PROCESSAMENTO DE LINGUAGEM NATURAL

AULA 1

Prof. Luciano Frontino de Medeiros

CONVERSA INICIAL

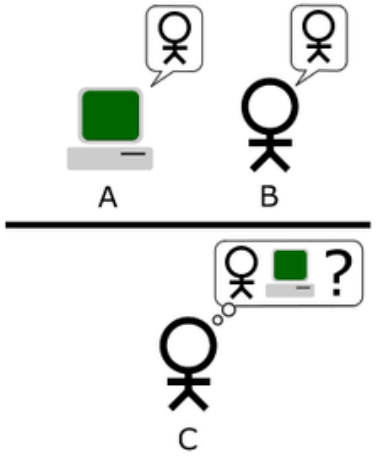
O processamento de linguagem natural (PLN) tem se popularizado cada vez mais na comunidade de desenvolvimento de sistemas, no sentido de permitir o desenvolvimento de interfaces ou *chatbots* para interação em linguagem natural com seres humanos. Outras ferramentas como análise de sentimento necessitam do processamento em linguagem natural para analisar grandes quantidades de textos e identificar se há conotação positiva ou negativa. Assim, conhecer PLN é fundamental para desenvolver tais tipos de ferramentas. Esse material trata dos fundamentos deste tópico, desde sua inserção no contexto da inteligência artificial (IA), um breve histórico, passando por conceitos e definições de termos relacionados à área, além do estudo das gramáticas e expressões regulares.

TEMA 1 – INTELIGÊNCIA ARTIFICIAL E PROCESSAMENTO DE LINGUAGEM NATURAL

De modo geral, o conceito de processamento de linguagem natural está presente na computação desde a conceptualização do teste proposto por Alan Turing, em 1950, em que o computador seria testado com um humano, sendo o teste conduzido por um interrogador humano que lança perguntas por escrito tanto para o computador quanto para o humano e recebe respostas deles, devendo decidir qual deles é o computador e qual é o humano (Turing, 1950).

> Se agora alguma máquina em particular pode ser descrita como um cérebro, temos apenas que programar nosso computador digital para imitá-la e ela também será um cérebro. Se for aceito que cérebros reais, como os encontrados em animais, e em particular nos homens, são uma espécie de máquina seguirá que nosso computador digital devidamente programado se comportará como um cérebro.

Figura 1 – Ilustração do teste de Turing



Fonte: Syncred View, [S.d.].

Conforme Russell e Norvig (2004, p. 4), para um computador ser considerado inteligente, ele precisa estar dotado das seguintes capacidades:

- processamento de linguagem natural;
- representação de conhecimento;
- raciocínio automatizado; e
- aprendizado de máquina.

Junto à visão computacional e à robótica, essas disciplinas representariam a maior parte da IA. Ainda assim, pouco se dedica à aprovação de um computador no teste de Turing (Russell; Norvig, 2004, p. 5).

Por ***linguagem natural***, entende-se uma linguagem usada para a comunicação diária por seres humanos; idiomas como inglês, espanhol, hindi, russo ou português. Ao contrário das ***linguagens artificiais***, como as de programação e notações matemáticas, as linguagens naturais evoluíram ao longo de gerações e são difíceis de definir com regras explícitas. O PLN, em sentido amplo, significa qualquer tipo de manipulação de linguagem natural por computador. Em um extremo, pode ser tão simples quanto contar a frequência das palavras para comparar diferentes estilos de escrita. No extremo oposto, envolve a compreensão de expressões humanas completas, pelo menos no sentido de ser capaz de proporcionar respostas úteis a elas (Bird; Klein; Loper, 2019).

De acordo com Palazzo (1997, p. 5), o PLN se reveste de alta importância para que se possa desenvolver ferramentas de comunicação entre homem e computador, requerendo não apenas a formalização sintática como também o grande problema: a formalização semântica, ou seja, o significado correto das

palavras, sentenças, frases e expressões etc. que povoam a comunicação natural humana. As tecnologias baseadas em PLN têm se tornando cada vez mais difundidas. Por exemplo (Bird; Klein; Loper, 2019):

- celulares e *notebooks* suportam texto preditivo e reconhecimento de escrita;
- motores de busca na *web* dão acesso a informações encerradas em texto não estruturado;
- tradução automática permite recuperar textos escritos em chinês e lê-los em espanhol; e
- análise de texto permite detectar sentimentos em *tweets* e *blogs*.

Ao proporcionar interfaces homem-máquina mais naturais e acesso mais sofisticado às informações armazenadas, o processamento de linguagem passou a desempenhar um papel importante na sociedade da informação multilíngue. O tamanho imensamente grande, a natureza irrestrita e a ambiguidade da linguagem natural levaram a dois problemas ao usar abordagens de análise padrão que se baseavam puramente em regras simbólicas feitas à mão (Nadkarni *et al*., 2011, p. 544).

- O PLN deve, em última instância, extrair o significado (**semântica**) do texto: gramáticas formais que especificam a relação entre as unidades do texto – partes do discurso como substantivos, verbos e adjetivos – endereçam principalmente a **sintaxe**. Pode-se estender as gramáticas para abordar a semântica da linguagem natural, expandindo muito à subcategorização, com regras/restrições adicionais (por exemplo, *comer* aplica-se apenas a substantivos de itens ingeríveis). Infelizmente, as regras podem agora se tornar excessivamente numerosas, frequentemente interagindo de forma imprevisível, com análises ambíguas mais frequentes (são possíveis várias interpretações de uma sequência de palavras).
- As regras escritas à mão lidam com a prosa falada *não gramatical* e (em contextos médicos) a prosa altamente telegráfica das anotações de progresso no hospital, embora tal prosa seja compreensível para humanos.

