PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Instituto de Ciências Exatas e Informática – ICEI

Departamento de Ciência da Computação

Engenharia de Computação

Hernane Velozo Rosa[1]

hernane.rosa@sga.pucminas.br[1]

ARTIGO SOBRE A LINGUAGEM DE USO GERAL PROLOG

Belo Horizonte

2022

ABSTRACT

Born without the focus of being implemented as a programming language, but rather of being a natural language processing tool, specifically for French, here comes the idea of Prolog in early July 1970, being consolidated as a language framed in the mathematical logic paradigm in the year 1973. It has data (object) orientation, with better application in heuristic processes and support for complex data

RESUMO

Nascida sem o foco de ser implementada como uma linguagem de programação, mas sim de ser uma ferramenta de processamento de linguagem natural, especificamente para o francês, eis que surge a ideia de Prolog no começo de julho de 1970, sendo consolidada como linguagem enquadrada no paradigma lógico matemático no ano de 1973. Possui orientação direcionada por dados (objetos), com melhor aplicação em processos heurístico e suporte a dados complexos

Artigo apresentado como requisito do projeto de seminários de estudo de uma linguagem de programação.

Professor: Dr. Marco Rodrigo Costa
Disciplina: Linguagens de Programação

Palavras chaves: Estudo de linguagens de Programação, Inteligência Artificial, Paradigma Lógico e Matemático.

Sumário

1 INTRODUÇÃO ESTENDIDA

Do francês, “**PRO**gramation et **LO**gique”, comumente conhecido como PROLOG, é o nome dado à linguagem de uso geral, enquadrada no paradigma de Lógica Matemática. Seus elementos básicos herdam características da lógica de primeira ordem de predicados, isto é, são fatos, regras e consultas. É a primeira e mais utilizada linguagem do paradigma da Programação em Lógica.

Prolog é uma linguagem cuja orientação é direcionada por dados, com melhor aplicação em processos heurístico e suporte a dados complexos, bem como estruturas de recursão. Também há suporte para processamento simbólico onde o uso da manipulação de símbolos pode ser utilizado ao invés de algoritmos definidos.

Bastante utilizada para resolução de problemas que envolvem objetos e relações, assim como como aplicações de inteligência artificial. No mundo, a linguagem tem bom proveito quando utilizada em disciplinas como Linguagens Formais e de Programação (paradigma lógico), Inteligência Artificial e Compiladores.

Para execução, Prolog utiliza um processo de combinação no qual quando o programa é instanciado, ele pedirá um objetivo e o usuário poderá entrar com esse objetivo para execução do programa.

1.1 SINTAXE E SEMÂNTICA

Programas em PROLOG são construídos a partir de termos. Termos podem ser constantes, variáveis ou estruturas. Na linguagem, cada termo é escrito como um conjunto de caracteres. Para fazer um comentário de bloco em dado trecho do código, basta envolvê-lo entre os símbolos de barra e asterisco, ou seja, /* */, ou caso seja necessário inserir um comentário de linha, é possível realizá-lo através do caractere de percentual (%) que deverá preceder o comentário. Desta maneira, Prolog irá ignorar o comentário.

% comentário de linha

/* Comentário de bloco
****/***

Para construir o nome das variáveis, será por meio de uma cadeia de caracteres. Além disso, a linguagem reconhece dois tipos de caracteres, aquele que podem ser impressos na tela do computador, e são divididos da seguinte maneira:

1.2 ÁTOMOS

Átomos são cadeias de caracteres composta pelos conjuntos:

- maiúsculas e mininusculas de Aa até Zz.
- Digitos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- Caracteres Especiais: + - * / < > = : . & _ ~

E podem ser construídos como cadeias de caracteres, dígitos e o sinal de subtraço, assim como cadeias de caracteres entre apóstrofos e especiais tal qual: ‘São_Paulo’

1.3 NÚMEROS

Além da representação em caractere, os números em Prolog ainda possuem os operadores aritméticos e os relacionais

Operadores Aritméticos	
adição	+
subtração	-
multiplicação	*
divisão	/
divisão inteira	//
resto de divisão	mod
potência	**
atribuição	is

Operadores Relacionais	
X maior que Y	X > Y
X menor que Y	X < Y
X maior ou igual a Y	X >= Y
X menor ou igual Y	X =< Y
X igual a Y	X =:= Y
X unifica com Y	X = Y
X diferente de Y	X = \ = Y

1.4 VARIÁVEIS E UNIFICAÇÕES

Uma **variável** pode estar em três estado sendo instanciada, livre ou não-instanciada:

- **Instanciada**: Momento em que a variável já está unificada a algum objeto
- **Livre ou não-instanciada**: quando a variável não é unificada a um objeto

Mas atenção: Uma vez a que variável seja instanciada, limita-se apenas a linguagem em transformá-la em não-instanciada através de seu mecanismo de inferência.

Em complemento, ainda há a **variável anônima**, que surge quando aparece em uma única cláusula não se faz necessário nomeá-la. Essa variável é escrita apenas com o um único caractere de subtraço "_".

