UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS GUARAPUAVA
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JOÃO GUILHERME FACCIN

VISUALIZAÇÃO INTERATIVA PARA O ENSINO DE PROGRAMAÇÃO LÓGICA EM PROLOG

TRABALHO DE CONCLUSÃO DE CURSO

GUARAPUAVA
2013

JOÃO GUILHERME FACCIN

VISUALIZAÇÃO INTERATIVA PARA O ENSINO DE PROGRAMAÇÃO LÓGICA EM PROLOG

Trabalho de Conclusão de Curso apresentado à disciplina de Trabalho de Conclusão de Curso II do curso de Tecnologia em Sistemas para Internet, Câmpus Guarapuava, da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Tecnólogo em Sistemas para Internet.

Áreas de concentração: Ensino de Programação de Computadores, Inteligência Computacional.

Orientador: Prof. Dr. Eleandro Maschio Krynski

GUARAPUAVA

2013

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Guarapuava
Curso Superior de Tecnologia em Sistemas Para a Internet

ATA DE DEFESA DE MONOGRAFIA DE TRABALHO DE CONCLUSÃO DE CURSO DO CURSO DE TSI

No dia 04 de dezembro de 2013, às 08:00 horas, nas dependências da Universidade Tecnológica Federal do Paraná Câmpus Guarapuava, ocorreu a banca de **defesa da monografia** de Trabalho de Conclusão de Curso intitulada: "**Visualização Interativa para o Ensino de Programação Lógica em Prolog**" do acadêmico **João Guilherme Faccin** sob orientação do professor **Prof. Dr. Eleandro Maschio** do Curso de Tecnologia em Sistemas para Internet.

Banca Avaliadora	
Membro	**Nome**
Orientador	Prof. Dr. Eleandro Maschio
Coorientador	
Avaliador 1	Prof. Me. Andres Jessé Porfirio
Avaliador 2	Prof. Me. Diego Marczal

Situação do Trabalho	
Situação	( ) Aprovado (x) Aprovado com ressalvas ( ) Reprovado ( ) Não Compareceu
Encaminhamento do trabalho para biblioteca	(x) Pode ser encaminhado para biblioteca. ( ) Manter sigilo para publicação ou geração de patente.

Guarapuava, 4 de dezembro de 2013.

A Folha de Aprovação assinada encontra-se na Coordenação do Curso (ou Programa).

AGRADECIMENTOS

Primeiramente agradeço ao meu orientador Prof. Dr. Eleandro Maschio Krynski, pelo apoio e paciência, especialmente nos meus momentos de ausência.

Aos meus pais, irmão e irmã que sempre me apoiaram e incentivaram nos momentos de dificuldade, sem vocês eu nada seria. Um agradecimento especial aos meus amigos, pelo apoio e incentivo durante os anos passados longe de casa. Aos que estavam longe, Artur, Rafael, Guilherme e Gustavo. Aos que aqui encontrei, David, Jonas, Rehnan, Karine.

Aos colegas de turma pelo apoio e pelas risadas proporcionadas. Ainda, um agradecimento especial àqueles que proporcionaram a oportunidade para que meus estudos se desenvolvessem, meus tios Valdir e Luciana. E a todos aqueles que colaboraram direta ou indiretamente e que de alguma forma torcem por mim.

Muito obrigado.

“Não é preciso ter olhos abertos para ver o sol,

nem é preciso ter ouvidos afiados para ouvir o trovão.

Para ser vitorioso você precisa ver o que não está visível.”

Sun Tzu

RESUMO

FACCIN, João G. Visualização Interativa para o Ensino de Programação Lógica em Prolog. 2013. Trabalho de Conclusão de Curso (Graduação em Tecnologia em Sistemas para Internet), Universidade Tecnológica Federal do Paraná. Guarapuava, 2013.

A presente pesquisa incide sobre a dificuldade enfrentada por aprendizes de linguagens baseadas no paradigma lógico de programação, voltando-se especificamente ao aprendizado da linguagem Prolog. A partir de uma introdução ao Prolog discutem-se conceitos como retroação e recursividade, bem como a utilização de representações externas como auxílio didático. A solução proposta busca amparo nas áreas de Visualização de Algoritmos e de Ambientes Interativos de Aprendizagem. A fim de atenuar as dificuldades que compõem a problemática da pesquisa, propõe-se a utilização de múltiplas representações externas implementadas em um ambiente interativo de aprendizagem.

Palavras-chave: Ensino de Programação de Computadores. Programação Lógica. Prolog. Múltiplas Representações Externas. Ambientes Interativos de Aprendizagem.

ABSTRACT

FACCIN, João G. Interactive Visualization for Teaching Logic Programming in Prolog. 2013. Trabalho de Conclusão de Curso (Graduação em Tecnologia em Sistemas para Internet), Universidade Tecnológica Federal do Paraná. Guarapuava, 2013.

This research focuses on the difficulties faced by learners of languages based on logic programming paradigm, turning specifically to the Prolog language learning. From an introduction to Prolog are discussed concepts like recursion and retroaction as well as the use of external representations as teaching aid. The proposed solution seeks support in the areas of Algorithms Visualization and Interactive Learning Environments. In order to mitigate the difficulties that make up the research problem, it proposes the use of multiple external representations implemented in an interactive learning environment.

Palavras-chave: Computer Programming Teaching. Logic Programming. Prolog. Multiple External Representation. Interactive Learning Environments.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A compreensão do modo e da lógica de funcionamento das coisas sempre foi buscado nos mais diversos meios, e não é diferente quando, em Ciência da Computação, se aborda a linguagem Prolog (*Programming Logic*).

Em Ciência da Computação os iniciantes no contato com o paradigma de programação lógico, além de buscar conhecimentos específicos da linguagem, como a sintaxe, por exemplo, devem dominar conceitos necessários para a aprendizagem de qualquer nova linguagem de programação, como algoritmos, estruturas de dados, entre outros. Ainda devem se aprofundar em temas como lógica de predicados, de primeira ordem, inferência lógica, *backtracking* (retroação) e recursividade. Sendo os dois últimos tópicos de difícil compreensão, porém, como no caso do conceito de recursão, de grande valia para qualquer profissional da área de Programação de Computadores.

Como a linguagem Prolog faz extenso uso desses recursos, é natural que surjam diversas ferramentas a fim de auxiliar e facilitar a compreensão dos conceitos correlatos. Muitas podem ser citadas, desde o início da linguagem, como a *Spy*, *Prolog Trace Package*, *Transparent Prolog Machine*, *Textual Tree Tracer* (MULHOLLAND, 1997), até as mais atuais como o diagrama *Logichart* (ADACHI, 2009).

Entretanto, ao se observar as diferentes representações propostas pelas referidas ferramentas, percebe-se que ainda há espaço de estudo, aprimoramento e desenvolvimento de um conjunto com múltiplas representações destinado ao ensino do Paradigma Lógico na linguagem Prolog. A proposta torna-se ainda mais pertinente quando associada a possibilidades adequadas de interação em uma ferramenta como um ambiente interativo de aprendizagem.

1.1 Objetivos

1.1.1 Geral

Dado o exposto, teve-se como objetivo geral desta pesquisa estudar e desenvolver um conjunto de múltiplas representações externas, instanciado em um ambiente interativo de aprendizagem, que permitisse a visualização e o entendimento pleno do processo de inferência da linguagem Prolog. Espera-se, com isto, auxiliar no entendimento de conceitos fundamentais e consequente desenvolvimento de perícia relacionada ao Paradigma Lógico. Demarcam-se como conceitos abordados: a retroação e a recursão.

1.1.2 Específicos

Foram, portanto, objetivos específicos:

1. Construir embasamento teórico sobre os temas de Múltiplas Representações Externas, Ambientes Interativos de Aprendizagem e Visualização de Algoritmos. Tratam-se de tópicos relevantes que associam as áreas de Inteligência Computacional e Ensino de Programação de Computadores. Ademais, não são abordados pela Matriz Curricular do curso de Tecnologia em Sistemas para Internet;
2. Definir um conjunto de múltiplas representações externas correlato à aquisição de conceitos e ao desenvolvimento de perícia na linguagem Prolog (remetendo-se fortemente aos princípios e técnicas do Paradigma Lógico de programação);
3. Desenvolver um protótipo de ambiente interativo de aprendizagem que auxilie no ensino da referida linguagem enfatizando conceitos importantes como a recursividade e a retroação.

1.2 Justificativa

Segundo Van Someren (1990), várias são as dificuldades encontradas por um programador ou estudante da área de Ciência da Computação ao iniciar o contato com uma linguagem declarativa, cujo foco principal reside no que deve ser feito, ao invés de instruções minuciosas de como se fazer. Tais dificuldades acabam sendo ainda mais específicas quando este contato se dá com linguagens declarativas baseadas no Paradigma Lógico de Programação, o qual se fundamenta em fatos e suas respectivas relações. Neste contexto, a linguagem Prolog passa a possuir destaque.

No que diz respeito ao Prolog, a maior parte da dificuldade encontra-se em conceitos inerentes ao próprio Paradigma Lógico, como retroação e unificação. Existem ainda aspectos relacionados à natureza recursiva com que o Prolog executa as tarefas, consistindo em um quesito de extrema importância para o desenvolvimento de perícia na linguagem. Além disto, o benefício da compreensão pode ser estendido e utilizado para consolidar a experiência em, praticamente, qualquer linguagem de programação.

Neste contexto, cabe destacar que o público-alvo desta proposta não se limita aos interessados na linguagem Prolog. Abrange, adicionalmente, aprendizes que buscam aprimoramento na compreensão dos conceitos de retroação e, principalmente, de recursão.

Contudo, poucos são os recursos didáticos oferecidos no sentido de permitir a visualização das características citadas. Exemplo disso são algumas representações abordadas no Capítulo 3 que, embora bastante embasadas teoricamente, mostram-se confusas na prática e oferecem limitadas opções de interação por parte do aprendiz.

1.3 Problemática do trabalho

O estudo proposto pela corrente pesquisa objetiva atenuar a dificuldade de assimilação de conceitos do Paradigma Lógico através de múltiplas representações

externas instanciadas em um ambiente interativo de aprendizagem. Adicionalmente será abordado o desenvolvimento de perícia no que se refere ao processo de inferência e ao mecanismo de prova da linguagem Prolog.

Neste sentido, foram estudados aspectos teóricos de múltiplas representações externas. Depois, se revisou a utilização destas representações em ambientes de ensino da área de programação (especialmente lógica).

O enfoque principal da pesquisa reside em atentar-se à perspectiva de visualização de algoritmos e em atingir uma representação externa adequada para denotar programas em Prolog. Com isto, pretende-se possibilitar oportunidades de aprendizado produtivas no ambiente interativo de aprendizagem prototipado pela pesquisa.

A maior dificuldade verificada residiu na forma de representar a máquina de inferência da linguagem Prolog. Em consequência, tratou-se o extenso uso de modelos recursivos exigidos para a compreensão dos assuntos abordados.

Por fim, pretendeu-se alcançar uma interação que permita ao aprendiz explorar a riqueza semântica oferecida pelo conjunto de representações externas envolvidos. A interface entre o aprendiz e as representações consiste no protótipo de ambiente interativo de aprendizagem implementado.

Como resultado subjacente, buscou-se conhecimento teórico e prático em áreas não contempladas pela Matriz Curricular do curso de Tecnologia em Sistemas para Internet.

2 REFERENCIAL TEÓRICO

2.1 Prolog

A Programação Lógica, por fazer uso de lógica simbólica como linguagem, difere das demais abordagens dos paradigmas essencialmente procedimentais. Destaca-se principalmente sua essência declarativa, cujo foco é a especificação dos resultados esperados. Em contrapartida aos paradigmas procedimentais, onde o que se informa é o passo a passo detalhado de como alcançar um resultado.

A capacidade de realizar um processamento baseando-se apenas na especificação de um resultado acontece quando se provê ao computador informações relevantes e um mecanismo de inferência para computar os resultados esperados (SEBESTA, 2012). Neste contexto, surge o Prolog, uma linguagem de programação lógica utilizada em uma gama de aplicações, que vão desde bancos de dados relacionais a aquelas relacionadas à inteligência artificial (CLOCKSIN; MELLISH, 2003).