Em termos de limitações da PLN, Bird *et al*. (2019) citam que, apesar de vários avanços, os sistemas de linguagem natural implantados para aplicativos do mundo real ainda não estão aptos a realizar o raciocínio de bom senso ou aproveitar o conhecimento do mundo de uma maneira geral e robusta. Pode-se esperar que esses difíceis problemas de inteligência artificial sejam resolvidos,

mas, enquanto isso, é necessário conviver com algumas limitações severas nas capacidades de raciocínio (de máquina) e conhecimento dos sistemas de linguagem natural. Consequentemente, desde o início, uma meta importante da pesquisa do PLN tem sido progredir na difícil tarefa de construir tecnologias que *entendam a linguagem*, usando técnicas superficiais, porém poderosas, em vez de conhecimento irrestrito e capacidades de raciocínio (Bird *et al*., 2019).

TEMA 2 – BREVE HISTÓRICO DO PROCESSAMENTO DE LINGUAGEM NATURAL

A noção geral por trás do PLN se estende à questão da **comunicação** entre agentes inteligentes interagindo com seres humanos. Um processo de comunicação envolve a troca intencional de informações pelo uso de sinais que são produzidos e percebidos, extraídos de um sistema de convenção de sinais compartilhados. A **linguagem**, uma das distinções dos seres humanos em relação aos animais, consiste num conjunto de mensagens estruturadas que permite a comunicação da maior parte do que se conhece do mundo. É interessante notar que Turing baseia o teste na linguagem, por estar conectada intimamente ao pensamento (Russell; Norvig, 2004, p. 765).

É interessante notar que a ideia da linguagem como **ação** já havia sido tratada ainda no início do século XX, no âmbito da Filosofia. Um dos trabalhos proeminentes é de Ludwig Wittgenstein, que publicou duas obras significativas quanto à natureza da linguagem (*Tractatus Logico-Philosophicus* e *Investigações Filosóficas*). Ludwig propôs uma teoria que considera a linguagem como **forma computada** e como **ação**. Ele caracterizou-se por trabalhar na fronteira entre a determinação e a indeterminação. A partir de um único ponto de vista teórico, explorou os dois lados, concentrando-se, no início, no **aspecto lógico** e, posteriormente, no **aspecto utilitário** (Buchholz, 2009).

Outra noção interessante explorada na Filosofia foram os **atos da fala**. John Searle explica que a produção de sinais e sons associados constitui a realização de um ato de fala. Para isso, a mente impõe intencionalidade à produção de sons ou sinais diversos, por meio da imposição das condições de satisfação do estado mental à produção dos fenômenos físicos. Quando alguém fala *está chovendo*, existe a expressão de uma crença que está na mente desse alguém por meio de um ato de fala (Searle, 2002, p. 229). Atos de fala podem

ser estendidos a exemplos, como envio de *e-mails*, e/ou a utilizar a linguagem, como Libras (Russell; Norvig, 2004, p. 766).

Em termos de história do PLN, um evento significativo foi o lançamento, por Noam Chomsky, do livro *Estruturas Sintáticas*. A teoria da obra serviu de base para que Chomsky contrapusesse uma crítica à teoria behaviorista, cujos fundamentos residiam no livro *Verbal Behavior*, publicado por B. F. Skinner, em 1957. A teoria behaviorista não explicava a criatividade na linguagem – como uma criança compreendia e formava frases sem nunca ter ouvido algo antes. Assim, Chomsky conseguia explicar esse fato por meio de modelos sintáticos, sendo um modelo dotado de suficiente formalidade para que pudesse ser expressada na forma de um programa (Russell; Norvig, 2004, p. 17).

A linguagem Prolog (acrônimo para *Programmation en Logique*) foi implementada por Alain Colmerauer e equipe na Universidade de Aix-Marseille, em 1972. Prolog foi imaginada como uma linguagem própria para o PLN. Em 1977, na Universidade de Edimburgo, foi formalizada por David H. D. Warren e Luiz M. Pereira, dentre outros, a versão conhecida atualmente de Prolog (denominado *Prolog de Edimburgo*). A programação em lógica implementada em Prolog utiliza o conceito de cláusulas de Horn, as quais eram adequadas para a representação de qualquer **gramática livre de contexto.** Além disso, permitiam a formulação de objetivos pudesse ser feita por questões expressas em sentenças de linguagem natural (Palazzo, 1997).

Até então, os métodos de PLN de caráter determinístico eram predominantes. Na década de 1980 e início dos anos 1990, há uma reorientação fundamental em que aproximações simples e robustas substituíram análises profundas: o **processamento estatístico de linguagem natural**. As pesquisas tiveram um incremento em **gramáticas livres de contexto probabilísticas**. A avaliação de sentenças também passa a ser menos rigorosa e entram em ascensão os métodos de *Machine Learning* que faziam uso de probabilidades. O próprio Chomsky questionou no *livro Estruturas Sintáticas* a utilidade de modelos de linguagem probabilísticos. Grandes corpos de texto anotados (**corpora**) começaram a ser empregados para treinar algoritmos de aprendizado de máquina – a anotação contém as respostas corretas – e forneceram padrões de ouro para avaliação (Nadkarni; Ohno-Machado; Chapman, 2011, p. 545).

Também na década de 1980 houve uma tendência ao desenvolvimento de inclusão de mais informações no léxico do que na gramática. O léxico se

refere à lista de palavras permitidas, agrupadas em categorias ou classes. Se a lexicalização é levada ao extremo, chega-se à **gramática de categorias**, com apenas duas regras gramaticais; ou mesmo à **gramática de dependência**, em que não existe nenhum sintagma, somente palavras (Russell; Norvig, 2004, p. 800). Entretanto, alguns trabalhos nos anos 2000, como o de Klein e Manning (2003), demonstraram que uma gramática não lexicalizada pode ter *performance* superior a outras gramáticas que contenham algum nível de léxico.

