para o estado '1'.*
- *se alguma dessas inspeções falharem,*
  *então inserção da transição de RetornoTeste para estado Falso.*
  *senão inserção da transição de RetornoTeste para estado Verdadeiro.*
- *se RetornoTeste aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $u_1'''$ para $u_1''''$*

| ESTADO_ORIGEM_TRANS | ESTADO_FINAL_XI_ | XI |
|---|---|---|
| $u_0$ | $u_1'$ | H1 |

*Ação adaptativa F'' deve executar:*

- *eliminação da transição de $u_2''$ para $u_2'''$.*

*Ação adaptativa F' deve executar:*

- *criação de um estado EST-P2 (ESTADO_X0).*
- *inserção de uma transição de ESTADO_CORR para ESTADO_X0.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo N.*
- *inserção da transição de ESTADO_X0 para estado '-', com estímulo V.*
- *inserção da transição de ESTADO_X0 para estado '2', com estímulo BAR.*
- *inserção da transição de ESTADO_X0 para EST-P1, com estímulo P1.*
- *inspeção para descobrir o estado F para o qual o EST-P aponta com estímulo FOOT-INST.*
- *se F existe e se de F parte alguma transição com estímulo SLASH*
  *então se de EST-P1 não parte nenhuma transição com estímulo SLASH*
  *Unifique (EST-P2, F, SLASH).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *(idem ação anterior, com estímulo WH)*
- *(idem ação anterior, com estímulo RE)*

- *se de ESTADO_INFO_XI correspondente a $X_1$ parte alguma transição com estímulo PFORM*
  *então Unifique (EST-P2, ESTADO_INFO_XI, PFORM).*
  *se Retorno aponta para estado Falso*
  *então Retorne.*
- *se Retorno aponta para estado Verdadeiro*
  *então inserção da transição em vazio, de $u_2''$ para $u_2'''$.*

*P1_LISTA_ESTADO_FINAL= {$u_2$}.*

Como nos experimentos referentes ao mapeamento de redes ATN para Autômato Adaptativo, submetendo-se o sintagma substantivo ‘a destruição da cidade’ ao analisador e etiquetador morfológico de [Menezes-00], obtém-se:

```
a /D-F destruição /N da /P+D-F cidade/N
```

Submetendo-se a saída do analisador e etiquetador morfológico referente ao sintagma substantivo ‘a destruição da cidade’ à rotina MapeamentoLexico, descrita no item 4.1 desta dissertação, obtém-se:

```
Cadeia de entrada:
<a> <destruição> <de> <a> <cidade>
```

sendo que <palavra> refere-se a uma etiqueta associada à palavra.