2.1.1 Contextualização histórica

Surgido em meados da década de 1970, o Prolog (do inglês, *Programming Logic*), teve seu projeto fundamental desenvolvido por Alan Colmerauer e Phillippe Roussel, da Universidade de Aix-Marseille, em conjunto com Robert Kowalski, da Universidade de Edimburgo. O primeiro interpretador Prolog foi desenvolvido em 1972, em Marseille.

Colmerauer e Roussel estavam interessados na utilização do Prolog para o processamento de linguagem natural, enquanto Kowalski, para a demonstração automatizada de teoremas. Este fato contribuiu para que, ao fim da cooperação entre as duas universidades, surgisse um grande número de dialetos distintos da mesma linguagem. Alguns oriundos do grupo da Universidade de Aix-Marseille[1], outros da Universidade de Edimburgo[2].

1 http://www.univ-amu.fr/
2 http://www.ed.ac.uk/home

A principal diferença entre esses dialetos consiste em sua forma sintática, porém, os componentes principais do Prolog permanecem. São eles: um método para especificar proposições de cálculo de predicado, associado à implementação de uma forma de resolução ou inferência.

2.1.2 Princípios do paradigma e visão geral da linguagem

Linguagens de programação lógica baseiam-se em declarações, e não em atribuições e instruções de fluxo de controle (SEBESTA, 2012). Assim é o Prolog, uma linguagem baseada na declaração de objetos e nas relações entre eles.

Clocksin e Mellish (2003), afirmam que a escrita de um programa em Prolog consiste em três etapas principais:

1. A especificação de fatos sobre os objetos e suas relações;
2. A definição de regras sobre estes objetos e relações; e
3. Questionamentos sobre os mesmos.

Ainda, um programa Prolog pode ser visto como uma base axiomática composta de fatos e regras (conceitos abordados na sequência). É um exemplo de uma instrução de fato:

```
pai(joao, ana).
```

que afirma que `joao` é pai de `ana`.

Um exemplo de instrução de regra seria o equivalente a:

```
avo(X, Z) :- genitor(X, Y), genitor(Y, Z).
```

que declara que, para determinados valores de `X`, `Y` e `Z`, `X` é avô de `Z` se `X` for genitor de `Y`, e `Y` for genitor de `Z`.

Uma consulta a esta base axiomática pode ser feita através de uma instrução de meta, buscando um retorno verdadeiro ou falso, ou por meio de uma instrução que busque uma atribuição válida a uma variável.

(1) `pai(joao, carlos).`

(2) `pai(X, joao).`

As instruções acima ilustram (1) uma consulta buscando um retorno quanto à validade do fato de `joao` ser pai de `carlos` e (2) uma atribuição à variável `X` que indique um possível pai de `joao`.

É possível perceber a simplicidade fornecida pela semântica declarativa em relação às instruções das linguagens imperativas, onde se deve considerar o contexto atual ou as sequências de execução. Trata-se, portanto, de uma das vantagens que as linguagens declarativas possuem sobre as imperativas (SEBESTA, 2012 *apud* HOGGER, 1984, p. 240-241).

Cabe reiterar que a programação lógica não é baseada em procedimentos que detalham como um resultado deve ser alcançado. Ela fundamenta-se na declaração da forma deste resultado, presumindo, para isso, que o computador possa determinar como o resultado deve ser obtido. Isto é feito a partir do fornecimento de informações relevantes juntamente a um mecanismo de inferência para o processamento dos resultados. O cálculo de predicados provê a comunicação ao computador, e o método de prova fornece a técnica de inferência (SEBESTA, 2012).

No decorrer do texto, a fundamentação do Paradigma Lógico sobre o Cálculo de Predicados será oportunamente evidenciada.

2.1.3 Fatos

A fim de que seja possível definir um fato, torna-se relevante o conhecimento sobre conceitos básicos de proposições do Cálculo de Predicados. Como define Sebesta (2012, p. 729), “uma proposição pode ser imaginada como uma declaração lógica que pode ser verdadeira ou não. Ela consiste em objetos e nas relações de objetos entre si”.

Estas proposições podem ser classificadas em proposições simples, também conhecidas por proposições atômicas, ou proposições compostas. Uma proposição atômica é composta basicamente por um functor, a constante que define a relação, e uma lista de parâmetros, também chamada de *n*-tupla, onde *n* é o

equivalente à quantidade de parâmetros existentes.

Em Prolog podemos definir um fato, portanto, como uma proposição considerada verdadeira, e que segue a mesma estrutura de uma proposição atômica do cálculo de predicados. São exemplos de fatos:

```
homem(joao).

irmao(joao, carlos).

mae(claudia, carlos).
```

Aqui, `homem`, `irmao` e `mae` fazem o papel de functor, sendo complementados por seus respectivos parâmetros.

2.1.4 Regras

O outro modelo de estrutura que compõe a base axiomática do Prolog é conhecido por regra. Regras são proposições do Cálculo de Predicados escritas em uma forma padrão, denominada forma clausal ou, cláusulas de Horn[1]. Este padrão tem por função minimizar redundâncias que possam ocorrer na declaração de proposições lógicas. Ainda, uma cláusula de Horn pode ser considerada um teorema do qual uma conclusão pode ser inferida desde que uma série de condições dadas seja satisfeita.

Uma regra possui então, duas partes, chamadas de antecedente e consequente. A parte antecedente consiste no lado direito da proposição, e é composta pelo termo ou conjunto de termos que, caso satisfeitos, farão com que a parte consequente (lado esquerdo da proposição) torne-se verdadeira. Podemos ainda fazer uma analogia aos termos *if* e *then*, presentes em grande parte das linguagens de programação. A parte antecedente seria equivalente ao *if*, a consequente, ao *then*.

Antecedentes que contenham mais de um termo são consideradas conjunções, separadas por um operador lógico *AND* implícito representado como uma vírgula. Regras que possuam um mesmo consequente, escritas

1 Um apanhado sobre as Cláusulas de Horn pode ser encontrado em: http://en.wikipedia.org/wiki/Horn_clause

separadamente, passam a ser disjunções (*OR*) lógicas.

São exemplos de regras:

```
irmao(X, Y) :-
          mae(Z, X),
          mae(Z, Y),
          homem(X),
          not(X = Y).
genitor(X, Y) :- mae(X, Y).
genitor(X, Y) :- pai(X, Y).
```

Onde a primeira define que `X` só será irmão de `Y` caso ambos possuam a mesma mãe, `X` seja homem e `X` seja diferente de `Y`. As duas regras seguintes definem que `X` será genitor de `Y` caso aquele seja mãe ou pai deste.

2.1.5 Metas, variáveis, unificação e instanciação

Pode-se atribuir ao Prolog a denominação de linguagem conversacional, visto que o modo interativo de uso remete-se a uma espécie de diálogo. Utiliza-se o teclado como método de entrada para fatos, regras e questionamentos, enquanto respostas são retornadas em tela, textualmente.

As respostas originam-se a partir de uma proposição informada pelo usuário, o qual deseja que a mesma seja provada ou refutada pelo Prolog. Estas proposições são conhecidas por metas e possuem a mesma estrutura de declaração de um fato. Uma meta pode ser, ainda, um fato composto por várias conjunções. Uma forma comum de meta é aquela que possui uma ou mais variáveis, as quais o usuário deseja que seja atribuído um valor, que torne a meta verdadeira. Uma variável pode ser identificada por ser um termo iniciado sempre por uma letra maiúscula.

De acordo com Sebesta (2012, p. 733), “este processo de determinação de valores úteis é chamado unificação. A atribuição temporária de valores às variáveis

para permitir a unificação é chamada instanciação". Ao unificar valores às variáveis informadas, o Prolog retorna-os ao usuário e aguarda uma instrução. Caso o usuário esteja satisfeito com a resposta obtida, é possível encerrar a busca. Do contrário, pode-se aguardar o retorno de outros resultados, ou mesmo interromper a busca.

As respostas a uma consulta no Prolog podem ser de três tipos básicos: verdadeiro, falso, ou um valor unificado a uma variável. Cabe destacar que os resultados obtidos pelo programa Prolog adotam a Presunção do Mundo Fechado (*Closed World Assumption*), ou seja, ao definir uma consulta como falsa, deve-se ter em mente que a resposta é fornecida com base apenas nos fatos providos pela base axiomática existente. Não possuindo, portanto, conhecimento relativo ao mundo fora de sua base axiomática. Em contraste a um sistema verdadeiro/falso, o Prolog passa a ser um sistema verdadeiro/falha (SEBESTA, 2012, p. 754).

2.1.6 Mecanismo de inferência e casamento de padrões (*matching*)

Uma meta pode ser composta por mais de uma preposição a ser provada ou refutada. Cada uma dessas preposições é conhecida por submeta, e tem papel fundamental dentro do mecanismo de inferência do Prolog. Para uma meta ser verdadeira, portanto, todas as suas submetas também devem ser verdadeiras.

Na busca por provar uma meta, o mecanismo de inferência tenta encontrar regras ou fatos que conectem tal meta à base axiomática. Isto é feito através de um processo de casamento de preposições, também conhecido por *matching* (SEBESTA, 2012). É importante destacar que o mecanismo de inferência trabalha através de busca em profundidade (*depth-first*) (RUSSEL; NORVIG, 2004, p. 76), ou seja, busca uma solução completa para a primeira submeta antes de trabalhar na submeta seguinte.

Ainda, a eficiência de um programa Prolog específico pode ser consideravelmente afetada pela ordenação adequada de fatos e regras. O mecanismo de inferência realiza o processo de *matching* sempre partindo do início da base axiomática e da esquerda para a direita de uma meta. O programador pode posicionar regras que possuam maiores chances de serem satisfeitas no início da base axiomática, a fim de otimizar o tempo de resposta de seu programa.

2.1.7 Retroação (*backtracking*)

Durante o processo de prova de uma meta que contenha várias submetas o mecanismo de inferência pode falhar ao satisfazer uma delas. O mecanismo, então, abandonará esta submeta e retornará à sua antecedente (caso exista), a fim de encontrar outra solução válida. Tal processo de reconsideração de uma submeta já satisfeita é conhecido por retroação, ou *backtracking*.

O processo de retroação acontece iniciando a busca por uma solução alternativa partindo de onde se tenha previamente parado. Tal processo pode se repetir até a submeta inicial e, caso esta não possa ser provada com uma solução alternativa, toda a meta falhará.

Com a finalidade de exemplificar este processo, considera-se o questionamento `genitor(joao, ana).` feito à seguinte base axiomática:

```
genitor(X, Y) :- mae(X, Y).

genitor(X, Y) :- pai(X, Y).

mae(claudia, joao).

pai(joao, carlos).

pai(joao, ana).
```

Ao encontrar a primeira regra que realize o *matching* com meta a ser provada (`genitor(X, Y) :- mae(X, Y).`), o mecanismo busca satisfazê-la. Não sendo possível, é realizado o *backtracking* ao nível anterior, buscando novamente um fato ou regra que realize a unificação com a meta definida, neste caso, `genitor(X, Y) :- pai(X, Y).` Novamente o mecanismo de inferência busca a prova da submeta. É encontrado um fato que unifica com o predicado, porém, não com suas variáveis. Assim, realiza-se mais uma retroação, encontra-se outro fato, desta vez válido, e a prova da meta inicial é realizada. Uma representação gráfica do exemplo pode ser acompanhada através das Figuras 1 e 2.