Além disso, a cada novo subtraço no código, eis que surge uma nova variável anônima.

Já para **unificação**, devido a condição declarativa, a ordem da sequência não é importante. Outra característica da unificação em Prolog é quando uma variável X ainda não tiver sido instanciada será possível unificá-la com um átomo, um termo ou com uma outra variável desde que essa também não tenha sido instanciada. Além disso, um átomo somente poderá ser unificado com o mesmo átomo.

1.5 GRAMÁTICA E CLÁUSULAS DE HORN

A gramática pode ser usada para descrever a estrutura de objetos e computações, onde:

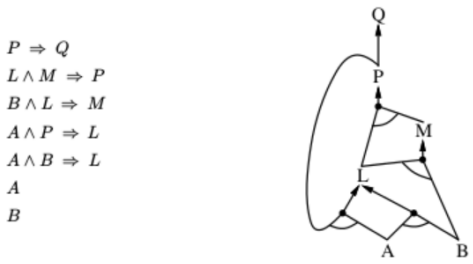
- Para descrever a estrutura de algoritmos: **Projetar**
- Para descrever a estrutura de entrada: **Analisar**
- Para gerar valor de saída: **Calcular**

Quando se tem uma cláusula definida, essa é uma cláusula de Horn que deverá possuir um literal positivo. Ela expressa um subconjunto de instruções de lógica de primeira ordem.

As cláusulas de Horn são fórmulas da forma ρ <= ρ1 ^ ρ2 ^ ... ^ ρn, em que n≥0, onde ρ é uma conclusão e ρ1, ..., pn são condições. Inclusive, fatos em Prolog são cláusulas de Horn com o corpo vazio, assim como as consultas são cláusulas de Horn, mas com a cabeça vazia. Quando não há literal positivo, a cláusula será chamada de meta.

Em Prolog são denotadas por <cabeça da cláusula> :- <corpo da cláusula>.

Figura 1.0 - Representação Com operações lógicas And e Or.

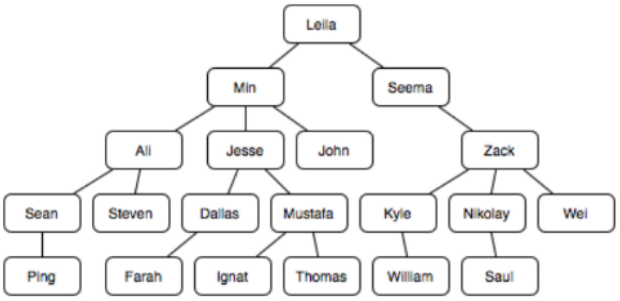


Disponível em: https://tex.stackexchange.com/questions/251896/and-or-graph-from-horn-clauses

1.6 FATOS

Mas afinal, o que é predicado? São característica inerente a um ser, atributo ou propriedade. Na matemática, predicado são relação. Em Prolog, os fatos são formados por predicados que contém argumentos/objetos cuja instrução é finalizada com o caractere ponto (.) que é equivalente ao ponto-virgula (;) como em C/C++. Portanto, o predicado é a relação na qual os objetos interagem.

Figura 1.1 - Representação genealógica.



Disponível em: < https://stackoverflow.com/calculating-cousin-relationship-in-prolog>

Além disso, é importante que os nomes dos predicados iniciem sempre com letras em minúsculas. A ordem de escrita é 1º predicado, 2º argumentos e 3º finalização da instrução. Por exemplo: pais(laura, daniel).

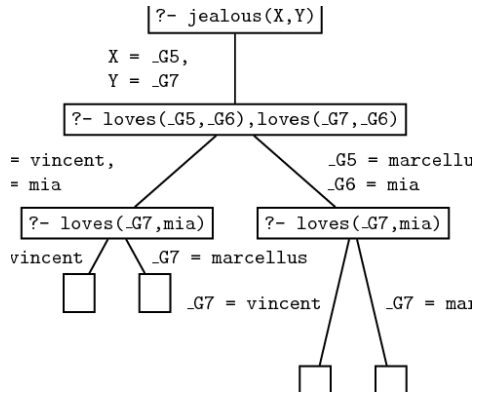
Pais é o nome da relação, isto é, o predicado laura e daniel são os argumentos. Todos os fatos em Prolog devem terminar com o caractere ponto '.' e os nomes das relações e argumentos deverão sempre ser escritos em minúsculo.

Um conjunto de fatos combinado com um conjunto de regras fazem uma base de dados Prolog.

1.7 CONSULTAS

Através de consultas é possível extrair informação de um programa. As variáveis que ocorrerem durante as consultas são existencialmente quantificadas. Quando se responde a uma consulta é o mesmo que que aferir se a questão é uma conclusão logica.

Figura 1.2 - Representação de consulta.



