ELEANDRO MASCHIO KRYNSKI

MODELAGEM DO PROCESSO DE AQUISIÇÃO DE CONHECIMENTO APOIADO POR AMBIENTES INTELIGENTES

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Alexandre Ibrahim Direne

CURITIBA

2013

ELEANDRO MASCHIO KRYNSKI

MODELAGEM DO PROCESSO DE AQUISIÇÃO DE CONHECIMENTO APOIADO POR AMBIENTES INTELIGENTES

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Alexandre Ibrahim Direne

CURITIBA

2013

K94m Krynski, Eleandro Maschio
Modelagem do Processo de Aquisição de Conhecimento Apoiado por Ambientes Inteligentes / Eleandro Maschio Krynski. – Curitiba, 2013.
187f. : il. color. ; 30 cm.

Tese (doutorado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-graduação em Informática, 2013.

Orientador: Alexandre Ibrahim Direne.
Bibliografia: p. 146-159.

1. Ensino auxiliado por computador. 2. Representação do conhecimento. 3. Sistemas tutoriais inteligentes. I. Universidade Federal do Paraná. II. Direne, Alexandre Ibrahim. III. Título.

CDD: 006.332

Ministério da Educação
Universidade Federal do Paraná
Programa de Pós-Graduação em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa do aluno de Doutorado em Ciência da Computação, Eleandro Maschio Krynski, avaliamos a tese de doutorado intitulada *"Modelagem do processo de aquisição de conhecimento apoiado por ambientes inteligentes"*, cuja defesa pública foi realizada no dia 26 de agosto de 2013, às 13:30 horas, no Departamento de Informática do Setor de Ciências Exatas da Universidade Federal do Paraná. Após avaliação, decidimos pela **aprovação** do candidato.

Curitiba, 26 de agosto de 2013.

Prof. Dr. Alexandre Ibrahim Direne
DINF/UFPR – Orientador

Prof. Dr. João Alberto Fabro
UTFPR – Membro Externo

Prof. Dr. Robinson Vida Noronha
UTFPR – Membro Externo

Prof. Dr. Wilson da Silva
DINF/UFPR – Membro Interno

Prof. Dr. Renato José da Silva Carmo
DINF/UFPR – Membro Interno

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

Às mãos imediatas.

AGRADECIMENTOS

Oh, I get by with a little help from my friends,
Mm, I get high with a little help from my friends,
Mm, gonna try with a little help from my friends.
(Paul McCartney)

À vida, pelas lições e oportunidades, por sempre me dar mais motivos para agradecer do que para lamentar e por haver me privilegiado com a amizade das pessoas valiosas que aqui demonstro verdadeira gratidão.

Aos meus pais, Cristian e Joceline, e à minha irmã, Christielli, pelo amparo, cuidado e incentivo sempre.

À minha namorada, Carolina, meu maior sorriso, pelo simples amor verdadeiro. Amo você!

À minha família como um todo, pela receptividade e confiança transmitida.

Às amizades de uma vida inteira, por compartilharmos alegres momentos e auxiliarmo-nos naqueles mais difíceis.

Ao professor Alexandre Direne, orientador e grande amigo, pelo esforço e compreensão em acolher alguém sem dedicação exclusiva. Sequer faz ideia do quanto sou grato. Inspiro-me no seu trabalho.

Aos membros da banca examinadora pela cordialidade com que atenderam ao convite. Em especial, aos professores Andrey Ricardo Pimentel e Robinson Vida Noronha (UTFPR), integrantes da banca de qualificação, pelas sugestões para aprimorar a pesquisa.

Aos professores André Guedes e Renato Carmo, pelas intervenções breves e pelos paradigmas desconstruídos. Gostaria que mais docentes tivessem comparáveis imparcialidade e maturidade científica.

Ao professor André Raabe (Univali) pela gentil colaboração com a pesquisa.

Ao professor Jomar Antonio Camarinha Filho, do Departamento de Estatística (UFPR), pela presteza e espontaneidade no auxílio quanto aos métodos estatísticos.

Ao Departamento de Informática da Universidade Federal do Paraná, pelo profissionalismo e comprometimento com suas atividades. Agradecimento especial à Jucélia pelo ânimo e prontidão com que nos ajuda.

Aos meus amigos doutorandos Alexandre Feitosa, Diego Marczal, Maici Leite e Tarcizio Alexandre Bini pelo apoio mútuo e decisivo em tantos momentos. Prosperem!

Aos professores Renata Carneiro Gomes (UTFPR) e Fábio Hernandes (Unicentro) pela visão coletiva e colaborativa institucional. Ajudas pontuais que sustentaram muito desta caminhada.

Aos demais professores e funcionários da Universidade Tecnológica Federal do Paraná, Câmpus Guarapuava, pelo companheirismo e suporte, principalmente nos momentos finais da pesquisa.

Aos acadêmicos dos cursos de Tecnologia em Sistemas para Internet (UTFPR) e Ciência da Computação (Unicentro). Obrigado pela amizade e pelo aprendizado.

À professora Adriana Dalla Vecchia (Faculdade Campo Real) pelo esmero na revisão dos textos que compõem o presente documento.

À Jyloo Software (Tussenhausen, Alemanha) pela concessão de licença, para propósitos científicos, dos componentes de interface Synthetica.

A todos que colaboraram e, por pedirem nada em troca, eu não tenha sequer me dado conta. Um obrigado sincero, espero retribuir.

Vos 10,000 premières photos sont vos pires.

Suas primeiras 10.000 fotografias são as suas piores.

(Henri Cartier-Bresson)

SUMÁRIO

LISTA DE SIGLAS E ACRÔNIMOS

ADG	Ambiente de Descoberta-Guiada
AIA	Ambiente Interativo de Aprendizagem
API	*Application Programming Interface* (Interface de Programação de Aplicativos)
BALSA	*Brown University Algorithm Simulator and Animator* (Simulador e Animador de Algoritmos da Universidade de Brown)
BIP	*BASIC Instructional Program*
CC	Complexidade Ciclomática
CE	Complexidade Estrutural
CMU	*Carnegie Mellon University* (Universidade de Carnegie Mellon)
Cusp	*Customer Programming*
DeFT	*Design, Functions, Tasks* (Projeto, Funções, Tarefas)
DI	Detalhes de Implementação
EA	*Electronic Arts*
GC	Grupo de Controle
GE	Grupo Experimental
GIL	*Graphical Instruction in LISP*
IA	Inteligência Artificial

IAC	Instrução Assistida por Computador
IDE	*Integrated Development Environment* (Ambiente Integrado de Desenvolvimento)
IHC	Interação Humano-Computador
IIAC	Instrução Inteligente Assistida por Computador
JVM	*Java Virtual Machine* (Máquina Virtual Java)
Lisp	*List Processing*
LOC	*Lines of Code* (Linhas de Código)
MIT	*Massachusetts Institute of Technology* (Instituto de Tecnologia de Massachusetts)
MREs	Múltiplas Representações Externas
MÚLTIPLA	Multiplicador de Linguagens Tipificado em Plataforma de Aprendizagem
OBM	Olimpíada Brasileira de Matemática
PARC	*Palo Alto Research Center* (Centro de Pesquisa de Palo Alto)
PDF	*Portable Document Format* (Formato de Documento Portátil)
RE	Representação Externa
RGB	*Red, Green, Blue* (Vermelho, Verde, Azul)

RI	Representação Interna
SBIE	Simpósio Brasileiro de Informática na Educação
STI	Sistema Tutor Inteligente
TPM	*Transparent Prolog Machine*
UFPR	Universidade Federal do Paraná
UML	*Unified Modeling Language* (Linguagem Unificada de Modelagem)
Unicentro	Universidade Estadual do Centro-Oeste
Univali	Universidade do Vale do Itajaí
UTFPR	Universidade Tecnológica Federal do Paraná
W4AP	Workshop de Ambientes de Apoio à Aprendizagem de Algoritmos e Programação
WYSIWYG	*What You See Is What You Get* (O que você vê é o que você obtém)
XML	*Extensible Markup Language* (Linguagem de Marcação Extensível)

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A presente tese concentra-se em descrever como o conhecimento de alto-nível sobre aspectos de experiência humana pode ser modelado, representado e, então, interpretado a fim de suportar interações de ensino e aprendizagem. A maioria das pesquisas anteriores na tutoria de Programação de Computadores tendia a se concentrar em princípios teóricos sobre a aquisição de experiência. Houve poucas implementações, ainda relacionadas a domínios específicos. Entretanto, o problema de prover uma epistemologia para a descrição de conhecimento sobre enunciados e estágios de conhecimento de aprendizes foi negligenciado. Esta pesquisa trata esse problema por meio de *(1)* um método baseado em grafos genéticos para gerenciar a complexidade na autoria de enunciados didáticos e *(2)* um processo geral para a modelagem dinâmica do conhecimento de aprendizes através da sobreposição das capacidades do domínio previamente descritas. O modelo e o método são ambos suportados por protótipos de ferramenta que foram implementados e integram o ambiente MÚLTIPLA. A avaliação do método e das ferramentas propostos foi realizada por estudos empíricos focados na generalidade dos grafos genéticos como linguagem de autoria com o objetivo de prover um arcabouço unificado para o desenvolvimento de perícias.

Palavras-chave: ensino de programação de computadores, representação de conhecimento, sistemas tutores inteligentes, ambientes interativos de aprendizagem, múltiplas representações externas.

ABSTRACT

This thesis describes how high-level knowledge about human expertise features can be modeled, represented and further interpreted to support learning and tutoring interactions. Most past work in the tutoring of computer programming has tended to concentrated on the theoretical principles of how humans acquire expertise. The few implementations there have been are domain-specific. However, the problem of providing an epistemology for describing knowledge about problem statements and students' states of belief has been neglected. We treat these problems through *(1)* a method based on genetic graphs for managing the complexity of courseware authoring of problem statements and *(2)* a general process for dynamic modeling a learner's knowledge by overlaying it against the domain expertise features. The method and the model are both supported here by implemented prototype software tools that integrate the educational environment MULTIPLA. To evaluate the method and tools, empirical observations have been carried out, focusing on the generality of genetic graphs as an authoring language to provide an unified expert-student framework for developing skills.

Keywords: teaching of computer programming, knowledge representation, intelligent tutoring systems, interactive learning environments, multiple external representations.

CAPÍTULO 1

INTRODUÇÃO

1.1 Problema Central

As disciplinas introdutórias de Programação de Computadores culminam grande responsabilidade em um curso de Ciência da Computação. Cabem-lhes a introdução dos princípios e consequente iniciação ao desenvolvimento de perícias na área de programação. Geralmente isso se faz em tempo restrito, caso considerada a exigência cognitiva da aprendizagem, mediante a intercalação de abordagens conceituais e práticas. O legado do trabalho nas citadas disciplinas (tanto competências quanto deficiências) desencadeia-se por entre todo o currículo do curso e tende a desembocar na prática profissional do egresso.

Contudo, conforme revisão feita sobre as pesquisas da área, pouco se investigou acerca do desenvolvimento de habilidades específicas do estereótipo de um programador perito. Nesse sentido, [84] manifesta-se contra tal desatenção, ressaltando a íngreme curva de aprendizado enfrentada por um iniciante que objetiva proficiência na área. Corrobora tal fato a vasta literatura que anuncia ensinar programação em poucos dias, se não em horas [1, 58, 94, 20, 111].

O citado estudo expõe maiores agravantes quando se trata não da suficiência, mas da excelência em determinado domínio de conhecimento. A exemplo disso, [49] documenta uma pesquisa, realizada com estudantes da Academia de Música de Berlim, que compara o tempo destinado à prática com o nível atingido pelos indivíduos. Constatou-se que todos iniciaram a tocar por volta dos cinco anos de idade. Nos primeiros anos, praticavam de duas a três horas semanais. Em torno dos oito anos de idade, diferenças reais apareceram. Aqueles que terminaram como melhores de suas turmas começavam a praticar mais do que o restante: 6 horas semanais aos 9 anos, 8 horas aos 12, 16 horas aos 14, aumentando para mais de 30 horas semanais de prática aos 20 anos. Nesta idade, os instrumentistas de elite totalizavam uma média de 10.000 horas de prática musical. Em contraste, aqueles

que eram somente bons acumularam 8.000 horas e os que não se destacaram, pouco mais de 4.000 horas.

Assim, [84, 39] apontam a prática deliberada como elemento decisivo no desenvolvimento de perícias em uma grande variedade de áreas. Sugere o desafio de tarefas que vão um pouco além da habilidade corrente do aprendiz, a fim de que ele tente desempenhá-las, analisar a própria performance durante e depois de realizá-las para, então, corrigir possíveis erros. Outro estudo [63] reforça que um aprendizado mais efetivo requer tarefas bem definidas, com nível de dificuldade apropriado ao aprendiz em particular, bem como *feedback* informativo, oportunidade para repetição e correção de erros.

Dessa forma, consta-se o sequenciamento dos enunciados[1] a serem oferecidos ao aprendiz como um dos elementos pedagógicos cruciais ao desenvolvimento de perícias. Assim, prover uma modelagem precisa, tanto do conhecimento do domínio quanto do aprendiz, para subsidiar decisões nesse âmbito, perfaz uma contribuição relevante na área de Inteligência Artificial (IA).

A critério de exemplo, os enunciados presentes nas figuras 1.1, 1.2 e 1.3 estão ordenados conforme a carga cognitiva exigida na resolução, da menos complexa para a mais complexa. Supondo a resolução do segundo enunciado pelo aprendiz, seria interessante que um Sistema Tutor Inteligente (STI) dispusesse de mecanismos que, conforme a performance na resolução, fosse ao encontro da necessidade de retroceder ao primeiro enunciado ou de avançar ao terceiro enunciado.

Tais mecanismos somente são possíveis mediante uma modelagem apurada e convencionada que situacione o aprendiz frente ao conhecimento do domínio. Faz-se também necessário que se classifique cada enunciado, conforme a fração exercitada do conhecimento.

Convém destacar que a modelagem de aprendizes não se trata de um conceito recente no campo de STIs. Contudo, pesquisas anteriores tendiam a se concentrar em questões pedagógicas em detrimento das psicológicas [93, 29, 48, 85, 112]. Como consequência, os esforços se destinaram à proposição de arcabouços que estabelecessem relações de arqui-

[1]Termo usado neste documento para referir-se, de maneira genérica, a exemplos e exercícios de um determinado domínio de conhecimento.

Enunciado 1:
Construa um programa que, fornecida a idade do indivíduo, responda se a maioridade já foi atingida.

```
1  program maioridade;
2  var
3     idade : integer;
4  begin
5     idade := 0;
6     writeln('Digite a sua idade:');
7     readln(idade);
8
9     if (idade >= 18) then
10       writeln('Atingiu a maioridade.')
11    else
12       writeln('Não atingiu a maioridade.');
13 end.
```

Figura 1.1: Maioridade (Enunciado 1)

Enunciado 2:
Construa um programa que, fornecida a idade do indivíduo, responda se o voto é obrigatório. No Brasil, a obrigatoriedade da votação incide sobre os cidadãos entre 18 e 70 anos.

```
1  program voto_obrigatorio;
2  var
3     idade : integer;
4  begin
5     idade := 0;
6     writeln('Digite a sua idade:');
7     readln(idade);
8
9     if ((idade >= 18) and (idade <= 70)) then
10       writeln('Voto obrigatório.')
11    else
12       writeln('Voto não obrigatório ou não
13                permitido.');
14 end.
```

Figura 1.2: Voto obrigatório (Enunciado 2)

Enunciado 3:
Construa um programa que, fornecida a idade do indivíduo, responda se o voto é obrigatório, facultativo ou não permitido. No Brasil, a obrigatoriedade da votação incide sobre os cidadãos entre 18 e 70 anos. O voto é facultativo para aqueles que têm entre 16 e 18 anos, ou mais de 70 anos.

```
1  program votos;
2  var
3     idade : integer;
4  begin
5     idade := 0;
6     writeln('Digite a sua idade:');
7     readln(idade);
8
9     if ((idade >= 18) and (idade <= 70)) then
10       writeln('Voto obrigatório.')
11    else
12       if (idade < 16) then
13          writeln('Voto não permitido.')
14       else
15          writeln('Voto facultativo.');
16 end.
```

Figura 1.3: Classificação do voto (Enunciado 3)

tetura entre os modelos pedagógico e do aprendiz.

Em Programação de Computadores, adicionalmente, percebe-se que pouco se fez acerca do suporte à aquisição de conhecimento no entremeio de princípios e perícias [98]. Algumas tentativas, no entanto, foram desempenhadas a fim de assistir, tanto passiva quanto ativamente, ao aprendiz em ambientes interativos no domínio. Especificamente em caráter passivo, houve construção de depuradores que se encaixassem no ensino de programação e permitissem ao aprendiz conduzir experimentos de execução simbólica do código fonte.

Porém, ainda resta um enorme vazio de pesquisa original a ser coberto para oferecer acompanhamento ao progresso das capacidades do aprendiz, como também para monitorar adequadamente experimentos através de representações compatíveis com as habilidades que se desenvolvem. Um caminho promissor foi proposto sob a denominação de Múltiplas Representações Externas (MREs) [5], tendo considerável sucesso na tentativa de correlacionar descrições de conhecimento internas com externas em ambientes de aprendizagem, inteligentes ou não.

Além de ressaltar tal compatibilidade, influenciando assim uma recente linha de projeto dos ambientes interativos de aprendizagem, a utilização de um conjunto multirrepresentacional no ensino tende a desembocar numa abordagem metacognitiva [24] por parte do instrutor e do aprendiz. Em teorias cognitivas aplicadas à educação, isso geralmente implica em uma divisão mais equitativa de responsabilidades e aprimora o desempenho naquilo que se ensina.

A partir disso, em contexto mais amplo, pode ser visto como desafio encontrar meios efetivos que propiciem o aprendizado, principalmente em Programação de Computadores. Na atualidade, os aprendizes têm o cotidiano imerso em tecnologias das mais variadas e sessões tradicionais de ensino mostram-se insuficientes para atraí-los e motivá-los ao estudo. Entretanto, pela característica imersiva, o ensino baseado em computadores tende a sobrepor muitas das abordagens obsoletas. Como consequência, a superação do platô constatado na modelagem do aprendiz tende a se mostrar naturalmente como o passo seguinte das pesquisas em Inteligência Artificial e Educação.

Diante do exposto, acredita-se que esforços voltados à intersecção de áreas aqui resenhadas podem contribuir de forma relevante para enriquecer o ensino de Programação de Computadores. Ademais, a pesquisa realizada procura remeter muitos dos resultados à generalização para a aprendizagem em domínios de natureza prática e complexa. Estima-se, pelas deficiências constatadas no nicho específico e pelas evidências apresentadas no decorrer do documento, que se atingiu uma contribuição relevante para a Ciência da Computação.

1.2 O Ambiente Interativo de Aprendizagem MÚLTIPLA

O protótipo MÚLTIPLA (Multiplicador de Linguagens Tipificado em Plataforma de Aprendizagem) [74] consiste em um ambiente interativo de aprendizagem destinado à aquisição de conhecimento na fronteira entre princípios e perícias de Programação de Computadores (Figura 1.4). Constituiu a dissertação de mestrado do proponente e se estabelece como início das pesquisas aqui realizadas em nível de doutorado.

O ambiente permite que o aprendiz construa algoritmos na representação gráfica de fluxograma e observe a geração automática do código equivalente na linguagem Pascal. Além disso, também há a possibilidade de que o aprendiz conduza experimentos de execução simbólica do algoritmo construído. Para a implementação do protótipo, trabalhou-se somente com um subconjunto da linguagem Pascal: os tipos booleano, inteiro, real e caractere; as instruções de entrada, saída e atribuição; a estrutura de seleção *se* (*if*); e as estruturas de repetição *enquanto-faça* (*while-do*), *repita-até* (*repeat-until*) e *para* (*for*).

Constrói-se um algoritmo por meio da seleção dos elementos de fluxograma em *menus* de contexto e da digitação das respectivas expressões em caixas de diálogo. Cada alteração feita no fluxograma é automaticamente transposta para o código fonte em Pascal, exibido no lado esquerdo. A declaração de variáveis foi alocada em uma tela independente.

O MÚLTIPLA oferece duas opções de execução: em modo passo-a-passo e temporizado. Ambas realçam o elemento de fluxograma e a linha de código onde se encontra o fluxo de controle, bem como sincronizam tais informações com a visualização da janela de saída e das células de memória (valores das variáveis).