O advento do paradigma de aprendizado profundo (*deep learning*) nos anos 2000 permitiu o surgimento de redes neurais artificiais com alta capacidade de processamento de linguagem. A partir do trabalho seminal de Hinton *et al*. (2006), foi demonstrado que o uso da estratégia do **pré-treinamento intensivo de camadas** (*greedy layer-wise pre-training*) pode melhorar a *performance* das RNA em problemas de alta complexidade. Isso se fundamenta em uma série de evidências a partir de estudos neurológicos de que o cérebro desenvolve os conceitos de nível superior em estágios, primeiro extraindo informação de várias camadas de representações úteis e gradualmente mais complexas (Battiti; Brunato, 2017, p. 104).

As **redes neurais recorrentes** (*recurrent neural networks*, ou RNN) são uma família de redes neurais para processamento de dados sequenciais. Do mesmo modo que uma rede convolucional é uma rede neural especializada para processar uma grade de valores X, como uma imagem, uma rede neural recorrente é uma rede neural especializada para processar uma sequência de valores $x^1 \ldots x^\tau$. Assim como as redes convolucionais podem escalar prontamente para imagens com grande largura e altura, e algumas redes convolucionais podem processar imagens de tamanho variável, as redes recorrentes podem escalar para sequências muito mais longas do que seria prático para redes sem especialização baseadas em sequência. A maioria das redes recorrentes também pode processar sequências de comprimento variável (Goodfellow; Bengio; Courville, 2016).

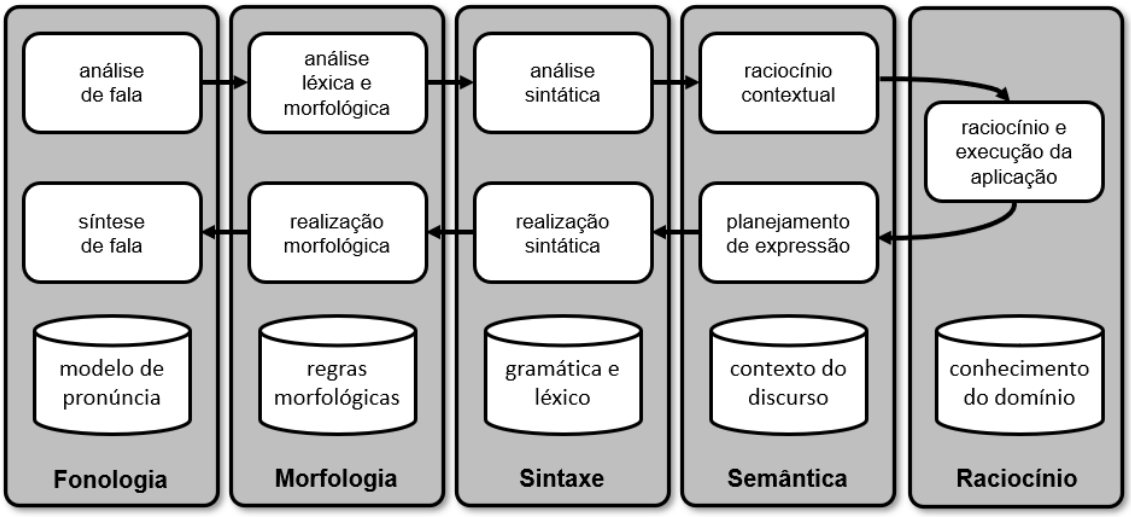
Uma integrante da família das RNN é a **LSTM** (do inglês, *long short-term memory*), que se refere a uma nova arquitetura de rede recorrente contendo um algoritmo de aprendizagem também baseado em descida de gradiente. A LSTM foi projetada para superar problemas de atraso de erro e pode aprender a transpor intervalos de tempo em excesso, mesmo no caso de sequências de entrada incompressíveis e com ruído, sem perda de recursos de intervalo de

tempo curto (Hochreiter; Schmidhuber, 1997). A LSTM tem sido bem-sucedida em aplicações como reconhecimento de escrita e de fala, geração de escrita, análise de sentimento e tradução de máquina.

TEMA 3 – CONCEITOS E DEFINIÇÕES

A arquitetura de sistema de diálogo simples pode ser visualizada na Figura 2. Ao longo da parte superior do diagrama, movendo-se da esquerda para a direita, há uma sequência de componentes de compreensão da linguagem. Tais componentes mapeiam a partir da entrada de fala por meio de análise sintática para algum tipo de representação de significado. Ao longo do meio, movendo-se da direita para a esquerda, está a sequência reversa de componentes para converter conceitos em fala. Esses componentes referem-se aos aspectos dinâmicos do sistema. Por fim, na parte inferior do diagrama, estão alguns corpos representativos de informações estáticas: os repositórios de dados relacionados à linguagem que os componentes de processamento utilizam para fazer seu trabalho.

Figura 2 – Arquitetura de um sistema de diálogo falado



Fonte: Elaborado com base em Bird *et al*. (2019).

A seguir, estão descritos alguns conceitos e definições fundamentais de linguagem, devido a Russell e Norvig (2004, p. 767).

- **Linguagem formal**: conjunto de cadeias que pode ser infinito; uma cadeia é uma concatenação de palavras ou símbolos terminais.
- **Linguagem natural**: não possuem nenhuma definição rígida como as linguagens formais, sendo utilizadas por uma comunidade de falantes; no

estudo de processamento de linguagem natural, busca-se tratar uma linguagem natural de maneira formal.