Disponível em: < http://www.let.rug.nl/bos/lpn//lpnpage.php?pagetype=html&pageid=lpn-htmlse6>

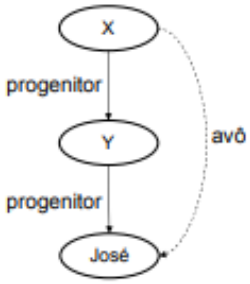
1.8 REGRAS

A utilização de regras permite avançar na linguagem e iniciar a construção de questões mais complexas. Em dadas situações, elas poderão ser verdadeiras quando satisfazerem as condições. Para denotar a regra, o símbolo ":-" indica uma condição. Resumidamente: um axioma.

É possível denotar as regras definindo relação entre avos ou entre tio e tia (XY):

avos(x, y) :- progenitor(X, Z), progenitor (Z, Y).
tios( ) :- p progenitor ais(Z,Y), progenitor (V, Z), progenitor (V, X).

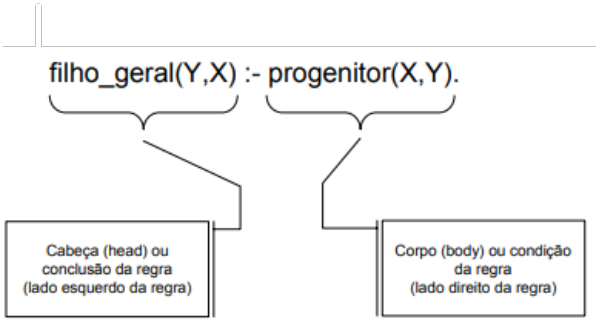
Figura 1.3 - Representação de consulta.



Disponível em: < https://silo.tips/download/inf-1771-inteligencia-artificial-14>

Onde a primeira parte da regra é chamada de cabeça ou conclusão, possui um símbolo condicional ":-" de "if" e, a parte posterior a essa condição é o corpo/parte condicional da regra.

Figura 1.4 - Representação de consulta.



Disponível em: < https://dcm.ffclrp.usp.br/~augusto/teaching/ia/IA-Prolog-Introducao-Tutorial.pdf>

1.9 AMARRAÇÕES

Amarração ou "binding", é o termo utilizado para descrever a atribuição de um valor à uma variável. Via de regra, em Prolog, ela ocorre durante a aplicação de regra ou quando é necessário satisfazer objetivos em uma consulta.

1.10 LISTAS EM PROLOG

Listas são estruturas de dados simples composta por uma sequência de elementos.

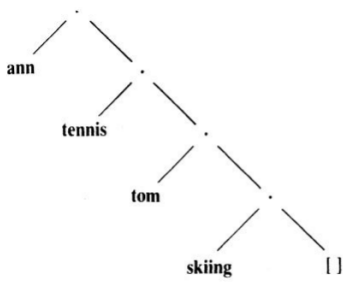
O primeiro item é chamado de cabeça "head" da lista. Já a parte remanescente é a calda. Pode ser escrito como:

[item¹, item², ...] ou [cabeça | Calda] ou ainda [item¹, item², ... | outros]

[ana, tenis, tom, skiing]

Como acima, ana é a cabeça e a cauda compõe a lista, ou seja**: [tenis, tom, skii]**

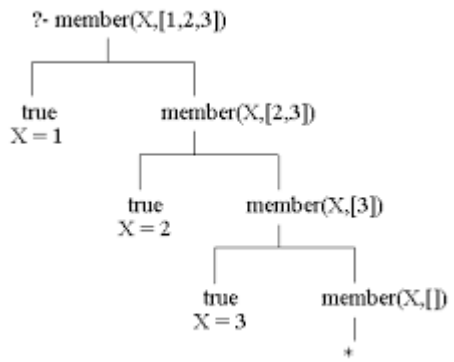
Figura 1.5 - Representação de lista.



A notação em Prolog é L = [a, b, c] e, para expressar essa notação, a linguagem fornece uma extensão da notação, que seria a barra vertical que separa a cabeça da calda, ou seja: **L= [a | Tail] Tail = [b, c].**

Além disso, essa notação vertical pode ser de uso mais geral na qual pode listar quaisquer números de elementos seguidos por ela e lista os elementos restantes, podendo ser denotado por **[a, b, c] = [a | [b, c]] = [a, b | [c] = [a, b, c | [ ]].**

Figura 1.6 - Representação de elementos da lista.



Disponível em: <https://www.cpp.edu/~jrfisher/www/prolog_tutorial/2_7.html>

1.11 APLICAÇÕES

Prolog tem diversas aplicações que vão desde a lógica matemática, prova automática de teoremas e semântica até a resolução de equações simbólicas, processamento de linguagem natural, sistemas especialistas e compiladores

1.12 O PODER DE EXPRESSIVIDADE

Implementando o paradigma declarativo, em contraste com paradigma imperativo de Pascal e C, onde descreve o que deve fazer e não como fazer. Em Prolog, temos que as regras de inferência e fatos são expressos de maneira declarativa em alto nível.

Ainda em alto nível, estruturas de árvores, grafos, listas, entre outros, são representadas de maneira natural em Prolog. Para resolução de lógica de cláusulas

definidas, a linguagem implementa um sistema que é um subconjunto da lógica de primeira ordem. Inclusive, a linguagem permite, através do uso da prototipagem rápida, a validação de especificações de sistemas de software.