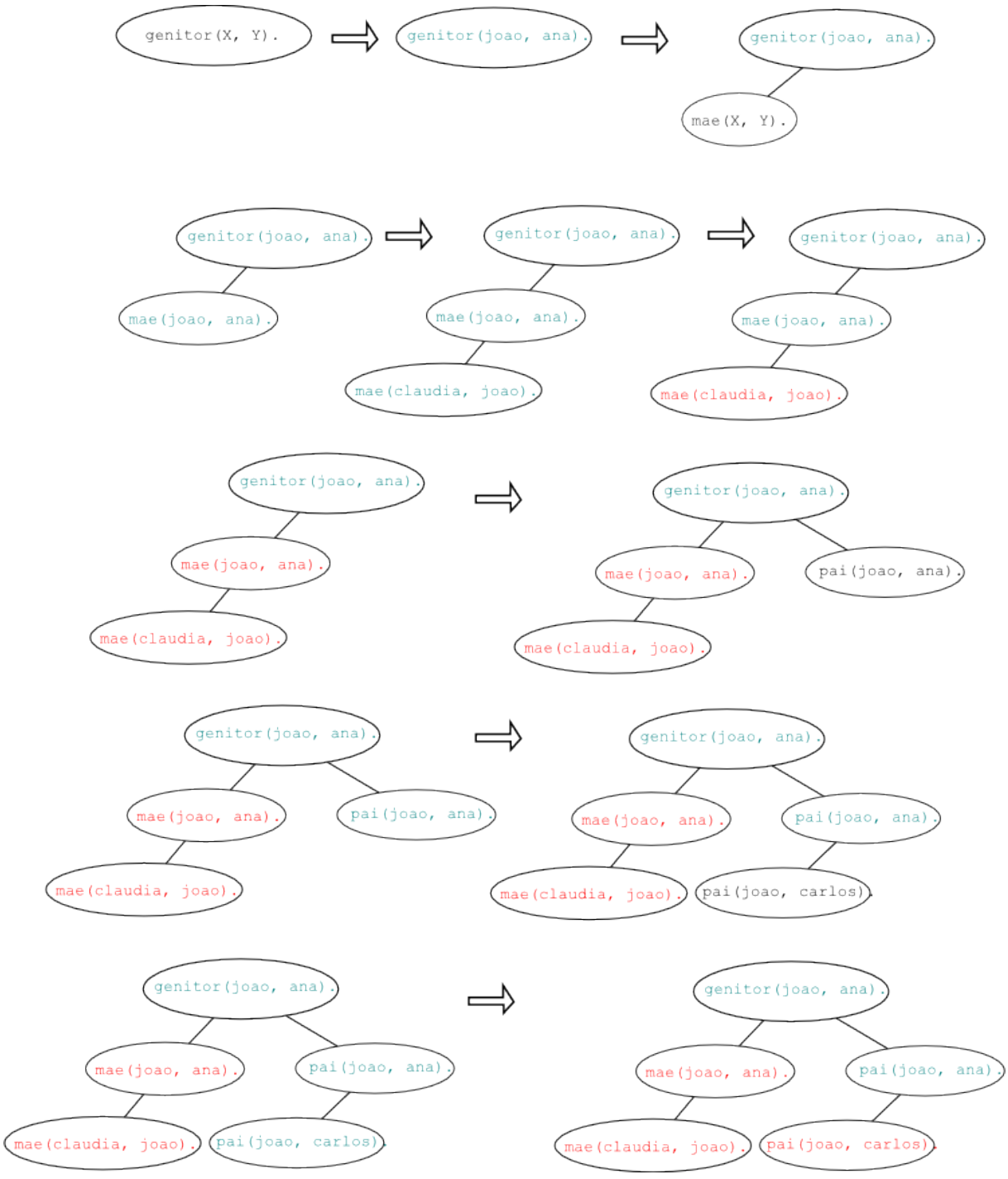

Figura 1 – Exemplo do processo de retroação realizado pelo mecanismo de inferência da linguagem Prolog.

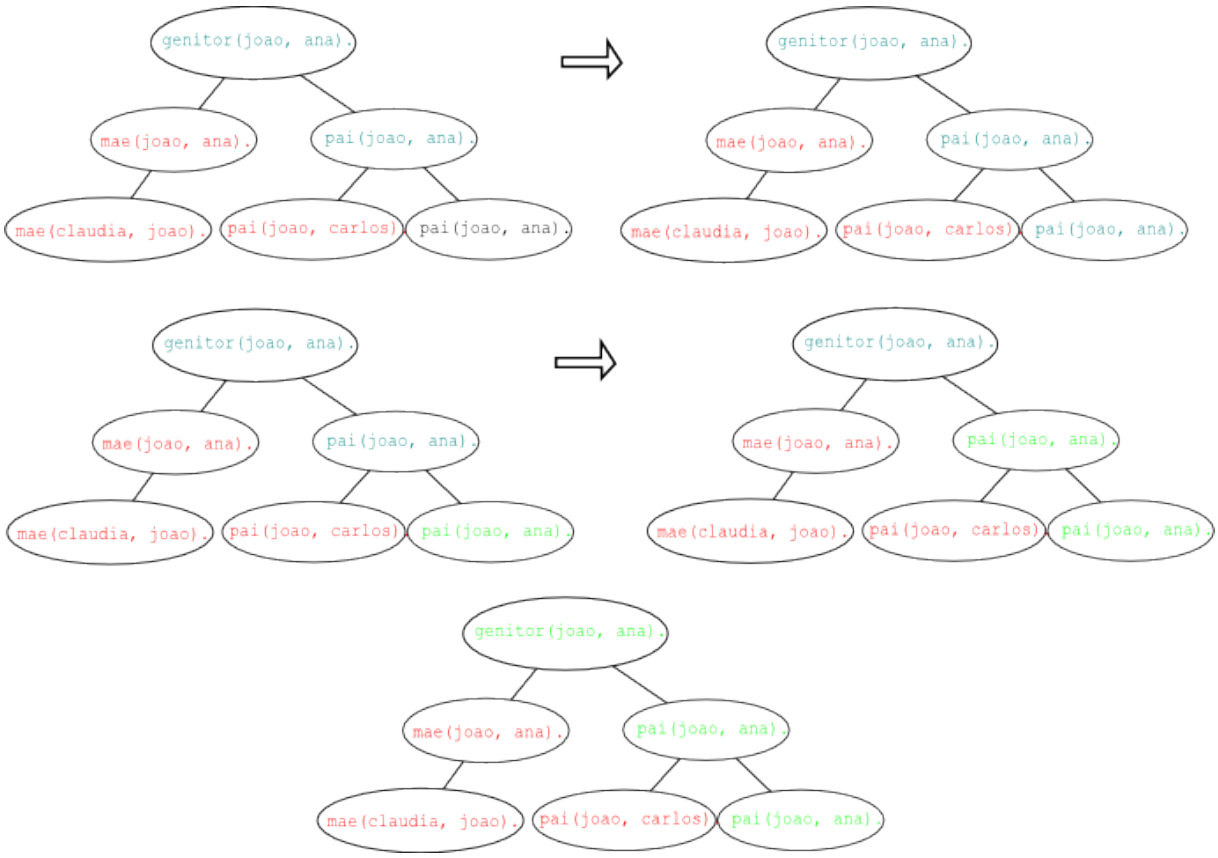

Figura 2 – Exemplo do processo de retroação realizado pelo mecanismo de inferência da linguagem Prolog.

2.1.8 Operadores lógicos

Conforme anteriormente mencionado na Seção 5.4, o Prolog possui em sua sintaxe operadores lógicos declarados de forma implícita. Proposições que, em seu lado direito, possuam mais de um termo são consideradas conjunções, tendo o operador lógico *AND* representado implicitamente por uma vírgula. Em contrapartida, proposições que possuam o mesmo termo em seu lado esquerdo, porém escritas separadamente, denotam uma operação de disjunção (OR).

O Prolog ainda provê o operador *NOT*, que não deve ser confundido com o operador lógico *NOT*. Enquanto um denota que seus operandos são prováveis verdades, o outro, para alcançar sucesso, especifica que seus operandos não podem ser provados.

Esta definição, também conhecida por “problema da negação” pode levar à uma certa confusão e ocasionar erros em uma programação mal-estruturada. Um

exemplo disso é o aninhamento de operadores *NOT* em uma proposição, que pode resultar em um retorno de variáveis não instanciadas ao usuário, como no exemplo:

```
not(not(irmao(X, carlos))).
```

2.1.9 Operadores aritméticos

O Prolog suporta o uso de expressões aritméticas simples em suas proposições. Estas expressões são definidas através da utilização do operador `is`, e permitem tanto a utilização de valores inteiros quanto variáveis.

O uso do operador `is` é realizado de maneira que exista uma variável em seu lado esquerdo e uma expressão aritmética em seu lado direito. Uma prerrogativa para este operador é de que todas as variáveis existentes no lado direito estejam instanciadas no momento de sua utilização, enquanto a variável em seu lado consequente não deve estar instanciada, a fim de receber o resultado da operação.

Um exemplo simples de operação aritmética em Prolog é:

```
A is B + 4.
```

que equivale a instanciar à variável `A` o valor da variável `B` adicionado a quatro.

2.1.10 Listas

Embora se trate de um aspecto não contemplado pela corrente pesquisa, cabe destacar o suporte oferecido pelo Prolog à criação e manipulação da estrutura de dados de lista. O Prolog utiliza a mesma sintaxe que as linguagens ML e Haskell para especificação de listas.

Uma lista Prolog possui grande semelhança com a estrutura análoga na linguagem LISP[1], baseando-se na determinação da cabeça (*head*) e cauda (*tail*). É com base nesta estruturação que a linguagem utiliza um mecanismo inerentemente

1 Um apanhado sobre a linguagem LISP pode ser encontrado no seguinte endereço: http://en.wikipedia.org/wiki/Lisp_(programming_language).

recursivo para manipulação de listas. A cabeça passa a ser o primeiro elemento da lista, enquanto a cauda é uma lista com os demais elementos, podendo até mesmo ser vazia.

2.1.11 Aplicações

Cabe destacar as principais áreas de aplicação de linguagens lógicas:

- Sistemas de gerenciamento de bases de dados relacionais: bases de dados relacionais armazenam seus dados em tabelas relacionadas entre si. A relação com o Prolog passa a ser explícita se considerarmos que os dados armazenados podem ser declarados como fatos, e as relações entre estes dados, como regras. O uso de uma linguagem como o Prolog no desenvolvimento de sistemas desta natureza traz o benefício de utilizar-se apenas uma linguagem tanto para definição, manipulação e consulta de dados, quanto para entrada e saída de informações. Além de prover ao sistema capacidades dedutivas. A principal deficiência passa a ser o desempenho mais lento;
- Sistemas especialistas: “Sistemas especialistas são sistemas computacionais desenvolvidos para emular a especialidade humana em determinado domínio” (SEBESTA, 2012, p. 757). Consistem basicamente em um conjunto de fatos e regras de inferência fornecidos inicialmente, e devem possuir a capacidade de “aprender” dinamicamente. Linguagens como o Prolog podem facilmente suprir estes requisitos básicos;
- Processamento de linguagem natural: esta é, atualmente, uma das principais áreas de aplicação de linguagens de programação lógicas. Linguagens como o Prolog são utilizadas a fim de descrever sintaxes, realizar modelagem semântica de linguagens, além de traduções, principalmente em gramáticas livres de contexto.

2.1.12 Considerações sobre o Prolog

Atualmente o Prolog é uma linguagem com um grande número de proponentes e

seguidores, especialmente em ambientes de pesquisa relacionados a Bancos de Dados e Inteligência Artificial, sendo considerada uma das principais linguagens utilizadas neste contexto. Assim, é inegável a contribuição que o Prolog vem oferecendo, tanto no que diz respeito ao desenvolvimento de tecnologias e conhecimento, quanto à popularização do paradigma de programação lógico.

Muitos, ainda, acreditam que o Prolog pode ser parte da solução para a resolução de problemas que linguagens imperativas atualmente não conseguem lidar (SEBESTA, 2003 *apud* CUADRADO E CUADRADO, 1985). Algumas das razões que levam a essa afirmação seriam: a capacidade que a linguagem possui de ser escrita e organizada logicamente, levando a menos erros e manutenção; o processamento naturalmente paralelo; e o reduzido tempo de desenvolvimento.

2.2 (Múltiplas) Representações Externas

Entende-se por Representação Externa (RE) o uso das mais diversas técnicas a fim de representar, organizar e apresentar conhecimento (MASCHIO, 2007). Neste contexto, consideram-se representações externas desde textos, diagramas, equações, tabelas, grafos, animações, sons, vídeos, até realidade aumentada e virtual.

Segundo Ainsworth (2006) apud Palmer (1977), uma representação externa consiste: (a) no mundo representado, (b) na representação do mundo, (c) em quais aspectos do mundo estão sendo representados, (d) em quais aspectos da representação do mundo fazem parte da modelagem e, (e) a correspondência entre os dois mundos. Assim podemos considerar as representações externas como parte do nosso dia-a-dia, seja no momento em se viaja guiado por um aparelho de GPS, até o momento em que se consulta o cardápio de um restaurante.