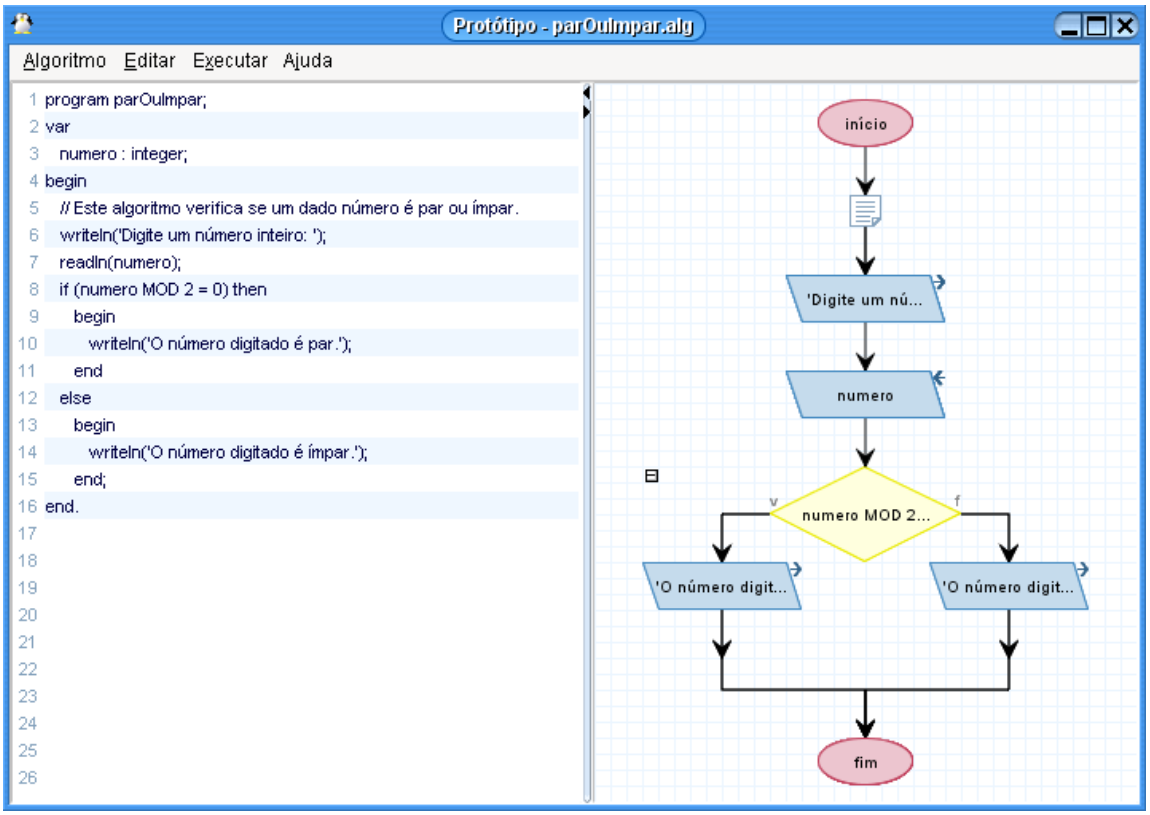


Figura 1.4: Tela do ambiente interativo de aprendizagem MÚLTIPLA

Também se evidencia que tanto a construção quanto a execução dos algoritmos são assistidas por múltiplas representações externas que atuam como ferramentas de visualização de diferentes aspectos desses algoritmos. Além daquelas acima mencionadas, destacam-se: cliques de correspondência entre a notação de fluxograma e o código em Pascal (o inverso também ocorre), recolhimento e expansão de blocos de elementos, cores distintas para cada categoria de elementos, comentários textuais e pontos de interrupção.

Investiu-se também na avaliação formal do MÚLTIPLA junto a aprendizes, no intuito de aferir os benefícios de utilização do protótipo desenvolvido. Como resultado subsequente, a coleta de informações a partir desse uso inicial tem validado a hipótese teórica que embasou o protótipo, como também tem fornecido indicativas de alterações para o aperfeiçoamento do referido ambiente.

Como passo imediato, motivador desta pesquisa, pretende-se futuramente investir em técnicas de Inteligência Artificial que elevem o nível do apoio fornecido pelo MÚLTIPLA de sintático para semântico. Intenciona-se propiciar um certo grau de automatização na

progressão dos enunciados a serem explorados.

Embora o MÚLTIPLA tenha sido citado como instância de aplicação, os aspectos de modelagem atingidos pela pesquisa pretendem ser gerais ao desenvolvimento de perícias em domínios de natureza prática.

1.3 Hipótese de Pesquisa

A hipótese principal desta tese apregoa que o conceito estruturado de múltiplas representações externas é adequado como complementação pedagógica que os micromundos podem oferecer. Adicionalmente, a técnica de representação de conhecimento por grafos genéticos (*overlay graphs*) oferece um formalismo de base computacional sólida para a modelagem de uma vasta gama de habilidades de um programador humano. Tais habilidades podem ser definidas por um autor de material eletrônico usando ferramentas específicas de autoria.

1.4 Objetivos Gerais e Contribuição

Diante do exposto, pretende-se tratar do problema apresentado investigando sobre como ocorre o desenvolvimento de perícias em Programação de Computadores, e também elevar os resultados obtidos à generalização para outros domínios de natureza prática e complexa. Propõe-se, portanto, explorar e construir modelos que atuem na especificação do processo de desenvolvimento pericial em Sistemas Tutores Inteligentes.

Para esse fim, busca-se embasamento, conforme resenha literária, em **(1)** abordagens sobre a capacidade de perícia, **(2)** ambientes de apoio ao aprendizado de Programação de Computadores e **(3)** modelagem de aprendizes em Sistemas Tutores Inteligentes. Conforme revisão feita acerca do estado da arte nesse conjunto de áreas, observou-se a inexistência de citações relacionadas ao foco específico desta tese. Considera-se, portanto, como um indicativo de relevância da ideia aqui apresentada.

Foram firmados como objetivos específicos pertinentes à hipótese apresentada:

1. Realizar experimento de avaliação formal do ambiente MÚLTIPLA em situações reais de aprendizagem a fim de aferir os benefícios de utilização do protótipo desenvolvido. Pretendeu-se atestar que o formalismo subjacente potencializa o estudo na área e mostra-se benéfico por empregar MREs no suporte a uma abordagem metacognitiva de ensino;

2. Estudar a extensão do formalismo, e consequentes benefícios, para o processo de autoria. Foram especialmente considerados os seguintes aspectos em domínios de natureza prática e complexa:

   (a) Definição das capacidades do domínio;

   (b) Elicitação e catalogamento de enunciados;

   (c) Modelagem do aprendiz;

   (d) Modelagem do processo de aprendizagem;

3. Formalizar um arcabouço conceitual de modelagem que subsidie o processo de desenvolvimento de perícias em Programação de Computadores;

4. Definir, como instância de aplicação do formalismo, um subconjunto das capacidades do domínio de Programação de Computadores, ou seja, das perícias que compõe o estereótipo de um programador experto[2];

5. Implementar, como validação de parte desta pesquisa, protótipos de ferramentas suportadas pelo formalismo proposto, sendo destinadas à:

   (a) definição do conhecimento do domínio por meio do detalhamento das capacidades componentes;

   (b) catalogação e elicitação de enunciados definidos em termos das capacidades do domínio descritas;

[2]Do latim *expertus*, denota uma pessoa com vasto conhecimento e capacidade em um domínio específico. Perito.

6. Disseminar, por meio de publicações da área, o conhecimento alcançado para que some, como referência, a outras pesquisas na contribuição com o estado da arte das áreas abordadas.

1.5 Estrutura da Tese

Depois desta introdução, no Capítulo 2, a resenha literária compreende abordagens sobre múltiplas representações externas, capacidades da perícia, ambientes de apoio ao ensino e à aprendizagem de Programação de Computadores e modelagem de aprendizes em Sistemas Tutores Inteligentes. O Capítulo 3 transcreve o experimento realizado com o ambiente MÚLTIPLA, cujos resultados amparam o início das pesquisas deste doutorado. Cabe ao Capítulo 4 formalizar o arcabouço conceitual assumido na solução do problema abordado, apresentando o embasamento em grafos genéticos e na proposta de um modelo de sobreposição para Programação de Computadores. Descrevem-se, no Capítulo 5, as ferramentas de autoria prototipadas que o formalismo adotado suporta.

No Capítulo 6, discutem-se as repercussões esperadas da pesquisa desenvolvida. Apresenta-se, no Capítulo 7, as conclusões alcançadas, bem como a perspectiva de trabalhos futuros. Os anexos encerram este documento.

CAPÍTULO 2

RESENHA LITERÁRIA

Concerne a este capítulo um levantamento de pesquisas relacionadas e a consequente avaliação crítica de trabalhos que abordaram temas correlatos ao problema tratado pela tese em defesa.

A revisão bibliográfica deste projeto embasa-se em quatro alicerces principais, detalhados no presente capítulo: Múltiplas Representações Externas (2.1), abordagens sobre capacidades de perícia (2.2), ambientes de apoio ao ensino e aprendizagem de Programação de Computadores (2.3) e, por fim, modelagem de aprendizes em Sistemas Tutores Inteligentes (2.4).

2.1 Múltiplas Representações Externas

O termo Representação Externa (RE) [28, 117], no escopo da corrente pesquisa, refere-se ao uso de técnicas para representar, organizar e apresentar conhecimento. Remete, portanto, a uma ampla variedade de representações que compreende desde modelos proposicionais/sentenciais até modelos gráficos/diagramáticos. Preenchendo a lacuna entre esses extremos, há representações intermediárias que associam elementos gráficos e textuais (e.g. tabelas, que denotam matrizes graficamente). Logo, são exemplos de RE: sentenças em linguagem natural, sentenças em linguagens formais (como lógica de primeira-ordem), tabelas, listas, grafos, mapas, projetos, diagramas, animações e até mesmo as realidades aumentada e virtual.

Segundo definição [86], qualquer RE deve ser descrita em termos:

1. do mundo representado;
2. do mundo representante;
3. de quais aspectos do mundo representado estão sendo expressos;

4. de quais aspectos do mundo representante compõem a modelagem;

5. da correspondência entre os dois mundos.

Como consequência do bom emprego de REs nas tarefas do dia-a-dia, tais técnicas tornaram-se um suporte valioso à resolução de uma diversidade de problemas de caráter mais formal, bem como um poderoso auxílio para diminuir a carga cognitiva inerentemente requerida. A aplicação de REs, portanto, passou a ser um amparo útil e promissor em várias áreas do conhecimento humano.

Muitos creem que o único benefício do uso de REs seja facilitar o processo de memorização, reduzindo a carga cognitiva necessária para se realizar determinada tarefa. Embora essa seja a característica mais notável, observam-se ainda as seguintes propriedades [117] permeando REs:

1. prover informações possíveis de serem diretamente percebidas e utilizadas sem nenhuma interpretação ou formulação explícita;

2. fixar e estruturar comportamento cognitivo, porquanto a estrutura física das REs restringe a gama de comportamentos (alguns são permitidos e outros proibidos);

3. modificar a natureza das tarefas em questão, tornando-as potencialmente mais fáceis de serem realizadas.

Além do supracitado, há evidências [5, 28] expondo vantagens da utilização de REs como suporte tanto ao ensino quanto à resolução de problemas. Estudos ressaltam que se eleva o desempenho do aprendiz ao proporcionar-lhe interação com REs apropriadas. Nesse sentido, ultimamente as pesquisas se concentram na combinação de diferentes REs para intermediar o aprendizado. Múltiplas Representações Externas (MREs) [5, 6] denominação dada ao conjunto, tem se mostrado um nicho próspero no atendimento às necessidades cognitivas de aprendizes diversos.

Os estudos precursores restringiam-se a investigar acerca da influência que a apresentação de imagens ao longo da informação escrita poderia exercer sobre a compreensão

de textos [5]. Posteriormente, a discussão foi estendida de modo a abranger uma ampla variedade de representações que incluía até mesmo som, vídeo, animação e simulação dinâmica. Na época, pesquisadores punham-se a indagar se representações específicas ofereciam vantagens equivalentes ao ensino e à resolução de problemas.

Houve outras pesquisas abordando exclusivamente a potencialidade e os possíveis problemas associados à aprendizagem com, simultaneamente, mais de uma RE [7]. Estas sustentam que MREs relacionadas resultam num aprendizado mais flexível e intuitivo, dando espaço a diferentes ideias e estratégias. Contudo, também se constatou que aprendizes podem não ter êxito em se beneficiarem das vantagens anteditas, pois nem sempre são facilmente atingidas.

Não obstante, generalizações concernentes à eficácia de MREs no aprendizado são um tanto difíceis de serem alcançadas [7] porque, para isso, se faz necessária a predição das condições sobre as quais MREs realmente são vantajosas. Ainda não são evidentes mesmo questões de natureza mais primária, como as circunstâncias sob as quais o desempenho da solução de problemas por MREs apresenta melhora [28]. Indaga-se, por exemplo, se a troca de representações durante o raciocínio é cognitivamente admissível.

Estudos específicos [28] relataram o emprego de REs em ambientes interativos de aprendizagem destinados à resolução de problemas. Constataram que sistemas como Bridge, Geometry Tutor e GIL valiam-se de grafos de prova como recurso metacognitivo, tendo-os como meio para tornar o processo de planejamento mais evidente. O primeiro deles, **Bridge**, abordava o detalhamento de planos em alusão simplificada a algoritmos. Já o segundo, **Geometry Tutor**, dedicava-se a provas geométricas no ensino colegial. O último, **GIL**, voltava-se ao ensino da linguagem Lisp.

Foram também relacionados, pelos mesmos estudos, ambientes em que o raciocínio com REs era essencial à atividade de ensino subjacente. **Hyperproof**, para lógica de primeira-ordem, e **Algebra (Word Problem Tutor)**, para habilidades básicas de manipulação algébrica, são exemplos disso [28]. Finalmente, na mesma linha de estimular o pensamento reflexivo por meio de uma abordagem de causa-efeito, o ambiente **SimForest** [82] simula o crescimento de árvores em florestas. Inovou ao introduzir em ambientes interativos de

aprendizagem os conceitos de caixa-preta e caixa-de-vidro para permitir que o aprendiz ajustasse o nível de detalhamento da representação dos mecanismos que conduzem a simulação.

Constatou-se também a incidência de MREs como recurso de depuração e/ou visualização em ambientes não-educacionais [98]. Evidência disso são os termos "visualização de algoritmos", "visualização de programas" e "visualização de *software*", todos análogos e referentes à re-representação gráfica de aspectos do código textual.

Diante do exposto, a utilização de um conjunto multirrepresentacional, no processo de ensino e aprendizagem de determinada área, tende a desembocar em uma abordagem metacognitiva por parte do instrutor e do aprendiz. Tal abordagem, amplamente estudada pela Informática na Educação [47], vem sendo apontada como progressista por delegar a quem aprende maior responsabilidade sobre o próprio aprendizado e, mediante essa autonomia, conseguir influenciar positivamente o desempenho naquilo que se ensina.

Embora não se defina metacognição de maneira unívoca, na intenção desta pesquisa refere-se ao conhecimento que o indivíduo tem sobre o próprio conhecimento [44], bem como sobre os próprios processos cognitivos, as formas de operá-los e a capacidade de controlá-los [12, 42]. Nuclearmente, diz respeito à "cognição da cognição", sendo portanto "um termo novo para uma ideia antiga, isto é, aprender a aprender" [12].

Uma abordagem metacognitiva, por tudo isso, ajuda o estudante a identificar e utilizar abordagens individuais de aprendizado [24], que podem ser as mais diversas, como realçar texto, fazer anotações ao longo do conteúdo, sintetizar em esquemas gráficos, elaborar resumos, procurar por palavras-chave, encontrar exemplos externos, explicar em voz alta, realizar experimentos, entre outros. Em consequência, remete-se ao aprendiz parte do controle sobre o próprio aprendizado, criando um vínculo de envolvimento muito mais intenso e ativo do que o proposto por abordagens tradicionais.

Considera-se, por fim, que coube a esta seção retomar os aspectos de MREs pertinentes à pesquisa atual. Um estudo pormenorizado em uma abordagem metacognitiva foi anteriormente realizado em [74] e embasou o ambiente interativo de aprendizagem MÚLTIPLA.

2.2 Abordagens Sobre Capacidades da Perícia

Faz-se, nesta seção, um apanhado geral sobre a maneira como se desenvolve perícia em domínios de natureza prática (2.2.1). Em seguida, descreve-se sobre a projeção destas considerações no âmbito da Programação de Computadores (2.2.2).

2.2.1 Desenvolvimento de Perícias em Domínios de Natureza Prática

O processo de aprendizagem em domínios de natureza prática e complexa fundamenta-se, segundo Lesgold [65], na aquisição de conhecimentos sobre **princípios** e consequente desenvolvimento destes até consolidarem **perícias** na área. As pesquisas de Lesgold centraram-se inicialmente no âmbito da Radiologia Médica. Entretanto, a contribuição logo foi verificada como genérica a domínios de natureza prática, como o Xadrez [41, 3, 73] e a Programação de Computadores [92]. Além disso, outras pesquisas seguiram o trabalho inicial ainda na área de Radiologia Médica [33].

No Xadrez [41, 3], por exemplo, a **aquisição de princípios** corresponde à assimilação das regras pelo aprendiz. Tais regras são conhecidas como as Leis do Xadrez e concentram aspectos como:

- configuração do tabuleiro;
- distinção, valor, movimento e captura de cada peça;
- movimentos extraordinários (roque, tomada *en passant* e promoção de peão);
- conceito de xeque-mate;
- critérios de vitória e de empate; e
- reconhecimento de movimentos irregulares.

Trata-se, portanto, de conhecimento formal (científico, em alguns casos) repassado inicialmente na formação do aprendiz. Princípios são os fundamentos do domínio ao qual se referem.

O **desenvolvimento de perícias**, por sua vez, diz respeito à construção do conhecimento experiencial, ou seja, a conquista de habilidade pela prática. Abrange a integração dos princípios aprendidos com a experiência obtida até o momento e, ainda possivelmente, campos distintos de conhecimento.

A perícia no Xadrez [41, 3] incorpora os aspectos de estratégia e táticas de jogo, desenvolvidos ao longo do tempo. A estratégia enxadrística consiste em definir e atingir objetivos de longo prazo durante uma partida (e.g. onde posicionar diferentes peças), enquanto a tática se concentra em manobras imediatas no tabuleiro. Essas duas componentes do raciocínio pericial no Xadrez não podem ser completamente separadas, uma vez que objetivos estratégicos são atingidos majoritariamente por meio de táticas, como também a razão das táticas se baseia em uma estratégia prévia.

Embora o conhecimento de princípios seja semelhante tanto em peritos (expertos) quanto em aprendizes, o conhecimento experiencial apresenta várias e contundentes diferenças. Em analogia, os princípios funcionam como um conjunto de ferramentas básicas disponíveis enquanto a perícia corresponde à destreza de manipulá-las. Mesmo que um aprendiz disponha das mesmas ferramentas, há distinção na habilidade com que as emprega na resolução de um problema.

Resumidamente, o desenvolvimento de perícias ocorre através da exposição do aprendiz a casos exemplares e não exemplares [33, 65]. Compreende-se cada caso, ao qual se expõe o aprendiz, como uma oportunidade para o desenvolvimento de perícias. A aprendizagem, portanto, acontece pela indução a partir desses casos. No Xadrez [3], um caso exemplar é a *harmonia de peões* que aponta como vantajosa, por enxadristas peritos, a disposição harmônica de peões no tabuleiro. Outro caso exemplar consiste no *domínio de centro* e considera como característica favorável dominar as quatro casas situadas no meio do tabuleiro, uma vez que as peças nesse território têm o raio de ação maximizado.

Nesse sentido, a pesquisa de Lesgold [65] foi um contraponto às outras teorias sobre o aprendizado até então. Estas focavam estudos curtos em laboratórios de psicologia e basicamente avaliavam pequenas alterações de velocidade e performance na resolução de problemas por aprendizes. Lesgold, entretanto, concentrou-se na precisão e na natureza

qualitativa do desempenho, estendendo-se por entre longos períodos de aprendizado.

Diante disso, Lesgold propôs que a escala de tempo da aquisição de princípios e da evolução de perícias fosse trazida para o campo da Psicologia do Desenvolvimento. Uma consequência direta seria que a variação das capacidades fosse observada no decorrer de anos ao invés de minutos.

Considerando-se cada caso como um experimento, no sentido do estudo psicológico clássico do aprendizado, o trabalho de Lesgold instanciado na Radiologia Médica [65] abordou entre 10.000 e 20.000 experimentos. Cita-se que um médico residente (aprendiz, neste escopo) tem contato com cerca de 40 casos por dia. Um radiologista perito, por sua vez, observa entre 65 e 70 casos no mesmo período.

Ainda que Lesgold [65] tenha focado a natureza da perícia em Radiologia Médica, a pesquisa verificou o contraste entre aprendizes e peritos em outros domínio de conhecimento. Em se tratando da resolução de problemas por peritos, constatou-se:

1. Aprendizes mais avançados tinham menor propensão ao diagnóstico correto do que os menos avançados. Portanto, a performance não figura em proporção crescente à experiência. Exemplo disso são crianças aprendendo a conjugação de verbos. Em um primeiro momento, produzem instâncias corretas de verbos irregulares. Depois, passam a conjugar verbos irregulares como regulares, de maneira incorreta. Por fim, alcançam controle sobre as irregularidades e cessam com os erros;

2. Um perito investe relativamente mais tempo em construir uma representação do problema antes de procurar por uma solução. Embora o aprendiz demore mais para alcançar uma solução, dedica apenas uma pequena parcela do tempo gasto para gerar a representação inicial do problema. Há domínios de conhecimento que mesmo a quantidade absoluta de tempo dedicada a elaborar tal representação é maior em peritos;

3. Como consequência, um plano de resposta com grande possibilidade de estar, pelo menos, no espaço correto de solução é gerado mais rapidamente por um perito. Esse plano tem utilidade em guiar o processamento futuro, incluindo a construção de

uma representação básica do problema;

4. Peritos são capazes de adaptar um plano às especificidades do caso em questão. Tal habilidade também permite que peritos testem, de maneira mais completa, se o plano inicial estava correto;

5. Outra constatação, mais evidente, assinala a menor eficiência de um aprendiz em modificar um plano em resposta frente a novas informações. Isso, se comparado a um perito que apresenta a devida flexibilidade de raciocínio;

6. A restrição do aprendiz a respostas óbvias também foi um aspecto preponderante no contraste com peritos. Uma possível explicação, segundo Lesgold, supõe que, quando existem uma hipótese dominante e uma possibilidade mais remota, considerar a segunda depende da capacidade de processos mentais subjacentes (fomentados por experiência). A ineficiência de algum desses processos ajuda a desconsiderar a suposição menos provável;

7. Aprendizes podem simplesmente não conceber toda a escala de possíveis respostas a um problema. Embora consigam acionar mentalmente a regra geral cabível à opção mais evidente, nem sempre têm mecanismos para cogitar casos especiais ou alternativas menos prováveis. Há, portanto, estruturas variadas de conhecimento que ajustam tal processo de decisão;

8. Apontou-se que peritos têm modelos mentais melhor definidos. Assim, conseguem respaldo para discriminações mais refinadas na resolução de problemas. Em adição, nota-se uma pequena disparidade entre o que se manifesta realmente em um problema e a representação feita por um aprendiz. Nesse sentido, frisa-se que a percepção tida do problema é fortemente dirigida pela representação (também mental) feita;

9. Capacidade oportuna de resolução foi outra característica observada em peritos, principalmente ao contemplar novas possibilidades quando surgidas. Dados externos ou fatos recém-percebidos no problema são informações adicionais inerentes a novas

oportunidades de resolução. Ambos os casos são aproveitados de maneira favorável por peritos;

10. Percebe-se que a resolução de problemas pelos peritos é mais fortemente orientada por padrões (anteriormente experienciados). Além disso, o reconhecimento de um padrão, em um problema, é acionado antes no raciocínio de peritos.