- **Gramática**: conjunto finito de regras que especificam uma linguagem; enquanto as linguagens formais possuem sempre uma gramática oficial, na linguagem natural há o empreendimento dos linguistas na codificação de uma gramática que possa fundamentar tal linguagem, a partir das pesquisas e descobertas.
- **Semântica**: associação de um significado a uma cadeia válida; isso acontece tanto para as linguagens formais como naturais.
- **Pragmática**: no caso de linguagens naturais, o significado de uma cadeia em uma situação específica; o significado não está apenas nas palavras, mas também na interpretação das palavras de forma situada.
- **Sintagma**: subcadeias que compõem as cadeias principais e que existem em categorias, tais como o **sintagma nominal**, que se refere a seres ou objetos no mundo (por exemplo, *o rei*), e o **sintagma verbal**, que se refere a ações ou estados (por exemplo, está vivo*"*). Sintagmas nominais podem se combinar a sintagmas verbais para formar **sentenças**; uma sentença é descrita como S; um sintagma nominal é simbolizado por SN; e o sintagma verbal é simbolizado por SV; os símbolos S, SN e SV são chamados também de **símbolos não terminais**.
- **Regras de reescrita**: são usadas para a definição de gramáticas; é comum o uso da forma BNF (*backus-naur form*) para as regras de reescrita; um exemplo de regra é *S -> SN SV*, que pode ser interpretada como *um S pode consistir em qualquer SN seguido por qualquer SV*.
- **Léxico**: lista ou conjunto de palavras permitidas; as palavras são agrupadas em categorias ou classes de palavras, tais como substantivos, pronomes, verbos, adjetivos etc. Os símbolos, em um léxico, são denominados **símbolos terminais**.
- **Classes fechadas**: classes de um léxico que são fixas e bem definidas. Por exemplo, classes de artigos definidos (o, a).
- **Classes abertas**: classes de um léxico nas quais é impossível listar todas as instâncias, existindo milhares ou mesmo novas instâncias sendo adicionadas a todo momento. Por exemplo, nomes próprios.

- **Análise sintática**: processo de construção de uma árvore de análise para uma cadeia de entrada. Os nós internos representam os sintagmas e os nós finais representam as palavras das classes do léxico.
- **Análise semântica**: constitui no processo de extração de significado de uma expressão vocal como uma expressão em uma linguagem de representação.
- **Interpretação pragmática**: as mesmas palavras podem ter significados diferentes, em situações distintas; expressões vocais que tenham o mesmo significado são denominadas ***ambíguas***.

Conforme Chomsky citado por Russell e Norvig (2004, p. 768), as gramáticas podem ser hierarquizadas da seguinte forma.

- **Recursivamente enumeráveis**: utilizam regras irrestritas; ambos os lados da regra podem conter qualquer número de símbolos terminais e não terminais.
- **Sensíveis ao contexto**: possuem restrições em relação à quantidade de símbolos que deve ser a mesma em ambos os lados da regra.
- **Livres de contexto**: o lado esquerdo possui apenas um símbolo não terminal; são populares para as linguagens naturais e de programação.
- **Regulares**: o tipo mais restrito de gramática; o lado esquerdo consiste em um único símbolo não terminal e o lado direito contém um símbolo terminal, opcionalmente seguido de um símbolo não terminal.

TEMA 4 – GRAMÁTICAS

Qualquer linguagem, seja natural ou de programação, precisa ter uma estrutura e organização, por mínima que seja, para que possa ser entendida pelas pessoas que a falam ou praticam. Ao longo do tempo, os linguistas notaram que existem padrões e regras na forma como as pessoas se expressam por meio da fala e escrita. Numa conversação, as pessoas intuitivamente analisam o que está sendo falado pela outra pessoa, para poderem emitir uma resposta apropriada. Assim, deve existir uma **gramática** para fornecer a ordem necessária das palavras e frases, em uma linguagem para proporcionar a compreensão, ou seja, o significado do que está sendo falado.

Uma linguagem de programação, por sua vez, possui regras rígidas para a escrita de símbolos, palavras ou *strings*; um programa escrito nessa linguagem

precisa seguir essas regras para ser interpretado ou compilado e assim executar a tarefa desejada. Portanto, uma gramática é um conjunto finito de regras que especificam uma linguagem, e define formalmente as regras de sintaxe de uma linguagem. Cada regra em uma gramática expressa a estrutura de uma subfrase, que, por sua vez, expressa outra estrutura, e assim sucessivamente, até chegar aos símbolos terminais (Parr, 2012, p. 21).

Uma gramática deve permitir que a partir da análise sintática seja construída uma árvore da análise (do inglês, *parse tree*). Essa árvore começa com o nó principal contendo a frase original e à medida que as expressões são processadas por um analisador (*parser*), o qual utiliza as regras definidas na gramática, os nós abaixo do nó principal vão formando o restante da árvore. Para descrever um exemplo, na língua inglesa, pode-se imaginar uma frase a ser analisada com base em uma gramática:

A small cat loves the white dog

Utilizando uma gramática para a língua inglesa bastante reduzida, porém permitindo a explanação sem perda de generalidade, essa gramática pode ser visualizada na Figura 3.

Figura 3 – Uma gramática para língua inglesa do exemplo do texto

	Regras
1	Sentença → SN SV
2	SN → Substantivo | Artigo Substantivo | Artigo Adjetivo Substantivo
3	SV → Verbo | Verbo SN | Verbo SN Advérbio
4	Substantivo → [house] | [cat] | [water] | [dog] | [John] | [Mary]
5	Artigo → [a] | [the]
6	Adjetivo → [big] | [small] | [white] | [black] | [tall] | [short]
7	Verbo → [goes] | [comes] | [fights] | [drinks] | [loves] | [has]
8	Preposição → [with] | [before] | [after] | [until] | [of] | [since]

*SN – Sintagma Nominal; SV – Sintagma Verbal.