Há evidências de que a adequação de uma RE à uma necessidade de aprendizado pode melhorar a performance e o entendimento do tema em questão (AINSWORTH, 2006). Ainda, várias REs relacionadas a um mesmo mundo representado podem conter benefícios distintos.

A partir disso surge o conceito de Múltiplas Representações Externas

(MREs). Consiste em um conjunto de duas ou mais REs a fim de extrair os benefícios advindos de cada uma delas.

Segundo Ainsworth (1999) e Maschio (2007) são três as funções principais exercidas por Múltiplas Representações Externas (MERs): complementar, restringir e construir (Figura 3).

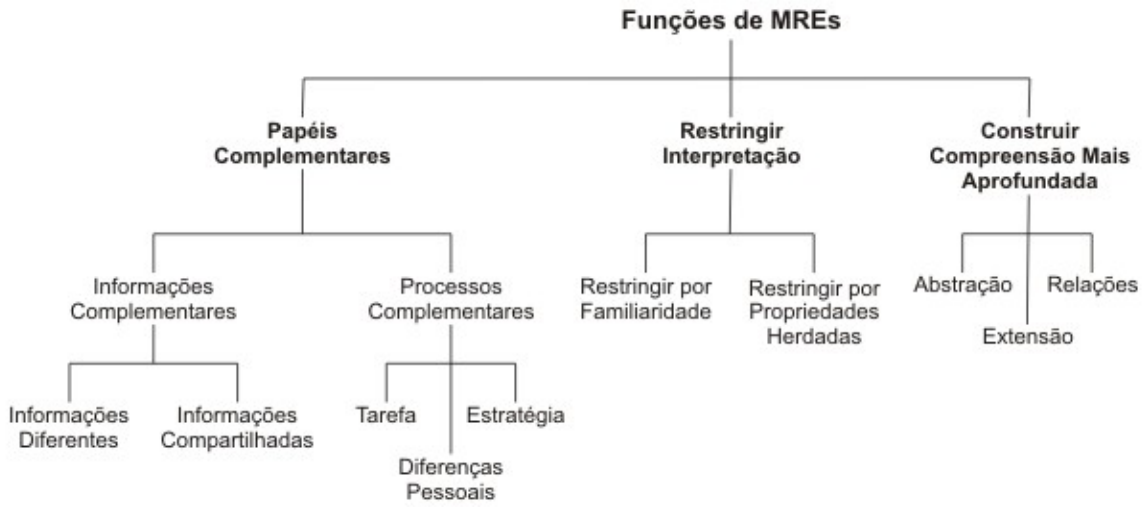

Figura 3 – Taxonomia funcional de Múltiplas Representações Externas.
Fonte: Maschio (2007, p. 10).

1. Papéis complementares:

   MREs complementam-se quando diferem na informação que apresentam ou processo ao qual suportam. A combinação de MREs com essas características busca prover o aprendiz com o benefício de cada uma das representações, individualmente.

   (a) Informações complementares: acontecem quando REs tratam diferentes tipos de informação. Ocorre quando uma única RE é insuficiente para tratar toda a informação necessária ou quando a complexidade de manter essa informação sob uma única RE é elevada.

   Subdivide-se ainda em:

   1. Informações diferentes: cada uma das REs aborda aspectos únicos do domínio.

2. Informações compartilhadas: existe uma intersecção de informações compartilhadas pelas REs.

(b) Processos complementares: apesar de poderem representar informações semelhantes, REs podem diferir nas inferências e processos que venham a sustentar. Esta situação pode ser vantajosa por uma série de razões.

1. Diferenças pessoais: provido de uma série de representações, o aprendiz pode fazer uso daquela que possua maior adequação às suas necessidades.
2. Tarefas: o desempenho passa a ser facilitado quando a estrutura de informação requerida pelo problema é compatível com a RE escolhida.
3. Estratégias: o provimento de diferentes formas de representação para a resolução de problemas encoraja o aprendiz a utilizar mais de uma estratégia para solução. Esta, pode ser mais ou menos efetiva.

2. Restrição de interpretação:

Consiste no uso de uma RE a fim de restringir uma interpretação errônea de uma segunda RE. Ocorre por basicamente dois meios:

(a) Restrição por familiaridade: a familiaridade do aprendiz com uma representação restringe a interpretação de uma menos familiar;

(b) Restrição por propriedades herdadas: ocorre quando uma RE com interpretação ambígua pode ser restringida por uma segunda RE com representação específica.

3. Construção de conhecimento aprofundado:

Ocorre quando a reunião de informações providas por MREs favorece o desenvolvimento de conhecimento que seria difícil de ser atingido através de apenas uma RE. Pode ocorrer através de:

(a) Abstração: processo pelo qual o aprendiz cria entidades mentais a

fim de embasar conceitos e procedimentos em um nível mais alto de organização;

(b) Extensão: considera-se o ato de estender o conhecimento prévio do aprendiz a novas situações sem, fundamentalmente, reconhecer a natureza deste conhecimento;

(c) Relação: o entendimento relacional é o processo pelo qual duas REs são associadas sem reorganização de conhecimento.

No que tange à representação de fenômenos não estáticos e baseando-se na pesquisa de Ainsworth e Labeke (2004), define-se o conceito de representação dinâmica como aquela que exibe processos que mudam no decorrer do tempo.

Desta forma, representações dinâmicas passam a ser classificadas de acordo com suas propriedades computacionais e referentes às informações que carregam. Surgindo, assim, três tipos de representações dinâmicas: *time-persistent*, *time-implicit*, e *time-singular*.

Resumidamente, pode-se introduzir a representação *time-persistent* como aquela que expressa a relação entre ao menos uma variável e o tempo, exibindo ambos (i) o valor atual e (ii) qualquer outro que tenha sido computado. Já a representação *time-implicit* é aquela que apresenta uma série de valores, mas não o tempo em que estes valores ocorreram. Ainda, uma representação *time-singular* apresenta uma ou mais variáveis em um único instante no tempo.

Conjuntos de MREs, incluindo representações dinâmicas, podem ser utilizados na representação e interpretação de algoritmos. Abre-se espaço, portanto, para a utilização destes conceitos na área de Visualização de Programas e Algoritmos.

2.3 Visualização de Programas e Algoritmos

Segundo Mulholland (1997) e baseando-se em sua pesquisa, uma visualização de programa consiste no uso das mais variadas técnicas a fim de representar o próprio programa e sua execução. A definição pode também ser estendida à visualização de algoritmos, onde estas técnicas representam vários estados e animam a transição

entre estes estados de um algoritmo (SHAFFER et al., 2010, p. 2).

De acordo com Shaffer et al. (2010), há uma quantidade significativa, porém não suficiente, de boas visualizações de algoritmos. Enquanto a maioria das visualizações existentes aborda temas mais simples do ponto de vista didático, ainda existe a necessidade de desenvolvimento de abordagens para temas mais complexos.

Grande parte das visualizações de algoritmos foram criadas na década de 1990, durante o advento da plataforma Java. Com o decorrer do tempo, muitas deixaram de ser mantidas por seus desenvolvedores, ou tiveram suas referências perdidas, impactando na quantidade de representações disponíveis atualmente.

No que diz respeito à representação e visualização de programas em Prolog, uma gama de visualizadores foi criado devido ao Prolog ter um modelo de execução particularmente complexo, onde aprendizes possuem dificuldades no seu entendimento (MULHOLLAND, 1997).

Porém, poucos dos visualizadores possuíam uma representação que não textual. Dentre os primeiros visualizadores a fazerem uso de elementos gráficos diversos, podemos citar o *Transparent Prolog Machine* (TPM), criado em 1986 por Eisenstadt e Brayshaw. Entretanto em testes realizando a comparação entre o TPM e três outros visualizadores essencialmente textuais, “o TPM demonstrou a pior performance geral no experimento, isto é, verificou-se ser menos eficaz no ensino do processo de execução do Prolog” (HENRY; INOUE, 2007, p. 1).

Desde então vários sistemas de visualização de programas foram desenvolvidos a fim de suprir esta lacuna de visualizadores que utilizassem representações gráficas. Dentre eles, podemos citar o *Fril Visual Tracer* (FVT) desenvolvido por Henry e Inoue (2007) e o *Logichart Diagram*, apresentado como sistema de visualização do Prolog por Adachi (2009). Representações e detalhamentos específicos de cada ambiente serão abordados em momento adequado.

2.4 Ambientes Interativos de Aprendizagem (AIAs)

Conforme define Maschio (2007), um AIA consiste em um ambiente passivo que não interfere ou interage de maneira pró-ativa com o usuário, apesar de possuir riqueza semântica. O desenvolvimento de um AIA surge da premissa de que o aprendiz desenvolverá seu conhecimento através de atividades exploratórias, investigativas e de descoberta. Neste contexto, o ambiente interativo de aprendizagem deverá prover os recursos necessários para que o uso da ferramenta pelo aprendiz atinja os resultados esperados.

Ademais, embora seja comum a relação feita entre AIAs e Sistemas Tutores Inteligentes (STIs), a diferença entre ambos deve ser destacada. Um STI interage ativamente com o aprendiz e atua na escolha do conteúdo e da maneira que se ensina, como também na avaliação de possíveis erros e intervenção. AIAs possuem um caráter tipicamente passivo, permitindo que o aprendiz explore o ambiente apresentado e construa seu conhecimento a partir das observações (experimentos) realizadas.

Para enfatizar a riqueza semântica proporcionada, os AIAs comumente buscam subsídio em representações externas. Na área de Programação de Computadores, REs têm sido constatadas como meios benéficos no auxílio ao aprendiz no desenvolvimento de perícia. De forma específica, elementos de interação e de correspondência entre representações também se mostram potencialmente úteis (Maschio, 2007).

Dentro disto, o ambiente interativo de aprendizagem proposto oferece ao aprendiz meios interativos de visualização do mecanismo de inferência da linguagem Prolog. Adicionalmente, serão enfocadas, durante a interação, as correspondências entre as representações utilizadas. Intenciona-se que tal riqueza semântica contribua para as atividades exploratórias no ambiente desenvolvido.

3 TRABALHOS RELACIONADOS

Neste capítulo são abordados alguns dos demais ambientes existentes desenvolvidos com o propósito de auxiliar na visualização de algoritmos em Prolog.

3.1 Spy

Criado por Byrd (1980), o *Spy* é um sistema de visualização textual que, abordando o processo de execução de um programa Prolog, é utilizado como mecanismo de rastreamento nas implementações mais comuns da linguagem. Faz uso dos identificadores *call*, *exit*, *redo* e *fail* para explicitar ações de chamada, retorno, retroação e falha, respectivamente, contando ainda com um identificador *UNIFY*, indicando operações de unificação.

Apesar de bastante simples, do ponto de vista de sua estrutura, o objetivo principal do desenvolvimento do *Spy* foi "fornecer uma descrição básica, porém completa do Prolog, sustentado por um modelo de execução consistente" (MULHOLLAND, 1997, p. 35).

Um exemplo de visualização fornecida pelo sistema *Spy* pode ser observado na Figura 4.

```
call  grandparent(_1, _2)

UNIFY 1   []
  call  parent(_1, _3)
  UNIFY 1   [_1 = tom _3 = liz]
  exit  parent(tom, liz)
  call  parent(liz, _2)
  fail  parent(liz, _2)
  redo  parent(tom, liz)
  UNIFY 2   [_1 = pam _3 = bob]
  exit  parent(pam, bob)
  call  parent(bob, _2)
  UNIFY 3   [_2 = ann]
  exit  parent(bob, ann)
exit  grandparent(pam, ann)
```

Figura 4 - Exemplo de visualização gerada pelo sistema *Spy*.
Fonte: Adaptado de Mulholland (1997, p. 35).

3.2 Prolog Trace Package (PTP)

Desenvolvido por Einsenstadt (1984) com a finalidade de prover uma representação mais detalhada e simples de ser entendida do que a encontrada no *Spy,* o PTP também é um sistema de visualização unicamente textual e faz uso de uma série de informações específicas a cada estado da execução de um programa Prolog.

Apesar de, inicialmente, buscar facilitar e simplificar o entendimento por parte de um usuário comum, o excesso de variantes de informação faz com que seja necessária a consulta constante a um referencial ou legenda que explicite o significado de cada representação de estado.