Adicionalmente, Lesgold [64] propôs uma teoria de currículo que privilegia aspectos de perícia. O currículo, em Sistemas Tutores Inteligentes (STIs), diz respeito à ordem pedagógica de apresentação na qual o conhecimento do domínio é oferecido ao aprendiz.

Mais recentemente, pesquisas sobre o ensino em diferentes domínios de conhecimento valem-se da **Taxonomia de Bloom** [38] para classificar e definir objetivos de aprendizagem. Também referida como **Taxonomia dos Objetivos Educacionais**, trata-se de uma estrutura hierárquica conceitual criada para auxiliar a definição de capacidades a serem desenvolvidas pelo aprendiz. A taxonomia resultou do trabalho de uma comissão multidisciplinar de especialistas, liderada por Benjamin S. Bloom, na década de 1950, que buscava uma maneira de facilitar a troca de enunciados entre professores de diferentes universidades.

A classificação proposta por Bloom organizou as possibilidades de aprendizagem em três grandes domínios [38], abaixo declarados:

- **Cognitivo:** recordação e identificação do conhecimento e o desenvolvimento de capacidades intelectuais;
- **Afetivo:** mudanças de interesse, atitudes e valores, bem como o desenvolvimento da sensibilidade e da adequação tanto emocional quanto social do indivíduo; e
- **Psicomotor:** execução de tarefas que envolvem aptidão física.

Concerne diretamente ao escopo deste trabalho apenas o **domínio cognitivo** [38] que, especificamente, abrange as categorias descritas a seguir:

1. **Conhecimento:** capacidade básica de recordar fatos. De forma mais ampla, consiste em recuperar informações isoladas relevantes da memória, tais como conceitos, fatos específicos, padrões e procedimentos;

2. **Compreensão:** capacidade de entender relações entre fatos. Ou seja, segundo [45], construir significado (ao conhecimento recordado) através de linguagem oral, escrita ou gráfica. Usam-se ferramentas como interpretação, exemplificação, classificação, sumarização, inferência e explicação;

3. **Aplicação:** capacidade de usar abstrações em contextos variados. Utiliza o aprendizado em novas situações e aborda ações como *aplicar*, *computar*, *demonstrar*, *manipular*, *modificar*, *produzir* e *resolver*;

4. **Análise:** capacidade de determinar como partes isoladas se correlacionam e cumprem com um propósito geral. Envolve a análise de elementos, relações e princípios de organização;

5. **Síntese:** capacidade de combinar informação variada a fim de constituir um todo coerente, abrangendo também o estabelecimento e anterior reconhecimento de padrões;

6. **Avaliação:** capacidade de construir e defender julgamentos, com base em critérios e padrões, bem como em evidências internas e externas.

As categorias são ordenadas da mais simples para a mais complexa. Além disso, trata-se em uma hierarquia cumulativa que pode ser representada como uma pirâmide (conforme Figura 2.1). Implica, portanto, que uma categoria mais básica é pré-requisito para outra mais complexa. A capacidade do aprendiz desenvolve-se à proporção que se avança ao pico da pirâmide. As três categorias mais básicas representam perícias de **baixo nível** enquanto as outras três são consideradas de **alto nível** por requererem processos cognitivamente mais elevados [45].

Durante a década de 1990, algumas revisões foram sugeridas à taxonomia original, compondo a então **Taxonomia de Bloom Revisada** [9, 62]. A pesquisa foi conduzida

Figura 2.1: Representação da Taxonomia de Bloom Original (adaptada de [45])

por Lorin Anderson, estudante de Benjamin S. Bloom, e valeu-se de uma comissão de especialistas assim como feito na concepção da taxonomia original. Basicamente, ilustradas na Figura 2.2, as alterações foram:

1. As categorias *conhecimento*, *compreensão*, *aplicação* e *análise* tornaram-se, respectivamente, **lembrar**, **entender**, **aplicar** e **analisar**;

2. A *síntese* evoluiu para **criar**, alocada no topo da pirâmide;

3. Consequentemente, *avaliação* foi renomeada para **avaliar**, sendo agora reduzida à segunda capacidade mais complexa.

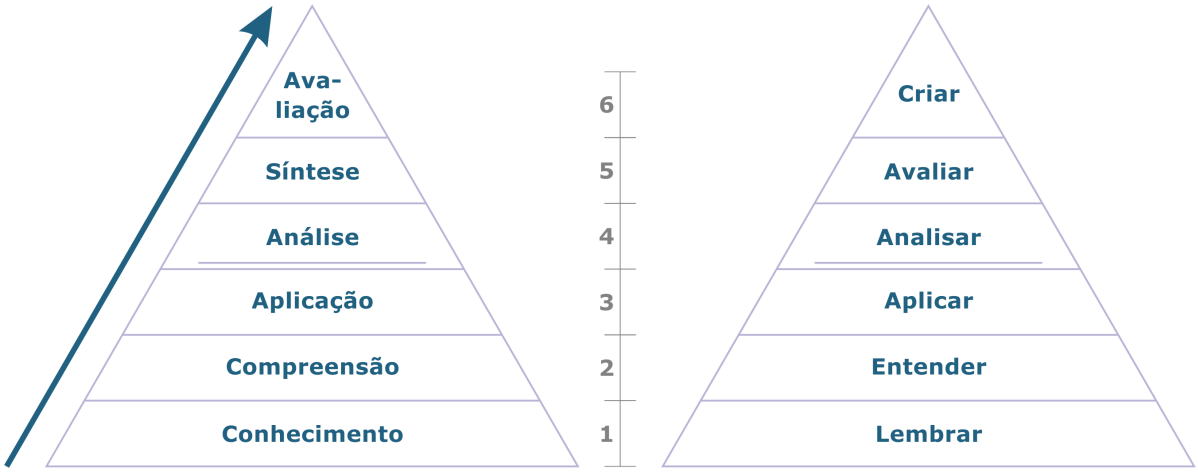


Figura 2.2: Representação da Taxonomia de Bloom Original e da Revisada (adaptada de [45])

Embora as alterações de nomenclatura sejam pequenas e tenham o intuito de esclarecimento, por meio da troca de substantivos por verbos, houve mudanças importantes nas perícias de alto nível, ou seja, naquelas três do topo da pirâmide. A capacidade de *criar* foi promovida em reconhecimento à complexidade inerente de construir algo novo. Portanto, *avaliar* foi declarada como uma capacidade menos complexa do que *criar*.

Adicionalmente, [30] justapôs os seis níveis da taxonomia revisada com os verbos associados, conforme Tabela 2.1. Tais verbos são especializações daqueles que denominam cada nível da taxonomia e têm o propósito de auxiliar na classificação de um enunciado nos níveis anteditos.

lembrar	**entender**	**aplicar**	**analisar**	**avaliar**	**criar**
reconhecer	interpretar	executar	diferenciar	verificar	gerar
relembrar	exemplificar	implementar	organizar	criticar	planejar
	classificar		atribuir		produzir
	sumarizar				
	inferir				
	comparar				
	explicar				

Tabela 2.1: Níveis da taxonomia revisada e verbos associados [30]

Por fim, [30] evidencia aspectos importantes apontados por [46]. Primeiro que, apesar das revisões, a Taxonomia de Bloom original ainda é a versão mais utilizada. Entretanto, critica-se a taxonomia original porque: **(1)** as categorias nem sempre são fáceis de serem aplicadas; **(2)** há sobreposição significativa entre as categorias; e **(3)** ainda existem discussões acerca da ordem das categorias *análise*, *síntese* e *avaliação* na hierarquia.

Convém supor que a sobreposição entre as categorias é esperada, visto constar de uma hierarquia cumulativa em que os pré-requisitos são evidentes. A ordem das categorias de alto nível, por sua vez, pode ser sensível ao domínio de aplicação considerado.

2.2.2 Impactos na Perícia em Programação de Computadores

O processo de aprendizagem em Programação de Computadores pode ser visto, conforme abordagem de Lesgold [65], como a aquisição de conhecimentos sobre os princípios de lógica de programação e consequente desenvolvimento destes até consolidarem perícias

na área. Distinguem-se os princípios das perícias [74], porquanto os primeiros tangem à compreensão de instruções primitivas isoladas, atendo-se à rigidez léxica, sintática e semântica determinada pela linguagem que se usa. As perícias, por sua vez, abrangem o encadeamento e o aninhamento de duas ou mais dessas instruções a fim de compor um algoritmo.

Somados, os princípios e as perícias alicerçam a análise e resolução de problemas por meio de algoritmos. No entanto, a tarefa de programar impõe uma alta carga cognitiva sobre os iniciantes, pois exige a aplicação de ambas as categorias de conhecimento na externalização do código fonte. Como consequência, os aprendizes incorrem em erros frequentes que antagonizam a evolução destes indivíduos enquanto programadores.

Dentro disso, [92] descreve **medidas cognitivas** no ensino de Programação de Computadores, as quais são uma forma de se obter meta-conhecimento acerca dos enunciados de uma base de conhecimento de um Sistema Tutor Inteligente (STI). Tais medidas funcionam como um forte elo entre o Modelo de Domínio e o Modelo do Aprendiz nesses sistemas.

As medidas citadas têm o papel de quantificar cognitivamente um enunciado de Programação de Computadores, ocupando-se de avaliar o quanto determinado enunciado exige de um aprendiz em termos de conhecimentos adquiridos e capacidades desenvolvidas na progressão do aprendizado. Nesse sentido, o mesmo estudo define a carga cognitiva de um enunciado como a capacidade que este tem em exercitar o aprendiz na construção de um programa de computador. Portanto, a carga cognitiva pode ser dividida em alguns componentes que individualmente se responsabilizam por medir um tipo de contribuição/exigência proporcionado pelo enunciado ao aprendiz.

A complexidade do software é um dos componentes sugeridos pelo citado estudo. Todavia, como os programas destinados a iniciantes são majoritariamente pequenos, a complexidade pode ser melhor determinada, usando linhas de código (LOC) em conjunto com complexidade estrutural (CE) e respectivos detalhes de implementação (DI)[1]. A complexidade estrutural avalia o número de estruturas de decisão e de repetição, bem como o grau

[1]O primeiro autor de [92], como integrante da banca avaliadora do exame de qualificação desta pesquisa, reiterou que a complexidade ciclomática (CC), conforme [78], pode ser a métrica mais indicada.

de dependência entre essas estruturas (aninhamentos). Por sua vez, os detalhes de implementação consideram a quantidade de condições específicas que dificultam o cumprimento do enunciado.

Além desses componentes que concernem à complexidade, cada enunciado contribui para que o aprendiz desenvolva características do estereótipo de um programador perito. Um conjunto de tais características foi identificado anteriormente por [92], a saber:

1. precisão sintática;
2. precisão semântica;
3. identificação de estruturas principais no programa fonte (busca por palavra-chave);
4. simulação mental dos estados do computador durante a execução;
5. catálogo de erros;
6. mapeamento mental das estruturas do programa;
7. checagem de pré-condições;
8. análise do problema;
9. integração dos subproblemas;
10. generalização da solução;
11. reutilização de soluções já conhecidas; e
12. catálogo de soluções.

Por consequência, os outros componentes da carga cognitiva de um enunciado baseiam-se nesse grupo de características e têm medidas definidas a partir do estereótipo de um programador perito. Assim, podem quantificar a contribuição/exigência de um enunciado frente a cada um dos atributos do estereótipo.

Portanto, a **carga cognitiva** de um enunciado constitui-se da ponderação das medidas cognitivas que o compõem. Embora cada medida contribua com uma parcela da

carga cognitiva, isso não ocorre de maneira uniforme. Diferentes medidas possuem pesos distintos na composição da carga cognitiva de uma instância. Tais pesos são valorados por meio de informações obtidas de especialistas no ensino de Programação de Computadores.

Evidencia-se então que as medidas cognitivas fornecem subsídios adequados para a escolha do próximo enunciado a ser trabalhado em uma sessão de ensino, colaborando para que a progressão do aprendizado seja mais concisa e coesa. Assim, há utilidade prática na classificação de um enunciado, por grau de exigência/contribuição, diante de um elenco preciso de medidas cognitivas. Conforme objetivo específico desta tese, uma galeria de enunciados poderá ser catalogada e ordenada formalmente frente às dimensões que denotam o (sub)conjunto de medidas cognitivas consideradas.

Parte da importância desse procedimento pode ser constatada, quando um problema de programação significativamente mais complexo do que o anterior é proposto a um aprendiz. Um fato pode que levar à ocorrência de erros múltiplos, impedindo o sucesso do indivíduo por longos períodos de tempo.

Partindo desse pressuposto, um fator adverso em ambientes de ensino destinados à Programação de Computadores reside na extrema rigidez na ordenação das respectivas galerias de enunciados. Mesmo que, mediante decisões fundamentadas, os projetistas e construtores de tais ambientes tenham alcançado determinada sequência de ensino, não há garantias de que ela privilegie toda a heterogeneidade de perfis e competências de diferentes aprendizes. Logo, pode-se deduzir que a sucessão dos enunciados atende a um grupo específico de aprendizes, geralmente de desempenho entre baixo e mediano, e negligencia aqueles que não se enquadram no estereótipo idealizado.

Uma contribuição anterior [109] antecipou a complexidade dos enunciados como um dos principais componentes de motivação do indivíduo no aprendizado. Nesse sentido, [92] explica que “a repetição sistemática de enunciados completamente diferentes, porém, de graus de complexidade aproximadamente iguais pode elevar consideravelmente a auto-confiança do aprendiz, mantendo-o motivado e produtivo”.

Logo, seria não somente desejável, como também necessária, a flexibilização de tais ambientes de ensino para que se adaptassem às competências de cada aprendiz. Nessa

direção, foi desenvolvida a ferramenta de autoria **Sequence** [91, 31], inicialmente concebida para a área médica e depois adaptada ao ensino de Programação de Computadores. A ferramenta baseia-se em medidas cognitivas e avalia um enunciado mediante a complexidade (LOC, CE e DI) aliada às perícias de precisão sintática/semântica, análise do problema, reutilização de soluções e simulação mental. Com isso, consegue sugerir uma sequência inicial de enunciados e, então, adaptá-la à situação corrente do usuário mediante informações oriundas do Modelo do Aprendiz no STI.

Em direção paralela à abordagem de Lesgold, [30] abordou a contribuição da Taxonomia de Bloom no contexto introdutório de Programação de Computadores. A referida pesquisa discute acerca da interpretação e utilização das categorias taxonômicas nas avaliações das disciplinas concernentes. Os resultados foram sintetizados na Tabela 2.2, transcrita neste documento, que apresenta as principais interpretações relacionadas a cada nível taxonômico no contexto de programação.

Além disso, [30] relata considerações relevantes de outras pesquisas. Primeiramente, são apontados os trabalhos de [79, 70, 114] que elaboraram instrumentos padronizados de avaliação para que o desempenho de diferentes turmas de programação introdutória fosse comparado. O instrumento de avaliação empregado por [70] foi aprimorado e reutilizado em [114], ambos seguindo a Taxonomia de Bloom.

Em segundo lugar, expõe-se que [114] sobreavisou a respeito da dificuldade de interpretar as descrições dos níveis taxonômicos no contexto dos exercícios de programação. Nesse sentido, [110] aponta situações prévias em que se observaram discrepâncias significativas entre as classificações sugeridas por diferentes professores para um mesmo enunciado. Com igual preocupação, [56] notifica que os professores da área não consideram as categorias de *síntese* e *avaliação* como úteis para a descrição dos objetivos de aprendizagem em programação, tendo a *aplicação* como a capacidade mais importante a se desenvolver.

Baseando-se nessas considerações, [30] propôs, como estudo de caso, um instrumento de avaliação cujas questões foram classificadas segundo as interpretações da taxonomia, conforme discutidas na publicação. Com esse apanhado, [30] declarou-se como um aprimoramento do estudo de [114], anteriormente descrito.

Categoria	Interpretação da categoria em Programação de Computadores
Lembrar	*Recuperação de conhecimento relevante da memória de longo termo.* - Identificar elementos específicos em um trecho de código. - Reconhecer a implementação de um determinado conceito. - Reconhecer a descrição mais apropriada para um determinado conceito. - Lembrar de um conceito, processo, algoritmo, etc. - Listar operadores de acordo com a ordem de precedência. - Definir o propósito de um método construtor. - Descrever um determinado padrão de projeto. - Citar os nomes dos tipos de *loops* em uma linguagem de programação. - Listar n métodos que executem operações de entrada e saída de dados.
Entender	*Construção de significados através de diferentes tipos de linguagens.* - Escrever em pseudo-código, fluxograma ou linguagem natural um programa que calcule uma fórmula bem conhecida. - Completar partes faltantes de um programa utilizando fragmentos de código. - Explicar com palavras o comportamento de um trecho de código. - Predizer valores de variáveis depois da execução de um trecho de código. - Traduzir um algoritmo de uma forma de representação para outra. - Explicar um conceito, algoritmo ou padrão de projeto. - Apresentar exemplos de um conceito, algoritmo ou padrão de projeto.
Aplicar	*Utilização de processos conhecidos para executar ou implementar.* - Implementar um programa utilizando como exemplo um código que resolva um problema semelhante. - Implementar ordenação de vetores não numéricos com alunos que já tenham ordenado vetores numéricos. - Executar mentalmente expressões seguindo as regras de precedência. - Resolver um problema familiar, mas com dados ou ferramentas não familiares. - Modificar o código de um *loop* do tipo *for* para um do tipo *while*.
Analisar	*Decomposição de um problema em suas partes constituintes e determinação das relações entre as partes e o todo.* - Dividir uma tarefa de programação em suas partes componentes. - Organizar as partes componentes para atingir um objetivo geral. - Identificar componentes críticos para o desenvolvimento. - Identificar componentes ou requisitos não importantes. - Diferenciar um método construtor dos demais métodos de uma classe.
Avaliar	*Realização de julgamentos baseados em critérios e padrões.* - Determinar se um código satisfaz os requisitos definindo uma estratégia de teste apropriada. - Criticar a qualidade de um código baseando-se em boas práticas de programação ou critérios de eficiência do código. - Avaliar qual de dois algoritmos, que resolvem a mesma tarefa, é mais adequado. - Encontrar um erro de lógica em um trecho de código dado.
Criar	*Juntar elementos para formar um todo coerente e funcional.* - Propor um algoritmo, processo ou estratégia alternativa para um problema. - Hipotetizar que uma nova combinação de algoritmos resolverá o problema. - Construir um programa, utilizando algoritmos inventados. - Aplicar algoritmos conhecidos em uma combinação não familiar para o aluno.

Tabela 2.2: Síntese da interpretação da Taxonomia de Bloom em Programação de Computadores (adaptada de [30])

Dadas as duas abordagens, de Lesgold e de Bloom, constata-se que ainda permanece um vasto espaço para pesquisa, pois faltam estudos que se estendam na focalização das características cognitivas que compõem a complexidade de um enunciado de Programação de Computadores. Ademais, seria interessante que diferentes pesquisas reavivassem o estado da arte dessa área e, como consequência, guinassem os ambientes de ensino destinados à Programação de Computadores a uma nova e promissora geração.

2.3 Ambientes de Apoio ao Ensino e Aprendizagem de Programação de Computadores

Muitos sistemas de ensino foram implementados para o domínio de Programação de Computadores [35, 81]. Tratava-se de uma direção bastante lógica a se seguir, haja vista que a maioria dos pesquisadores tinha por especialidade a programação.

Mesmo com esse progresso, houve um período em que o campo de sistemas destinados ao ensino de programação permaneceu estagnado. Uma hipótese cabível era que, inicialmente, se tinha bastante facilidade por se dispor de especialistas no domínio ensinado entre os pesquisadores. Com a evolução dos estudos, percebeu-se que muito dos sistemas se devia aos especialistas em ciências cognitivas sendo, portanto, o domínio de programação tão difícil de ser ensinado quanto qualquer outro.

Portanto, frisa-se que, desde o início da década de 90, poucas pesquisas se dedicaram ao ensino dessa área. Somente nos últimos anos, estudos concernentes ao domínio têm ressurgido, o que justifica a lacuna de referências a trabalhos desse período. Também se percebe, nas poucas publicações recentes, que a maior proporção das citações consta de pesquisas feitas até a primeira metade da década mencionada.

Pode-se, além disso, apontar a instauração do Workshop de Ambientes de Apoio à Aprendizagem de Algoritmos e Programação (W4AP), dentro do Simpósio Brasileiro de Informática na Educação (SBIE), como evidência concreta da necessidade de despertar pesquisas nesse âmbito e de concentrar esforços que estavam sendo feitos de forma dispersa. O SBIE é o principal evento de Informática na Educação e Educação em Informática no

território nacional. O W4AP conta com as edições de 2007, 2008 e 2010.

A corrente seção aborda os ambientes para livre exploração (2.3.1) e aqueles que realizam diagnóstico ativo de erros de programação (2.3.2). Embora não sejam classificações exclusivas, perfazem duas linhas de contribuição distintas para esta pesquisa.

2.3.1 Ambientes para Livre Exploração

Os tradicionais sistemas de **Instrução Assistida por Computador**[2] (IAC) [112] eram recipientes organizados de forma estática, estruturados para incorporar o conhecimento tanto do domínio quanto pedagógico de professores especialistas. Seguindo a linha evolutiva, esses sistemas passaram a agregar técnicas da Inteligência Artificial[3] (IA) e a adotar outras abordagens para o ensino. Por algum tempo, tais pesquisas foram conduzidas sob a denominação de **Instrução Inteligente Assistida por Computador**[4] (IIAC), coexistindo com a então designação de **Sistemas Tutores Inteligentes**[5] (STIs). Contudo, a última foi ganhando a preferência dos pesquisadores em detrimento da primeira, pois os STIs representam, sob diversos aspectos, uma mudança de metodologia muito maior do que a simples adição do "I" em IAC.