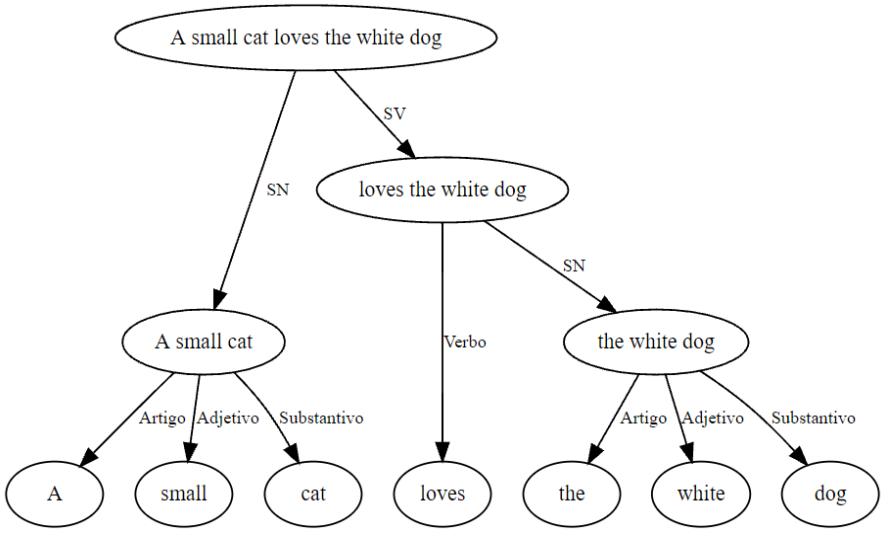
Fonte: Elaborado com base emForouzan e Mosharraf (2011).

A definição das regras segue o formato BNF (*backus-naur form*), comum à definição de gramáticas em geral. Uma regra contém uma seta (→), a qual indica que o antecedente (esquerda da seta) é composto pelo consequente (direita da seta) da regra. Se existe mais de uma opção possível, o consequente é separado por barras verticais (|).

Na Figura 3, a primeira regra que define uma sentença que é composta de um sintagma nominal e um sintagma verbal. A segunda regra define que um sintagma nominal pode ser um substantivo, um artigo seguido de um substantivo ou um artigo seguido de um adjetivo e seguido, por sua vez, de um substantivo. Na terceira regra, um sintagma verbal pode ser apenas um verbo; um verbo seguido de um sintagma nominal; ou um verbo seguido de um sintagma nominal, que é seguido de um advérbio. Essas regras definem símbolos não terminais.

As regras seguintes tratam dos símbolos terminais (ou seja, o léxico). Para um substantivo, existem seis opções (*house*, *cat*, *water*, *dog*, John e Mary), e assim sucessivamente. Na construção da árvore da análise (*parsing tree*), os símbolos do léxico farão parte dos nós inferiores da árvore. Dessa forma, o analisador (*parser*) toma como entrada a frase e produz na saída a árvore da análise, conforme consta na Figura 4.

Figura 4 – Análise da frase *A small cat loves the white dog*, conforme a gramática da Figura 3



Fonte: Medeiros, 2021.

Para os símbolos não terminais, a regra 1 produz:

Sentença → SN SV		
A small cat loves the white dog	*A small cat*	*loves the white dog*

A regra 2 produz para o sintagma nominal originado pela regra 1:

SN → Artigo Adjetivo Substantivo		
SN: A small cat		
Artigo	Adjetivo	Substantivo

A	*small*	*cat*

A regra 3 produz para o sintagma verbal originado pela regra 1:

SV → Verbo SN	
SV: loves the white dog	
Verbo	SN
loves	*the white dog*

E a regra 2 novamente produz para o sintagma nominal originado pela aplicação da regra anterior:

SN → Artigo Adjetivo Substantivo		
SN: the white dog		
Artigo	Adjetivo	Substantivo
the	*white*	*dog*

Outra maneira explorada pelas linguagens de programação e *toolkits*, como a NLTK, é expressar a árvore sintática pelo uso de parênteses:

```
(S
  (SN (Artigo a) (Adjetivo small) (Substantivo cat))
  (SV
    (Verbo loves)
    (SN (Artigo the) (Adjetivo white) (Substantivo dog))))
```

Com essa representação, à medida em que a árvore desce na análise, um novo nó é originado com o conteúdo dentro de novos parênteses, e assim sucessivamente. O exemplo demonstrado serve também para ilustrar como uma gramática livre de contexto pode ser utilizada para analisar sentenças em uma determinada linguagem. A gramática da Figura 3 é uma gramática livre de contexto, pois as regras possuem no antecedente apenas um símbolo não terminal. Esse processo é o mesmo para a análise de linguagens formais, como as linguagens de programação. É claro que uma gramática deve ser robusta o suficiente para dar conta de um grande número de combinações que podem ser produzidas.

Pode-se notar também que o léxico estava junto das regras que definiam os símbolos não terminais. Na prática, entretanto, os *softwares* analisadores de linguagem costumam implementar uma gramática dividindo as regras que contêm símbolos não terminais das que contêm o léxico. Por exemplo, o analisador de uso genérico ANTLR (Parr, 2012) apresenta um analisador léxico (denominado de *lexer*), que é separado do analisador sintático (denominado *parser*).

Na implementação de uma linguagem de programação, o processo de análise de expressões escritas numa determinada linguagem é similar ao PLN.

Dessa forma, é necessário construir um aplicativo que leia as sentenças e reaja apropriadamente a frases e símbolos de entrada que descobrir. Em termos gerais, se um aplicativo computa ou *executa* sentenças, chamamos esse aplicativo de ***interpretador***. Exemplos de interpretadores incluem calculadoras, leitores de arquivo de configuração e mesmo os interpretadores Python. Na conversão de frases de um idioma para outro, chamamos esse aplicativo de ***tradutor***. Os exemplos incluem conversores de Java para C # (Parr, 2012, p. 9).