Um exemplo de visualização gerada pelo sistema *Prolog Trace Package* pode ser observado na Figura 5.

```
 1: ? grandparent(_1, _2)
 2: > grandparent(_1, _2) [1]
 3:   ? parent(_1, _3)
 4:   +*parent(tom, liz) [1]
 5:   ? parent(liz, _2)
 6:   -~parent(liz, _2)
 7:   ^ parent(tom, liz)
 8:   < parent(tom, liz) [1]
 9:   +*parent(pam, bob) [2]
10:   ? parent(bob, _2)
11:   +*parent(bob, ann) [3]
12: + grandparent(pam, ann) [1]
```

Figura 5 – Exemplo de visualização gerada pelo sistema *Prolog Trace Package*.
Fonte: Adaptado de Mulholland (1997, p. 35).

3.3 Transparent Prolog Machine (TPM)

O *Transparent Prolog Machine*, conhecido usualmente por TPM, é um sistema de visualização criado, segundo Mulholland (1997), com a intenção de apresentar a mesma representação detalhada provida pelo PTP de uma maneira mais acessível ao usuário comum. Criado por Brayshaw e Einsenstadt (1991), foi um dos primeiros sistemas de visualização a apresentar uma representação gráfica do mecanismo de execução do Prolog. Esta representação é feita através de um modelo de árvore *AND/OR* de busca em profundidade gerada pela execução do programa Prolog,

onde cada nó da árvore representa uma meta de determinada proposição.

Assim, este modelo permite a visualização do programa Prolog em dois níveis de granularidade: o primeiro através do modelo de árvore gerado, o segundo, mais detalhado, exibe informações específicas de determinado nó através da seleção do mesmo na árvore.

Em estudo conduzido por Mulholland (1997), o TPM foi considerado o sistema menos efetivo para o ensino do processo de execução do Prolog, em comparação com outros três sistemas de visualização. Este resultado deveu-se principalmente à dificuldade dos aprendizes em determinar a posição temporal na execução exibida em determinado ponto (MULHOLLAND, 1998, p. 11).

Um exemplo de visualização gerada pelo TPM pode ser verificada na Figura 6.

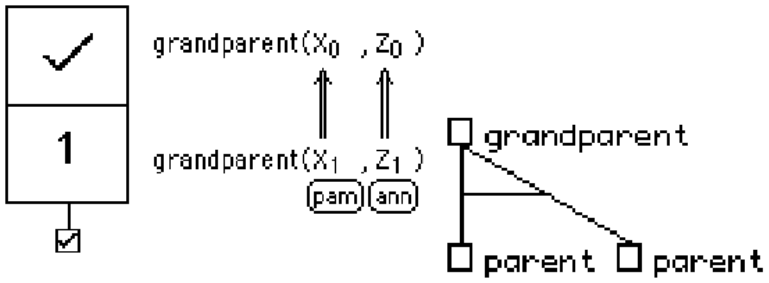

Figura 6 – Exemplo de visualização gerada pelo sistema *Transparent Prolog Machine*. Fonte: Adaptado de Mulholland (1997, p. 35).

3.4 Textual Tree Tracer (TTT)

Desenvolvido por Taylor, du Boulay e Patel (1991), este sistema de representação utiliza-se de uma representação textual de árvore para prover uma visualização da execução do programa Prolog.

O TTT faz uso da indentação da informação gerada a fim de relacioná-la a determinada proposição. O principal objetivo durante o desenvolvimento desta ferramenta de visualização, segundo Mulholland (1997), foi o de prover a mesma riqueza de informações encontrada no *Transparent Prolog Machine* de uma maneira que se assemelhe ao próprio código encontrado na base axiomática de um

programa Prolog. Dado este fato, abdicou-se de uma representação gráfica em detrimento de uma representação puramente textual.

Ainda assim, o excesso de informação talvez cause certa confusão ao aprendiz de Prolog, dada a existência de elementos alheios à representação usual da base axiomática e que demandam um maior tempo para entendimento. Um exemplo de visualização gerada pelo TTT pode ser encontrada na Figura 7.

```
>>>1: grandparent(X, Z)    1S

|1     X = pam
|      Z = ann
 ***2: parent(X, Y_1)     1SF/2S
 |1     X ≠ tom
 |      Y_1 ≠ liz
 |2     X = pam
 |      Y_1 = bob
 ***3: parent(liz, Z)     Fm
 ***4: parent(bob, Z)     3S
 |3     Z = ann
```

Figura 7 – Exemplo de visualização gerada pelo sistema *Textual Tree Tracer*.
Fonte: Adaptado de Mulholland (1997, p. 35).

3.5 Logichart Diagram

Logichart é uma linguagem de descrição de diagramas de programas voltado ao Prolog. Um diagrama *Logichart* consiste em uma estrutura semelhante a uma árvore, onde os nós representam predicados e metas. Os nós encontram-se dispostos de modo que cabeça e termos do corpo dos predicados alinhem-se horizontalmente, enquanto uma chamada de uma meta e as cabeças dos predicados que esta chama, alinham-se verticalmente.

Baseando-se neste diagrama, Adachi (2009) desenvolveu um sistema de visualização do fluxo de execução de um programa Prolog. Tal sistema provê a representação animada da execução Prolog em um diagrama *Logichart*, além da apresentação de mudanças dinâmicas através da chamada aos predicados `assertz` e `retract`. Este sistema permite, ainda, o acompanhamento do processo

de instanciamento de variáveis.

Segundo Adachi (2009), é necessária uma investigação empírica a fim de determinar a utilidade deste programa de visualização em Prolog no ensino de programação desta linguagem.

Um exemplo de diagrama resultante do sistema desenvolvido por Adachi (2009) é apresentado na Figura 8.

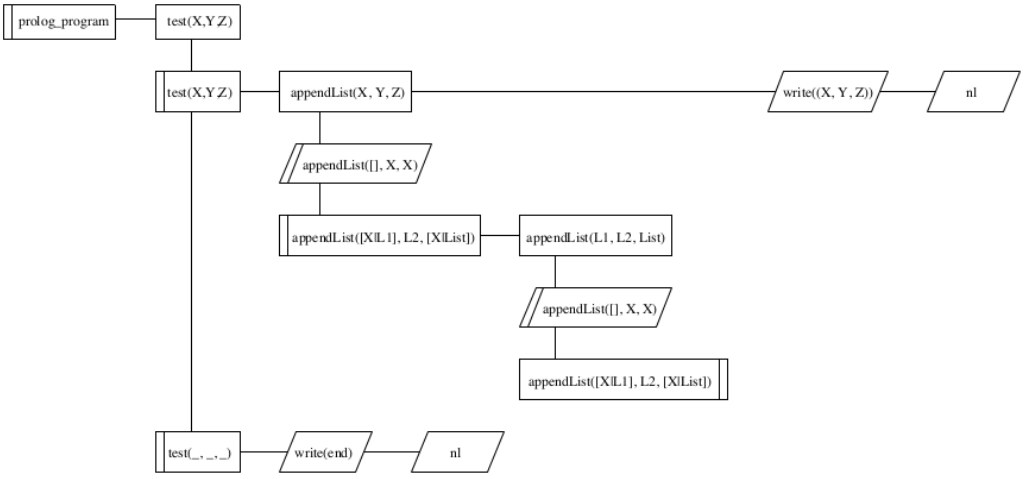

Figura 8 – Exemplo de visualização gerada a partir do modelo *Logichart Diagram*. Fonte: Adachi (2009, p. 14).

3.6 Discussão

Dado o exposto nos tópicos precedentes, abre-se espaço a uma discussão quanto a validade da proposta de um novo conjunto de representações para o sistema de inferência da linguagem Prolog, voltando-se especificamente aos conceitos de retroação e recursão.

Percebe-se a existência de representações tanto textuais quanto gráficas. Naquelas, destaca-se a busca pelo fornecimento do máximo de informação atrelada à maior simplicidade possível. Nestas, o apelo visual, ao mesmo tempo em que se propõe a fornecer quantidade significativa de informação, busca facilitar o entendimento por parte do usuário.

Porém, em alguns casos, nota-se a ineficiência em atingir estes objetivos.

Um exemplo disto é o TPM que, segundo estudo de Mulholland (1997), possuiu o pior desempenho quando se considera o entendimento por parte do aprendiz.

Destaca-se ainda o fato de que nenhuma das representações apresentadas tem por origem um objetivo didático. A representação Logichart, talvez por ser a mais atual, pode ser a que melhor se adeque a esta finalidade, mas ainda abre espaço para o desenvolvimento de uma representação que vise especificamente o ensino dos conceitos de retroação e recursão.

4 TECNOLOGIAS ENVOLVIDAS

O corrente capítulo discorre sobre as tecnologias envolvidas no desenvolvimento do ambiente interativo proposto. Também se discute a motivação de cada uma dessas tecnologias presentes na pesquisa.

4.1 Java

Java consiste em uma linguagem de programação e plataforma computacional, tendo sua primeira versão lançada no ano de 1995[1]. Baseada no paradigma orientado a objetos, dispõe de uma gama de componentes de *software* voltados às mais diversas aplicações. Um destaque desta linguagem consiste na sua alta portabilidade, visto que, após sua compilação, um código Java será interpretado por uma Máquina Virtual da própria plataforma, sendo portanto, independente de *hardware* ou sistema operacional.

A comunidade de desenvolvedores dedicada a esta plataforma é especialmente ativa, sendo extremamente fácil encontrar suporte referente a dúvidas ou implementações em praticamente todos os idiomas.

Como fatores fundamentais para a escolha desta linguagem no desenvolvimento deste projeto, podemos citar, portanto:

- Compatibilidade com componentes e bibliotecas que permitem a expansão da linguagem;
- Existência de padrões de projetos especificamente documentados para esta linguagem, facilitando sua adoção e implementação;
- Grande disponibilidade de documentação *on-line*;
- Possibilidade de personalização de interfaces;
- Alta portabilidade para os mais variados sistemas operacionais;
- Possibilidade fornecida pela linguagem de, posteriormente, desenvolver uma

1 Mais informações podem ser obtidas no seguinte endereço: https://www.java.com/en/

versão *web* para o protótipo implementado;

- Familiaridade do proponente com a linguagem.

4.2 GNU Prolog for Java

A *GNU Prolog for Java*[1], ou simplesmente GPJ, é uma biblioteca de código aberto, desenvolvida em Java, com o intuito de permitir o uso dos recursos da linguagem Prolog. Esta biblioteca provê, através de sua implementação, meios que permitem a execução de consultas a uma base axiomática escrita em Prolog de forma bastante intuitiva, semelhante à consulta realizada nativamente nesta linguagem.

Dado o exposto, a principal motivação para a escolha desta biblioteca passou a ser exatamente a simplicidade na realização destas consultas a partir de uma entrada de texto no formato das implementações mais comuns do Prolog. Além disso, o acesso facilitado aos resultados retornados em operações inerentes ao mecanismo de inferência do Prolog, como o rastreamento da execução de um programa, contribuíram para a decisão.

4.3 JGraphx

Biblioteca Java para criação e manipulação de grafos, o JGraphx[2] consiste em uma ferramenta robusta e rica em funcionalidades. Possui total compatibilidade com Swing[3], o que permite sua integração plena com o modelo de protótipo apresentado.

Por ter sua concepção baseada na Teoria dos Grafos, esta ferramenta oferece mecanismos para visualização, interação, desenho e disposição deste tipo de estrutura. Dispõe ainda de funcionalidades relacionadas à teoria em que se fundamenta (obtenção do menor caminho entre dois vértices, por exemplo).

A escolha desta biblioteca deveu-se, principalmente, à possibilidade de modelagem e interação adequada à respectiva proposta de representação externa,

1 Mais informações podem ser obtidas no seguinte endereço: http://www.gnu.org/software/gnuprologjava/
2 Mais informações sobre esta biblioteca podem ser encontradas em: https://github.com/jgraph/jgraphx
3 Uma definição sobre esta API pode ser encontrada em: http://docs.oracle.com/javase/tutorial/uiswing/start/about.html

tais quais a criação de painéis contendo predicados e disposição destes painéis conforme necessidade. Ainda, a relativa facilidade de manipulação destes objetos dentro da linguagem Java contribuíram para a escolha desta ferramenta.