Com suas regras gramaticais, também conhecido como DCC (*Definite Clause Grammar*), que é essencial durante o desenvolvimento de aplicações e/ou protótipos. Além disso, permite a "metaprogramação", na qual um programa consegue se auto modificar, bem como se examinar. Metaprogramação pode ser interpretador como um sinônimo de linguagem de alta ordem, isto é, por exemplo, interpretadores de regras para Inteligência Artificial.

1.13 INTELIGÊNCIA ARTIFICIAL

O IBM Watson utiliza um software chamdo IBM DeepQA além do frameWirk Apach Unstructure Information Management Architeture (UIMA), sendo escrito em vasta linguagens, o que inclui Prolog.

1.14 NO CONTEXTO LINGUISTICO

É possível realizar o processamento linguístico em Prolog tendo como base as gramáticas formais. Dispondo de um gerador, as sentenças serão geradas em conformidade com o léxico e gramática, tendo um aceitador a responsabilidade de verificar a conformidade das sentenças. Em conjunto, é claro, com um analisador que reconhecerá frase permitirá que construa uma "arvore decorada", o que seria uma abstração.

2 HISTÓRICO ESTENDIDO SOBRE A LINGUAGEM

Como será mostrado mais adiante, a programação lógica surge em meados dos anos 1970 como um paradigma no qual a abordagem tem como base a expressão de programa de forma lógica simbólica, utilizando processo de inferência para produção de resultados a partir de um conjunto de base de dados fatos e regras.

Nessa década, no começo de julho de 1970, Philippe Roussel e Robert Pasero chegam em Montreal, Canadá, e são convidados por Alain Colmerauer, um então

cientista da computação francês que estava liderando um projeto de tradução automática na Universidade de Montreal.

Ainda muito jovens, Philippe e Robert, ambos com vinte e cinco anos, estavam em momento importantes em suas carreiras tendo sidos premiados com cargos de ensino em Ciência da Computação na Faculdade de Ciências Luminy. Alain era o mais velho, com vinte e nove anos. Ele passa por três anos no Canadá e tão logo retornaria a França. Durante a estadia em Montreal, Philippe e Robert rapidamente se familiarizam com o processamento de linguagens naturais, tendo eles escrito diversos analisadores livres de contexto não determinísticos em Algol 60, assim como um gerador de paráfrases francês, fazendo uso do Q-systems, linguagem cuja autoria fora de Alain para o projeto de tradução. Ao mesmo tempo, Jean Trudel, um pesquisador canadense e doutorando de Alain, que escolhe para trabalhar na provação automatizada de teoremas.

Devido sua experiência com o curso de lógica que realizou, tendo sido ministrado por Martin Davis, Jean tem certa vantagem sobre um artigo de referência de Alan Robinson, publicado em 1965, e que era considerado difícil de ler. A partir dessa experiência, Jean desenvolve um provador de teoremas completo, escrito num estilo de programação atualizado onde toda a computação se consistiram na modificação de ponteiros.

Já no começo de 1971, todos regressam à Marselha, na França. Alain consegue um cargo de professor em Ciência da Computação, a título de “maître de conférence”, do francês, que pode ser traduzido como "professor experiente”, trazendo consigo Jean Trudel devido à um incentivo adquirido pela Hydro-Quebec. No então projeto que estavam trabalhando, que tinha como intuito realizar deduções com base em textos escritos em francês, Jean Trudel e Philippe Roussel são incumbidos de trabalha na parte de dedução, enquanto Alain e Robert trabalhariam com a parte da linguagem natural.

Jean Trudel consegue melhorar seu provador e, em maio daquele ano, Philippe produz toda uma série de provadores escritos em Algol. Um sistema primitivo de comunicação em linguagem natural é desenvolvido pela equipe de Colmerauer. A interface das fórmulas lógicas e o francês são compostas por cinquenta regras de entradas para dezessete regras de saída do Q-system. O raciocínio então é

implementado por um dos provadores de Philippe, permitindo a seguinte conversa com o computador:

Usuário:	**Computador:**
Gatos matam ratos. Tom é um gato que não gosta de ratos que comem queijo. Jerry é um rato que come queijo. Max não é um rato. O que Tom faz?	*Tom não gosta de ratos que comem queijo. Tom mata ratos.*
Usuário: *Quem é um gato?*	**Computador:** *Tom.*
Usuário: *O que Jerry come?*	**- Computador** *Queijo.*
- Usuário: *Quem não gosta de ratos que comem queijo?*	**- Computador:** *Tom.*
- Usuário: *O que Jerry Come?*	**- Computador:** *Queijo.*

Algum tempo depois, Trudel encontrou um método muito interessante, a resolução SL feita por Robert Kowalski, um então cientista da computação francês que desenvolve uma interpretação procedural de cláusulas de Horn (que é a disjunção de literais) e, também, demonstra que o famoso axioma "**A ocorre se B ocorre"**, permitindo o entendimento que esse axioma poderia ser lido como um procedimento em uma linguagem recursiva na qual **A é a cabeça da cláusula, assim como B o seu corpo**.