Outro exemplo comum, presente também nas linguagens de programação, é a interpretação de **expressões matemáticas**. Uma expressão matemática, dentro de um texto, deve ter um tratamento específico de forma a atender à sintaxe correta, para que seja uma expressão válida e proporcione uma interpretação semântica, que é o próprio resultado da expressão. Por exemplo, veja a seguinte expressão:

```
25 + 56
```

A validade da expressão estará garantida no uso correto dos números entre os operadores. Um operador (soma, subtração, multiplicação, divisão) precisa estar entre dois números para que exista uma interpretação, isto é, um resultado). Tais operadores são ditos ***binários***. Outro fator a considerar é a prioridade das operações sobre outras: a multiplicação e a divisão precedem a soma e subtração (no sentido de que devem ser executadas antes). É possível também expressar operadores tais como:

```
- 100
```

Neste caso, um número negativo também é um número válido, desde que o operador - (menos) seja colocado antes do número. Esses operadores são denominados ***unários***. Isso também deve ser válido com números positivos, nos quais o sinal + (mais) seja colocado antes do número. Assim, os números

```
100
```

e

```
+ 100
```

são expressões sintáticas diferentes, mas irão produzir a mesma interpretação semântica. Uma expressão matemática pode ser composta por várias operações que devem ser resolvidas com prioridade. O uso de parênteses

em uma expressão permite a definição de quais operações devem ser executadas antes de outras. Veja a seguinte expressão:

```
25 + 56 * (40 / 15)
```

Os parênteses indicam que primeiro deve ser efetuada a operação 40 / 15, depois o resultado deverá ser multiplicado por 56 e, em seguida, somado com 25. Uma expressão válida requer que para cada parêntese aberto deve haver um parêntese fechado. Uma expressão complexa pode conter, dessa forma, vários conjuntos de parênteses, indicando a prioridade no cálculo.

Um exemplo de gramática livre de contexto para ser utilizada em expressões matemáticas está expresso na Figura 4. Essa gramática contém 12 regras, sendo que para os símbolos não terminais estão atendidas as operações algébricas e cada símbolo tem sua regra lexical. Ainda há uma regra (6) que simplifica a extração de um número; ela está expressa de modo abstrato. Em uma linguagem de programação, é necessário que seja detalhado que tipo de número será capturado pela gramática. Dependendo da implementação ou da biblioteca utilizada, isso pode ser feito por meio de expressões regulares ou ainda por algum pré-processamento que possa generalizar os números.

Figura 5 – Uma gramática para expressões matemáticas do exemplo do texto

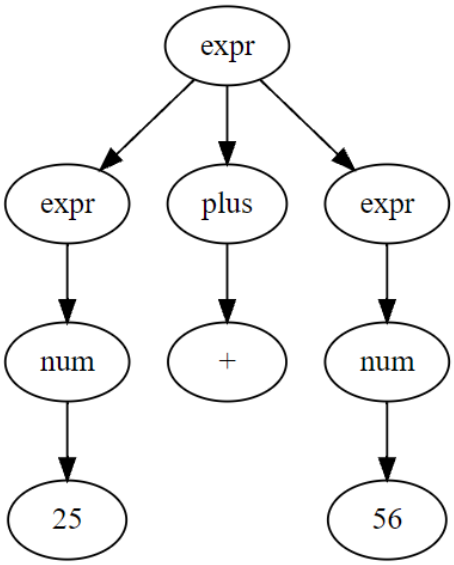
	Regras
1	`expr → LPAR expr RPAR`
2	`expr → expr POW expr`
3	`expr → expr DIV expr | expr MUL expr`
4	`expr → MINUS expr | PLUS expr`
5	`expr → expr PLUS expr | expr MINUS expr`
6	`expr → num`
7	`POW → [^]`
8	`DIV → [/]`
9	`TIMES → [*]`
10	`PLUS → [+]`
11	`MINUS → [-]`
12	`LPAR → [(]`
13	`RPAR → [)]`

Fonte: Medeiros, 2021.

Com a gramática da Figura 5, a análise da expressão 25 + 56 produzirá a árvore sintática da Figura 6. A expressão que casa com a soma está na regra 5,

a qual chama a regra 10 para extrair o símbolo + e depois chama a regra 6 duas vezes para os números 25 e 56.

Figura 6 – Árvore sintática para a expressão 25+56, conforme a gramática da Figura 5



Fonte: Medeiros, 2021.

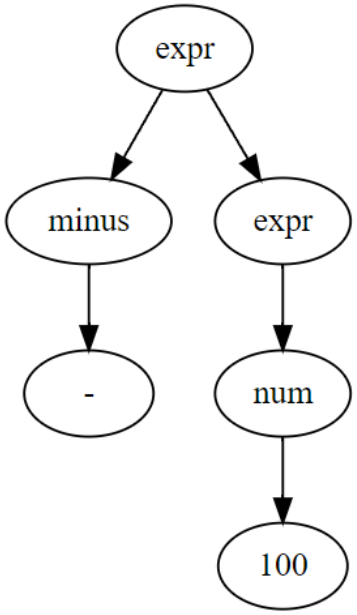
A mesma expressão da árvore sintática pode ser escrita no formato de parênteses.

```
(expr (expr (num 25)) (plus +) (expr (num 56)))
```

Para um segundo exemplo, utilizando uma expressão unária, como -100, a árvore sintática da Figura 7 representa a análise feita a partir da gramática da Figura 5. A análise começa com a regra 4, que extrai o sinal - (menos) pela regra 11 e na sequência chamando a regra 6 para a extração do número ‘100’. A mesma árvore expressa na forma simplificada com parênteses é:

```
(expr (minus -) (expr (num 100)))
```

Figura 7 – Árvore sintática para a expressão -100, conforme a gramática da Figura 5



Fonte: Medeiros, 2021.