Destaca-se, portanto, que a aplicação da biblioteca JGraphx teve importância não apenas para os aspectos de visualização do ambiente. Trouxe, além disto, facilidades pertinentes a questões de modelagem e representação interna daquilo que se visualiza.

5 FORMALISMO ADOTADO

Neste capítulo aborda-se o conjunto de múltiplas representações externas proposto, através da apresentação de uma simulação e da correlação entre cada uma das representações e suas finalidades didáticas.

5.1 Conjunto Multirrepresentacional Proposto

Como parte da solução exposta neste trabalho, várias categorias de representações externas foram estudadas, definidas e até mesmo criadas. A fim de evidenciar o conjunto de múltiplas representações externas composto por estas diversas categorias de REs, torna-se válida a apresentação de uma simulação contendo o conjunto proposto.

Conforme pode ser observado nas Figuras 9 e 10, destaca-se a utilização do conjunto multirrepresentacional proposto na simulação do cálculo do fatorial de determinado número. A simulação tem início com a definição da meta a ser provada ou refutada e, a partir disso, inicia-se a criação de painéis contendo predicados da base axiomática relacionados ao questionamento definido. As cores azul, vermelha e verde definem diferentes estados de um predicado, respectivamente: em avaliação, falha, e sucesso. Ainda, cada painel sobreposto representa a descida de um nível no processo recursivo, assim como sua remoção indica o retorno ao nível anterior. A simulação tem fim com o sucesso em obter a prova da meta solicitada.

A partir da apresentação das diversas REs como conjunto, em seu contexto, torna-se conveniente a abordagem individualizada de cada uma delas, assim como o relacionamento destas com suas respectivas finalidades didáticas. Destaca-se que as representações aqui apresentadas são passíveis de aperfeiçoamento após a coleta de resultados a partir da utilização da solução proposta, em um trabalho futuro.

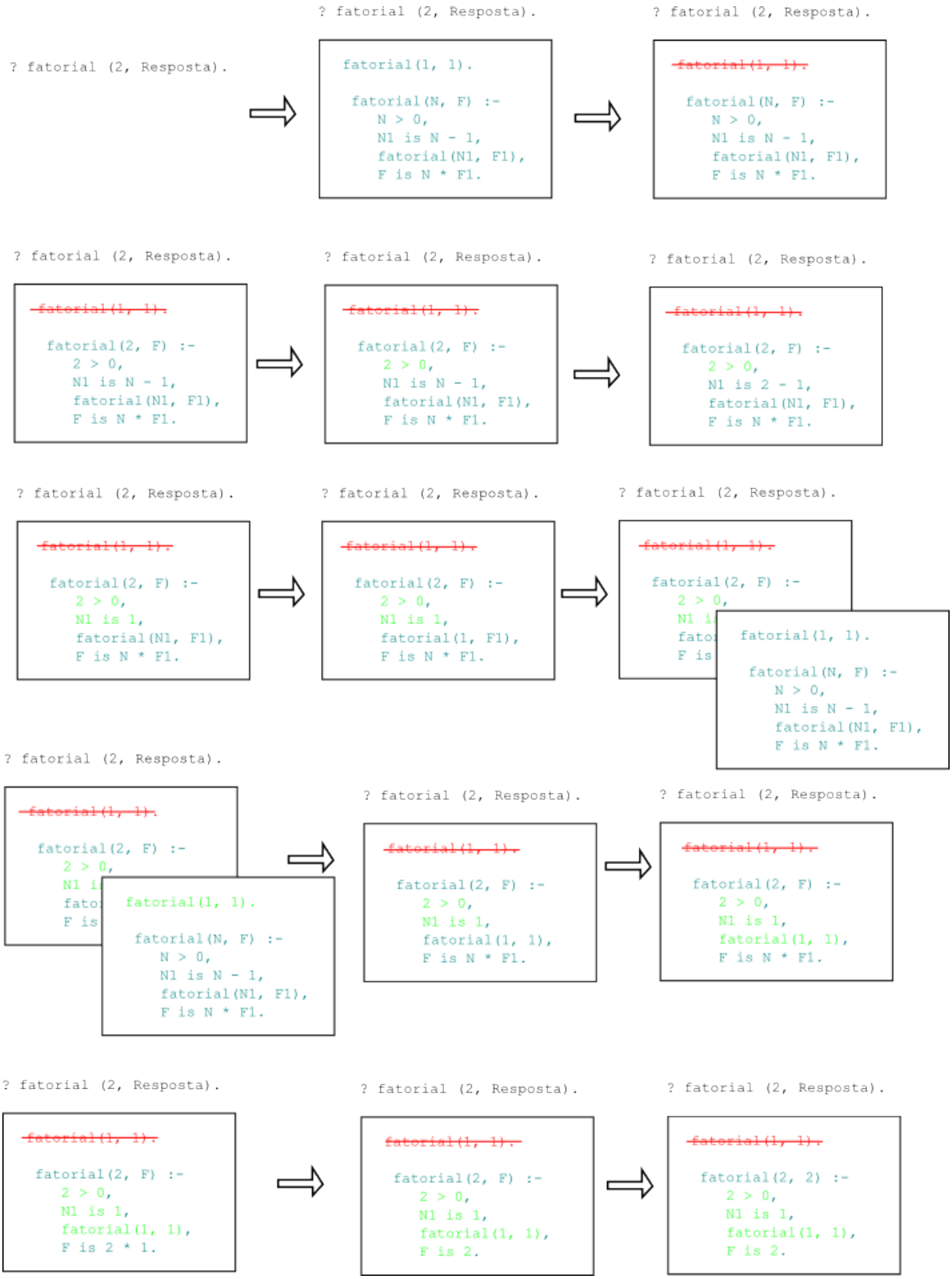

Figura 9 – Simulação do cálculo de fatorial utilizando o conjunto multirrepresentacional proposto.

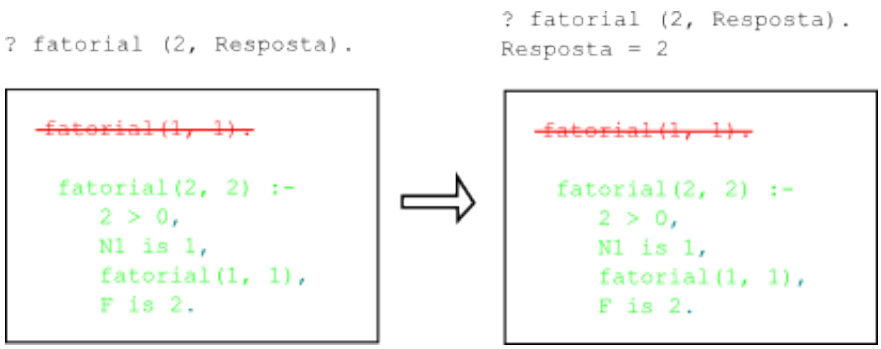

Figura 10 – Simulação do cálculo de fatorial utilizando o conjunto multirrepresentacional proposto.

5.1.1 Prompt de conversação do Prolog

Foi implementado um console que se remetesse ao existente e utilizado em interpretadores Prolog. Este console tem por função permitir que o aprendiz insira as metas a serem provadas ou refutadas pelo mecanismo de inferência da linguagem.

Sendo semelhante a consoles já existentes, seu objetivo passa a ser adaptar o usuário ao ambiente de interação, através do papel de restrição por familiaridade, com uma estrutura previamente conhecida.

5.1.2 Painel com ativação do predicado

Ao realizar uma consulta, o protótipo exibe ao aprendiz um painel contendo o predicado a ser validado, indicando assim o início do processo de inferência. Além disso, são exibidas todas as entradas da base axiomática relacionadas àquele predicado.

Busca-se, com isso, que o usuário identifique desvios no fluxo de execução do mecanismo de inferência, bem como identifique o início de um novo processo de inferência.

Um exemplo de painel contendo a ativação de um predicado pode ser observado na Figura 11.

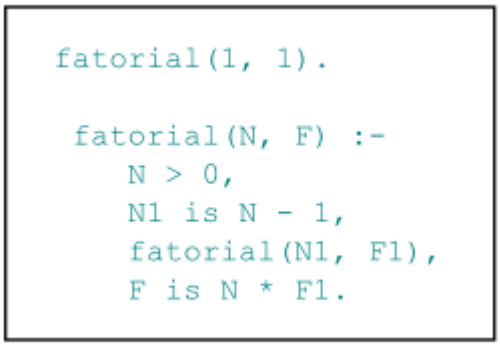

Figura 11 – Painel com ativação do predicado.

5.1.3 Cores e tachado

Com o intuito de representar os diferentes estados possíveis de um predicado durante o processo de inferência provido pelo Prolog, buscou-se um conjunto de cores adequado a esta finalidade. Esta representação tem por função permitir que o aprendiz identifique rapidamente, ao visualizar determinado estado do processo de inferência, quais predicados já passaram por validação ou não, assim como o retorno desta validação.

Assim, utilizou-se a cor azul a fim de representar predicados sob o estado de chamada (*Call*) e reunificação (*Redo*). O estado de retorno com sucesso (*Exit*) foi definido com a cor verde, enquanto para o retorno com falha (*Fail*) utilizou-se a cor vermelha, em conjunto com uma linha tachando o predicado, ressaltando assim sua natureza de insucesso.

Um exemplo da utilização das cores na representação dos estados de um predicado durante o processo de inferência do Prolog pode ser observado na Figura 12.

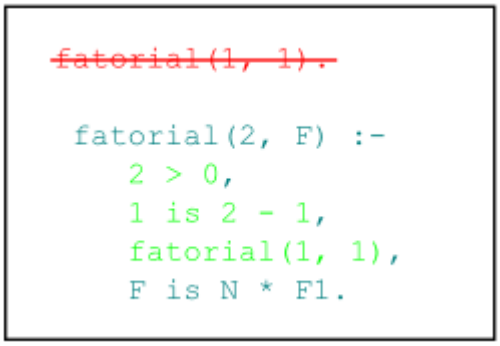

Figura 12 – Cores e tachado.

5.1.4 Encadeamento da cascata recursiva

O encadeamento da cascata recursiva é representado no conjunto multirrepresentacional proposto, assim como na simulação anteriormente vista, pela sobreposição dos painéis contendo os predicados avaliados. Busca-se, desta forma, remeter o aprendiz ao conceito de recursão, explicitando a natureza hierárquica contida neste conceito, visto que cada sobreposição representa a descida de um nível na recursão corrente.

Além disso provê-se a demonstração da necessidade de uma condição de parada, conceito crucial para o entendimento do modelo recursivo. Um exemplo da representação de encadeamento da cascata recursiva pode ser observada na Figura 13.

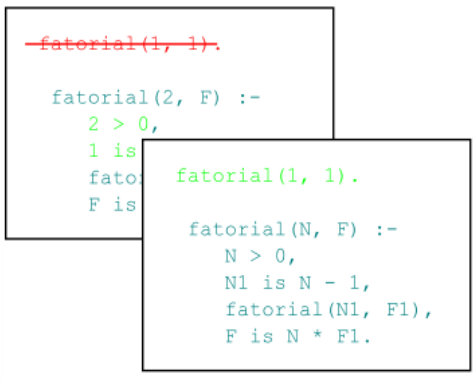

Figura 13 – Encadeamento da cascata recursiva.

5.1.5 Recolhimento da cascata recursiva

Esta representação, assim como a anterior, remete ao conceito hierárquico da recursão e consiste basicamente na remoção do painel de nível mais externo. Tem por função explicitar o retorno de uma resposta a sua chamada após encontrar uma condição de retorno.

5.1.6 Aspecto conclusivo da prova

O aspecto conclusivo da prova por si já corresponde a uma múltipla representação externa, visto que compreende o recolhimento da cascata recursiva e a utilização de cores e tachado. Sua finalidade é a de apresentar a conclusão de um processo de inferência, provando ou refutando uma meta. Consiste basicamente na apresentação do primeiro painel de ativação de predicado com o seu resultado definido na cor verde ou vermelha com tachado, indicando, respectivamente, a prova ou refutação do predicado inicialmente inserido.