Jean Trudel convence Alain a convidar Robert Kowalski a visitá-los durante uma semana de junho de 1971. O encontro foi muito importante pois era a primeira vez que Alain e sua equipe conversaram com um especialista em demonstração automatizada de teoremas, bem como foi uma experiência interessante para Robert Kowalski que conhecera pessoas realmente interessadas em sua pesquisa e que estavam interessadas em utilizá-la em processamento de linguagem natural.

Em setembro daquele ano, Alain e Trudel participaram de uma palestra na IJCAI onde de Terry Winograd discursava acerca de processamento de linguagem natural. Para eles, foi muito intrigante dado ao fato de que Terry não utilizou do formalismo unificado. Nessa época, eles tomam conhecimento da linguagem Planner, de Carl

Hewitt, criada em 1969. Devido a pouca formalização da linguagem e da falta de conhecimento quanto a linguagem Lisp, somados ao fato deles serem extremamente dedicados a lógica, e este trabalho acarretou numa não contribuição em suas pesquisas futuras.

No ano seguinte, foi um período mais produtivo, pois em fevereiro de 1972, o grupo havia obtido uma bolsa equivalente de vinte mil dólares, oriundo do Institut de Recherche d'Informatique et d'Automatique, uma instituição de pesquisa voltada para área de computadores que era afiliada ao ministério da Indústria francês, por dezoito meses. Eles investem esse aporte na aquisição de um terminal de teletipo com trinta caracteres por seguindo e integram ele ao IBM 360-67 na Universidade de Grenoble, via link dedicado. Além disso, Kowalski obtém da OTAN um financiamento para numerosos intercâmbios entre Edimburgo e Marselha.

Figura 2.0: IBM 360-67

Disponível em: <https://blogs.ncl.ac.uk/cs-history/hardware-ibm-360-67-cabinets/>

Convidam novamente Robert Kowalski, mas por um período de dois meses, entre maio e junho. Juntos, a equipe havia conhecimento computacional suficiente quanto a prova automatizada de teoremas. Tinham experiência na concatenação de listas, adição de inteiros, reversão de listas, entre outros.

Alain finalmente encontra uma maneira de desenvolver poderosos analisadores e os associou à um predicado binário n(a,b), onde cada símbolo não terminal N da gramática, que significaria que a e b são cadeias terminais para quais a cadeia definida a e b por listas, permitindo que cada regra gramatical pudesse ser codificada por uma cláusula com exatamente o mesmo número de literais que prescindiam da concatenação de listas. Além disso, ele introduz parâmetros adicionais em cada não terminal para propagar e computar informações. Neste ponto, nada impedia a criação de um sistema de comunicação homem-máquina inteiramente lógica.

Mais adiante, no outono de 1972, o sistema de teoremas foi batizado de Prolog na qual incorporou a interpretação procedural de R. Kowalski. Ele, por sua vez, afirma que sua contribuição para linguagem foi de maneira filosófica, enquanto Alain e Philippe são quem de fato contribuíram para a concepção de Prolog.

Ainda naquele ano, Philippe implementa o sistema na linguagem Algol, de Niklaus Wirt e, paralelamente, Alain e Robert Pasero criam o tão aguardado sistema de comunicação homem-máquina em francês.

Ainda em 1972, duas outras aplicações foram desenvolvidas utilizando uma versão preliminar de Prolog, que seriam um sistema de computação simbólica, inciado em 1972 e concluído no ano seguinte, assim como um sistema de resolução de problemas, concluído em 1974.

No primeiro semestre de 1973, o então grupo formado é oficialmente reconhecido pelo Centro Nacional de Pesquisa Cientifica - CNRS (Centre national de la recherche Scientifique) como uma equipe de pesquisa associada, na qual tinha como o objetivo a área Diálogo homem-máquina em linguagem natural. Além do reconhecimento, o grupo recebe outro aporte financeiro no valor de seis mil e quinhentos dólares.

Enquanto doutorando, Robert Pasero continuou a utilizar seu trabalho sobre a semântica francesa, o que lhe concedeu uma bem-sucedida tese de doutorado em maio daquele ano.

Sua primeira versão foi desenvolvida em Fortran por Gérard Battani e Henri Meloni, em 1973. Sua principal aplicação é dada na área de computação simbólica, que concerne na lógica matemática, prova de teoremas, solução de equações simbólicas, em bancos de dados relacionais, sistemas especialistas, compiladores e análises bioquímicas.

2.1 PROLOG II – UM NOVO MODELO TEÓRICO

Em 1984 é criada a empresa **PrologIA** para comercializar o Prolog II. Nesta versão melhorada do Prolog, que substitui a noção de unificação pela de resolução de equações em dado domínio, a linguagem introduz na programação lógica o conceito de árvores infinitas e restrições na teoria das árvores racionais, bem como a possibilidade de verificar se dois objetos são desiguais sem a utilização do corte do espaço de busca.