Portanto, STIs são programas de computador projetados para incorporar técnicas de IA de modo a prover tutores que saibam como ensinar, o que ensinar e a quem ensinar. Para isso, há intersecção das áreas de Ciência da Computação, Psicologia Cognitiva e Pesquisa Educacional; sendo o novo campo normalmente denominado Ciência Cognitiva [85].

Um **Ambiente Interativo de Aprendizagem**[6] (AIA) típico distingue-se de um STI por ser de caráter mais passivo, não interferindo/interagindo (pró-)ativamente com o aprendiz, embora seja semanticamente rico. Parte de atividades como exploração, investigação e descoberta para que o aprendiz construa individualmente seu conhecimento. São também conhecidos como **Ambientes de Livre Exploração**.

[2] Do inglês, *Computer-Aided Instruction* (*CAI*).
[3] *Artificial Intelligence* (*AI*).
[4] *Intelligent Computer-Aided Instruction* (*ICAI*).
[5] *Intelligent Tutoring Systems* (*ITS*).
[6] *Interactive Learning Environment* (*ILE*).

Como casos bastante característicos de AIAs, merecem destaque os chamados micromundos[7] ou microcosmos. São ambientes com riqueza semântica que atuam como instrumento de abstração para representar situações reais. Os elementos disponibilizados por um micromundo agem, portanto, como um conjunto metafórico dentro das situações representadas.

Apesar do exposto, salienta-se que a fronteira entre AIAs e STIs, além de tênue, é pouco definida. Evidência disso são os Ambientes de Descoberta-Guiada[8] (ADGs).

Outrossim, pode-se encarar o atributo "inteligência" (até então próprio dos STIs) como decorrente de um alto grau de interatividade provido por determinado ambiente. Nesse panorama, um AIA seria inteligente à devida proporção que dispusesse de uma escala variada de oportunidades de interação (ou exploração) ao aprendiz. Tal ambiente, no dado contexto, tornaria ainda mais sutil e intricada a demarcação entre STIs e AIAs.

Na sequência, resenha-se sobre ambientes que fomentam a programação livre. Podem ser STIs, AIAs ou incorporar características de ambos. Primeiramente, são abordados os ambientes com apoio sintático e depois aqueles com apoio semântico. Recursos que demarcam o respectivo apoio, sintático ou semântico, são introduzidos e contextualizados juntamente com os ambientes correlatos.

2.3.1.1 Programação Livre com Apoio Sintático

O **BALSA** (*Brown University Algorithm Simulator and Animator*) [15, 16] foi um sistema pioneiro desenvolvido na década de 1980 que provê uma interface de alto-nível e permite a atuação do aprendiz sobre representações externas gráficas dos algoritmos em execução. Tais representações denotam não somente variáveis e estruturas de dados, mas também estados significativos (denominados pelo BALSA como *interesting events*) na execução/compreensão de um algoritmo.

Convém frisar que o BALSA não propõe, como objetivo principal, a construção de algoritmos por aprendizes. Propicia, entretanto, a livre exploração sobre múltiplas repre-

[7] *Microworlds.*
[8] *Guided-Discovery Environments* (*GDEs*).

sentações dos algoritmos em execução simbólica. Para essa interação, dispõe de funcionalidades como: realçar aspectos de correspondência entre representações, interromper a simulação, diminuir a velocidade, avançar um passo e estabelecer pontos de interrupção (*breakpoints*). Dentre as ferramentas avançadas, cita-se a possibilidade de que o algoritmo retroceda passos, ou mesmo seja visualizado de trás para frente, como também de que um segundo algoritmo execute simultaneamente, com a finalidade de comparação.

O BALSA promove outros papéis de interação, além do aprendiz, progressivamente de níveis mais baixos, a saber: **(1)** preparo dos roteiros de simulação a serem destinados aos aprendizes; **(2)** escrita dos procedimentos para representação gráfica dos algoritmos; e **(3)** construção, na linguagem Pascal, dos algoritmos a serem visualizados. Sobre o último, frisa-se que a edição do código fonte era dirigida pela sintaxe, algo novo até então.

As atividades desempenhadas pelos papéis descritos contemplam um processo de autoria para a simulação de algoritmos no BALSA. Cada simulação dura cerca de 15 minutos e requer entre 15 e 30 horas de preparo.

O ambiente sucessor, **BALSA-II** [14], foi desenvolvido com a ideia de oferecer facilidades para a criação dos roteiros de simulação. O ambiente proporciona, por exemplo, que a interação do aprendiz com o algoritmo simulado seja registrada para posterior reprodução. Além disso, busca transpor o papel do aprendiz, de um visualizador passivo a um explorador cada vez mais ativo dos algoritmos executados.

Na época, outra pesquisa da mesma instituição (Universidade Brown) valeu-se dos resultados inerentes ao BALSA. O projeto **PECAN** [96, 97] era dito uma "família de sistemas para o desenvolvimento de programas", que ansiava ser empregado tanto para aprimorar a perícia de aprendizes quanto para incrementar a produtividade de programadores experientes.

Hoje, pode-se perceber que o PECAN procurou ser modularizado em subsistemas de forma semelhante aos atuais IDEs[9] (Ambientes Integrados de Desenvolvimento). Também antecipou ferramentas úteis no suporte aos desenvolvedores quanto à rigidez sintática, como:

[9] *Integrated Development Environments.*

1. editor dirigido pela sintaxe para o algoritmo textual (linguagem Pascal);

2. compilação incremental e *feedback* imediato de erros sintáticos (e alguns semânticos) enquanto o aprendiz edita o algoritmo;

3. disponibilização de gabaritos (*templates*), através de *menu* próprio, para escrita de instruções como alternativa à digitação;

4. flexibilidade de permitir a entrada manual de instruções em detrimento do uso exclusivo dos gabaritos citados; e

5. admissão de *pretty-printers*, ou marcadores de sintaxe, que consistem em convenções de formatação, atuantes como facilitadoras na compreensão semântica dos elementos isolados do algoritmo textual trabalhado.

O PECAN representa internamente o algoritmo por uma árvore sintática abstrata [4]. Conta com múltiplas representações externas como o diagrama de Nassi-Shneiderman, o fluxograma e a visualização de expressões em estrutura de árvore. A execução é puramente baseada nas representações do ambiente BALSA.

Concomitantemente, surgiu o **Tinker** [67, 66], ambiente desenvolvido por Henry Lieberman, pesquisador Instituto de Tecnologia de Massachusetts (MIT). Esse sistema baseia-se nas ideias de que programar se compara a ensinar procedimentos a um computador e de que exemplos concretos são uma abordagem próspera para o ensino/aprendizagem. Assim, nele se propõe uma metodologia em que o aprendiz programa em Lisp através da demonstração de exemplos ao ambiente.

Na utilização do Tinker, o aprendiz fornece os dados de entrada, bem como o tratamento dado a esses exemplos (através de interface gráfica ou expressões Lisp). O ambiente registra os passos envolvidos na resolução do problema e, mediante à apresentação de múltiplos exemplos, formula um programa capaz de tratar do caso geral presumido. Ou seja, programa-se por meio da demonstração de exemplos ao ambiente.

Embora o Tinker não propicie uma interface de manipulação direta e a programação puramente visual, preserva o aprendiz de aspectos sintáticos que podem ser empecilhos

na abstração lógica ou precisão semântica durante a experiência de ensino. Além disso, diz-se capaz de aprender quaisquer procedimentos que possam ser expressos em Lisp, inclusive quando exigem desvios condicionais, recursividade e complementação de definições parciais.

Outro ambiente contemporâneo foi o **TPM** (*Transparent Prolog Machine*) [36, 37], destinado à animação de programas na linguagem Prolog. O ambiente foi concebido como uma ferramenta para aprendizes e programadores experientes e propõe a representação gráfica fiel dos mecanismos internos do interpretador Prolog, fornecendo um panorama preciso dos predicados durante a execução. Além disso, permite que a visualização obedeça o fluxo progressivo ou retrospectivo, sob diferentes níveis de granularidade.

Com isso, o TPM popularizou o recurso de seguir ou inspecionar a execução de um programa passo a passo (*program tracing*). Esse ambiente teve também mérito em tratar espaços de busca bastante grandes (milhares de nós) e de determinar metáforas simples para especificidades como o predicado *cut* (abortamento da retroação) do Prolog.

No Brasil, por sua vez, foi desenvolvido o **SATELIT-2** [32], um tutor baseado em simulação que objetivava suporte inteligente ao treinamento para operação de centrais telefônicas digitais. Tal sistema permite interações ativas e passivas com o aprendiz, concentrando-se nos âmbitos léxico e sintático do diálogo. Nele, houve esforço diferencial no que tange ao tratamento de erros sintáticos das instruções fornecidas pelo aprendiz durante a interação. Outra singularidade do SATELIT-2 reside nos aspectos genéricos de correção, bem como nas explicações dinâmicas dos erros cometidos pelos aprendizes. Através dessas explicações, técnicos ou engenheiros em treinamento desenvolvem perícia na operação das referidas centrais.

Caso ocorra mudança de treinamento de uma central telefônica específica para outra, basta que se forneça a descrição sintática da nova linguagem de operação ao SATELIT-2. Dessa forma, o tutor é capaz de ensinar a segunda linguagem baseado nos mesmos princípios pedagógicos já praticados, porquanto conserva a característica de empregar mecanismos genéricos de ensino e consequente reutilização destes em diversas situações, sobre bases de conhecimento descritas em metalinguagem por meio de uma ferramenta de au-

toria.

Ressalta-se que os mecanismos pedagógicos do tutor SATELIT-2 apenas têm poder reativo sobre os aspectos sintáticos da linguagem ensinada. Os aspectos semânticos, embora inicialmente estudados, não chegaram à implementação.

Atualmente, tem-se algumas iniciativas amplamente utilizadas em instituições de ensino para abordar programação orientada a objetos. O **BlueJ** [10], uma destas, consiste em uma IDE para a iniciação no referido paradigma e no emprego da linguagem Java. Destaca-se pela correspondência gráfica que faz com UML[10] e facilita a visualização da modelagem de classes, atributos e relacionamentos. Embora o BlueJ seja carente de algumas formas básicas de apoio sintático, como a funcionalidade de autocompletar, dispõe de recursos visuais para a instanciação e manipulação de objetos sem a escrita de código. Permite, também, que se verifique o comportamento dos objetos na memória durante a execução. Pode-se, por exemplo, inspecionar valores de atributos e invocar métodos através da interface gráfica.

O **Greenfoot** [43, 61], outro empreendimento para o ensino de orientação a objetos, é um ambiente de desenvolvimento para Java destinado à criação de aplicações gráficas em duas dimensões, como simulações e jogos. Utiliza-se o BlueJ como ferramenta de edição. O arcabouço fornecido pelo Greenfoot abrange duas classes principais: *world*, representando o mundo em que os objetos serão alocados e onde ocorrerá a execução (uma área bidimensional na tela); e *actor* que denota todos os outros objetos presentes no jogo que podem atuar com o mundo criado. Programar, no Greenfoot, baseia-se na construção de subclasses de *world* e *actor*.

Pedagogicamente, o Greenfoot objetiva motivar o aprendiz, facilitando-lhe o acesso a figuras, animações, sons e possíveis interações. O ambiente encoraja a exploração e experimentação construtivista. Nesse sentido, busca enfatizar abstrações e conceitos importantes em orientação a objetos, como a relação classe-objeto, métodos, parâmetros e comunicação entre objetos. Para esse propósito, o Greenfoot vale-se de imagens e de interações guiadas que atenuam a rigidez sintática dos comandos. Assim, consegue construir

[10] *Unified Modeling Language*.

e suportar um modelo mental que representa o alto nível de abstração do mundo concreto e dos modernos programas orientados a objeto.

Com proposta similar, **Alice** [26, 88] é um ambiente interativo destinado à criação de animações e jogos em 3D que promove o aprendizado de conceitos introdutórios de orientação a objetos. Esse ambiente emprega o recurso de arrastar-e-soltar como precaução contra muitos erros sintáticos do aprendiz. A programação dos objetos alocados no cenário (micromundo) dá-se através de *scripts* baseados na linguagem Python.

Extensamente utilizado, o Alice está a caminho da terceira versão [43] e permitirá a digitação de códigos Java para a construção de projetos. Também possibilitará a interoperação entre a IDE própria e outra externa, como o NetBeans. Destaca-se que, para esta versão, houve incentivo da Electronic Arts[11] (EA).

O **Scratch** [43, 72], outro ambiente atual, foi criado como proposta aos centros de aprendizado extraclasse para comunidades economicamente desfavorecidas. Tal ambiente enfatiza a manipulação de mídias e trabalha com atividades de programação que repercutem interesses como a criação de histórias animadas, jogos e apresentações interativas.

No Scratch, cada projeto conta com um cenário e objetos que podem ser trazidos para esse lugar. A programação baseia-se na metáfora de blocos de construção que devem ser arrastados e encaixados como peças de quebra-cabeça. Os blocos figuram instruções que podem ser parametrizadas, encadeadas e aninhadas. Como resultado, tem-se pilhas de blocos que determinam o comportamento dos objetos presentes no cenário. Muitos dos eventuais erros sintáticos são restringidos pelo fato dos blocos de instruções oferecerem canais precisos de interação.

Em âmbito nacional, o jogo **Belesminha** [27] foi criado com o propósito de contribuir para o aprendizado dos conceitos de recursividade nas disciplinas introdutórias à Programação de Computadores. Nesse jogo cabe ao aprendiz controlar uma lesma que, mediante instruções programadas, deve percorrer trajetos definidos por folhas, na intenção de coletá-las.

A lesma pode executar três comandos: *avance uma casa*, *vire à direita* e *vire a es-*

[11] Desenvolvedora de jogos que lançou títulos como *Need For Speed*, *FIFA* e *The Sims*.

querda. No entanto, estimula-se que blocos desses comandos sejam identificados e alojados em funções. Cada nível do jogo estabelece requisitos que a solução (algoritmo e respectivas funções) deve atender. Conforme os níveis são avançados, o uso de recursividade se intensifica e os desafios exigem maior abstração. Há um mínimo auxílio sintático e também dicas semânticas prefixadas ao início de cada nível.

De outros ambientes construídos mais recentemente, verificou-se a ausência de recursos relevantes que não tenham sido apresentados pelos ambientes supracitados.

2.3.1.2 Programação Livre com Apoio Semântico

Nos ambientes destinados à programação livre com apoio semântico, observam-se dois grupos não exclusivos: os dirigidos por comandos e aqueles que possuem interface de manipulação direta. Do primeiro grupo, são descritos aqui os ambientes PETAL, JUNO e JUNO-2. Também abordados, Pygmalion, SmallStar e GIL, representam o segundo grupo.

O ambiente **PETAL** [11] aborda o domínio recursivo das linguagens Lisp e Scheme. Foi motivado pelo problema de que aprendizes se concentram tanto na sintaxe que externaliza soluções que relegam aspectos semânticos importantes.

Logo, o PETAL propôs uma interface que promove suporte à exatidão sintática de uma solução. Portanto, formulações sintaticamente incorretas não são permitidas. Além disso, o ambiente auxilia na construção dos modelos mentais requeridos na resolução eficiente de um problema. Este segundo auxílio, semântico, pode inclusive descartar instruções desnecessárias à solução.

O suporte fornecido pelo PETAL é expresso pelo termo *scaffolding*, que denota, em tradução literal, "andaime" ou "estrutura de amparo", tendo o papel de suster o aprendiz nos estágios iniciais do desenvolvimento de perícias. Proporcionalmente ao avanço do aprendiz, o suporte torna-se dispensável e, então, desnecessário.

O **JUNO** [83], outro ambiente, foi desenvolvido pelo lendário Palo Alto Research Center[12] (PARC) da Xerox, integra uma linguagem textual para a descrição de figuras

[12]O PARC foi uma importante divisão de pesquisa da Xerox Corporation, situado em Palo Alto,

geométricas e um editor gráfico WYSIWYG[13], fazendo a transposição automática entre essas duas representações que podem ser livremente manipuladas. A especificação precisa de uma imagem, de forma construtiva, aqui foi interpretada como um ato de programar.

A linguagem de programação subjacente ao JUNO é baseada na METAFONT de Donald Knuth, constituindo-se de três características principais: **(1)** a posição dos pontos em uma imagem é formalizada por restrições declarativas; **(2)** a imagem é renderizada através de comandos imperativos de desenho que são parametrizados por tais pontos; e **(3)** uma abstração procedimental da imagem é constituída pela sequência desses comandos. A imagem final possui estrutura hierárquica e fundamentada em restrições.

No JUNO, o editor responsável pela representação gráfica da imagem permite que o aprendiz distribua os pontos constituintes de forma imprecisa. Para que isso seja possível, há subsídio semântico de um sofisticado mecanismo que avalia as restrições de cada figura geométrica e alinha os pontos com exatidão. Pode até que sejam exigidas, do aprendiz, informações adicionais (dicas) de pontos desconhecidos pelo mecanismo.

Ademais, o ambiente provê suporte sintático ao aprendiz porquanto permite a alteração de estado através tanto do editor gráfico quanto da linguagem textual, bem como a sincronia automática entre essas representações. O JUNO pode ser, assim, compreendido na fronteira entre os sistemas dirigidos por comandos e aqueles de manipulação direta.

Sucedendo o trabalho, **JUNO-2** [55] estendeu a linguagem de programação utilizada. Também em decorrência disso, precisou incrementar o mecanismo que auxiliava semanticamente o aprendiz.

Em se tratando de ambientes que oferecem manipulação direta, o **Pygmalion** [106, 107], desenvolvido na década de 1970 por David Canfield Smith (então pesquisador do PARC), tem concepções de vanguarda até mesmo para a atualidade. Esse ambiente alicerça-se no pensamento de que a forma mais fluente de construir um algoritmo deve requerer a mínima tradução entre a representação interna concebida (na mente do aprendiz) e a externa no meio (linguagem de programação).

Califórnia. Fundado em 1970, tornou-se companhia autônoma em 2002. Fez-se famoso por ter sido incubador de invenções como interface gráfica (usufruída e popularizada pela Apple), Ethernet e impressão a laser, como também pioneiro em tecnologias de discos ópticos e LCD.

[13] *What You See Is What You Get.*

O criador do Pygmalion identificou a abstração excessiva das instruções como um dos maiores obstáculos para que os computadores se tornassem máquinas de fácil operação e programação. Para confrontar isso, ele introduziu o conceito de programação por demonstração (ou através de exemplos), o qual consiste em ensinar o computador a atingir o comportamento desejado por meio de situações reais. Isso ocorre por meio da demonstração dos passos do procedimento em um exemplo concreto, cabendo ao computador registrar tais passos e generalizá-los para que o algoritmo possa lidar com situações semelhantes.

Outro embasamento do Pygmalion defende que o processo criativo é intenso em atividades de construção e visualização de representações gráficas. Assim, o ambiente trouxe as ideias de ícones metafóricos e de manipulação direta em interfaces. Esta última, presente em recursos como arrastar-e-soltar para passagem de argumentos a sub-rotinas no ambiente.

Embora o Pygmalion tenha implementado em completude o que idealizou, permitia que apenas algoritmos simples fossem programados. Ocorria que o ambiente esbarrava em limitações de memória ou de processamento. Entretanto, como legado imediato, possibilitou o gerenciamento de arquivos por meio de ícones e metáforas[14] e propulsionou as pesquisas de Henry Lieberman [68] com o ambiente Tinker. Lieberman frisa a contribuição do Pygmalion enquanto ambiente de programação criativa, além da necessidade de que um sistema de caráter prático leve a programação por meio de exemplos ao reconhecimento [69].

O **SmallStar** [53], por sua vez, é uma simulação que reimplementa parte do sistema operacional Xerox Star. A ferramenta inspirou-se nos ambientes Pygmalion e Tinker, todavia se propunha a mostrar que a programação por meio de exemplos poderia ser compreendida e empregada por usuários que não fossem programadores. A intenção era que o usuário final do Xerox Star pudesse automatizar, por demonstração, tarefas cotidianas repetitivas.

O próprio Xerox Star já dispunha da linguagem de programação Cusp (*Customer*

[14]David Canfield Smith participou do projeto Xerox Star, criador da área de trabalho como metáfora.

Programming), destinada à escrita de procedimentos para automação de tarefas. Em SmallStar, um programa é gravado e, ao final, as ações que causaram mudança de estado no ambiente são transcritas em uma linguagem de fácil leitura, inspirada na Cusp. A edição do programa resultante apenas pode ser feita gravando novos passos ou apagando existentes. A execução ocorre de forma fluente ou a partir de passos isolados.

Convém evidenciar que a contribuição do SmallStar não reside em registrar os passos explícitos que alteram os estados do ambiente. Além disso, o SmallStar captura os critérios que fundamentam a intenção do usuário ao executar os passos de uma tarefa. Contudo, ao invés de utilizar inferência indutiva, método comum para esse propósito, apoia-se na descrição dinâmica intencional dos objetos de dados. Tais descrições são supostas pelo ambiente e permitem mudança, pelo aprendiz, no momento da edição do programa. Igualmente, o SmallStar também admite as estruturas condicional e de repetição, embora o uso de tais recursos não seja assimilado pelo ambiente e precise ser expresso pelo usuário.

O sistema tutor inteligente **GIL** (*Graphical Instruction in LISP*) [95], na década seguinte, foi o mais representativo ambiente de uma família em que cabe ao aprendiz interconectar objetos visuais como metáfora à construção de algoritmos em grafos. Ao fim, o grafo denota o fluxo de dados e de controle no algoritmo.

A interface gráfica do GIL foi elaborada com o objetivo de facilitar a aquisição de modelos causais na programação. Tal interface também propicia o encadeamento progressivo e regressivo dos objetos visuais, recurso que colabora com o processo de planejamento de uma solução pelo aprendiz. Ademais, mantém o registro gráfico do histórico de soluções e das metas correntes.

Além de o GIL suprimir muito da rigidez sintática, incorpora a orientação semântica baseada no rastreamento de modelo (*model tracing*). Essa abordagem, de monitorar a resolução de problemas pelo aprendiz, foi anteriormente desenvolvida por John R. Anderson [8] e implementada no sistema CMU Lisp. A intervenção semântica, portanto, ocorre ativamente quando o aprendiz se afasta da resolução conhecida pelo ambiente. Em adição, no GIL, geram-se explicações dinâmicas feitas diretamente sobre o conhecimento do domínio.