5.2 Correspondência do Conjunto com a Taxonomia Funcionalista

Torna-se conveniente explicitar a correspondência do conjunto multirrepresentacional proposto à taxonomia funcionalista abordada na Seção 2.2, demonstrando assim as funcionalidades abordadas pela solução desenvolvida.

5.2.1 Papéis complementares

Neste contexto, podemos destacar tanto a classificação de informações complementares compartilhadas quanto a de informações complementares diferentes. A primeira toma por base o encadeamento e recolhimento da cascata recursiva, as quais representam o mesmo processo recursivo em momentos distintos, compartilhando, desta forma, da mesma informação. A segunda refere-se à utilização de cores e tachado que, além do conceito recursivo, adiciona ao modelo multirrepresentacional proposto a alusão aos estados do processo de inferência do

Prolog.

5.2.2 Restrição de interpretação

A abordagem adotada neste trabalho enquadra-se, nesta classe, como restrição por familiaridade, pois um conhecimento prévio da sintaxe do Prolog é necessário para a utilização do protótipo proposto.

A sintaxe da linguagem é utilizada como recurso em diversas representações, sendo abordada desde o *prompt* de conversação, passando pelos painéis de ativação de predicados até o aspecto conclusivo da prova de uma consulta. Assim, ambiguidades que possam surgir frente às representações propostas e que façam uso deste recurso, passam a ser exauridas devido ao conhecimento prévio do aprendiz em relação à linguagem utilizada.

5.2.3 Construção de conhecimento aprofundado

No contexto do corrente trabalho, a função desta classe da taxonomia funcional que se destaca é a de abstração. Isto pois a RE no formato de cascata, com painéis sobrepostos, busca auxiliar o aprendiz no desenvolvimento de uma entidade mental que o remeta à natureza hierárquica e, deste modo, o leve à assimilação do conceito de recursividade.

Ainda, a função de relação se evidencia no momento do encadeamento e recolhimento da cascata recursiva através da possibilidade de manipulação da representação por parte do aprendiz e identificação de desvios no fluxo de inferência quando do surgimento e recolhimento dos painéis. Permite, deste modo, que o aprendiz localize exatamente em qual ponto no tempo se encontra o processo de inferência.

Ademais, a função de extensão, apesar de não evidenciada no protótipo, ocorrerá fora do ambiente interativo, quando o aprendiz, além de dominar a linguagem Prolog e seu processo de inferência, utilizar o conhecimento adquirido a partir da manipulação da ferramenta na escrita de algoritmos recursivos nas mais variadas linguagens.

6 PROTÓTIPO IMPLEMENTADO

O capítulo corrente aborda a interação do aprendiz com o protótipo implementado, destacando funcionalidades da solução, assim como particularidades e características da implementação.

6.1 Interação com o Aprendiz

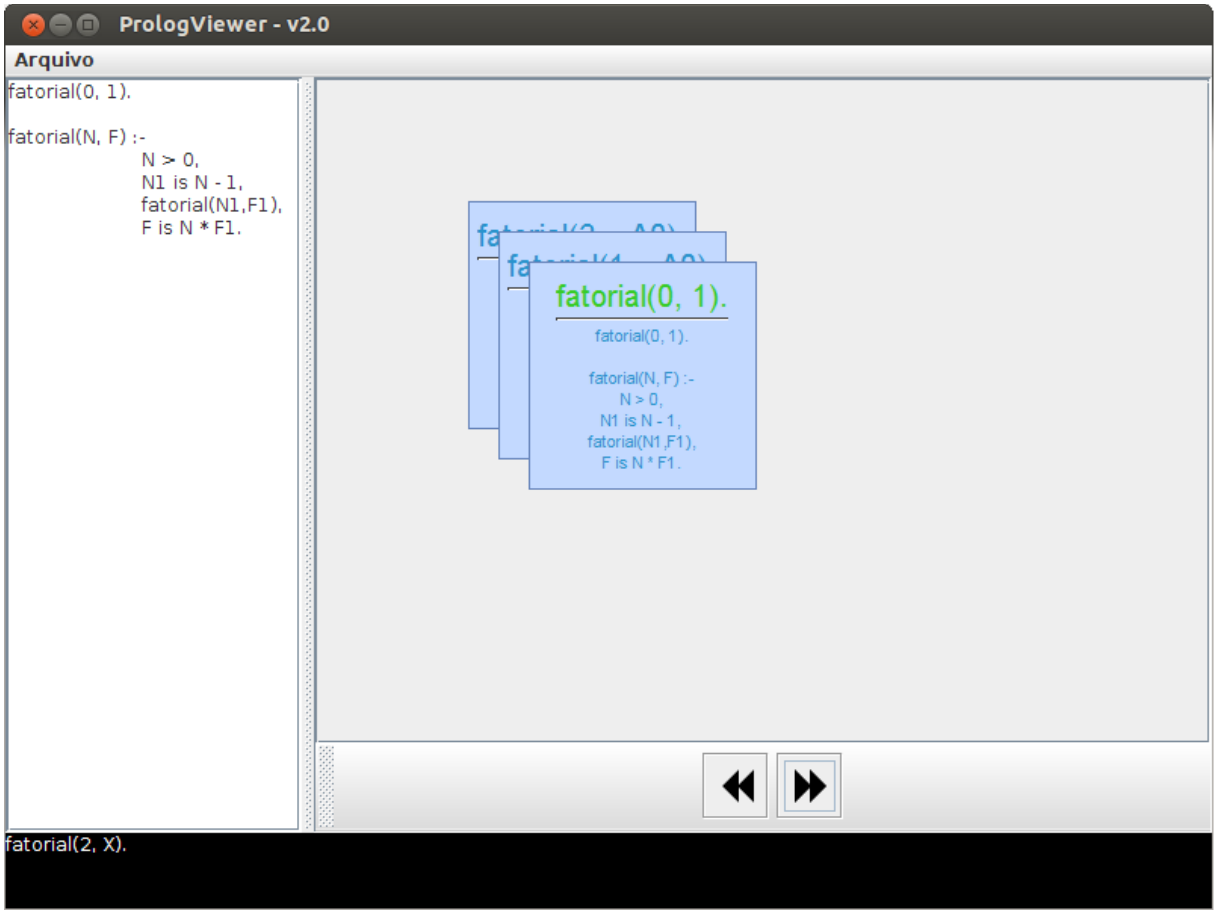

Figura 14 – Visão geral da interface da aplicação.

O desenvolvimento da uma interface gráfica para interação do aprendiz com a aplicação foi norteado pela ideia da simplicidade e funcionalidade.

A interface consiste, basicamente, em uma barra de *menus* localizada na parte superior, um painel central contendo a área de visualização do arquivo

utilizado como base axiomática e o painel para visualização e manipulação da representação gerada.

A área inferior exerce a função de console, onde o usuário inserirá os questionamentos à base axiomática previamente carregada. Uma visão geral da interface da aplicação pode ser obtida através da Figura 14.

6.1.1 Barra de *Menus*

Figura 15 – Barra de *menus*.

Localizada no canto superior esquerdo da aplicação, a barra de menus compreende duas opções: *Abrir...* e *Sair*.

A primeira detém a funcionalidade que permite o acesso a uma base axiomática pré-existente na máquina utilizada. Ao clicá-la, abre-se uma janela de seleção de arquivos, onde o aprendiz faz a seleção da base axiomática desejada. Após selecionar determinado arquivo, ele será carregado em um painel de visualização específico.

A segunda opção permite a finalização da aplicação. A barra de menus pode ser observada na Figura 15.

6.1.2 Painel de visualização da base axiomática

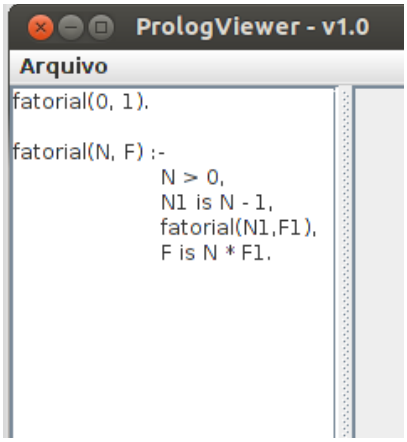

Figura 16 – Painel de visualização da base axiomática.

O painel de visualização da base axiomática corresponde à área definida para a exibição do conteúdo do arquivo utilizado como base axiomática e que será carregado pelo processador Prolog. Consiste basicamente em uma área em branco, que será preenchida com o conteúdo do arquivo selecionado através do menu *Abrir.* Caso o tamanho deste conteúdo ultrapasse as dimensões do painel, surgirá uma barra de rolagem vertical a fim de que o mesmo possa ser visualizado por completo. As dimensões laterais também podem ser alteradas através de uma barra de *split*.

Porém, o painel de exibição não permite que o seu conteúdo seja editado. Entretanto, futuramente, objetiva-se que seja habilitada esta propriedade de alteração durante a execução do ambiente interativo, tornando-o assim uma espécie de ambiente de desenvolvimento específico à linguagem Prolog.

O painel de visualização da base axiomática pode ser visualizado na Figura 16.

6.1.3 Painel de visualização da representação gerada

Figura 17 – Painel de visualização da representação gerada.

O painel de visualização da representação gerada compreende a área de exibição da RE oriunda da execução da máquina de inferência Prolog. Existe também uma barra de ferramentas que permite a interação com esta representação.

A área de exibição da RE gerada consiste em um painel em branco, localizado ao lado do painel de visualização da base axiomática, e separado deste pela mesma barra de *split* que permite o redimensionamento de ambos. É neste painel que surgirão as representações apresentadas na simulação da Seção 5.1.

A barra de ferramentas localiza-se na parte inferior do painel de visualização da representação gerada e compreende dois botões com finalidades específicas:

- Botão de avanço: permite o avanço da exibição da RE apresentada. É o equivalente a solicitar à máquina de inferência do Prolog que realize o próximo passo em seu processo. Corresponde à função *do* e *redo*.
- Botão de retrocesso: permite o retrocesso da exibição da RE apresentada. É

o equivalente a solicitar à máquina de inferência do Prolog que retorne ao passo anteriormente realizado. Corresponde à função *undo*.

Uma visão geral do painel de visualização da representação gerada pode ser observada na Figura 17.

6.1.4 Console

Figura 18 – Console.

O console localiza-se na área inferior da interface da aplicação e busca se relacionar a uma interface padrão de processadores Prolog comumente utilizados. Para isso, alteraram-se algumas características padrões de uma caixa de inserção de texto. Seu plano de fundo passou a ser preto, enquanto a cor da fonte utilizada tornou-se branca.

Assim, é através do console que o usuário irá inserir suas consultas à base axiomática, no intuito de prová-las ou refutá-las com a máquina de inferência Prolog. Uma visão do console pode ser obtida através da Figura 18.

6.2 Características da Implementação

Durante o desenvolvimento do protótipo de ambiente interativo de aprendizagem procurou-se aplicar conceitos e técnicas assimilados durante o curso de Tecnologia em Sistemas para Internet. Destes conceitos, destacam-se a utilização do Paradigma Orientado a Objetos, assim como a aplicação de conhecimentos, oriundos da área de Padrões de Projetos, relacionados a este paradigma.

6.2.1 Uma visão geral do projeto

O desenvolvimento do protótipo de Ambiente Interativo de Aprendizagem fez uso de

uma série de recursos advindos principalmente de bibliotecas externas como a *GNU Prolog for Java* e a *JGraphx*.

- *GNU Prolog for Java* (GPJ):

  Após a realização de uma consulta, em formato texto e na sintaxe padrão das implementações mais comuns do Prolog, na aplicação, esta, através da utilização de expressões regulares, é dividida e instanciada em diversos objetos de classes pré-existentes na biblioteca GPJ.

  Esta divisão realizada através da utilização de expressões regulares resulta em objetos que representam termos, variáveis, functores e parâmetros que compunham uma consulta. Todos estes objetos são, então, passados ao motor de inferência fornecido pela GPJ para que sejam processados.

  Como resultado do processamento é dada uma saída em texto que, novamente através de expressões regulares, é dividida e instanciada em objetos que servirão de base para a criação de parte dos elementos gráficos baseados no conjunto multirrepresentacional proposto. Estes elementos gráficos surgem com a utilização da biblioteca *JGraphx.*

- *JGraphx*:

  Tendo por função gerar os elementos que compunham a maior parte do conjunto multirrepresentacional proposto, esta biblioteca é abastecida com os objetos gerados pelo processamento oriundo da GPJ.