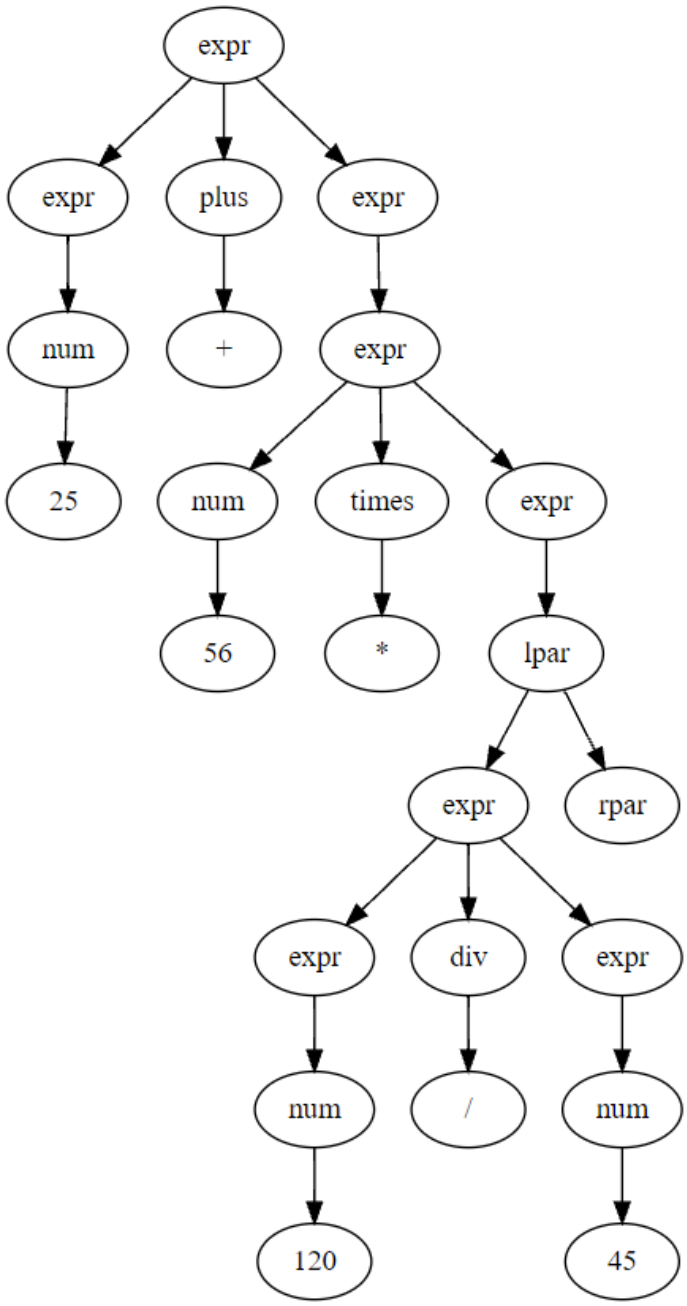
Para um terceiro exemplo utilizando uma expressão com parênteses,

```
25 + 56 * ( 120 / 45 )
```

a árvore sintática para essa expressão se encontra na Figura 8. A regra 5 é chamada para a operação de soma, sendo seguida pela regra 3 para a multiplicação. Na multiplicação, a regra 1 é acionada para a captura dos parênteses; em seguida, a análise continua para a extração da operação de divisão com a regra 3, novamente.

É importante mencionar a extração da expressão na forma de uma árvore sintática com relação à precedência de operações. Primeiramente, o conteúdo dentro das expressões precisa ser avaliado; portanto, a interpretação semântica a ser feita deve promover uma busca em profundidade na árvore, de forma a alcançar os nós mais profundos e fazer as operações nos nós mais abaixo na árvore, para depois efetuar as operações que estão nos nós mais acima. Em algumas ferramentas (como a ANTLR), a ordem em que as regras estão na gramática são suficientes para definir a precedência dos nós que precisam ser visitados primeiramente. No entanto, em outras, talvez seja necessário desdobrar a gramática para contemplar uma precedência entre as operações.

Figura 8 – Árvore sintática para a expressão 25 + 56 * (120 / 45), conforme a gramática da Figura 5



Fonte: Medeiros, 2021.

A mesma árvore expressa na forma simplificada entre parênteses é:

```
(expr
  (expr (expr (num 25)) (plus +) (expr (num 56)))
  (times *)
  (expr
    (lpar ()
    (expr (expr (num 120)) (div /) (expr (num 45)))
    (rpar ))))
(expr
```

```
(expr (num 25))
(plus +)
(expr
  (expr (num 56))
  (times *)
  (expr
    (lpar ()
    (expr (expr (num 120)) (div /) (expr (num 45)))
    (rpar )))))
```

TEMA 5 – EXPRESSÕES REGULARES

As expressões regulares desempenham um papel importante no contexto da análise sintática. A noção por trás das expressões regulares é o problema relacionado com certos símbolos terminais difíceis de serem armazenados em um léxico. Basta imaginar como representar números associados a quantidades ou moedas, por exemplo, que fica clara a manifestação dessa dificuldade: se a cada valor possível de um número, tais valores tivessem que ser representados em um léxico, ficam evidentes tais problemas de representação.

As expressões regulares têm o objetivo de definir uma gramática regular para uma única cadeia de texto (Russell; Norvig, 2004, p. 821). Por meio de uma simbologia padrão, essas cadeias podem ser representadas por tal gramática regular. Expressões regulares também são úteis em processos de extração de texto dentro das linguagens de programação, bem como nos próprios interpretadores e compiladores de linguagens. Por exemplo, para representar apenas um dígito numérico, a seguinte cadeia regular permite tal possibilidade:

```
[0-9]
```

Dessa forma, qualquer dígito de 0 a 9 seria enquadrado nessa cadeia. Entretanto, se é necessário interpretar um número com vários dígitos, a representação seria da seguinte forma:

```
[0-9]+
```

Um número como 3429 seria capturado nessa cadeia. Caso seja necessário representar um número com dois dígitos para o ponto decimal, uma cadeia regular possível seria:

```
.[0-9][0-9]
```

Assim, .56 seria capturado por essa cadeia. Mas um número (moeda, por exemplo) pode ou não ter casas decimais. Outra forma de expressar essa possibilidade seria:

```
(.[0-9][0-9])?
```

Com esse formato, um número com ponto decimal ou seguido por nada seria enquadrado. Por fim, se queremos representar um valor de moeda de maneira completa com uma expressão regular, tem-se a seguinte cadeia:

```
R$[0-9]+(.[0-9][0-9])?
```

Essa expressão regular interceptaria valores de moeda tais como R$92.27 ou R$1000. A presença dos parênteses seguido pelo ponto de interrogação permite que as casas depois do ponto decimal sejam facultativas.