É importante realçar a escassez de trabalhos recentes que tenham ênfase no apoio semântico. Uma hipótese cabível seria que o apoio sintático se manteve por tornar-se característica indispensável em IDEs (recursos como autocompletar, *templates*, *pretty-printers*).

Argumenta-se também que o apoio sintático pode poupar muitos detalhes complexos de implementação. No desenvolvimento do MÚLTIPLA (introduzido na Seção 1.2 e detalhado no Capítulo 3), por exemplo, evitou-se a criação de um compilador completo porquanto a construção do algoritmo foi restrita à representação gráfica. Assim, os únicos textos realmente digitados pelo aprendiz são as expressões aritméticas, lógicas e relacionais. Reduzem-se, portanto, os erros e a necessidade de tratamento pelo ambiente.

Em contrapartida, o apoio semântico exige esforço redobrado para resultados bastante pequenos. Inicialmente, quando a Programação de Computadores era objeto fundamental de pesquisas em STIs, não se poupavam esforços para que houvesse ganhos, mesmo que mínimos, em apoio semântico. Dada a fase na qual se percebeu que o domínio de programação era tão difícil de ser ensinado quanto qualquer outro, devido às questões pedagógicas e de representação que muito remetem às ciências cognitivas, tais pesquisas foram cessando.

Por fim, lembra-se que parte da motivação desta pesquisa fixa-se justamente em elevar o nível de apoio do MÚLTIPLA, anteriormente citado, de sintático para semântico. A intenção consiste em atingir uma modelagem precisa do conhecimento do domínio e do aprendiz para que se proporcione automatização no avanço dos enunciados a serem explorados.

2.3.2 Ambientes que Realizam Diagnóstico Ativo de Erros de Programação de Computadores

Os ambientes que realizam diagnóstico ativo[15], ou automático, de programas objetivam encontrar e classificar erros no código fonte construído pelo aprendiz [35]. Tais ambientes, além de orientarem a interação do aprendiz acerca dos erros cometidos, também consis-

[15]Também conhecidos como *bug finders*.

tem em ferramentas úteis na atualização do Modelo do Aprendiz em um Sistema Tutor Inteligente.

Em algumas áreas, que não a programação, a dificuldade de avaliar uma resposta pode ser reduzida. Na Aritmética, por exemplo, a resposta pode se resumir a um resultado numérico, facilitando a avaliação. Entretanto, não se pode diagnosticar facilmente se um algoritmo construído por um aprendiz cumpre os requisitos do enunciado. Nisso reside, portanto, a dificuldade em descrever uma meta-solução em nível adequado de detalhes e precisão.

Basicamente, os ambientes que proveem diagnóstico podem ser classificados em três categorias [35], conforme o critério adotado para localizar e isolar erros, detalhadas na sequência:

1. Diagnóstico por solução de referência;
2. Diagnóstico por análise de especificação;
3. Diagnóstico por diálogo de depuração.

Esclarece-se que o diagnóstico de erros em programação possui aspecto periférico ao escopo que esta pesquisa adquiriu. Contudo, acredita-se ser conveniente demarcá-lo, mesmo com menor intensidade, por resumir muito do potencial de utilização do mapeamento de perícias em Programação de Computadores.

2.3.2.1 Diagnóstico por Solução de Referência

No método de diagnóstico por solução de referência [35], fornece-se ao ambiente uma solução de exemplo (*specimen answer*) junto à resposta do aprendiz, para que sejam confrontadas. Os erros são encontrados na tentativa de transformar ambas as soluções a fim de que se compatibilizem.

Uma dificuldade crucial desse método reside no fato de que os ambientes não têm entendimento dos objetivos das soluções (tanto do aprendiz quando de referência). Assim, tais ambientes mostram-se pouco capazes de julgar com critério a finalidade específica do algoritmo e os erros na tentativa de solução.

Outro obstáculo consiste em localizar com precisão os erros na resposta do aprendiz e isolá-los. A intenção seria comunicar em quais particularidades a solução de referência não compatibiliza com aquela do aprendiz.

Um ambiente de notoriedade foi **LAURA** [2, 112, 35], que teve destaque por utilizar-se de soluções de referência para diagnosticar os algoritmos do aprendiz. Basicamente, LAURA transforma ambas as soluções em grafos de fluxo de controle e tenta casar (mediante transformações) o grafo do aprendiz com o de referência. Contudo, o processo de casamento de padrões é bastante complexo e sofisticado, uma vez que concede ao sistema a capacidade de aceitar algoritmos corretos embora não idênticos à solução de referência. Cita-se isso como vantagem, porquanto não se induz o aprendiz a restringir sua criatividade em prol do caminho especificado pela solução prevista. No entanto, salienta-se que LAURA tem pouca capacidade de resposta, sendo apenas capaz de diagnosticar erros de baixo nível de abstração. Como o LAURA trabalha de forma genérica, as mensagens de erro são geradas dinamicamente e o ambiente desconsidera o contexto e a capacidade de planejamento do aprendiz dentro disso.

LAURA teve papel fundamental no desenvolvimento da pesquisa de [34] que procurou estender as ferramentas de diagnóstico anteriormente criadas e possibilitar *feedback* de alto valor cognitivo. Buscou-se flexibilidade suficiente para aceitar soluções não apenas idênticas àquelas de referência, como também generalidade quanto à linguagem de programação utilizada (desde que Imperativista).

2.3.2.2 Diagnóstico por Análise de Especificação

Ambientes que implementam o diagnóstico por análise de especificação [35] são providos de uma descrição (*specification*) em alto nível das metas do problema. A análise é feita no sentido de verificar em que proporção essas metas foram atingidas pela solução do aprendiz.

Teoricamente, tal abordagem pode fornecer informações mais detalhadas acerca dos erros, uma vez que estes são detectados por meio da mesma especificação do problema utilizada pelo aprendiz. Adicionalmente, a solução do aprendiz é avaliada em um âmbito

mais global. Nessa categoria, há a maior variedade de ambientes, sendo exemplos [34, 35]: MYCROFT, AURAC, PUDSY e PROUST.

O ambiente **MYCROFT** foi desenvolvido por Goldstein [50] e concentra-se na detecção e reparo de erros em programas para produzir desenhos em LOGO [87]. O problema é especificado por meio de assertivas sobre as propriedades geométricas do desenho a ser gerado. O mecanismo de diagnóstico tem a capacidade de encontrar inconsistências entre a solução do aprendiz e as assertivas, bem como de sugerir modificações que corrijam a solução.

Embora tenha atingido resultados notáveis, MYCROFT beneficiou-se pela simplicidade do domínio de aplicação e a possibilidade de utilizar propriedades geométricas para especificar o problema. Não é complexo determinar se uma solução cumpre com os objetivos. Ademais, as estratégias de análise são privilegiadas pela facilidade da linguagem LOGO.

O **AURAC** [54], por sua vez, foi outro ambiente favorecido pelas características da linguagem tratada (Solo), cujo processo de diagnóstico consiste em três fases: **(a)** casamento de produções que representam erros sintáticos; **(b)** comparação de segmentos de código com uma biblioteca de clichês[16]; e, finalmente, **(c)** análise do fluxo de dados.

Levando em conta que o processo de diagnóstico considera trechos de código, o AURAC tem condições de apresentar mensagens de erros significativas em uma análise de mais alto nível. Entretanto, o caráter restritivo da linguagem Solo viabilizou a abordagem por meio desses trechos, assim como a admissão da citada biblioteca de clichês. Caso uma linguagem de programação mais robusta fosse escolhida, tal forma de diagnóstico ficaria inviável devido à variedade de possíveis fragmentos de código a serem ponderados.

PUDSY [71], um outro ambiente, destina-se à linguagem Pascal e funciona a partir de uma especificação abstrata, sendo capaz de eliminar uma limitada classe de erros em pequenos programas. Para isso, o ambiente utilizava duas fases de depuração. A primeira, executava uma busca *bottom-up* no código procurando indícios de erros. A próxima, realizava uma análise do fluxo de dados de trechos do programa e, com isso,

[16]Fragmentos corretos de código utilizados con frequência.

gerava dinamicamente a especificação do programa analisado.

Como diferencial, PUDSY conseguiu trabalhar com uma linguagem de programação menos restrita e, ainda assim, diagnosticar e corrigir erros. Todavia, o próprio desenvolvedor apontou como limitações: a falta de flexibilidade, a incapacidade de avaliar o comportamento do programa em tempo de execução e a admissão de abordagens que admitem pouca generalização. Além disso, apesar de categorizado entre ambientes de diagnóstico por análise de especificação, mostra-se limitado na capacidade de explicar os erros ocorridos, principalmente por não prever o caráter dos erros que poderia encontrar.

Também dedicado à linguagem Pascal, **PROUST** [57] figura entre os mais importantes da categoria de diagnóstico. A especificação do programa é expressa por meio de um conjunto de objetivos. Cada objetivo tem informações declarativas concernentes armazenadas em um objeto associado. Tal objeto relaciona um ou mais planos que atingem o objetivo proposto. Os planos, individualmente, são relacionados com um gabarito (*template*) e também admitem um conjunto de sub-objetivos próprios. Assim, a definição de objetivos é recursiva e os gabaritos ainda podem ser usados para casamento com trechos de código em Pascal.

Evidencia-se positivamente, na abordagem do PROUST, a capacidade de relacionar trechos de código aos objetivos da especificação. Por conta disso, mais do que qualquer outro ambiente contemporâneo, PROUST consegue fazer comentários sobre problemas na lógica do programa, sendo inclusive capaz de sugerir entradas em que o comportamento não seria bem sucedido para a solução analisada.

Como desvantagem, PROUST era sensível a pequenas diferenças na resposta e inapto a aceitar soluções equivalentes, restringindo as possibilidades de resolução do aprendiz. Também se destaca que o ambiente versava apenas sobre um subconjunto de Pascal e, por consequência, atingia um conjunto limitado de problemas.

2.3.2.3 Diagnóstico por Diálogo de Depuração

O método de diagnóstico por diálogo de depuração (*debugging dialogue*) [35] foi concebido tendo em foco linguagens de programação declarativas. Nessa categoria de ambientes, a

análise da solução é orientada pelo diálogo com o aprendiz.

O diálogo consiste em perguntas sobre a finalidade das várias sub-rotinas componentes da solução. O acompanhamento (*tracing*) das chamadas dessas rotinas, associado à informação dialogal, propicia que erros sejam isolados em partes específicas da solução.

Shapiro [35, 104] é um ambiente da categoria e baseia-se no fato de que os programas em questão podem ser descritos em termos de uma hierarquia de procedimentos e funções. Ainda que pensado para atender a uma variedade de linguagens, o ambiente restringe-se ao Prolog.

A execução do programa no Shapiro é suportada pelo aprendiz e constrói uma árvore contendo a sequência da chamada dos predicados envolvidos. O ambiente então examina a árvore produzida e pergunta ao aprendiz sobre o comportamento desejado daqueles predicados. Por meio da comparação dos valores desejados com os produzidos, determinam-se quais definições de predicado que apresentam falhas. O processo de diagnóstico foi otimizado para reduzir o número de perguntas ao aprendiz, limitando-se rapidamente a predicados com falhas.

No Shapiro, o comportamento desejado do programa (informado pelo aprendiz) pode ser armazenado como um conjunto de assertivas, então consultadas pelo mecanismo de diagnóstico. Assim sendo, a análise não se distancia daquela praticada pelo MYCROFT [50], baseada na especificação do problema. Contudo, o Shapiro se beneficia da natureza declarativa do Prolog, tendo facilitadas tarefas como relatar possíveis alterações de código e tratar de passos intermediários. O fato de o ambiente estar fortemente atrelado ao Prolog, deixa dúvidas sobre o quanto os métodos descritos podem ser generalizados a outros paradigmas e linguagens de programação.

2.4 Modelagem de Aprendizes em Sistemas Tutores Inteligentes

Embora não haja arquitetura única para construção de STIs, uma abordagem bastante clássica compõe-se dos seguintes módulos [85], esquematizados na Figura 2.3:

1. **Modelo do Domínio (ou do Especialista):** representação por meio de objetos,

regras ou procedimentos, que codifica explicitamente o conhecimento do especialista;

2. **Modelo do Aprendiz:** denota a proficiência do aprendiz no conteúdo tutorado (deduzido com base no Modelo do Domínio) e, adicionalmente, as preferências do aprendiz relativas ao processo de aquisição do conhecimento;

3. **Modelo (Didático-)Pedagógico:** abrange as estratégias de ensino que, interagindo com os modelos anteriores, permite que a apresentação do conteúdo seja adaptada às preferências e ao progresso do aprendiz; e

4. **Módulo de Interface:** basicamente, provê a dinâmica de troca de informações (interação) entre o aprendiz e o sistema.

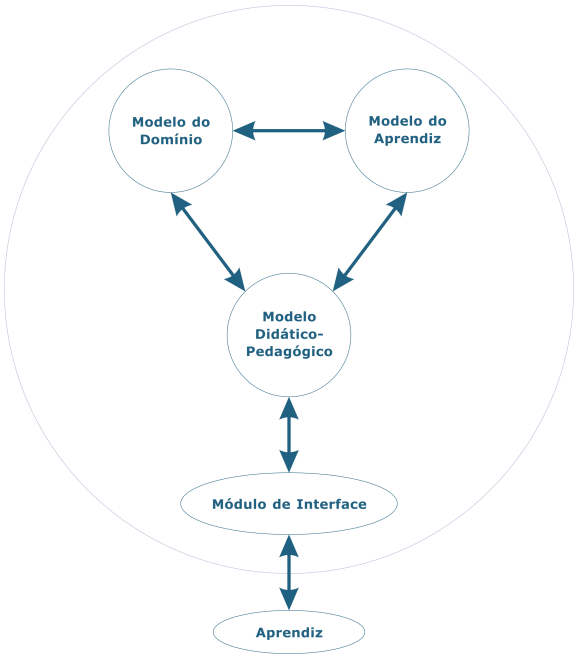


Figura 2.3: Arquitetura geral de um STI (adaptada de [85])

Além disso, [85] também registra a perspectiva histórica de que, até o início da década de 1990, era raro encontrar dois STIs baseados numa mesma arquitetura. Entretanto, verificou notável consenso de que uma arquitetura seria basicamente composta dos três primeiros componentes citados (modelos do Domínio, do Aprendiz e Pedagógico). Na mesma época, pesquisas identificaram e adicionaram um quarto componente, o Módulo de Interface [112].

Outras arquiteturas são dirigidas por diferentes enfoques, podendo incluir elementos ou mesmo decompor e/ou suprimir os módulos descritos. Apenas como exemplo, há arquiteturas que contém catálogo de erros prováveis do aprendiz para que, quando este cometer um erro durante a resolução de problemas, o STI possa avaliar o desvio do curso de aprendizado implicado.

Relativamente ao **Modelo do Aprendiz**, [85] destaca se tratar de uma representação dinâmica do conhecimento e habilidades emergentes deste usuário específico. Em adição, o mesmo estudo enfatiza a inviabilidade de um STI se manter sem tais informações. Idealmente, o referido modelo deveria incluir todos os aspectos cognitivos e comportamentais do indivíduo que pudessem repercutir no aprendizado. Entretanto, a complexidade de construir um modelo tão completo seria tanto não-trivial quanto praticamente impossível. Convém ainda considerar a natureza restritiva da interface convencional entre o aprendiz e um STI, pois não captura dados como expressões faciais e vocais, além de fatores psicológicos como motivação ou tédio, cruciais no aprendizado.

Dentre muitas das técnicas utilizadas para constituir o Modelo do Aprendiz, [48] destacam-se:

1. reconhecer padrões no histórico de respostas fornecidas pelo aprendiz;
2. comparar a conduta do aprendiz com um especialista para então observar semelhanças e discrepâncias;
3. considerar as preferências do aprendiz, bem como objetivos particulares;
4. detectar falhas recorrentes nas resoluções e respostas; e
5. indicar objetivos particulares.

Nesse sentido, [103] constatou que as funções exercidas pelo Modelo do Aprendiz podem ser classificadas em seis tipos. Abaixo, cita-se cada uma delas e a incidência da respectiva contribuição ao integrarem um STI:

1. **Corretiva:** eliminar falhas no conhecimento do aprendiz;

2. **Elaborativa:** construir o conhecimento, então incompleto, do aprendiz;

3. **Estratégica:** propor mudanças significativas na estratégias de tutoria, além de embasar decisões táticas que abranjam as funções anteriores 1 e 2;

4. **Diagnosticadora:** constatar falhas no conhecimento do aprendiz;

5. **Preditiva:** determinar prováveis respostas do aprendiz às ações tutoriais do sistema;

6. **Avaliativa:** estimar o conhecimento do aprendiz e/ou do STI.

Em complemento, considera-se que a adaptação às características do aprendiz seja um dos principais propósitos de um STI. Busca-se, com isso, promover um processo de ensino eficiente e individualizado. Diante disso, [29] constatou que o Modelo do Aprendiz, bem como a arquitetura clássica de um STI apresentam limitações, pois tendem a não promover:

1. descentralização das atividades e a especialização dos módulos;

2. múltiplas representações externas do conhecimento;

3. aprendizagem colaborativa;

4. modificação mais profunda em representações, exemplos e conteúdos para se adaptar ao aprendiz;

5. detecção do estado motivacional do aprendiz;

Uma distinção evidente na formalização do Modelo do Aprendiz baseia-se na maneira como as informações obtidas são representadas internamente, apoiando-se em alguns modelos de descrição. Seguem quatro principais modelos cognitivos, conforme [48, 93]:

1. **Modelo de Medição de Performance:** consiste numa maneira simples de descrever um aprendiz em relação a um domínio de conhecimento, baseada na medição da respectiva performance na resolução de problemas da área. Compara-se, a performance (não o conhecimento) do aprendiz com a de um perito. A medição pode

ser feita de formas variadas, como notas de avaliações, desempenho em exercícios, observação direta, entre outras;

2. **Modelo de Sobreposição (*Overlay*):** trata da representação do conhecimento do aprendiz como um subconjunto do conhecimento do domínio. Com a sobreposição desses conjuntos, fica evidente quais objetivos devem ser atingidos pelo aprendiz. Todavia, implica que a representação de conhecimento de ambos os modelos sejam comparáveis, ou a mesma;

3. **Modelo de Perturbação ou de Descrição de Erros (BUGGY):** também relaciona o Modelo do Aprendiz como subconjunto do conhecimento do domínio. Assume que os erros do aprendiz podem provir não somente da ausência de algum conceito, como também de concepções errôneas do que foi adquirido durante o processo de ensino;

4. **Modelo de Simulação:** compõe-se de um modelo (geralmente procedural) cuja execução simula um comportamento que tenha desempenho próximo ao aprendiz no domínio de conhecimento abordado. Tal comportamento acaba sendo uma predição das ações do aprendiz para resolver determinada tarefa. Com isso, pode-se tanto explicar os passos do aprendiz durante a resolução, quanto a solução propriamente dita.

Diante disto, foi selecionado o **Modelo de Sobreposição** para aprofundamento de estudos como solução mais cabível ao objetivo desta proposta. As seções 4.1 e 4.2 justificam como a descrição de perícias do domínio de Programação de Computadores, por meio do grafo genético, é privilegiada pelo modelo escolhido.

CAPÍTULO 3

EXPERIMENTO COM O AMBIENTE MÚLTIPLA

O ambiente interativo de aprendizagem MÚLTIPLA (Multiplicador de Linguagens Tipificado em Plataforma de Aprendizagem), conforme já abordado na Seção 1.2, adveio da necessidade de contemplar o aprendizado na fronteira entre aquisição de princípios e desenvolvimento de perícias em Programação de Computadores. Integrou o trabalho de mestrado do proponente [74] e foi resultado de intensa pesquisa no que se remete a Múltiplas Representações Externas (MREs) e metacognição.

Como passo subsequente, para início das pesquisas em nível de doutorado, investiu-se na avaliação formal do MÚLTIPLA em situações reais de aprendizagem. O objetivo foi aferir os benefícios de utilização do protótipo desenvolvido. Consequentemente, a coleta de informações a partir do uso inicial tem validado a hipótese teórica que embasou o protótipo, como também tem fornecido indicativas de alterações para o aperfeiçoamento do referido ambiente.

Portanto se assumiu, como hipótese legada para o experimento, que o MÚLTIPLA potencializa o estudo na área e mostra-se benéfico por empregar MREs no suporte de uma abordagem metacognitiva de ensino. Tratou-se de um ponto de partida para reconhecer lacunas de contribuição, no ensino de Programação de Computadores, que conduzissem a uma hipótese de pesquisa em nível de doutorado.

Com a realização do experimento, conjeturou-se sobre a extensão do formalismo, e consequentes benefícios, para o processo de autoria. Foi especialmente considerada a influência de representações adequadas na elicitação e catalogamento de enunciados em domínios de natureza prática. A suposição recai tanto sobre representação interna (RI) quanto externa (RE) e impactou na hipótese firmada para o presente doutorado (explicitada na Seção 1.3).

Aponta-se, como pesquisa futura, o investimento em técnicas de Inteligência Artificial

que elevem o nível do apoio fornecido pelo MÚLTIPLA de sintático para semântico. Nesse âmbito, a pesquisa atual estabelece critérios que propiciam certo grau de automatização na progressão dos enunciados a serem explorados (conformes formalismos delineados no Capítulo 4).

3.1 Coleta e Análise de Resultados a Partir do Uso Inicial do MÚLTIPLA

O escopo desta etapa residiu em mensurar/quantificar a contribuição do ambiente MÚLTIPLA para com o aprendizado em Programação de Computadores. A pesquisa foi realizada pelo proponente enquanto professor do Departamento de Ciência da Computação na Universidade Estadual do Centro-Oeste[1] (Unicentro). Consistiu em um projeto credenciado junto à Pró-Reitoria de Pesquisa e Pós-Graduação da universidade mediante aprovação do departamento mencionado. Houve também autorização e apoio dos acadêmicos envolvidos para que a coleta fosse conduzida. Os resultados, por fim, foram documentados em um relatório técnico final daquela pró-reitoria [75].