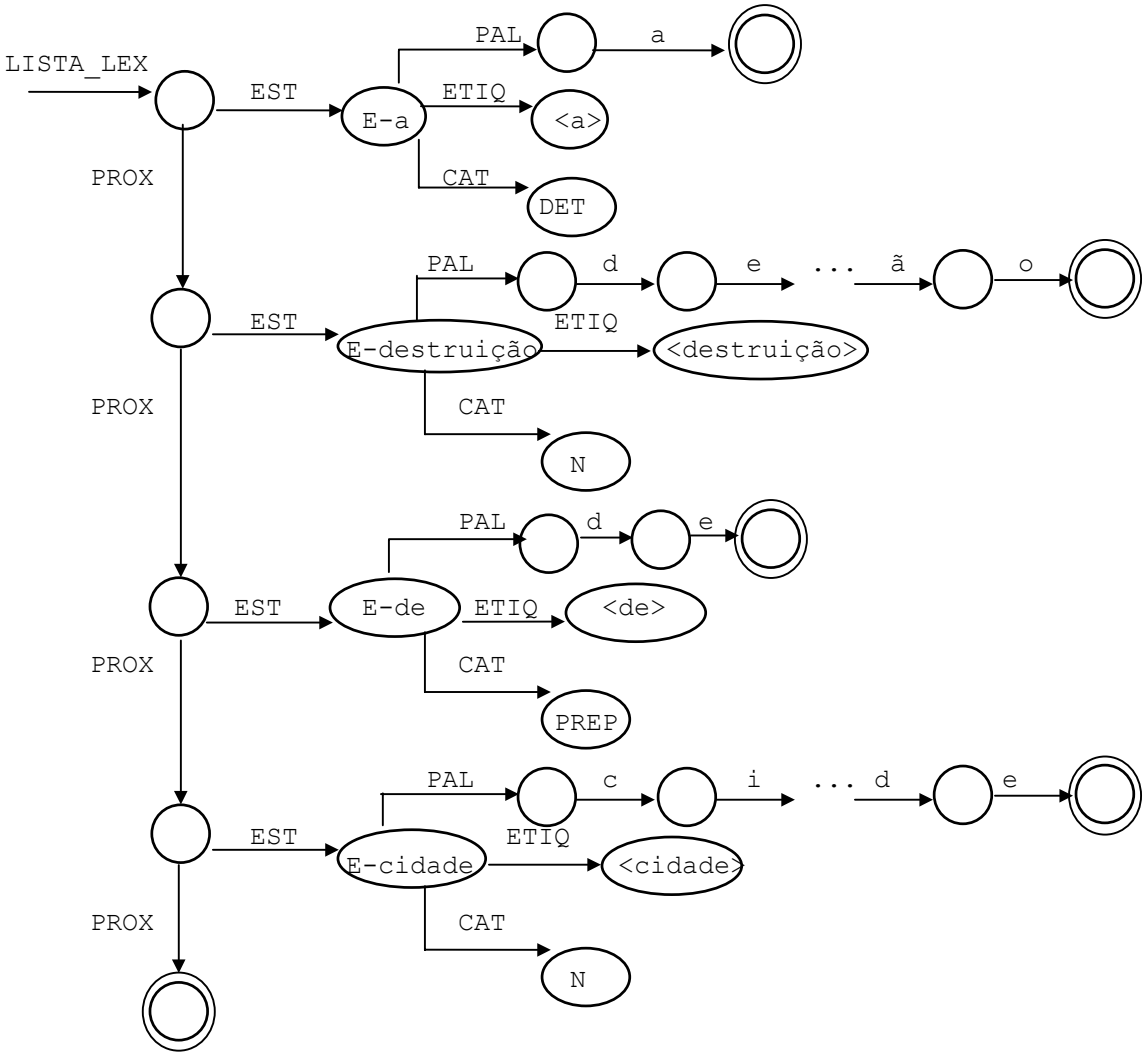



***Figura 40 - Autômato que representa a lista léxica correspondente ao sintagma substantivo 'a destruição da cidade'.***

Na Figura 40, por questões de simplicidade, foram omitidas as ilustrações referentes às demais características de cada palavra. Assim, do estado ‘E-a’ devem partir transições correspondentes às características de ‘a’, para os estados que representam os respectivos valores: [SUBCAT D, AGR NP[PLU-, MASC -]].

Do estado ‘E-destruição’, omitiu-se as transições e estados referentes às características e valores: [N+, V-, BAR 0, SUBCAT 1.1, PER 3, PLU-, MASC-].

Do estado ‘E-de’, omitiu-se as transições e estados referentes às características e valores: [N-, V-, BAR 0, SUBCAT 3.0, PFORM de].

Do estado ‘E-cidade’, omitiu-se as transições e estados referentes às características e valores: [N+, V-, BAR 0, SUBCAT 1.0, PER 3, PLU-, MASC-].

Submetendo-se a saída da rotina MapeamentoLéxico referente ao sintagma substantivo ‘a destruição da cidade’ aos autômatos adaptativos gerados pelo mapeamento das regras ID 1a a 1f, obtém-se a seguinte estrutura, correspondente à análise sintática do sintagma. (A simulação passo a passo foi omitida, para que a dissertação não ficasse muito extensa).