2.2 PROLOG III – A CHEGADA DA PROGRAMAÇÃO DE RESTRIÇÃO

Em meados de 1989 e 1990 a versão III da linguagem integra três novos solucionadores sendo eles as listas, os booleanos e números racionais. É nesta versão que Prolog redefine seu processo fundamental; a unificação. Ele integra a esse mecanismo o processamento refinado de listas e arvores, assim como o processamento de álgebra booleana, bem como o de números de dois valores, o que permite que se tornem tipos específicos de árvores e é possível realizar o processamento exato de sistemas de restrições numéricos tal qual expressos em álgebra linear nas relações de igualdade (=), diferente de (≠) e menor ou igual (≤).

Para lidar com desigualdades lineares, a linguagem recorre a um algoritmo do tipo Simplex que consiste em conhecer uma solução básica inicial e que seja viável, testar a solução se caso for ótima e, por fim, melhorar essa solução a partir de um conjunto de regras.

2.3 PROLOG IV - A APROXIMAÇÃO DE RESTRIÇÕES NÃO LINEARES

Na quarta versão da linguagem, com o custo de aumento dez vezes maior que o mecanismo da segunda versão, Prolog IV oferece uma biblioteca composta por um conjunto de mais de restrições, que conta com listas, booleanos, inteiros e reais, o que permite que ir além das restrições lineares, permitindo que todos os cálculos numéricos tenham precisão garantida.

2.4 DERIVADOS: O SWI, TURBO E O VISUAL PROLOG

O **SWI PROLOG** é a versão gratuita da linguagem e está disponível em versões para Windows e Linux. Essa versão está em conformidade com os padrões além de conter muitas bibliotecas. Sua desvantagem é que ele não possui um depurador de nível visual e é lento.

O **TURBO PROLOG,** lançado em 1997, é uma versão muito rápida da linguagem e possui um depurador elegante. No entanto, so há suporte para MS-DOS. O Mundo Tarskies foi implementado em Turbo Prolog.

O **VISUAL PROLOG** é o sucessor da versão turbo. Visual Prolog é uma linguagem multiparadigma, de alto nível, poderosa e segura. Combina os recursos dos

paradigmas lógico, funcional e orientado a objetos, de maneira elegante e concisa. Visual Prolog foi baseada na lógica do Prolog Original de Alain e teve como objetivo facilitar soluções de problemas complexos.

3 PARADIGMA

Importante ressaltar que Programação em Lógica não um sinônimo de Prolog. Com a Programação em Lógica, há duas notórias classes de programas, as definidas (Horn) e não definidas. Prolog executa apenas programas que contenham cláusulas definidas (Horn). Prolog pertence ao paradigma Lógico Matemático e Declarativo.

3.1 PROLOG VS PROGRAMAÇÃO FUNCIONAL

Al linguagens funcionais são baseadas no conceito de função que recebe argumentos e calcula valor. Prolog, por sua vez, não possui funções. Os predicados são utilizados para provar o teorema, além disso, não computam valor, mas respondem com true ou false. Linguagens funcionais ganharam muita força, enquanto Prolog tornou-se algo mais nichado.

Portanto, é possível definir que Prolog é uma linguagem declarativa, tal qual um programa de alto nível implementado nela descreverá o que o computador deve executar, bem como ter correspondência com a lógica matemática e que não tenha efeitos colaterais, isto é, que haja referenciamento transparente. Logo, Prolog é uma linguagem mais direcionada ao conhecimento e menos a algoritmos.

3.2 PROGRAMAÇÃO RECURSIVA

Fazendo referência a estrutura de dados recursiva, listas em Prolog faz a definição das operações. Logo, o predicado é definido por dupla regra, onde temos em uma lista a regra recursiva, que é composta de uma cabeça e sua cauda, temos que:

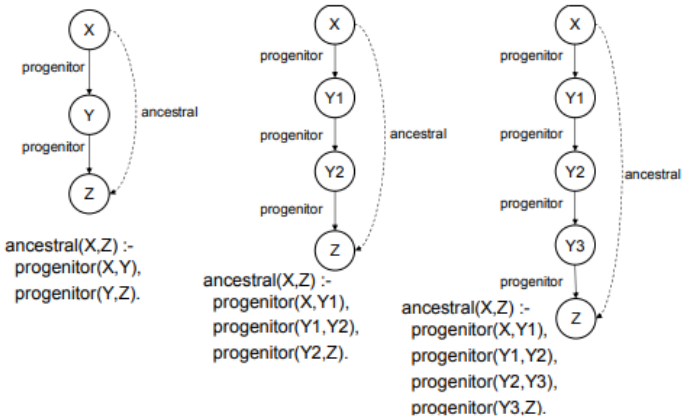
> ***Comprimento da lista [A \| Ab] é 1 somado ao comprimento da cauda Ab;***
> ***comprimento(L, T): - L = [A \|Ab], comprimento(Ab, T1), T=1+T1.***

Bem como sua regra base, que definirá quando a recursividade é finalizada devido a uma lista vazia, na qual:

Comprimento da cauda [ ] é 0.
comprimento(L,T): - L = [ ], T=0.