3.1.1 Sujeitos

Foram sujeitos desta pesquisa os alunos das turmas **A** e **B** da disciplina de **Programação de Computadores I**, ministrada pelo proponente na primeira série do curso de Ciência da Computação da Universidade Estadual do Centro-Oeste. Os alunos foram classificados, por um pré-teste (de acordo com critérios relatados na sequência), em dois grupos sem intersecção: o grupo de controle (GC) e o grupo experimental (GE).

Grupo de controle: composto por 10 alunos, presenciou as aulas regulares da disciplina (que contemplam teoria e prática, tendo sido ministradas em laboratório) e utilizou dos recursos de monitoria, bem como atendimento docente na condição de atividades complementares.

[1] Câmpus Santa Cruz, Guarapuava, Paraná.

Grupo experimental: constituído por outros 10 alunos, os quais participaram das aulas regulares e, em horário extra, de atividades dirigidas no ambiente MÚLTIPLA.

Salienta-se que os grupos de controle e experimental foram compostos de alunos provenientes tanto da turma A quanto da B. Portanto, receberam tratamento imparcial quanto ao conteúdo ministrado em carga-horária letiva e quanto às avaliações.

Dentro disso, havia ainda a incongruência de que o grupo experimental, sendo submetido a atividades extras, teria melhor desempenho independentemente da utilização do MÚLTIPLA. Como o ambiente contempla somente exercícios que são resolvidos com o compilador Pascal em carga-horária letiva, orientou-se os alunos do grupo experimental de que resolver os mesmos exercícios também em horário de aula seria facultativo. Em contraste, apenas o grupo de controle participou das atividades de monitoria e de atendimento docente ofertadas no semestre.

Para a análise deste estudo, documentam-se os resultados dos 10 alunos de cada grupo. Em separado, consideram-se também os resultados somente dos alunos que não desistiram da disciplina ou do curso.

3.1.2 Instrumentos

Os instrumentos utilizados para a observação e análise foram:

- **Pré-teste:** avaliação de habilidades pré-algorítmicas;
- **Médias da disciplina:** primeiro semestre, segundo semestre e médias anuais;
- **Faltas:** primeiro semestre, segundo semestre e total anual;
- **Desistência:** da disciplina e do curso;
- **Pós-teste:** saldo de aprovações, exames e reprovações.

3.1.3 Procedimentos

O MÚLTIPLA abrange a exploração dos assuntos de manipulação de variáveis, instruções de entrada e de saída, estruturas condicionais e de repetição. Todos compreendidos no

primeiro semestre do programa correspondente à referida disciplina.

Ambos os grupos assistiram às aulas da disciplina em horário regular. Tanto as aulas teóricas quanto as práticas foram ministradas em um dos laboratórios de informática do curso. A sequência de ensino para as aulas compunha-se da exposição e explicação do conteúdo, com o uso de apresentações digitais e da lousa, bem como a resolução de exercícios (implementação de programas) na linguagem Pascal[2].

Em particular, o grupo de controle teve participação nos horários destinados à monitoria e naqueles de atendimento docente. Sendo o curso de Ciência da Computação oferecido em período integral, é tão frequente a presença dos acadêmicos em tais atividades que são vistas como uma extensão dos horários de aula propriamente ditos.

O recurso de monitoria foi ofertado em regime de 6 horas-aula semanais, totalizando uma carga-horária de 114 horas-aula (19 semanas). O estudante monitor dispunha do laboratório de informática com lousa, assim como ocorria nas aulas da disciplina, e auxiliava no entendimento do conteúdo, como também na resolução de exercícios na linguagem Pascal.

Por sua vez, o atendimento docente, realizado pelo ministrante da disciplina, compreendia 4 horas-aula semanais e acumulou 76 horas-aula naquele semestre. A assistência igualmente incidia sobre a compreensão do conteúdo e de exercícios relacionados. Era prestada no laboratório de informática nas mesmas condições que a monitoria.

O grupo experimental, exclusivamente, participou de encontros em que atividades orientadas ao ambiente MÚLTIPLA eram prescritas. Ao todo foram feitos cinco encontros de 2 horas-aula cada. Portanto, totalizaram-se 10 horas-aula de trabalho com o grupo experimental, equivalente a $7,35\%$ da carga-horária da disciplina (136 horas-aula). Os encontros ocorreram todos no primeiro semestre, com periodicidade variável de acordo com o tempo disponível dos alunos (calendário de provas e trabalhos). Durante aquele semestre, não houve frequência (mas inexistiu restrição) do grupo nos horários destinados à monitoria ou ao atendimento docente institucionalizado. Os encontros consistiram nas únicas atividades complementares em Programação de Computadores.

[2]As IDEs FreePascal e PascalZim eram utilizadas.

3.1.4 Resultados Coletados

Os resultados provieram dos instrumentos de observação e análise especificados na Seção 3.1.2. São interpretados conforme segue.

3.1.4.1 Pré-Teste

Todos os alunos da disciplina foram sujeitados a um teste com o objetivo de sondagem e critério para constituição dos grupos. O teste enfocou a avaliação de habilidades pré-algorítmicas, ou seja, conhecimentos que indicassem a estruturação de raciocínio lógico conforme o que demanda a prática de Programação de Computadores. Objetivou-se avaliar tais conhecimentos sem explicitamente se remeter a instruções específicas de linguagem de programação.

O teste aplicado, transcrito juntamente com o gabarito no Anexo A, foi composto por dezesseis questões que avaliavam implicitamente diferentes tópicos contemplados pelo ambiente e pela ementa da disciplina. Cada questão foi criteriosamente escolhida para remeter a um tópico específico de Programação de Computadores. Os alunos tiveram o tempo de 4 horas-aula para resolver o teste.

Com os resultados do pré-teste, teve-se condições de classificar os alunos em dois grupos bastante parecidos, ou seja, equiparáveis (fenômeno estatístico denominado homocedasticidade). Além da média de ambos os grupos serem idênticas (algo difícil de ocorrer, pois se buscava médias próximas), cuidou-se para que houvesse paridade de alunos entre os grupos. A paridade ocorre no sentido de que dois alunos de desempenho próximos são preferencialmente alocados em grupos distintos, desde a dupla de melhor desempenho até aquela de pior. A decisão de qual aluno da dupla ficaria em determinado grupo foi também influenciada pela disponibilidade de horário para os encontros periódicos para utilização do ambiente MÚLTIPLA.

Os resultados do pré-teste são apresentados pela Tabela 3.1.

	GE	GC
Média aritmética	5,0309	5,0309
Desvio padrão	1,6676	1,9235
Mediana	5,1531	4,8500
Menor Valor	1,5156	1,2125
Maior Valor	7,2750	7,8813
Amplitude	5,7594	6,6688

Tabela 3.1: Avaliação de Problemas Pré-Algorítmicos

3.1.4.2 Médias do Primeiro Semestre

A média do primeiro semestre foi composta por duas avaliações presenciais e dois trabalhos correlatos às avaliações. O grupo experimental apresentou média semestral $13,6\%$ superior à obtida pelo grupo de controle. Excetuando-se um aluno que desistiu do curso, componente do grupo de controle, a superioridade decrementa para $3,2\%$. Não houve desistentes no grupo experimental.

No entanto, merece registro o percentual de crescimento dos grupos com relação ao pré-teste aplicado. O grupo experimental apresentou crescimento médio de $32,91\%$, com crescimento máximo de $163,92\%$. O grupo de controle, por sua vez, apresentou crescimento médio de $4,74\%$ e crescimento máximo de $128,57\%$. Com a exceção do aluno desistente, o crescimento médio do grupo de controle alcança $11,80\%$.

As referidas informações estão justapostas pelas tabelas 3.2, 3.3 e 3.4.

	Média	Crescimento	% Crescimento
Média aritmética	6,1900	1,1581	32,91%
Desvio padrão	1,8150	0,9961	49,31%
Mediana	6,3500	1,2906	22,79%
Menor Valor	3,6000	-0,3406	-8,64%
Maior Valor	9,3000	2,4844	163,92%
Amplitude	5,7000	2,8250	172,56%

Tabela 3.2: Grupo Experimental - Médias do Primeiro Semestre

	Média	Crescimento	% Crescimento
Média aritmética	5,4500	0,4181	4,74%
Desvio padrão	3,0457	2,2211	52,37%
Mediana	4,6000	-0,0406	-0,03%
Menor Valor	0,5000	-2,5531	-58,76%
Maior Valor	9,7000	5,4563	128,57%
Amplitude	9,2000	8,0094	187,33%

Tabela 3.3: Grupo de Controle - Médias do Primeiro Semestre

	Média	Crescimento	% Crescimento
Média aritmética	6,0000	0,5438	11,80%
Desvio padrão	2,6519	2,3178	50,25%
Mediana	5,0000	0,2656	7,58%
Menor Valor	2,6000	-2,5531	-49,55%
Maior Valor	9,7000	5,4563	128,57%
Amplitude	7,1000	8,0094	178,12%

Tabela 3.4: Grupo de Controle - Médias do Primeiro Semestre (Desistentes Excetuados)

3.1.4.3 Médias do Segundo Semestre

O cálculo da média do segundo semestre baseou-se somente em avaliações presenciais escritas. O grupo experimental teve 2 desistentes. O grupo de controle somou 4 desistentes. Os resultados das médias do segundo semestre levou em conta apenas os alunos que continuaram frequentando a disciplina.

Estima-se que, embora o MÚLTIPLA incidisse mais sobre os conteúdos do primeiro semestre, os resultados de um aprendizado produtivo são mais visíveis no segundo semestre, pois o ensino de Programação de Computadores não ocorre de forma compartimentada. A proficiência no segundo semestre é o produto de todos os conteúdos anteriormente vistos na disciplina.

O grupo de experimental teve média $59,21\%$ superior do que o grupo de controle. O crescimento agora é relativo à média do primeiro semestre. Dentro disso, o grupo experimental apresentou crescimento médio de $10,96\%$, contra um decréscimo do grupo de controle de $32,50\%$.

Percebe-se ainda que os valores do grupo experimental são menos discrepantes do que

aqueles do grupo de controle. Chama também atenção o fato de que a mediana do grupo experimental seja $8,15$, enquanto a do grupo de controle constitui-se em $4,35$.

Estas informações estão apresentadas nas tabelas 3.5 e 3.6.

	Média	**Crescimento**	**% Crescimento**
Média aritmética	7,5625	0,8500	10,96%
Desvio padrão	2,3157	1,2433	24,06%
Mediana	8,1500	1,0000	14,84%
Menor Valor	2,4000	-1,6000	-40,00%
Maior Valor	9,5000	2,2000	40,00%
Amplitude	7,1000	3,8000	80,00%

Tabela 3.5: Grupo Experimental - Médias do Segundo Semestre (Desistentes Excetuados)

	Média	**Crescimento**	**% Crescimento**
Média aritmética	4,7500	-1,3667	-32,50%
Desvio padrão	3,5664	1,7025	40,50%
Mediana	4,3500	-1,4000	-27,38%
Menor Valor	0,0000	-3,7000	-100,00%
Maior Valor	10,000	1,0000	11,11%
Amplitude	10,000	4,7000	111,11%

Tabela 3.6: Grupo de Controle - Médias do Segundo Semestre (Desistentes Excetuados)

3.1.4.4 Médias Anuais

A média anual de um aluno é dada sobre a média aritmética simples das semestrais. Apresentam-se aqui quatro tabelas que contrastam, além dos dois grupos, a exceção dos alunos desistentes. Nas duas primeiras, que incluem todos os 10 alunos de cada grupo, percebe-se $43,5\%$ de superioridade na média do grupo experimental. O crescimento apontado é de $26,43\%$ para o grupo experimental e de $-22,50\%$ para o grupo de controle.

Outros fatos relevantes, superiores no grupo de controle, são: **(a)** mediana: $6,85$ contra $4,15$; **(b)** menor nota: $1,8$ em contraste com $0,3$; e **(c)** maior crescimento: $111,13\%$ contra $34,32\%$.

Todas essas informações são mostradas pelas tabelas 3.7 e 3.8.

	Média	Crescimento	% Crescimento
Média aritmética	6,1400	1,1081	26,43%
Desvio padrão	2,7359	1,8633	49,27%
Mediana	6,8500	1,8109	31,04%
Menor Valor	1,8000	-2,5500	-54,32%
Maior Valor	9,3000	2,7375	111,13%
Amplitude	7,5000	5,2875	165,46%

Tabela 3.7: Grupo Experimental - Médias Anuais

	Média	Crescimento	% Crescimento
Média aritmética	4,2800	-0,7519	-22,50%
Desvio padrão	2,9914	1,8616	40,50%
Mediana	4,1500	-1,2406	-24,44%
Menor Valor	0,3000	-3,8531	-75,26%
Maior Valor	9,5000	1,8375	34,32%
Amplitude	9,2000	5,6906	109,57%

Tabela 3.8: Grupo de Controle - Médias Anuais

Os mesmos cálculos foram feitos desconsiderando os alunos que desistiram da disciplina ou do curso, conforme tabelas 3.9 e 3.10. Notaram-se diferenças ainda maiores quanto à superioridade do grupo experimental: **(a)** média: $31,4\%$ superior; **(b)** mediana: $7,55$ contra $5,35$; e **(c)** maior crescimento: $111,13\%$ contra $30,31\%$.

	Média	Crescimento	% Crescimento
Média aritmética	7,1625	1,9715	46,40%
Desvio padrão	1,9056	0,4388	29,01%
Mediana	7,5500	1,9813	38,56%
Menor Valor	3,2000	1,3313	19,96%
Maior Valor	9,3000	2,7375	111,13%
Amplitude	6,1000	1,4063	91,17%

Tabela 3.9: Grupo Experimental - Médias Anuais (Desistentes Excetuados)

3.1.4.5 Saldo de Aprovações, Reprovações e Exames

Tanto o grupo experimental quanto o grupo de controle tiveram 2 alunos para exame (com a exclusão dos desistentes). Embora o número de alunos em exame seja o mesmo, no grupo experimental apenas 3 obtiveram reprovação direta, contra 5 do grupo de con-

	Média	Crescimento	% Crescimento
Média aritmética	5,4500	-0,7135	-15,17%
Desvio padrão	2,9838	2,1366	37,67%
Mediana	5,3500	-1,2594	-19,85%
Menor Valor	1,3000	-3,8531	-74,77%
Maior Valor	9,5000	1,8375	30,31%
Amplitude	8,2000	5,6906	105,08%

Tabela 3.10: Grupo de Controle - Médias Anuais (Desistentes Excetuados)

trole. Ao final, o grupo de controle contou com $33,33\%$ de aprovação, enquanto o grupo experimental totalizou $87,50\%$ de aprovação.

Tais informações foram todas compiladas nas tabelas 3.11, 3.12, 3.13 e 3.14.

	GE		GC	
	Alunos	**%Alunos**	**Alunos**	**% Alunos**
Aprovados	5	50%	2	20%
Reprovados	3	30%	5	50%
Em Exame	2	20%	3	30%

Tabela 3.11: Saldo de Aprovações, Reprovações e Exames

	GE		GC	
	Alunos	**%Alunos**	**Alunos**	**% Alunos**
Aprovados	5	62,50%	2	33,33%
Reprovados	1	12,50%	2	33,33%
Em Exame	2	25,00%	2	33,33%

Tabela 3.12: Saldo de Aprovações, Reprovações e Exames (Desistentes Excetuados)

	GE		GC	
	Alunos	**%Alunos**	**Alunos**	**% Alunos**
Aprovados	7	70%	2	20%
Reprovados	3	30%	8	80%

Tabela 3.13: Saldo de Aprovações e Reprovações

3.1.4.6 Média do Percentual de Faltas

Um provável indicativo da motivação dos alunos que compuseram o grupo experimental reside no índice de faltas constatado (vide Tabela 3.15). Os alunos do grupo de controle

	GE		GC	
	Alunos	**%Alunos**	**Alunos**	**% Alunos**
Aprovados	7	87,50%	2	33,33%
Reprovados	1	12,50%	4	66,67%

Tabela 3.14: Saldo de Aprovações e Reprovações (Desistentes Excetuados)

faltaram $24,15\%$ a mais do que aqueles do grupo experimental. Sendo a carga horária da disciplina 136 horas-aula, destas, 76 foram ministradas no primeiro semestre e 60 no segundo semestre. Exige-se, ao menos, 75% de frequência.

	GE	**GC**
Primeiro Semestre	10,00%	15,79%
Segundo Semestre	26,00%	28,00%
Ano Letivo	17,06%	21,18%

Tabela 3.15: Média do Percentual de Faltas

3.1.4.7 Desistência da Disciplina

Há alto índice de desistência das disciplinas e do curso no primeiro ano de Ciência da Computação. Tal fato remeteu ao estudo e aperfeiçoamento da grade curricular do curso por parte da Coordenação do Departamento de Computação. Os docentes observaram existir grande exigência das disciplinas das área de Matemática e Física, que ultrapassam Programação de Computadores I em termos de carga-horária, mas desmotivam o aluno porquanto (em perspectiva discente) assumem o papel de descaracterizar o curso em um primeiro contato.

A despeito dessa situação, ainda se percebeu diferenças entre os alunos do grupo experimental e os do outro grupo. Além do número de alunos desistentes ser menor no grupo experimental (Tabela 3.16), o momento da desistência ocorreu posteriormente neste grupo se comparado ao de controle. Em média, os alunos do grupo experimental assistiram $77,94\%$ das aulas da disciplina antes de desistirem (suficiente para aprovação por frequência, inclusive), enquanto os alunos do grupo de controle desistiram em média em $70,96\%$ da carga-horária da disciplina (conforme Tabela 3.17).

	GE		GC	
	Alunos	**%Alunos**	**Alunos**	**% Alunos**
Desistência da Disciplina	1	10,00%	3	30,00%
Desistência do Curso	1	10,00%	1	10,00%

Tabela 3.16: Desistência da Disciplina

	GE		GC	
	Alunos	**%Alunos**	**Alunos**	**% Alunos**
Última Aula Assistida	106	77,94%	97	70,96%

Tabela 3.17: Última Aula Assistida por Alunos Desistentes (Média)

3.2 Discussão dos Resultados à Luz do MÚLTIPLA

Conforme antedito no início deste capítulo, diante do que se pesquisou até então, inexistia ambiente, no nicho especificado de Programação de Computadores, que propiciasse uma abordagem em que mais de uma representação externa do algoritmo fosse utilizada para promover o controle do aprendiz sobre o próprio desenvolvimento. O arcabouço conceitual adotado no MÚLTIPLA teve relevante embasamento nas concepções ligadas a micromundos e MREs, ambas apresentadas na sequência.

Os exemplos citados no decorrer desta seção são apresentados na íntegra ao final do documento (anexos B, C e D).

3.2.1 O Potencial de Micromundos

Conforme explicitado em resenha bibliográfica, um micromundo consiste num ambiente semanticamente rico por oferecer uma escala variada de recursos de interação. Tais ambientes, constituem ferramentas de autoestudo que assistem à construção de conhecimento por parte do aprendiz. Nesse sentido, o aprendiz deve ser visto como um agente ativo na interação, e o micromundo, um agente passivo.

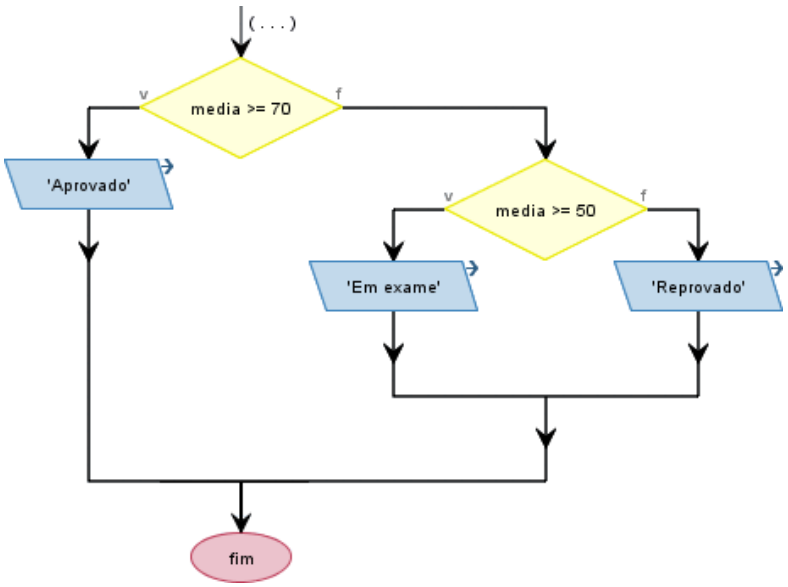
Convém ressaltar que um micromundo voltado à aprendizagem de programação difere de um depurador convencional. Embora este último possa ter utilidade em cursos introdutórios, um micromundo merece destaque por focar representações de mais alto nível, bem

como a possível correspondência e sincronização entre tais representações, cujo caráter é exclusivamente educacional.

3.2.1.1 A Finalidade de Fluxogramas

Baseou-se a escolha da notação de fluxograma, como correspondente ao algoritmo textual, na notável necessidade do aprendiz de desenvolver conceitos de mais baixo nível que precedem a noção de escopos aninhados das linguagens de alto nível de abstração. Entre tais conceitos, sobressai a compreensão do funcionamento de desvios progressivos e regressivos, mais explícitos no fluxograma.

O trecho de algoritmo apresentado pela Figura 3.1 expõe um caso de desvio progressivo. A instrução de saída que será executada depende exclusivamente do valor assumido pela variável *media*. Supondo, então, que lhe seja atribuído *100*, no fluxograma fica evidente o passo dado até a impressão da mensagem e, posteriormente, a progressão ao fim do algoritmo. Porém, num código em linguagem de alto nível, o mesmo desvio pode não se mostrar transparente ao aprendiz.