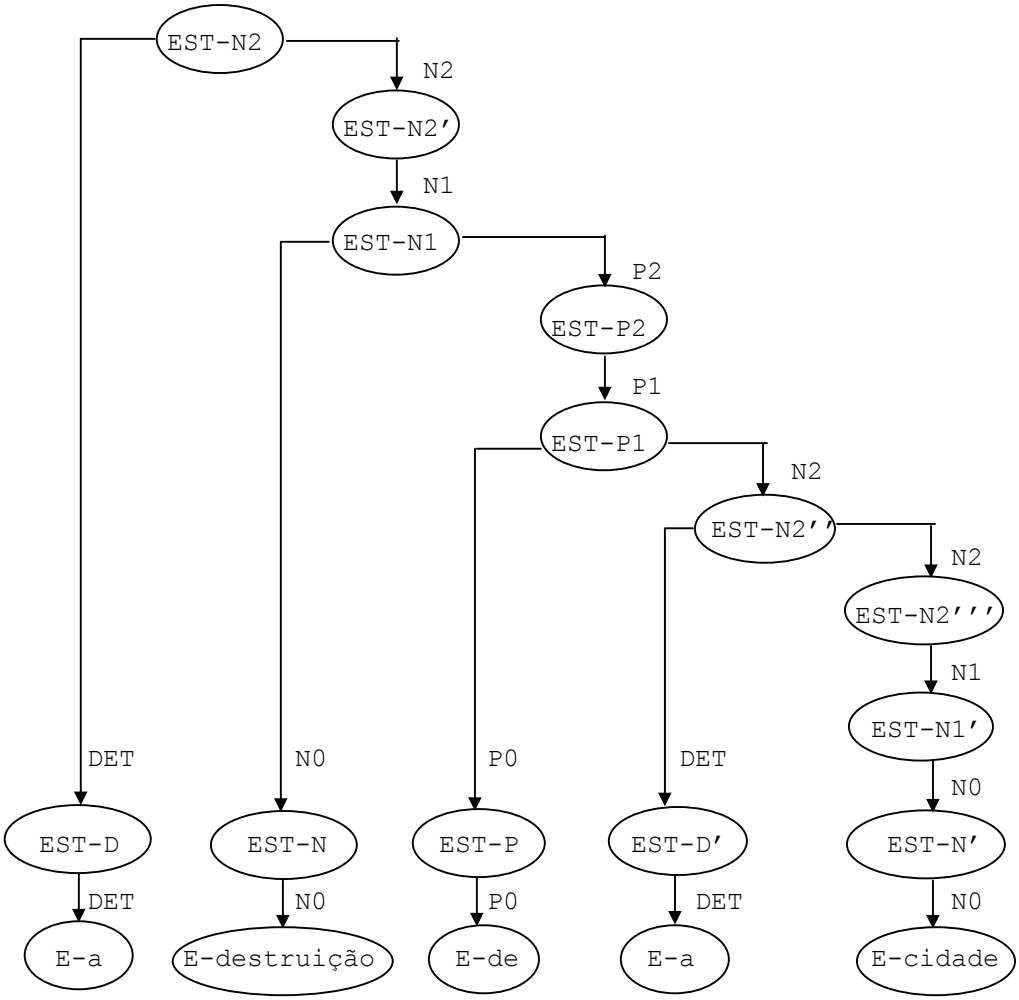



***Figura 41 - Estrutura obtida como resultado do reconhecimento do sintagma substantivo 'a destruição da cidade', pelos autômatos adaptativos correspondentes às Regras ID 1a a 1f.***

Neste exemplo, através da aplicação do princípio HFC, os estados EST-N1’ e EST-N2’’’ recebem os valores [PER3, PLU- , MASC-] , referentes à palavra ‘cidade’. Aplicando-se o princípio CAP, verifica-se a concordância entre os estados EST-D’ e EST-N2’’’ (ambos apresentam-se no singular e no gênero feminino).

Pela aplicação do princípio HFC, os estados EST-P1 e EST-P2 recebem o valor [PFORM de], o que é essencial para que se possa aplicar a regra 1b (`N1 → H[1.1], P2[de]`).

Também pela aplicação do princípio HFC, os estados EST-N1 e EST-N2' recebem os valores [PER3, PLU-, MASC-], referentes à palavra 'destruição'. Aplicando-se o princípio CAP, verifica-se a concordância entre os estados EST-D e EST-N2' (ambos apresentam-se no singular e no gênero feminino).

Constata-se, assim, que os autômatos adaptativos resultantes do mapeamento das Regras GPSG 1a a 1f são capazes de reconhecer e analisar sintaticamente um sintagma substantivo da língua portuguesa, criando uma estrutura sintática em decorrência dessa análise.

Dessa forma, pode-se comprovar a funcionalidade dos algoritmos de mapeamento de GPSG para Autômato Adaptativo.

O próximo item apresenta uma avaliação dos experimentos realizados.

## 5.2. Avaliação

Pelos experimentos realizados, verificou-se a funcionalidade dos algoritmos de mapeamento propostos nesta dissertação.

Não foram descritos mais experimentos de simulação dos algoritmos, devido à extensão da descrição dos mesmos. No entanto, apesar de poucos experimentos serem registrados, foi possível constatar que os algoritmos são funcionais e que os autômatos adaptativos resultantes dos mapeamentos são capazes de reconhecer e analisar sintaticamente o trecho proposto da oração da língua portuguesa, como era desejado.

Com relação ao método utilizado para o desenvolvimento da proposta, verificou-se que possibilitou um resultado prático, uma vez que demonstrou a viabilidade de utilização do Formalismo Adaptativo no processamento de linguagem natural, mais especificamente na etapa de análise sintática.

Comparando-se o mapeamento dos dois formalismos escolhidos, percebe-se que o mapeamento do ATN foi mais direto do que o do GPSG, o que já era esperado, dada a similaridade estrutural entre o ATN e o Autômato Adaptativo. O mapeamento do GPSG já não foi tão direto, resultando num algoritmo extenso, e em ações adaptativas extensas. Talvez, se fosse escolhido algum outro formalismo baseado em restrição, o mapeamento poderia ter sido mais simples.

Porém, justifica-se a escolha do GPSG pela existência da especificação da gramática superficial da língua portuguesa nessa notação. Um bom exercício futuro seria o mapeamento completo dessa especificação GPSG, embora seja bem extensa, para que se obtenha um analisador sintático em Autômato Adaptativo, que fosse capaz de analisar as várias nuances da língua portuguesa.

O próximo capítulo apresenta as conclusões obtidas ao final deste trabalho.

# 6. CONCLUSÃO

Este capítulo apresenta uma avaliação geral desta pesquisa, assim como considerações finais a respeito de suas contribuições e futuros trabalhos que possam dar continuidade a essa dissertação.