  Para isto foram criadas duas classes: *Graph* e *GraphComponent*, que, estendendo classes padrões da biblioteca, geram os elementos de um grafo conforme o formato desejado. Mais especificamente, a classe *Graph* tem por função controlar o grafo como um todo, possuindo em seu interior uma lista de objetos da classe *GraphComponent,* que nada mais são do que os vértices do grafo.

  É baseando-se na classe *Graph*, ainda, que surge a implementação do Padrão de Projeto *Memento*.

6.2.2 Padrão de Projeto *Memento*

O Padrão de Projeto *Memento* pode ser considerado elemento fundamental na visualização do algoritmo. Através do *memento* é possível realizar o processo de *do*/*redo* e *undo* controlado pelo aprendiz, permitindo assim a interação desejada entre usuário e aplicação.

O padrão consiste em basicamente três classes, que auxiliam na extração e armazenamento do estado de determinado objeto. São elas:

- Originador: objeto que origina os dados para o *memento*;
- *Memento*: representa o estado do objeto armazenado;
- *Caretaker*: é a classe responsável pela custódia do *memento*.

No contexto desta aplicação, a classe *Graph* foi a selecionada como originador, visto que a mesma é a responsável por manter o grafo e seus respectivos componentes.

7 DIFICULDADES ENFRENTADAS

Durante o desenvolvimento deste trabalho, procurou-se ferramentas que facilitassem o desenvolvimento do protótipo de ambiente interativo de aprendizagem contendo o conjunto multirrepresentacional implementado.

Assim, após uma busca entre as opções existentes, verificou-se que a biblioteca *GNU Prolog for Java* atendia às necessidades iniciais do trabalho, possuindo métodos de tracejamento do processo de inferência da linguagem Prolog, assim como saídas que se adequavam aos requisitos inicialmente desejados.

Porém, após realizar o desenvolvimento do protótipo de ambiente interativo de aprendizagem proposto, verificou-se que a opção selecionada não possuía suporte ao tracejamento de determinados predicados nativos da linguagem Prolog, tais como operadores aritméticos, de atribuição, além do predicado *not*.

Este fato fez com que algumas das representações externas do conjunto multirrepresentacional desenvolvido, como o painel de ativação do predicado contendo em seu corpo os predicados a este relacionados, não se adequassem completamente à proposta apresentada. Dada a impossibilidade de tracejar os predicados anteriormente citados, não houve meios de sinalizar o *status* de determinados termos durante o processo de inferência.

Além disso, contribuiu como dificuldade, a abordagem dada às saídas fornecidas pela biblioteca *GNU Prolog for Java*. Utilizou-se a saída em tela provida pela biblioteca, em detrimento de extensões personalizadas da mesma que, porventura, otimizassem a aplicação.

Estes fatos demonstram a necessidade e importância de uma adequada elicitação e análise de requisitos, e o grande impacto que estas podem apresentar em projetos das mais variadas dimensões.

8 RESULTADOS E LIMITAÇÕES

Este capítulo tem por finalidade elencar resultados obtidos, baseando-se nos objetivos previamente estabelecidos, assim como identificar limitações concernentes ao protótipo desenvolvido.

8.1 Resultados Obtidos

Partindo de objetivos previamente estabelecidos, podem-se elencar os seguintes resultados obtidos pelo corrente trabalho:

- Desenvolvimento de um conjunto de MREs consolidado, composto pelas diversas representações abordadas na Seção 5.1 como o encadeamento da cascata recursiva, cores e tachado, painel de ativação do predicado, entre outros. Buscou-se o desenvolvimento deste conjunto de MREs com foco no auxílio ao entendimento de conceitos correlatos à Programação de Computadores em várias linguagens. Destacando-se o conceito de recursão, abordado especificamente na linguagem Prolog;
- Desenvolvimento de um protótipo de ambiente interativo de aprendizagem, com a instanciação parcial do conjunto de múltiplas representações externas desenvolvido;
- Aprofundamento, por parte do acadêmico, em conceitos não abordados pela Matriz Curricular do curso de Tecnologia em Sistemas para Internet, tais quais Programação Lógica, Representações Externas e Ambientes Interativos de Aprendizagem.

8.2 Resultados Esperados

Pretendeu-se contribuir para o ensino na área de Programação de Computadores, mais especificamente no que diz respeito ao conceito de recursão, que pode ser estudado através da utilização da linguagem Prolog. Espera-se que o protótipo resultante atinja a robustez necessária para que seja livremente utilizado de maneira

didática. Assim podem ser elencados como resultados esperados desta pesquisa:

- Esclarecimento de conceitos primordiais para a compreensão do paradigma lógico, como a retroação e recursão, através da utilização das múltiplas representações externas instanciadas no ambiente interativo de aprendizagem desenvolvido;
- Espera-se que o mesmo conjunto de representações seja também útil à consequente evolução dos conceitos adquiridos, até consolidarem perícias na área.
- Extensão dos benefícios que possam surgir à outras linguagens de programação, por meio do embasamento subsidiado pelo ambiente interativo de aprendizado. Visto que muitos dos conceitos sustentados pelo conjunto multirrepresentacional são comuns e podem ser remetidos a outras linguagens e paradigmas;
- Por fim, espera-se também que a pesquisa realizada possa motivar o desenvolvimento de novas ferramentas ou métodos para o ensino de conceitos ainda pouco explorados ou que apresentem um grau de dificuldade elevado. Ou, ainda, estimular o aperfeiçoamento das diversas ferramentas existentes mas que, porventura, possuam um grau reduzido de eficiência.

8.3 Limitações da Solução Desenvolvida

Documentam-se como limitações apresentadas pela solução atingida:

- Conforme implementação do protótipo, percebe-se a necessidade de um aprimoramento do mesmo, a fim de cobrir todos os aspectos da linguagem utilizada (Prolog). Isto se deve ao fato de a biblioteca utilizada como motor de inferência da linguagem não prover, em sua essência, suporte ao rastreamento de predicados nativos, como operadores aritméticos;
- A adequação ao conjunto de múltiplas representações externas ainda é parcial dada deficiência anteriormente citada;
- Com a utilização do protótipo, vislumbra-se a necessidade de um

aperfeiçoamento em sua interface no que diz respeito a aspectos visuais, tornando-a mais amigável ao usuário e facilitando seu uso. Isto seria feito, por exemplo, através da inserção de *tooltips* contendo informações de auxílio ao aprendiz;

- Necessidade de um tratamento de exceções mais completo, a fim de restringir ações do usuário, assim como fornecer um *feedback* adequado às suas ações.

9 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Apresentou-se neste trabalho uma abordagem utilizando um conjunto de múltiplas representações externas como meio de atenuar dificuldades de assimilação de conceitos por parte de aprendizes das áreas de Ciência da Computação e Programação de Computadores.

Primeiramente, foi realizada uma revisão do estado da arte. Foram abordados conceitos referentes a linguagem Prolog, buscando fornecer uma base de conhecimento acerca da mesma, inserindo-a no contexto do Paradigma Lógico de programação.

Baseando-se em estudo realizado por Van Someren (1990) o qual identificou e elencou as principais dificuldades encontradas por aprendizes, propôs-se um conjunto multirrepresentacional.

Ainda, elucidaram-se aspectos referentes à Representações Externas e Ambientes Interativos de Aprendizagem, com destaque ao modelo de Taxonomia Funcional de Múltiplas Representações Externas proposto por Ainsworth (2006) e retomado por Maschio (2007) como ferramenta teórica utilizada.

Em seguida, descreveu-se o formalismo adotado no desenvolvimento do ambiente proposto, categorizando seus componentes e remetendo-os à Taxonomia Funcional previamente citada.

Finalmente, explicitaram-se resultados obtidos e expectativas acerca desta pesquisa.

9.1 Trabalhos Futuros

Durante o desenvolvimento deste projeto, percebeu-se possíveis extensões aplicáveis a trabalhos futuros, com a finalidade de aperfeiçoá-lo:

1. O primeiro aperfeiçoamento a ser implementado é o tratamento adequado de exceções, principalmente as geradas pela biblioteca *GNU Prolog for Java*. Permitindo assim um melhor retorno às ações executadas pelo aprendiz,

permitindo ao mesmo maior proveito na utilização do protótipo;

2. Outra oportunidade de aperfeiçoamento encontra-se na extensão da biblioteca utilizada como motor de inferência da linguagem Prolog, a fim de prover recursos não fornecidos atualmente. Assim será possível o rastreamento de predicados pré-existentes na linguagem, como operadores aritméticos, de atribuição, além do predicado `not`, não cobertos pela implementação atual;
3. Pode-se ainda, prospectar a possibilidade de estender o ambiente interativo de aprendizagem proposto a um ambiente de desenvolvimento específico da linguagem. Isso pode ocorrer permitindo que o aprendiz realize a modificação do arquivo referente a base axiomática diretamente a partir da interface da aplicação. Atualmente, modificações na base axiomática devem ser realizadas de maneira externa. Esta mudança permitiria maior interação e possibilidade do aprendiz permanecer utilizando a ferramenta por mais tempo;
4. Ainda, a adequação do protótipo implementado ao conjunto de múltiplas representações externas proposto é um tópico essencial a ser abordado em futuras implementações. Atualmente, algumas das representações não estão adequadamente implementadas na aplicação, como é o caso do painel de ativação de predicados, que não possui a mudança dinâmica de cor dos predicados sob inspeção em seu corpo.

Por fim, tem-se consciência de que as ideias aqui expostas podem ser de difícil desenvolvimento dentro do modelo proposto, porém, podem servir de base a novas concepções dentro do estudo voltado ao desenvolvimento de aplicações para o Ensino de Programação e Ciência da Computação.

REFERÊNCIAS

ADACHI, Yoshihiro. Prolog Visualization System Using Logichart Diagrams. **CoRR,** p. 8-18. 12 mar. 2009.

AINSWORTH, Shaaron. A functional taxonomy of multiple representations. **Computers and Education**, p. 131-152. 1999.

AINSWORTH, Shaaron; LABEKE, Nicolas Van. Multiple Forms of Dynamic Representation. **Learning And Instruction,** p. 241-255. Jun. 2004.

AINSWORTH, Shaaron. DeFT: A Conceptual Framework For Considering Learning with Multiple Representations. **Learning And Instruction,** p. 183-198. Dez. 2006.

CLOCKSIN, Willian F.; MELLISH, Christopher S. **Programming in Prolog:** using the ISO standart. 5. ed. Berlin: Springer, 2003. 299 p.

HENRY, Michael D.; INOUE, Atsushi. Visual Tracer For Logic Programming. **Proceedings Of The International Symposium On Advanced Intelligent Systems (ISIS2007),** Sokcho, 2007.

MASCHIO, Eleandro. **Uma abordagem metacognitiva através de múltiplas representações externas para o ensino de programação de computadores.** 2007. 94 f. Dissertação (Mestrado) - Universidade Federal do Paraná, Curitiba, 2007.

MULHOLLAND, Paul. Incorporating Software Visualization into Prolog teaching: a challenge, a restriction, and an opportunity. In: ICLP'97 POSTCONFERENCE WORKSHOP ON LOGIC PROGRAMMING ENVIRONMENT, 14., 1997, Leuven. **Proceedings of ICLP'97 Postconference Workshop on Logic Programming Environment.** Mit Press, 1997. p. 33 – 42.

MULHOLLAND, Paul. **A Principled Approach to the Evaluation of SV**: a case-study in Prolog . 1998.

RUSSEL, Stuart J.; NORVIG, Peter. **Inteligência Artificial**: tradução da segunda edição. Rio de Janeiro: Elsevier, 2004. 1021 p.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. 5 ed. São Paulo: Bookman, 2003.

SEBESTA, Robert W. **Concepts of Programming Languages**. 10 ed. New Jersey: Pearson, 2012. 795 p.

SHAFFER, Clifford A. et al. Algorithm Visualization: the state of the field. **Trans. Comput. Educ.,** New York. Ago. 2010.

VAN SOMEREN, Maarten W. What's wrong? Understanding beginners' problems with Prolog. **Instructional Science,** Dordrecht, p. 257-282. 1990.