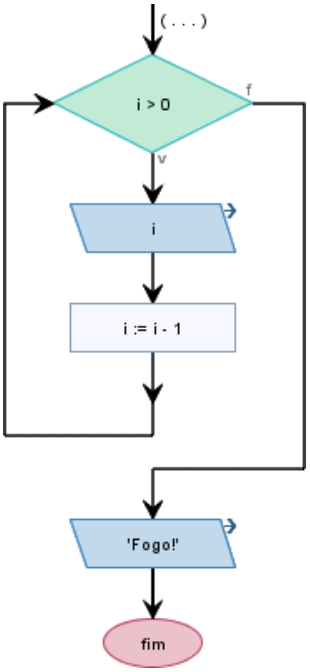



```
1  //...
2    if (media >= 70) then
3       writeln('Aprovado')
4    else
5       if (media >= 50) then
6          writeln('Em exame')
7       else
8          writeln('Reprovado');
9  end.
```

Figura 3.1: Desvio progressivo em um trecho de algoritmo

De forma análoga, o trecho de algoritmo expresso pela Figura 3.2 demonstra uma ocorrência de desvio regressivo. Independentemente de qualquer outra circunstância, se o

fluxo de controle está na atribuição $i := i + 1$, haverá em seguida um desvio regressivo para a avaliação do condicional da estrutura *enquanto-faça*. No fluxograma, o desvio sucede de forma bastante natural. Em uma linguagem de alto nível, contudo, isso não se manifesta, exigindo conhecimento prévio.




```
1  //...
2    while (i > 0) do
3       begin
4          writeln(i);
5          i := i - 1;
6       end;
7    writeln('Fogo!');
8  end.
```

Figura 3.2: Desvio regressivo em um trecho de algoritmo

Diante disso, acredita-se que o entendimento do fluxo de controle em instruções isoladas se faz muito necessário para consolidar a aquisição de princípios de Programação de Computadores. Ademais, tal conhecimento não apenas orientará o aprendiz para a aquisição de perícia no referido campo, como também implicará positivamente no decorrer dessa atividade.

3.2.1.2 A Finalidade de Códigos em Pascal

A intenção de correlacionar o fluxograma com o código em Pascal adveio, primeiramente, da necessidade de se remeter o aprendizado a uma situação concreta de Programação de Computadores. Elegeu-se Pascal por ser uma linguagem de alto nível que tem se mostrado adequada ao ambiente acadêmico[3]. Dentre outras características que impactaram na

[3]Na época da construção do MÚLTIPLA, utilizada em ambas as instituições que o proponente tinha contato direto: Universidade Federal do Paraná (UFPR) e Universidade Estadual do Centro-Oeste (Unicentro).

escolha da linguagem, cita-se a correspondência ao paradigma tratado (Imperativista), alta difusão de códigos escritos e variedade de compiladores[4].

Além disso, Pascal particulariza a ideia de programação estruturada ao redor do conceito de embutimento de escopos (ou contextos). Neste, um bloco de código contido em uma instrução (e.g. estrutura condicional *if*) corresponde a um subcontexto daquele mais geral que abriga a instrução recipiente (contexto).

Dessa forma, sobressai o fato de que a sequência de processamento das instruções de um bloco ocorre sem saltos. Isso não diz respeito às instruções alocadas em subcontextos (aninhamentos), que, por sua vez, podem ser desviadas por eventuais comandos condicionais. Assim, o final lógico da execução coincide com o final sintático do texto.

O trecho de algoritmo exibido na Figura 3.3 exemplifica o embutimento de contextos. No código, as instruções *writeln(i)* e *i := i - 1* estão contidas em um subcontexto delimitado pelas palavras reservadas *begin* e *end* que correspondem à estrutura *while-do*. Esta, por seu turno, encontra-se em um contexto maior, do algoritmo, delimitado por *begin* e *end.* (o último, conforme sintaxe, com ponto final).

Convém fazer aqui uma digressão para relatar que o próprio conceito de programação estruturada proveio, em parte, como reação à má legibilidade causada pelas instruções de controle [101]. O cenário precedente destacava-se pelo uso indiscriminado de instruções *go to*, que permitiam ao fluxo de controle saltar de uma instrução a outra não-adjacente, implicando redução crítica na legibilidade do código, como também na perturbação do fluxo de leitura do algoritmo como um todo. Como consequências típicas, havia a diminuição da capacidade de escrita (visto que fazê-lo exige uma releitura frequente do que foi redigido) e o aumento do custo de manutenção dos sistemas (relacionado com a legibilidade).

Isso posto, em favor da prática de programação estruturada e como característica evolutiva, muitas das linguagens atuais sequer implementam instrução equivalente ao *go to*. Portanto, vem a ser importante que o aprendiz tenha pleno domínio do conceito de embutimento de escopo.

Por fim, diante do apresentado, verifica-se a necessidade de melhor detalhar o conjunto

[4]Havendo compiladores em versões livres para várias plataformas.

```
1  program contagem_regressiva;
2  var
3    i : integer;
4  begin
5    i := 10;
6    writeln('Iniciando contagem regressiva...');
7    while (i > 0) do
8       begin
9          writeln(i);
10         i := i - 1;
11      end;
12   writeln('Fogo!');
13 end.
```

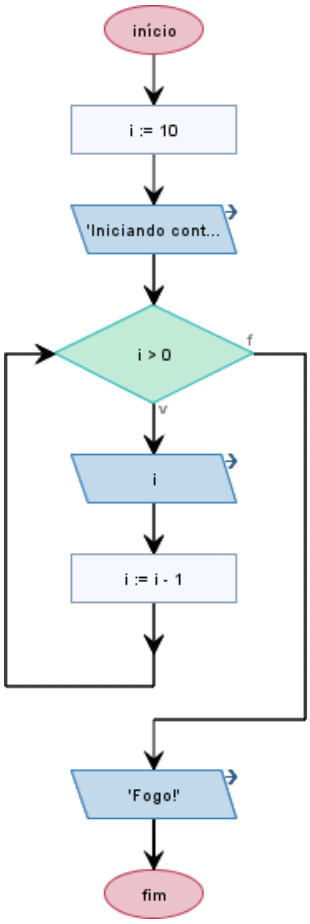


Figura 3.3: Subcontexto em um trecho de algoritmo

de MREs definido. Faz-se isso nas seções seguintes. A criação de controles de correspondência entre as funções didático-pedagógicas das representações envolvidas são descritas em [74].

3.2.2 Conjunto Multirrepresentacional Envolvido

Cabe a esta seção introduzir o conjunto multirrepresentacional estabelecido como embasamento do MÚLTIPLA. Foram estudadas e definidas (ou mesmo criadas), como parte das soluções da dissertação que originou o ambiente, várias categorias de RE. Houve cautela no desempenho de tal atividade porquanto se remete às prescrições delineadas pelo trabalho de Ainsworth [5, 6], cujas referências foram essenciais na pesquisa de elaboração do protótipo. Diante disso, como parâmetro de projeto, para que se disponha de um conjunto de MREs bem-definido, tem-se como tarefa indispensável precisar dimensões como número, informação, forma, sequência e tradução das REs componentes.

Assim, relaciona-se nesta seção o elenco das categorias de REs empregadas no protótipo, associadas ao devido valor metacognitivo. Embora se tenha consolidado tais categorias, leva-se em consideração que, com o recolhimento de resultados adicionais, estas possam ainda ser aperfeiçoadas em alguns aspectos, assim como outras mais possam vir a compor o conjunto. Frisa-se que as representações de fluxograma e o código em Pascal foram abordadas na seção anterior.

3.2.2.1 Cliques de Correspondência

Mediante um clique em determinado elemento do fluxograma (Figura 3.4), destaca-se automaticamente a linha de código Pascal equivalente. A situação inversa também ocorre. Esta RE tem suma importância para se relacionar o algoritmo na notação de fluxograma com a textual[5] associada.

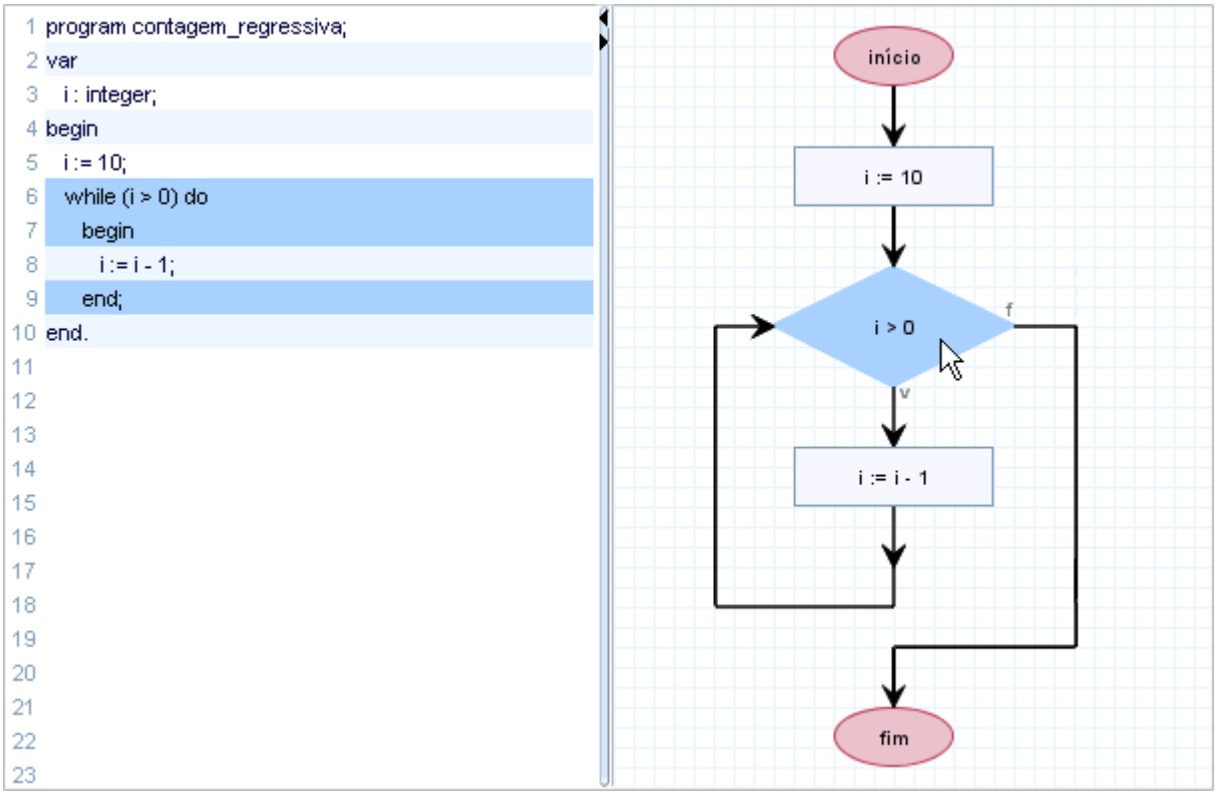


Figura 3.4: Clique de correspondência

[5]O algoritmo expresso pelo fluxograma é transposto para a representação textual. Conforme o módulo de Gramática do Domínio do MÚLTIPLA, que define uma metalinguagem e provê mecanismos de tradução, o fluxograma pode ser convertido tanto para um código equivalente em uma linguagem de programação determinada (utilizou-se Pascal) quanto para o algoritmo em Portugol (pseudocódigo).

3.2.2.2 Realce de Elementos

Durante a execução do algoritmo, são realçados o elemento de fluxograma e a linha de código onde se encontra o fluxo controle. Intenciona-se que isso ajude o aprendiz a assimilar a correspondência entre as notações, bem como a entender o funcionamento das estruturas empregadas (condicionais e de repetição).

Difere-se do Clique de Correspondência por ter caráter dinâmico, ou seja, o destaque é feito automaticamente de acordo com a progressão do fluxo de controle.

3.2.2.3 Recolhimento e Expansão de Blocos de Elementos

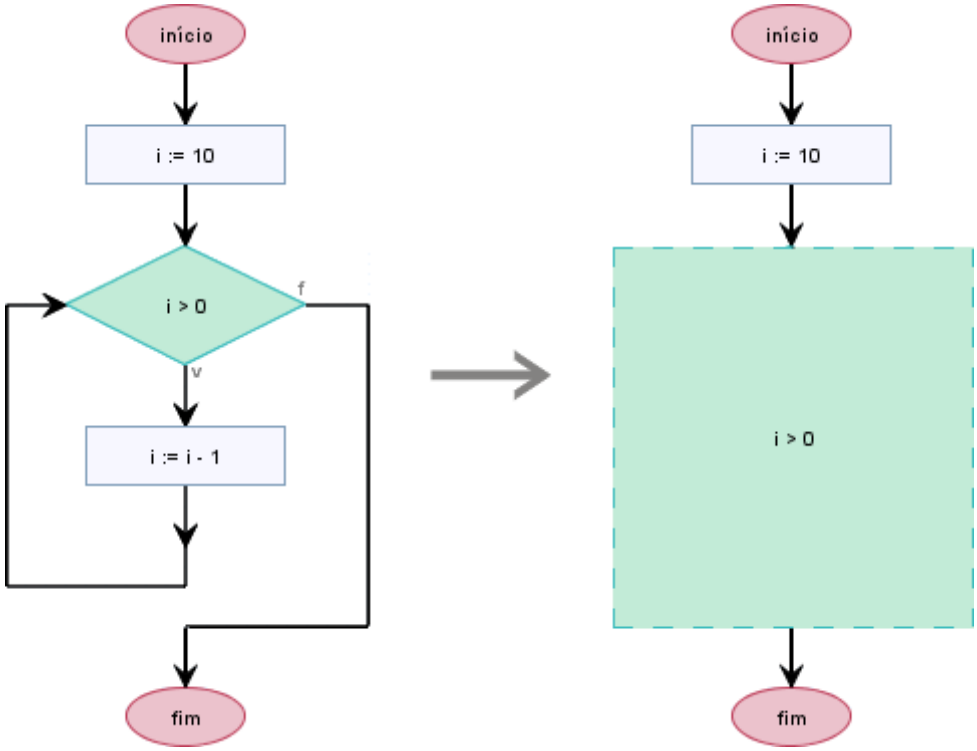


Figura 3.5: Recolhimento e expansão de blocos de elementos

Há situações em que se tem algumas estruturas condicionais e de repetição aninhadas e deseja-se limitar a visualização, por exemplo, à estrutura mais externa (que contém todas as outras). Esta representação (Figura 3.5) permite encapsular visualmente o que está contido dentro da estrutura escolhida, transfigurando-a numa espécie de "caixa-preta" (pois não se terá acesso ao que se passa dentro dela). Com isso, pretende-se focar a atenção

do aprendiz ao que lhe convém sem que ele se atenha aos elementos e ao fluxo que ocorre dentro da estrutura recolhida.

Conforme algumas observações, apontam-se melhorias à implementação desta RE no protótipo. A mais substancial consiste em recolher (e expandir) não somente o elemento no algoritmo gráfico, mas encontrar maneiras condizentes de fazê-lo também no textual. Uma solução que se antecipa consiste no emprego de reticências e/ou sinais de adição e subtração ao lado do início das estruturas, no algoritmo textual, indicando as possibilidades de expansão e recolhimento. Outro aperfeiçoamento possível seria o redimensionamento de elementos recolhidos a um tamanho menor (padronizado e não proporcional).

3.2.2.4 Cores

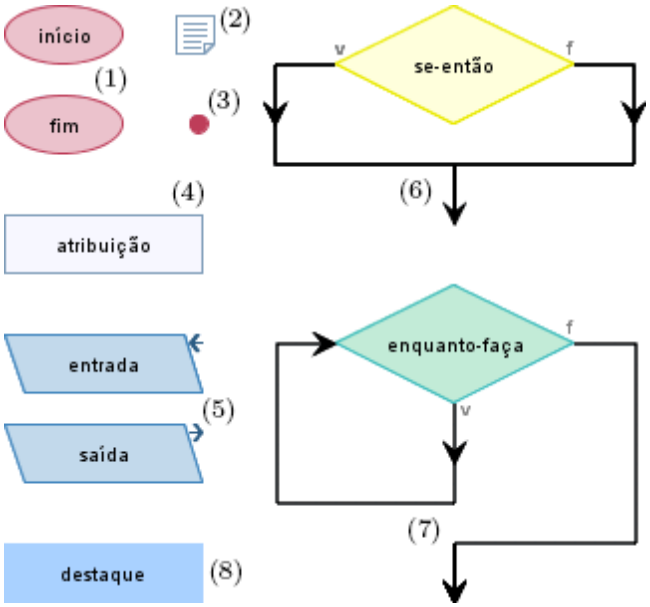


Figura 3.6: Cores

Investigaram-se cores que atuassem como elo indireto entre o algoritmo textual e o representado pelo fluxograma. Considerou-se a associação de cores a categorias de elementos representados, com a intenção de que o aprendiz fosse induzido a classificar e agrupar elementos semanticamente próximos. Além disso, preferiram-se cores brandas por, ao que se acredita, causarem uma menor exaustão visual ao aprendiz.

Conforme a Figura 3.6, a seguinte relação foi estabelecida: **(1)** demarcadores de início

e fim, róseo; **(2)** comentário textual, branco com contorno azul; **(3)** ponto de interrupção, vermelho escuro; **(4)** atribuição, variante de cinza; **(5)** entrada e saída, azul; **(6)** estruturas condicionais, amarelo; **(7)** estruturas de repetição, verde; e **(8)** qualquer elemento selecionado, azul claro.

Este último, a cor de destaque, necessitou de mudanças quanto à concepção inicial. Primeiramente, seria utilizado como realce para cada elemento, o complemento da cor original no padrão RGB[6] (efeito de "negativo"). Assim, as instruções de entrada e saída, por exemplo, adquiririam um tom laranja escuro, quase marrom. Todavia, além das cores resultantes serem discrepantes à interface, requeriam uma maior carga cognitiva por parte do aprendiz. Como são 7 categorias de elementos, com o destaque feito dessa maneira, haveria 14 cores para elementos de fluxograma. Evitando tal situação, optou-se pelo azul claro, semelhante àquele das canetas marca-textos, como cor padronizada para todos os destaques. Assim, há somente 8 cores para elementos do fluxograma.

3.2.2.5 Comentários Textuais

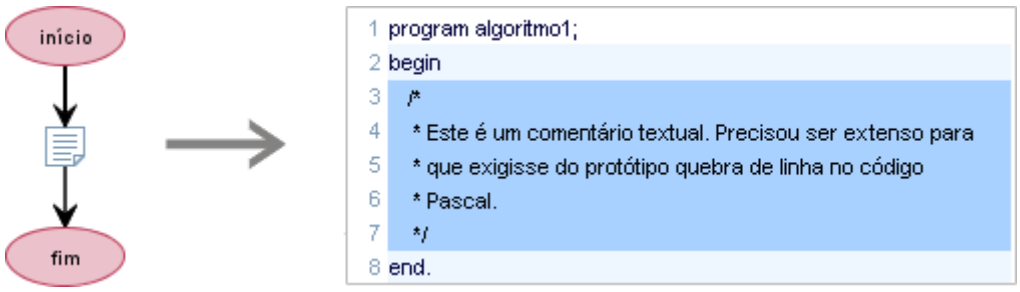


Figura 3.7: Comentários

São uma prática comum, tratada pela sintaxe das maioria das linguagens, que possibilita ao aprendiz inserir texto a fim de comentar linhas de código (aqui também elementos do fluxograma). Esses comentários foram incorporados ao algoritmo gráfico, sendo representados por ícones (Figura 3.7), inseridos em arcos, que podem ser expandidos para exibirem (ou permitirem edição) dos comentários digitados. Estes, juntamente aos demais elementos do fluxograma, são transpostos para o código em Pascal.

[6]Do inglês, *Red, Green, Blue* (Vermelho, Verde, Azul).

3.2.2.6 Pontos de Interrupção (*Breakpoints*)

Presente na generalidade dos depuradores, consiste aqui em permitir ao aprendiz apontar previamente elementos do fluxograma (ou de código) para que a execução do algoritmo seja interrompida, quando o fluxo de controle atingir os elementos demarcados. Têm grande utilidade porque o aprendiz pode verificar, entre outras coisas, se há blocos inatingíveis do algoritmo, inspecionar o valor de variáveis em determinados trechos e mesmo perceber erros semânticos bastante difíceis de serem detectados por compiladores (e.g. *loops* infinitos).

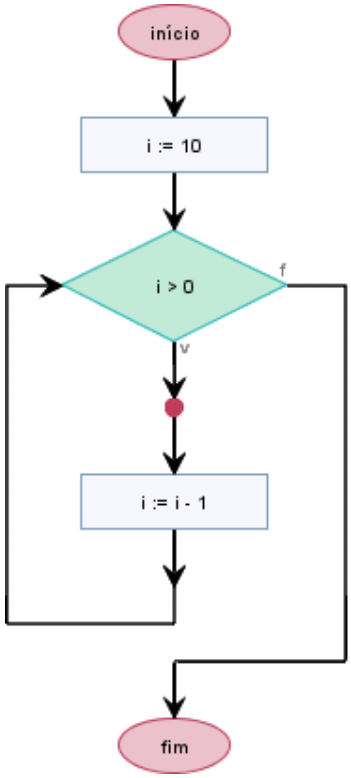


Figura 3.8: Ponto de interrupção precedendo uma atribuição

Inicialmente, esta representação perfazia-se de um ícone que era adicionado ao lado do elemento de fluxograma demarcado. Contudo, implicava ambiguidade, pois não ficava explícito se a parada ocorreria antes ou depois da execução daquele elemento demarcado. Levando isso em conta, alterou-se o ponto de interrupção de modo que fosse associado a um arco e não a um elemento particular (Figura 3.8). Consequentemente, transparece que, se a parada precede uma dada instrução, o fluxo de controle será interrompido antes

que esta última seja alcançada. Convém lembrar que o ponto de interrupção não possui equivalente sintático no algoritmo textual, portanto fica somente expresso no fluxograma.