## 6.1. Avaliação Geral

Pelos experimentos realizados, pode-se concluir que os Formalismos Adaptativos podem ser empregados no processamento de linguagem natural, especificamente no que se refere à representação de informações lingüísticas e à análise sintática.

Os algoritmos propostos mostraram-se funcionais, e os métodos utilizados possibilitaram demonstrar a viabilidade de utilização prática dos Formalismos Adaptativos na análise sintática da linguagem natural.

O mapeamento do GPSG para Autômato Adaptativo foi bem menos direto do que o do ATN, resultando em algoritmos extensos, bem como em ações adaptativas extensas. Talvez algum outro formalismo baseado em restrições apresentasse um mapeamento mais direto do que o GPSG.

Embora não se tenha registrado, nesta dissertação, experimentos mais significativos, que possibilitariam uma melhor avaliação dos resultados, pode-se dizer, de maneira geral, que o objetivo proposto foi atingido.

## 6.2. Contribuições

A contribuição deste trabalho é apresentar uma utilização prática do Formalismo Adaptativo, à medida que se verifica a sua capacidade de uso como ferramenta para a representação de informações e análise sintática no processamento de linguagem natural.

Através do desenvolvimento da proposta desta dissertação, são apresentadas formas de mapeamento, para Formalismo Adaptativo, de duas notações clássicas de representação de linguagem natural: as redes *Augmented Transition Network* e as especificações GPSG (*Generalized Phrase Structure Grammar*).

Uma contribuição importante decorrente desses experimentos é a utilização da especificação da gramática da língua portuguesa em GPSG, desenvolvida em [Chin-96], o que representa a evolução de um trabalho lingüístico e o aproveitamento de esforços já despendidos, possibilitando a geração de futuros trabalhos sobre o assunto, a respeito do qual pouco existe disponível na literatura.

Para realizar os experimentos, esse trabalho também contribuiu para possibilitar uma integração com o analisador e etiquetador morfológico desenvolvido em [Menezes-00].

Através do desenvolvimento do mapeamento, também foi possível constatar que o Formalismo Adaptativo é capaz de resolver diversos problemas de linguagens complexas, tais como especificações incompletas e dependências de contexto, utilizando apenas recursos sintáticos.

## 6.3. Trabalhos Futuros

Há a necessidade de ensaios exaustivos, envolvendo uma implementação completa dos algoritmos propostos nesta dissertação, bem como da especificação da gramática da língua portuguesa, para que se possa avaliar melhor o alcance das técnicas apresentadas. Disso resultaria um analisador sintático para a língua portuguesa, implementado através de Autômato Adaptativo.

Para dar continuidade a esse trabalho, o que pode ser realizado futuramente é a especificação da representação da linguagem e de um analisador sintático através do Formalismo Adaptativo, sem que haja mapeamentos intermediários de outros formalismos, sendo possível, assim, melhor aproveitar os recursos de adaptabilidade dos Formalismos Adaptativos.

Uma vez constatada a viabilidade de emprego dos Formalismos Adaptativos no processamento de linguagem natural, poder-se-ia pensar também em futuras pesquisas nessa área, utilizando os Formalismos Adaptativos como ferramenta.

# REFERÊNCIAS BIBLIOGRÁFICAS

ALLEN, J. **Natural language understanding.** 2.ed. Redwood City, California: The Benjamin/Cummings Publishing Company, Inc., 1994.

ARNOLD, D. **Lexical functional grammar.** Disponível em: <http://clwww.essex.ac.uk/LFG/>. Acesso em: 16 jun. 2001.

BARBERO, C.; LOMBARDO, V. Wide-Coverage Lexicalized Grammars. In: CONGRESS OF THE ITALIAN ASSOCIATION FOR AI, 5, Roma, 1997. **AI*IA 97: Advances in AI: Proceedings.** Lecture Notes in Artificial Intelligence, 1321, p. 60-71.

BARBOSA, C.R.S.C.de; CASTILHO, J.M.V.de Gramática Livre de Contexto Lexicalizada para a Análise Sintática da Língua Portuguesa – uma experiência na geração de consultas de uma Interface em Linguagem Natural para Banco de Dados. In: PROPOR00, Atibaia, 2000. **Anais do V Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR00)**, p. 155-164.

BATES, M. The Theory and Practice of Augmented Transition Network Grammars. **Natural language communication with computer.** Berlim, 1978. Lecture Notes in Computer Science, 63, p. 191-259.

BATES, M.; BOBROW, R.J.; WEISCHEDEL, R.M. Critical challenges for natural language processing, p. 3-34. In BATES, M.; WEISCHEDEL, R.M. (Eds.) **Challenges in Natural Language Processing (Studies in Natural Language Processing).** Cambridge University Press, 1993.

BICK, E. Automatic parsing of Portuguese. In: PROPOR96, Curitiba, 1996. **Anais do II Encontro para o Processamento Computacional de Português Escrito e Falado (PROPOR96)**, p. 101-107.