O exemplo abaixo demonstra que todo X e Z, X é um ancestral de Z se há algum Y tal que X é um progenitor de Y e Y é um ancestral de Z.

Figura 3.0: Ilustração de recursividade



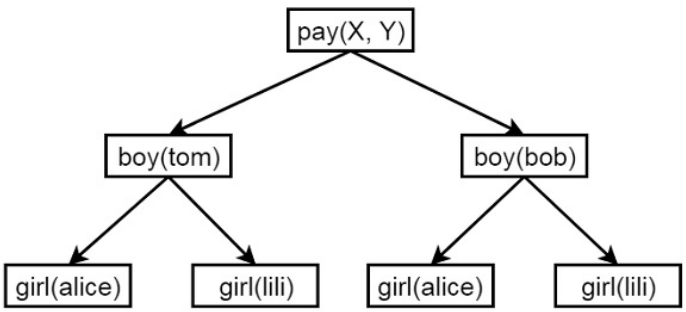
Disponível em: <https://dcm.ffclrp.usp.br/~augusto/teaching/ia/IA-Prolog-Introducao-Tutorial.pdf>

4 CARACTERÍSTICA MARCANTE DA LINGUAGEM

4.1 BACKTRACKING

O termo, cunhado pelo matemático Derrick Henry Lehmer, backtracking é um refinado algoritmo de força bruta, cujo objetivo é eliminar múltiplas soluções sem que essas sejam explicitamente examinadas pela linguagem e, quando necessário, Prolog efetuará o backtracking sempre que for preciso para satisfazer uma cláusula.

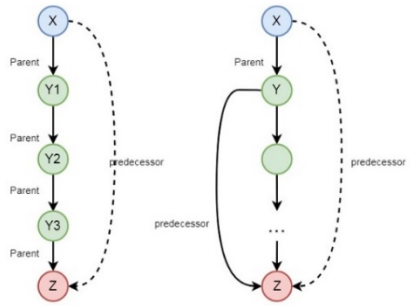
Figura 4.0: Ilustração de retrocesso(backtracking)



Disponível em: < https://www.tutorialspoint.com/prolog/prolog_backtracking.htm >

Mas há uma desvantagem de se utilizar deste artificio; se não forem programadas “constraints” (restrições), sempre irá realizar uma busca exaustiva e tenderá a uma explosão combinatória, já que programas backtracking são combinatórios, além de gastar muita memória já no stack devido ao uso de sucessivas chamadas recursivas, o que faz com que um programa com essas características cresça exponencialmente conforme o tamanho do problema.

Figura 4.1: Ilustração de recursão



Disponível em: < https://www.tutorialspoint.com/prolog/prolog_recursion_and_structures.htm>

5 LINGUAGENS SIMILARES

É possível citar LISP, que pertence ao paradigma funcional, enquanto Prolog ao Lógico. Comparando-as, temos o que cada uma oferece:

PROLOG

- Manipulação de vinculação variável por meio da unificação
- Regras e fatos na estruturação de um banco de dados
- Padrão, busca de controle direcionado, recursão

LISP:

- Retornos de função, passagem de parâmetro por valor, vinculação de variáveis.
- Avaliação de função, avaliação condicional, recursão e loop
- Funciona em um ambiente global, deixe blocos

6 CONSIDERAÇÕES FINAIS

Não é possível negar que graças a linguagem Prolog contribuiu para grandes avanços no âmbito computacional, tendo Alain Colmerauer sido uma grande influência para as próximas gerações no ramo da ciência da computação. A partir das experiências adquiridas com o grupo de pesquisa no desenvolvimento do processamento de linguagens natural, tivemos o primeiro interpretador para o francês. Importante também foi Robert Kowalski que trouxe uma maneira procedural para o processamento das cláusulas de Horn. Além desses marcos na história da computação, no estudo realizado sobre a linguagem é possível dizer que, independentemente de suas várias versões, é uma ótima opção para o uso acadêmico, de pesquisa, bem como para Inteligência Artificial.

7 ALGORITMOS IMPLEMENTADOS NA LINGUAGEM

Bubble Sort

```
bubblesort(L,Sorted) :-
        swap(L,L1), !,
         bubblesort(L1,Sorted).
bubblesort(L,L).

swap([X,Y|T],[Y,X|T]) :-
        gt(X,Y).
swap([Z|T],[Z|T1]) :-
        swap(T,T1).
```

Quick Sort

```
split(H, [A|X], [A|Y], Z) :-
  order(A, H), split(H, X, Y, Z).
split(H, [A|X], Y, [A|Z]) :-
  not(order(A, H)), split(H, X, Y, Z).
split(_, [], [], []).
quicksort([], X, X).
quicksort([H|T], S, X) :-
  split(H, T, A, B),
  quicksort(A, S, [H|Y]),
  quicksort(B, Y, X).
```