A Figura 9 mostra outras codificações para expressões regulares. É possível notar que várias combinações podem ser obtidas de maneira a prover um analisador sintático de capturar diferentes tipos de símbolos terminais que se encaixassem em uma expressão regular específica. No caso da gramática do exemplo anterior, para que a seguinte afirmação *John has $100.00,* pudesse ser analisada, a gramática teria de ser alterada de forma que a regra 2 fosse expressa como

```
SN → Substantivo | Artigo Substantivo | Artigo Adjetivo Substantivo | Número
```

E mais uma regra seria adicionada para capturar o valor numérico na afirmação

```
Número → $[0-9]+(.[0-9][0-9])?
```

Assim, qualquer valor numérico de moeda poderia ser analisado e transformado em um símbolo não terminal.

Figura 9 – Alguns símbolos encontrados em expressões regulares	Descrição	Exemplo
Símbolo		

|	Separa possíveis alternativas	[a-z|A-Z]+ Palavra com maiúsculas ou minúsculas
?	Indica 0 ou 1 ocorrência do elemento precedente	(.[0-9][0-9])? Presença ou não duas casas decimais
*	Indica 0 ou mais ocorrências do elemento precedente	[0-9]* Nenhum ou vários dígitos
+	Indica 1 ou mais ocorrências do elemento precedente	[0-9]+ Pelo menos um dígito

Fonte: Medeiros, 2021.

FINALIZANDO

O estudo de PLN iniciado nesta aula permitiu a contextualização dentro da área da Inteligência Artificial. Também foi estudado um breve histórico enraizando a área de PLN em estudos que se originam na própria filosofia da linguagem. Conceitos e definições pertinentes à área também foram vistos, assim como o estudo de uma das principais áreas de PLN, as gramáticas, que são fundamentais na análise sintática de uma sentença ou expressão. Essa análise sintática precede a interpretação semântica e a pragmática, sendo importante tanto para PLN quanto para a construção de linguagens de programação e interpretadores.

Por fim, as expressões regulares desempenham um papel importante na PLN, assim como em outras áreas tais como a recuperação de informações, no sentido de busca símbolos de acordo com certo padrão de expressão. O estudo desta aula prepara, portanto, para os estudos subsequentes que dão continuidade ao estudo de PLN.

REFERÊNCIAS

BATTITI, R.; BRUNATO, M. **The LION Way**: Machine Learning plus Intelligent Optimization. Trento: University of Trento, 2017. *E-book*.

BIRD, S.; KLEIN, E.; LOPER, E. **Natural Language Processing with Python**. [*S. l.*], 2019. Disponível em: <http://www.nltk.org/book/>. Acesso em: 1 fev. 2022.

BUCHHOLZ, K. **Compreender Wittgenstein**. 2. ed. Petrópolis: Vozes, 2009.

FOROUZAN, B.; MOSHARRAF, F. **Fundamentos da Ciência da Computação**. São Paulo: Cengage Learning, 2011.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge-MA: MIT Press, 2016. *E-book*.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A Fast Learning Algorithm for Deep Belief Nets. **Neural Comput**., Cambridge, MA, USA, v. 18, n. 7, p. 1527–1554, 2006. Disponível em: <https://doi.org/10.1162/neco.2006.18.7.1527>. Acesso em: 1 jan. 2022.

HOCHREITER, S.; SCHMIDHUBER, J. Long Short-Term Memory. **Neural Computation**, [*s. l.*], v. 9, n. 8, p. 1735–1780, 1997. Disponível em: <https://doi.org/10.1162/neco.1997.9.8.1735>. Acesso em: 1 jan. 2022.

KLEIN, D.; MANNING, C. D. Accurate unlexicalized parsing. [*s. l.*], p. 423-430, 2003. Disponível em: <https://doi.org/10.3115/1075096.1075150>. Acesso em: 1 jan. 2022.

NADKARNI, P. M.; OHNO-MACHADO, L.; CHAPMAN, W. W. Natural language processing: an introduction. **Journal of the American Medical Informatics Association**, [*s. l.*], v. 18, n. 5, p. 544–551, 2011. Disponível em: <https://doi.org/10.1136/amiajnl-2011-000464>. Acesso em: 1 jan. 2022.

PALAZZO, L. A. M. **Introdução à Programação PROLOG**. Pelotas-RS: Editora da Universidade Católica de Pelotas, 1997.

PARR, T. **The Definitive ANTLR 4 Reference**. Dallas, Texas: The Pragmatic Programmers, LLC, 2012.

RUSSELL, S.; NORVIG, P. **Inteligência Artificial**. Tradução da 2. ed. Rio de Janeiro: Editora Campus, 2004.

SEARLE, J. **Intencionalidade**. 2. ed. São Paulo: Martins Fontes, 2002.

TURING, A. M. **Can Digital Computers Think?**. Cambridge-UK: The Turing Digital Archive, 1951. Disponível em: <http://www.turingarchive.org/viewer/?id=459&title=1>. Acesso em: 1 jan. 2022.

TURING, A. M. Computing Machinery and Intelligence. **Mind**, [*s. l.*], v. LIX, n. 236, p. 433–460, 1950. Disponível em: <https://doi.org/10.1093/mind/LIX.236.433>. Acesso em: 1 jan. 2022.