BICK, E. Structural Lexical Heuristics in the Automatic Analysis of Portuguese. In: Nodalida'98, Copenhaguen, 1988. **The 11th Nordic Conference on Computational Linguistics (Nodalida '98), Proceedings,** p. 44-56.

BICK, E. **The Parsing System "Palavras": Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework.** Denmark: Aarhus University Press, 2000.

BOLC, L. **The design of interpreters, compilers and editors for augmented transition networks.** Berlin: Springer-Verlag, 1983.

BRÖKER, N; KRUIJFF, G. **Dependency-Based Approaches to Natural Language Syntax.** 1999. Disponível em: <http://ufal.mff.cuni.cz/dg.html>. Acesso em: 30 abr. 2001.

CHANOD, J.; TAPANAINEN, P. Tagging French -- comparing a statistical and a constraint-based method. In: EACL95, Dublin, 1995. **Proceedings of Seventh Conference of the European Chapter of the ACL**, p.149-156. Disponível em: <http://xxx.lanl.gov/abs/cmp-lg/9503003>. Acesso em: 15 jun. 2001.

CHARNIAK, E. **Statistical language learning.** Cambridge: MIT Press, 1993.

CHIN, E. **Tradução por computador: dicionário e componentes de análise e transferência.** 1996. 330p. Tese (Doutorado) - Departamento de Lingüística da Faculdade de Filosofia, Letras e Ciências Humanas da Universidade de São Paulo. São Paulo.

DERANSART, P.; JOURDAN, M.; LORHO, B. **Attribute Grammars: Definitions, systems and bibliography.** New York: Springer-Verlag, 1988. Lecture Notes in Computer Science, 323.

DE ROECK, A. An Underview of Parsing, p. 3-17. In: KING, M. (Ed.) **Parsing Natural Language.** London: Academic Press, 1983.

GAZDAR G.; KLEIN E.; PULLUM G.; SAG I. **Generalized Phrase Structure Grammar.** Cambridge: Harvard University Press, 1985.

GAZDAR G.; MELLISH, C. **Natural Language Processing in PROLOG: an Introduction to Computational Linguistics.** Wokingham: Addison-Wesley Publishing Company, 1994.

HARRIS, M.D. **Introduction to natural language processing.** Virginia: Reston, 1985.

HUDSON, R. **An Encyclopedia of English Grammar and Word Grammar.** 2001. Disponível em: <http://www.phon.ucl.ac.uk/home/dick/encyclopedia/encframe.htm>. Acesso em: 29 jun. 2001.

IWAI, M.K. **Um formalismo gramatical adaptativo para linguagens dependentes de contexto.** 2000. 191p. Tese (Doutorado) - Departamento de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo. São Paulo.

JOHNSON, R. Parsing with Transition Networks, p. 59-72. In KING, M. (Ed.) **Parsing Natural Language.** New York: Academic Press, 1983.

JOSÉ NETO, J. **Contribuição à metodologia de construção de compiladores.** 1993. 272p. Tese (Livre-Docência) - Escola Politécnica, Universidade de São Paulo. São Paulo.

JOSÉ NETO, J. Adaptive automata for context-dependent languages. **ACM SIGPLAN Notices**, v.29, n.9, p.115-24, 1994.

JOSÉ NETO, J. Adaptive rule-driven devices – general formulation and case study. In: CONFERENCE OF IMPLEMENTATIONA AND APPLICATIONS OF AUTOMATA, Pretoria, 2001. **Proceedings of the 6th conference on implementationa and applications of automata.** p.158-176.

KINOSHITA, J. **Aspectos de implementação de uma ferramenta de auxílio à tradução inglês-português.** 1997. 163p. Tese (Doutoradop) – Departamento de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo. São Paulo.

KNIGHT, K. Unification: A Multidisciplinary Survey. **ACM Computing Surveys**, 21, p. 93-124, 1989.

LEECH, G. **Corpus Annotation: linguistic information from computer text corpora.** London: Addison-Wesley, 1997.

Lombardo, V. Parsing Dependency Grammars. In: CONGRESS OF THE ITALIAN ASSOCIATION FOR AI, 2, Palermo, 1991. **Proceedings of the Congress of the Italian Association for AI.** Lecture Notes in Artificial Intelligence, 549, p. 291-300, 1991.

MENEZES, C.E.D. **Um método para a construção de analisadores morfológicos, aplicado à língua portuguesa, baseado em autômatos adaptativos.** 2000. 117p. Dissertação (Mestrado) – Departamento de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo. São Paulo.

NUNES, M.G.V.; HASEGAWA, R.; KAWAMOTO, S.; OLIVEIRA, M.C.F. DE; TURINE, M.A.S.; GHIRALDELO, C.M.; OLIVEIRA JR. O.N.; RIOLFI, C.R.; SIKANSKI, N.S.; MARTINS, T.B. Style and grammar checkers for brazilian portuguese. **Notas do ICMSC – Série Computação**, 25, São Carlos, 1996.