Merge Sort

```
mergesort([],[]) :- !.
mergesort([X],[X]) :- !.
mergesort(L,Sorted) :-
        divide(L,L1,L2),
        mergesort(L1,Sorted1),
        mergesort(L2,Sorted2),
        intercala(Sorted1,Sorted2,Sorted).
```

Torre de Hanoi

```
hanoi(N) :- move(N, left, center, right).
move(0, _, _, _) :- !.
move(N, A, B, C) :-
  M is N-1,
  move(M, A, C, B), inform(A, B), move(M, C, B, A).
inform(X, Y) :-
  write('move a disc from the '),write(X), write(' pole to the '), write(Y), write(' pole'),  nl.
```

Árvore Binária

```
istree(nil).
istree(t(_,L,R)) :- istree(L), istree(R).

tree(1,t(a,t(b,t(d,nil,nil),t(e,nil,nil)),t(c,nil,t(f,t(g,nil,nil),nil)))).
tree(2,t(a,nil,nil)).
tree(3,nil).
```

Hello World

```
?- write('Hello World!'), nl.
Hello World!
true.

?-
```

Completeza de Turing

```
turing(Tape0, Tape) :-
    perform(q0, [], Ls, Tape0, Rs),
    reverse(Ls, Ls1),
    append(Ls1, Rs, Tape).

perform(qf, Ls, Ls, Rs, Rs) :- !.
perform(Q0, Ls0, Ls, Rs0, Rs) :-
    symbol(Rs0, Sym, RsRest),
    once(rule(Q0, Sym, Q1, NewSym, Action)),
    action(Action, Ls0, Ls1, [NewSym|RsRest], Rs1),
    perform(Q1, Ls1, Ls, Rs1, Rs).

symbol([], b, []).
symbol([Sym|Rs], Sym, Rs).

action(left, Ls0, Ls, Rs0, Rs) :- left(Ls0, Ls, Rs0, Rs).
action(stay, Ls, Ls, Rs, Rs).
action(right, Ls0, [Sym|Ls0], [Sym|Rs], Rs).

left([], [], Rs0, [b|Rs0]).
left([L|Ls], Ls, Rs, [L|Rs]).
```

8 BIBLIOGRAFIA

BRATKO, Ivan. **Prolog Programming for Artificial Intelligence - (2nd edition).**
Addison-Wesley, 1993.

STERLING, L. & **SHAPHIRO**, E. **The Art of Prolog: advanced programming techniques - (2nd edition).** MIT Press, 1994.

HOGGER, Christopher John. **Essentials of Logic Programming.**
Oxford University Press, 1990.

TOWNSEND, Carl. **Advanced techniques in Turbo Prolog**
Sybex, 1987.

KOWALSKI, Robert A. **Lógica, Programación e Inteligencia Artificial**
Díaz de Santos, 1986.

CLOCKSIN, William. **Clause, and Effect. Prolog Programming for the Working Programmer.**
Springer-Verlag, 1997.

AMBLE, Tore**. Logic Programming and Knowledge Engineering**
Ed. Addison-Wesley, 1987.

ZHANG, J. and Grant. **An Automatic Diference-list Transformation Algorithm for Prolog**, P. W., 1988.

Grammar Rules in Prolog. **Yorku**, 2011. Disponível em: <https://www.eecs.yorku.ca/course_archive/2011-12/F/3401/slides/22a Grammars.pdf>. Acesso em: 04 de outubro. de 2022.

Curriculum Of Alain Colmerauer. **Alain Colmerauer**, 2007. Disponível em: < http://alain.colmerauer.free.fr/alcol/ArchivesPublications/CurriculumVitae/Cve06a4.pdf>. Acesso em: 02 de outubro. de 2022.

Prolog and Logic Programming Historical Sources Archive. **Paul McJones**, 2022. Disponível em: <https://www.softwarepreservation.org/projects/prolog/>. Acesso em: 03 de outubro. de 2022.

The Birth of Prolog. **Alain Colmerauer & Philippe Roussel**, 1993. Disponível em: <https://dl.acm.org/doi/10.1145/155360.155362>. Acesso em: 04 de outubro. de 2022.

Reference Manual. **SWI Prolog**.
Disponível em: <https://www.swi-prolog.org/pldoc/doc_for?object=root>. Acesso em: 04 de outubro. de 2022.

Introdução à Linguagem Prolog. **Silvio Lago**. Disponível em: < https://www.ime.usp.br/~slago/slago-prolog.pdf>. Acesso em: 04 de outubro. de 2022.

O método Simplex. **Volmir Eugênio Wilhelm**. Disponível em: < https://docs.ufpr.br/~volmir/PO_I/DOC_simplex_algebrico.pdf>. Acesso em: 05 de outubro. de 2022.

Backtracking. **Aldo von Wangenheim**. Disponível em: < https://docs.ufpr.br/~volmir/PO_I/DOC_simplex_algebrico.pdf>. Acesso em: 05 de outubro. de 2022.

Visual Prolog. **Volmir Eugênio Wilhelm**. Disponível em: < https://www.visual-prolog.com/>. Acesso em: 05 de outubro. de 2022.