PEREIRA, F.C.N.; WARREN, D.H.D. Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. **Artificial Intelligence** 13, p. 231-278, 1980.

PROCESSAMENTO COMPUTACIONAL DO PORTUGUÊS. Oslo, Noruega. 1998. **Processamento Computacional do Português.** Santos, Diana. SINTEF Telecom and Informatics. Disponível em: <http://www.portugues.mct.pt>. Acesso em: 30 abr. 2001.

RAVERA, S.B. Information-Based Linguistics and Head-Driven Phrase Structure. In: EAIA'90, Guarda, Portugal, 1990. **Natural Language Processing.** Lecture Notes in Artificial Intelligence, 476, p. 55-101.

RICH, E.; KNIGHT, K. **Inteligência Artificial,** 2. Ed. São Paulo: Makron Books, 1993.

SANTOS, D. O Projecto Processamento Computacional do Português: Balanço e Perspectivas. In: PROPOR00, Atibaia, 2000. **Anais do V Encontro para o Processamento Computacional da Língua Portuguesa Escrita e Falada (PROPOR00)**, p. 105-113.

SAVADOVSKY, P. **Introdução ao Projeto de Interfaces em Linguagem Natural.** São Paulo: SID Informática, 1988.

SCHAÜFELE, S. **SYNTHINAR: The Syntactic Theory Seminar.** 1999. Disponível em: <http://www.jtauber.com/linguistics/synthinar/>. Acesso em: 3 ago. 2001.

SHIEBER, S.M. **An introduction to unification-based approaches to grammar. CSLI Lectures Notes**, 4. Stanford: Center for the Study of Language and Information, 1986.

SHIEBER, S.M. Separating Linguistic Analyses from Linguistic Theories. **Natural Language Parsing and Linguistic Theories**, p. 33-68, D. Reidel Publishing Company, 1988.

SHIEBER, S.M. **Constraint-based grammar formalisms: parsing and type inference for natural and computer languages.** Cambridge: The MIT Press, 1992.

TYCHO BRAHE CORPUS OF HISTORICAL PORTUGUESE. **Corpus Tycho Brahe.** Instituto de Estudos da Linguagem. Universidade de Campinas, SP, 1998. Disponível em: <http://www.ime.usp.br/~tycho/corpus>. Acesso em: 10 Maio 2001.

USZKOREIT, H.; ZAENEN, A. Grammar Formalisms. In: VARILE, G.; ZAMPOLLI, A. **Survey of the state of the art in human language technology.** Oregon, Center for Spoken Language Understanding, Oregon Graduate Institute, 1995. Disponível em: <http://cslu.cse.ogi.edu/HTLsurvey/>. Acesso em: 02 Maio 2001.

VISL – VISUAL INTERACTIVE SYNTAX LEARNING. Odense, Dinamarca. 2001. **Visual Interactive Syntax Learning.** Bick, Eckhard. University of Southern Denmark. Institute of Language and Communication. Disponível em: <http://visl.hum.sdu.dk>. Acesso em: 30 abr. 2001.

WEDEKIND, J. Approaches to Unification in Grammar: A Brief Survey. **Specifying Syntactic Structures.** Stanford: CSLI Publications, 1997.

WINOGRAD, T. **Language as a cognitive process. Volume I - Syntax.** Reading: Addison-Wesley, 1983.

WOODS, W. A. Transition Network Grammars for Natural Language Analysis. **Communications of the ACM**, 13, n. 10, Out. 1970, p. 591-606.

# APÊNDICE A

## Especificação da Gramática da Estrutura Superficial da Língua Portuguesa em GPSG

Há um trabalho [Chin-96] que apresenta uma especificação da gramática da estrutura superficial da língua portuguesa em GPSG, com a finalidade de se indicar como seria uma tradução automática de textos da área médica, do português para o inglês.

Nesta dissertação, adotou-se essa especificação no desenvolvimento do mapeamento do formalismo GPSG para Autômato Adaptativo, tornando-se possível trabalhar com os elementos da gramática da língua portuguesa.

Neste Apêndice, encontram-se relacionadas as regras para os diversos sintagmas e orações da língua portuguesa, segundo essa especificação. Maiores detalhes sobre cada regra podem ser encontrados em [Chin-96].

## Sintagma substantivo

```
N1 → H[1.0]
N1 → H[1.1], (P2[de])
N1 → H[1.2], (P2[sobre])
N1 → H[1.3], (P2[de]), (P2[com])
N1 → H[1.4], (P2[de]), (P2[a])
N1 → H[1.5], (P2[entre])
N1 → H[1.6], (P2[de]), (P2[em])
N1 → H[1.7], (P2[de]), (P2[para])
P1 → H[3.0], N2
P1 → H[3.1], V2[INF]
P1 → H[3.2], S[FIN]
P2 → H1
N2 [DEF-] → {[SUBCAT A]}, H1
N2 [DEF+] → {[SUBCAT B]}, H1
N2 [DEF+] → {[SUBCAT C]}, H1
N2 [QUANT+] → {[SUBCAT D]}, H1
N2 [QUANT+] → {[SUBCAT D]}, H2[DEF+]
N2 → {[SUBCAT E]}, H2[QUANT+]
```

N2[DEF+]→{[SUBCAT G]},{[SUBCAT F]},H1

N2 → H1

N1 [SUBST +] → H[1.0, SUBST +], (X2)

N1 [SUBST +] → H1[1.0, SUBST +], (A2)

N1 [SUBST +] → H1[1.0, SUBST +], (P2)

N2 → H1[SUBST +]

N2[Q+] → {[SUBCAT D]}, H2[DET +, Q -]

N2 → {[SUBCAT E]}, H2[Q+]

FCR: [N+, V-, BAR 2] ⊃ ~[SUBST]

N1[PRON+] → H[1.0, PRON+]

N2 → H1[PRON+]

N2 → {[SUBCAT E]}, H1[PRON+, Q+]

FSD 10: [N+, V-, BAR 2] ≡ [ACC]

FCR: [V+, N-, BAR 2, FIN] ⊃ [CASO NOM]

FCR: [N-,V+,BAR 0,SUBCAT α]⊃[CASO ACC]

FCR: [N-, V-, BAR 0] ⊃ [CASO OBL]

TYP(N2[AGR V2])=<TYP(V2[FIN]), TYP(S)>

TYP(N2[AGR V0])=<TYP(V0[SUBCAT α]), TYP(V1)>

α ∈ {verbos que têm Caso acusativo a atribuir}

TYP(N2[AGR P0])=<TYP(P0), TYP(P1)>

Q1 → H[n]

Q2 → {[SUBCAT G]}, H1

Q2 → {[SUBCAT H]}, H2

N2 → Q2, H1

N2 → H2, N2

# Sintagma adjetivo

A1 → H[2.0]

A1 → H[2.1], P2[a]

A1 → H[2.2], P2[por]

A1 → H[2.3], P2[de]

A1 → H[2.4], P2[com]

A1 → H[2.5], P2[sobre]

A1 → H[2.6], P2[em]

A1 → H[2.7], P2[para]

A1[AGR S[FIN, COMP que]] → H[2.8], (P2[para])

Meta-regra de extraposição: X2[AGR S] → W

X2[AGR N2[NFORM pro2, PLU-]] → W,S

A2 → (A2[ADV+]), H1

A2 → (A2[ADV+, GRAU+]), H1

A2 → ([SUBCAT J]), H1

A2 → H2, (P2[PFORM α]) sendo que α ∈ {do que, tanto}

A2 → H2, S[CONJS α] sendo que α ∈ {quanto}

A2[ADV+, GRAU+] → N2[SEP+], H1[ADV+, GRAU+]

# Sintagma preposicional

P1 → H[3.0], N2

P1 → H[3.1], V2[INF]

P1 → H[3.2], S[FIN, COMP que]

P1 → H[3.3], N2[PLU+]

P1 → H1

P1 → H1[PFORM α], P1[PFORM β] sendo que α ∈ {de, desde} e β ∈ {a, até}

P1 → H[PFORM a], Q2

P2 → (A2[ADV+]), H1

P2 → ([SUBCAT J]), H1

P2 → N2[SEP+], H1

# Sintagma verbal

V1 → H[4.0]

V1 → H[4.1], N2

V1 → H[4.2], N2

V1 → H[4.3], N2, P2[em]

V1 → H[4.4], N2, P2[a]

V1 → H[4.5], N2, P2[para]

V1 → H[4.6], N2, (P2[para])

V1 → H[4.7], N2, (P2[de])

V1 → H[4.8], N2, P2[como]

V1 → H[4.9], P2[de]

V1 → H[4.10], P2[em]

V1 → H[4.11], P2[a]

V1 → H[4.12], P2[para]

V1 → H[4.13], P2[por]

V1 → H[4.14], P2[como]

V1 → H[4.15], (P2[de]), P2[para]

V1 → H[4.16], N2[PLU $\alpha$, MASC $\beta$], [N+, BAR 2, PLU $\alpha$, MASC $\beta$]

V1 → H[4.17], S[FIN]

V1[AUX+] → H[4.18], X2[PRD+]

V1 → H[4.19], (N2), V2[INF]

V1[AGR N2[NFORM pro2, PLU -]] → H[4.20], (N2), S[FIN]

V1[AGR N2[NFORM pro1, PLU $\alpha$]] → H[4.21], N2[PLU $\alpha$]

V1 → H[4.22], V2[INF]

V1 → H[4.23], V2[PSP]

V1[AGR N2[NFORM pro1, PLU-]] → H[4.24], N2

Meta-regra da passiva analítica:

V1 → H[- SUBCAT n], N2, W  sendo que n ∈ {4.2, 4.18, 4.19}

V1[PAS] → H[- SUBCAT n], N2[NULL+], W, (P2[por])

Meta-regra da passiva sintética:

V1 → H[-SUBCAT n], N2, W  sendo que n ∈ {4.18, 4.19}

V1 → H[-SUBCAT n], N2[NFORM se], N2[NULL+], W

Meta-regra da passiva sintética 2:

V1 → H[4.17], S[FIN]

V1[AGR N2[NFORM pro2, PLU-]] → H[4.17], N2[NFORM se], S[FIN]

Meta-regra da indeterminação do sujeito:

V1 → H[-SUBCAT n], (P2)  sendo que n ∈ {4.13, 4.14}

V1[AGR N2[NFORM pron2, PLU-]] → H[-SUBCAT n],N2[NFORM se],(P2)

V2 → H2, (A2[ADV+])

V2 → H2, (P2)

# Oração modificadora

N1 → H1, S[R+]

S → X2, H[SUBJ-]

N1[SUBST+] → H1[SUBST+], S[R+]

V2 → H2, X2

V2 → H2, X2[NULL+]

N1 → H1, S[R+]

V2 → H2, (P2)

V2 → H2, (A2[ADV+])

V2 → H2, P2[NULL+]/P2

V2 → H2, A2[ADV+, NULL+]/A2[ADV+]

S → X2, H/X2  sendo X2= P2 ou A2[ADV+] onde se encontra o pronome relativo

N1 → H1, S[R+]

N2 → {[SUBCAT A]}, H1

S → X2, H[SUBJ-]

V1 → H[4.1], N2

V1 → H[4.3], N2, P2[em]

V1[PAS] → H[-SUBCAT n], N2[NULL+]/N2, (P2[por])

V1[PAS] → H[-SUBCAT n], N2[NULL+]/N2, P2[em], (P2[por])

N1 → H1, V1[PAS, AGR N1[PLU, MASC]]/N2

TYP(V1[PAS, AGR N1]) = <TYP(N1), TYP(N1)>

V2 → H2, (V1[PSP])

V2 → H2, V2[PRP]

# Oração

S → X2, H[SUBJ-]

S → X2, H/X2

S[COMP a] → {[COMP α]}, H[COMP NIL] sendo que α ∈ {que, se}

FCR 11: [SUBJ] ⊃ {+V, -N, BAR 2]

FCR 15: [COMP] ≡ [+SUBJ]

FCR 16: [WH, +SUBJ] ⊃ [COMP NIL]

FSD: [NFORM] ⊃ ([NORM] ∨ [pron1])

FSD: [V+, N-, BAR 0] ⊃ ([AGR N2[NORM]] ∨ [AGR N2 [pron1]]

pron1: [N+, V-, BAR 2, NFORM pron1, PER 3, PLU-]

pron1: [N+, V-, BAR2, NFORM pron1, PER 3, PLU+]

# Sintaxe de coordenação e subordinação

Esquema de coordenação iterativa:

X → H[CONJ α0], H[CONJ α1]+

sendo que α ∈ {<e,NIL>, <NIL, e>, <ou, NIL>, <NIL, ou>, <nem, nem>,

<NIL e/ou>, <e/ou, NIL>, <NIL, NIL>}

Esquema de coordenação binária:

X → H[CONJ α0], H[CONJ α1]

sendo que α ∈ {<NIL, mas>, <NIL, porém>, <NIL, pois>, <NIL, bem como>, <tanto, como>, <ou, ou>, <nem, nem>}

Regra LP para o esquema de coordenação binária:

[CONJ α0] < [CONJ α1]

sendo que α0 ∈ {NIL, tanto, ou, nem} e α1 ∈ {como, ou, nem, bem como}

FCRlex: [N+, V+] ⊃ [ADV]

FSDlex: [ADV-]

A2[ADV+] → H[N+, V+, BAR 2, ADV-, CONJα0], H[N+, V+, BAR 2, ADV+, CONJα1]

sendo que α0 ∈ {NIL} e α1 ∈ {e, ou}

V2 → H2, (X2,[ADV+])

S → H[SUBJ+], S[CONJS α]

sendo que α ∈ {se, como, embora, caso, até que, enquanto, quando, já que, visto que, a menos que, desde que, porque, uma vez que, na medida em que, antes que, quanto, do que}

S[CONJ α] → {[SUBCAT α]}, H

sendo que α ∈ {se, como, embora, caso, até que, enquanto, quando, já que, visto que, a menos que, desde que, porque, uma vez que, na medida em que, antes que, quanto, do que}

FCR: [CONJS] ⊃ [FIN, SUBJ+]

# Marcador do discurso e palavra denotativa

V2 → {[SUBCAT K]}, H2

X2 → {[SUBCAT M]}, X2