3.2.2.7 Destaque de Variáveis com Valores Alterados

Cresce, junto ao número de variáveis empregadas, a dificuldade do aprendiz em gerenciá-las. Quando poucas variáveis são utilizadas, é fácil atentar-se a quando cada uma delas tem o respectivo valor alterado. Porém, dependendo da quantidade, tais ocorrências tornam-se difíceis de serem percebidas. Cabe a esta RE evidenciar, durante a execução, a variável que teve valor alterado em um dado instante. Faz-se isso por meio de uma tabela constando os nomes das variáveis seguidos dos valores atuais (Figura 3.9), sendo sempre destacada a variável que sofreu a última atribuição. Variáveis apenas declaradas são demarcadas com *<nulo>*.

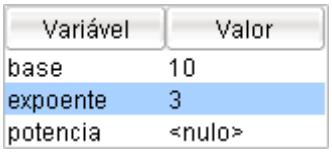


Figura 3.9: Destaque de variáveis com valores alterados

Cogita-se também, como melhoria, a representação das informações como Teste-de-Mesa, em que as variáveis são representadas por colunas de uma tabela cujas linhas são os valores em determinados instantes. Porém, quando implementada, provavelmente deva ser exibida mediante alternância com a área ocupada pelo algoritmo textual. Ou seja, opcionalmente, pode-se monitorar as variáveis pela tabela Teste-de-Mesa ao invés de se observar a correspondência dinâmica do algoritmo em fluxograma com o textual.

CAPÍTULO 4

FORMALISMOS ADOTADOS NA SOLUÇÃO DO PROBLEMA

Concerne ao atual capítulo a formalização do arcabouço conceitual adotado no presente projeto. Houve relevante delineamento perante aspectos de grafos genéticos e de um modelo de sobreposição para Programação de Computadores propriamente dito. Ambos são discorridos na sequência.

4.1 Grafos Genéticos

A corrente seção objetiva transcrever os conceitos de grafo genético[1] fundamentados por Ira P. Goldstein [51], bem como apresentar exemplos de utilização relacionados. Trata-se, essencialmente, de um modelo evolutivo para representação de conhecimento procedimental que serve como base para o desenvolvimento de tutores inteligentes.

O termo "genético", nesse contexto, é usado em sentido primordial, oriundo do chamado Método Genético que consiste no estudo da gênese e consequente desenvolvimento de fenômenos, ou seja, objetos de estudo. A tarefa de modelagem, em consequência, pode ser vista como um exercício de Epistemologia Genética[2], concentrando-se no estudo de como o conhecimento se origina e se desenvolve [90]. Inexiste, portanto, associação direta com o termo homônimo, da Computação Natural ou Bioinspirada, referente à hereditariedade.

Na época em que os grafos genéticos foram disseminados por Goldstein, no final da década de 70, anunciava-se um novo paradigma de Instrução Assistida por Computador (IAC), baseado no aprendiz. Os grafos genéticos foram levados à comunidade científica por meio de um relatório de pesquisa do Laboratório de Inteligência Artificial do Instituto

[1] Do inglês, *genetic graph*. Também denominado *overlay graph* (grafo de sobreposição).

[2] Teoria, desenvolvida por Jean Piaget, que estuda a complexa relação entre o sujeito e o objeto de conhecimento e, mais particularmente, as mudanças histórico-evolutivas que ocorrem nessa relação [90].

de Tecnologia do Massachusetts[3], posteriormente publicado no *International Journal of Man-Machine Studies*. Esses grafos eram introduzidos a partir do desenvolvimento de uma nova geração de sistemas de IAC que se destacava pelo uso de técnicas da Inteligência Artificial. A citada geração foi responsável por cunhar a Instrução Inteligente Assistida por Computador (IIAC), cuja denominação evoluiu para os conhecidos Sistemas Tutores Inteligentes (STI). A fim de contexto, a Seção 2.3.1 aborda com mais detalhes a transição mencionada.

Haviam recém-surgido tutores baseados na IIAC, destacados pelo próprio Goldstein, abrangendo uma diversidade de domínios, a saber: Geografia [21], Eletrônica [13], Teoria dos Conjuntos [108], Espectroscopia [105] e Jogos Matemáticos [18, 52]. As pesquisas relacionadas objetivaram superar a natureza restritiva das ferramentas baseadas em *scripts* (roteiros). A nova geração provia ambientes de aprendizagem reativos que podiam analisar uma ampla escala de respostas frente ao conhecimento do domínio embarcado.

A inclusão de um módulo que representasse o conhecimento do domínio era um avanço em relação aos ambientes anteriores baseados em *scripts*. Entretanto, as estratégias pedagógicas incorporadas aos tutores citados, responsáveis por transmitir tal conhecimento, eram bastante primárias.

Basicamente, o ensino era abordado sob a perspectiva de subconjuntos, conforme a Figura 4.1. A perícia, em um domínio de conhecimento, consiste em um conjunto de fatos ou regras. O conhecimento do aprendiz é modelado como um subconjunto da perícia do domínio. A tutoria resume-se em estimular o crescimento do subconjunto, geralmente, por intervir em situações em que um fato ou uma regra omissa é componente crítico para que se atinja a resposta correta.

Embora a perspectiva de subconjuntos fosse uma simplificação do processo de ensino, permitiu que as pesquisas se concentrassem na tarefa crítica de representar a perícia de um domínio. Todavia, a abordagem da época privava-se de descrever a maneira como o novo conhecimento evoluía a partir do conhecimento prévio. Omitiam-se processos evolutivos como a analogia, a generalização, a depuração e o refinamento.

[3]Artificial Intelligence Laboratory (Massachusetts Institute of Technology).

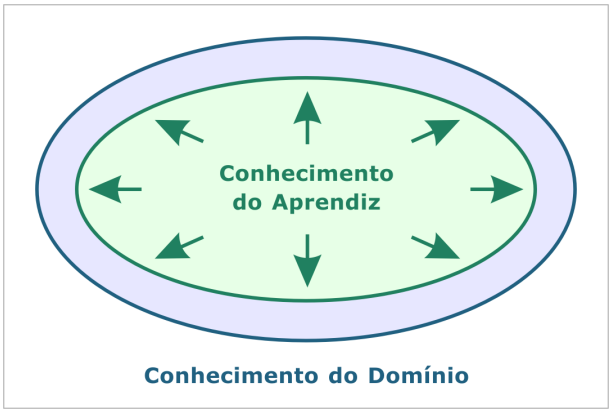


Figura 4.1: Ensino abordado sob a perspectiva de subconjuntos

Em resposta às deficiências mencionadas, Goldstein propôs o grafo genético como um arcabouço para representação de conhecimento procedimental sob uma perspectiva evolucionária. Com isso, pretendia-se auxiliar para que a IIAC fosse baseada no aprendiz, e não no especialista (perito). Mudanças consequentes foram:

1. Modelagem mais sofisticada do conhecimento do aprendiz e do próprio estilo de aprendizagem [18, 59, 22];

2. Aumento das possibilidades de comunicação entre o aprendiz e o tutor por meio de interfaces em linguagem natural [19];

3. Desenvolvimento de uma teoria própria para o ensino de perícias [25].

A sequência do presente texto procura reproduzir, com a fidelidade que foi possível, o conteúdo trazido por Goldstein ao meio científico sobre a introdução dos grafos genéticos. A maior adaptação, nesta tese, ocorreu na mudança do domínio de conhecimento que exemplifica os conceitos apresentados, remetido para o contexto da Programação de Computadores.

A primeira seção busca mostrar o contexto histórico em que ocorreu a divulgação dos grafos genéticos. São apresentados outros ambientes que também contribuíram nesse sentido e que motivaram as pesquisas de uma epistemologia genética para o conhecimento. Depois, definem-se os conceitos de grafos genéticos (Seção 4.1.2) e a ideia de um ambiente para a simulação de aprendizes (4.1.4). Extensões para o grafo genético são sugeridas

na Seção 4.1.5. Em seguida, descreve-se como aperfeiçoar a amplitude da orientação tutorial fornecida por um ambiente de aprendizagem (4.1.6) e a precisão da modelagem construída do aprendiz (4.1.7). Então se discute a teoria de aprendizagem que existe implícita no grafo genético e a consequente sugestão de uma métrica para a complexidade de aprendizado em termos das propriedades topológicas do grafo, conforme Seção 4.1.8. As considerações finais sobre o modelo de representação proposto pelos grafos genéticos (Seção 4.1.9) encerram o presente formalismo.

4.1.1 Contextualização Histórica

Goldstein especifica que a representação em grafo do conteúdo programático (*syllabus*) tem raízes nas pesquisas de IIAC. **Scholar** [21], o primeiro tutor da geração, emprega uma rede semântica[4] para representar fatos declarativos sobre Geografia. Contudo, a representação exprime apenas relações específicas do domínio. Não incorpora, portanto, relações evolucionárias entre os diversos fatos geográficos a fim de constituir níveis de conhecimento progressivamente mais refinados.

SOPHIE-1 [13] foi considerado o próximo marco na IIAC, destinando-se à resolução de problemas em Eletrônica. Como estratégia, esse tutor compara a hipótese de resolução do aprendiz, para um determinado circuito eletrônico, com a de um especialista, previamente incorporada ao ambiente. Oferece, então, recomendações quanto as hipóteses de resolução se desviam. O SOPHIE-1 utiliza uma representação procedimental, não divulgada, para o conhecimento do domínio. Entretanto, segundo Goldstein, o SOPHIE-1 não dispõe de acesso a uma representação detalhada, modularizada e orientada a humanos e também não possui representação da gênese das perícias abordadas.

O sucessor, **SOPHIE-2** [60], incorporou uma representação modularizada e dita antropomórfica para o conhecimento do domínio. Embora se possua uma fundamentação mais consistente para a tutoria baseada no especialista, ainda não há modelagem de como um aprendiz evolui até tal nível de competência.

[4]Representação de conhecimento definida como um grafo direcionado, cujos vértices denotam conceitos e as arestas expressam relações semânticas entre conceitos.

BUGGY [59] realiza a construção de modelos procedimentais de perícias aritméticas. O ambiente incorpora tanto a representação em grafo de perícias básicas quanto as relações evolucionárias. A representação de perícias básicas ocorre em termos de vértices interligados por arestas que denotam a relação de perícia-subperícia. A componente evolucionária reside nas relações de desvio para concepções errôneas de algumas perícias.

Por sua vez, **BIP-II** (*BASIC Instructional Program*) [113] tem por domínio a Programação de Computadores. Também utiliza um tipo específico de grafo para representar perícias básicas, mas incorpora um conjunto diferente de relações evolucionárias. O tutor implementa relações de analogia, generalização, especialização, pré-requisito e dificuldade relativa. Ainda assim, não dispõe da relação de desvio, recém-mencionada. O BIP-II conta, particularmente, com enunciados fornecidos pelo autor e, então, relacionados com perícias relevantes do grafo.

Isso posto, observa-se que o grafo genético pode ser considerado um descendente das representações anteriormente descritas. Os vértices caracterizam as perícias do domínio especificado. As arestas determinam as relações de analogia, especialização, generalização e pré-requisito (trazidas do BIP-II), mais a relação de desvio, provinda do BUGGY.

4.1.2 Definições, Conceitos e Terminologia

A introdução dos grafos genéticos no meio científico, feita por Goldstein, foi instanciada no domínio experimental do *Mundo de Wumpus*[5]. Trata-se de um jogo, proposto por Gregory Yob em 1972, que exercita habilidades básicas de Lógica e Probabilidade [99]. Com isso, definições e conceitos de grafo genético, na publicação original, foram exemplificados em termos de uma versão do jogo. O autor formulou uma epistemologia evolucionária do conhecimento de um jogador perito, assim provendo suporte para que um projeto baseado no aprendiz fosse aperfeiçoado.

Especificamente para esta tese, tendo em vista a contextualização dos exemplos, preferiu-se que as definições e conceitos fossem já aplicados à Programação de Computadores. Desse modo, pode-se oportunamente discutir regiões do grafo genético que formaliza

[5] *Hunt the Wumpus*. Existem muitas implementações *on-line* gratuitas do jogo.

o domínio de conhecimento abordado pela corrente pesquisa. O grafo genético construído representa um subconjunto de perícias que compõem o estereótipo de um programador experto. Consiste em uma versão experimental, proposta para conduzir a validação do trabalho e, assim sendo, sujeita a alterações pela comunidade científica. Detalhes sobre a modelagem no domínio de Programação de Computadores são expostos pela Seção 4.2.

Conforme antedito, o grafo genético formaliza a evolução de conhecimento procedimental através da representação de regras como vértices e de relações como arestas. A Figura 4.2 ilustra a disposição desses elementos e adianta uma região da modelagem do conhecimento no domínio de Programação de Computadores. Concerne às subseções seguintes apresentar as diferentes relações evolucionárias previstas pelo grafo genético. Posteriormente, na Seção 4.1.5, extensões são propostas para tornar mais apropriada a representação de conhecimento em casos e domínios particulares.

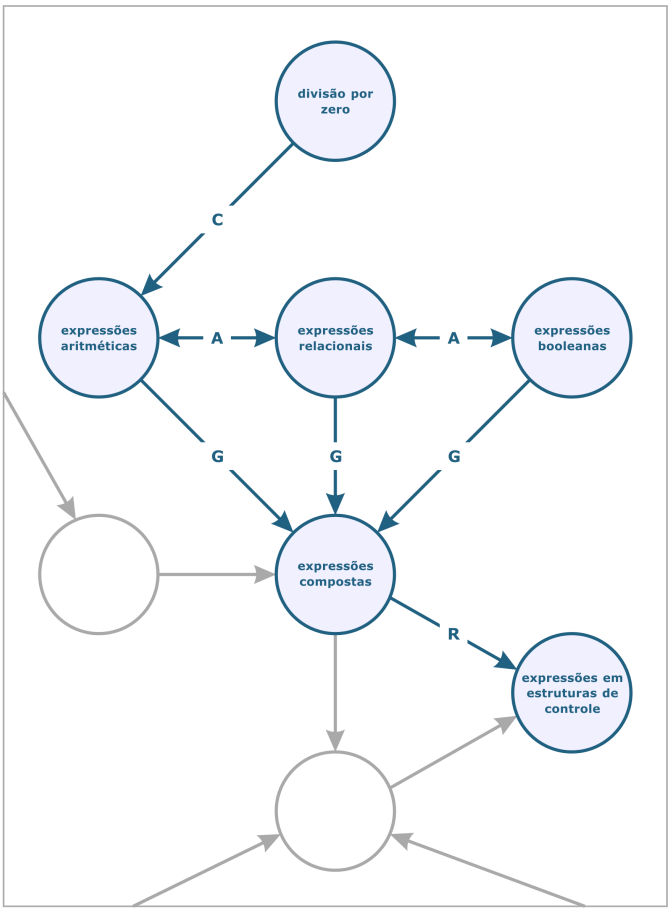


Figura 4.2: Exemplo de grafo genético contextualizado no domínio de Programação de Computadores

4.1.2.1 Regras

As regras procedimentais de um domínio de conhecimento são representadas pelos vértices do grafo genético. Equivalem, portanto, às perícias ou, mais especificamente, às subperícias do domínio representado. O processo de aprendizagem, nesses termos, diz respeito à apropriação, pelo aprendiz, de cada regra mediante o desenvolvimento da perícia correspondente.

4.1.2.2 Relações Genéticas

As relações genéticas especificam as associações evolucionárias entre regras, então representadas como vértices do grafo. Discutem-se, na sequência, as relações de generalização/especialização, analogia, desvio/correção, refinamento/simplificação. Depois disso, todas são devidamente exemplificadas e contextualizadas em Programação de Computadores.

Generalização / Especialização

R' é uma **generalização** de R, se R' é obtida a partir de R quantificada sobre alguma constante. Por sua vez, a **especialização** é a relação inversa. O conceito de quantificação provém da Lógica Matemática e a definição de Cálculo de Predicados foi trazida ao contexto de grafos genéticos para exercer a quantificação sobre fórmulas que representam regras, ao invés de sentenças lógicas.

Analogia

R' é uma analogia de R, se existe um mapeamento a partir das constantes de R' para as constantes de R. Goldstein aponta que se trata da definição estrutural empregada por [80]. É a única relação bilateral, ou seja, se R' é uma analogia de R, então R também é uma analogia de R'. Ademais, a transitividade se aplica à analogia, isto é, se A é análoga a B e B é análoga a C, então A e C são análogas.

Obviamente, nem todas as analogias definidas dessa maneira são relevantes. Entretanto, o grafo genético empenha-se em representar aquelas que se fazem úteis.

Simplificação / Refinamento

R' é um **refinamento** de R, se R' manipula um subconjunto dos dados manipulados por R, com base em algumas propriedades especializadas. Representa a evolução de uma regra para considerar um conjunto de distinções mais peculiares. **Simplificação** é a relação inversa.

Desvio / Correção

R' é um **desvio** de R, se R' tem o mesmo propósito de R, mas não cumpre o objetivo em alguma circunstância. **Correção** é a relação inversa. Os desvios surgem naturalmente no aprendizado como consequência de simplificações, supergeneralizações, analogias equivocadas, dentre outras causas. Enquanto cada regra pode ter uma forma anômala, ou seja, um desvio, o grafo genético também possui a finalidade de registrar os erros mais comuns inerentes ao aprendizado.

As regras de desvio assimiladas pelos grafos genéticos dizem respeito aos erros que surgem da aplicação correta de regras incorretas. Existe outra classe de erros provenientes da aplicação incorreta de regras corretas. Nesta última, os erros decorrem de causas como: **(a)** eventual falha ao se verificar todas as pré-condições de uma determinada regra; **(b)** má interpretação de dados; ou **(c)** engano no processo de busca por uma regra específica.

4.1.3 Exemplo em Programação de Computadores

A Figura 4.2, situada anteriormente, apresenta uma região do grafo genético que modela parte do conhecimento no contexto de Programação de Computadores. Diz respeito, essencialmente, às perícias associadas ao domínio sobre expressões compostas em Pascal. Os vértices são rotulados com identificadores breves para evitar sobrecarga visual e cognitiva. A aplicação dos diferentes elementos do grafo (perícias e conexões evolucionárias), recém-abordados, é exemplificada no contexto.

De acordo com a região ilustrada, tem-se que:

1. A perícia de *avaliar e construir expressões compostas* **generaliza** e interopera o

conhecimento específico das *expressões aritméticas*, *relacionais* e *booleanas*. Igualmente, as três últimas perícias incidem sobre uma fração **especializada** do conhecimento acumulado pela primeira;

2. As perícias de *avaliar e construir expressões aritméticas*, *relacionais* e *booleanas* são **análogas** entre si, pois encerram conhecimento semelhante embora versem sobre conjuntos de operadores distintos;

3. A perícia de *projetar (conjuntos de) expressões condicionais coesas e concisas para estruturas de controle* **refina** o conhecimento retido em *avaliar e construir expressões compostas*. Isto é, evolui a perícia original (tida como uma **simplificação**) a fim de considerar um conjunto de distinções mais peculiares;

4. A *incorrência da divisão por zero* é um **desvio** na tentativa (e da perícia) de *avaliar e construir corretamente expressões aritméticas*. A última perícia incorpora o conhecimento depurado[6] (sendo uma **correção**) do mesmo aspecto. Todo desvio tem uma origem genética natural e, no caso da divisão por zero, proveio de se generalizar que o operador de divisão, assim como o de multiplicação, é aplicável a quaisquer operandos inteiros ou reais, inclusive admitindo o zero como dividendo.

4.1.4 Ambiente para a Simulação de Aprendizes

Goldstein explorou o *Mundo de Wumpus* como domínio experimental. O grafo genético construído para tal finalidade conteve, inicialmente, 100 regras e 300 relações. A razoabilidade do grafo elaborado foi testada através de um ambiente para a simulação de aprendizes[7].

No referido ambiente, foi analisado o desempenho de vários aprendizes simulados, definidos em termos de diferentes regiões do grafo genético (conforme Modelo de Sobreposição detalhado em 4.1.7.1). Os aprendizes correspondiam, portanto, a estágios evolutivos distintos do desenvolvimento de perícias no domínio do jogo.

[6] *Debugged*.

[7] *Student simulation testbed*. Em outras áreas, como a Engenharia de Software, o termo "ambiente de teste" é uma tradução mais apropriada.

Os ambientes tradicionais da IIAC, baseados no especialista, suportam apenas a simulação de aprendizes constituídos em termos de um subconjunto de perícias do domínio representado. O grafo genético amplia tanto as possibilidades de simulação quanto a compreensão que se tem da tutoria, pois permite a geração de aprendizes estabelecidos em termos de especializações, desvios e simplificações das perícias que perfazem o domínio. Assim se percebe, em comparação, que um aprendiz em estágio anterior a um segundo carece desenvolver perícias mais avançadas diante das relações evolucionárias prescritas.

Adicionalmente, Goldstein comenta que um ambiente de simulação também atende a outros objetivos. A simulação de aprendizes pode ser usada para testar a modelagem e as estratégias pedagógicas em ambientes de aprendizagem [23, 113]. Outra possibilidade reside em auxiliar na modelagem de aprendizes reais, enriquecendo a perspectiva de tutores humanos que observem o andamento da simulação.

Por fim, Goldstein enfatiza que as relações evolucionárias apresentadas podem se encontrar subespecificadas ou incompletas. Há inúmeras formas de analogias, generalizações e correções. Também existem outros processos para aquisição de conhecimento, como a indução a partir de exemplos anteriores e a dedução baseada em regras previamente conhecidas. Considerando isso, a próxima seção sugere extensões para o grafo genético.

4.1.5 Extensões do Grafo Genético

A Seção 4.1.2.2 definiu um conjunto de relações genéticas entre regras (perícias) individuais. Agora, o próprio Goldstein [51] estende o grafo genético para que incorpore relações entre grupos de regras, como também entre os fatos declarativos que explicam e justificam o uso de regras. Tratam-se de ferramentas que ampliam o potencial de representação do grafo em casos e domínios específicos. O modelo evolutivo passa a ser denominado **grafo genético estendido**.

4.1.5.1 Agrupamento em Ilhas

Na proposição original do grafo genético exposta, não há representação explícita dos conjuntos de regras estreitamente relacionadas e, em geral, aprendidas em grupos. Tal

limitação é excedida, agrupando-se regras dentro de ilhas. Com isso, um critério natural para a formação de ilhas consiste em agrupar regras que têm o mesmo objetivo.

Ilhas permitem que a tutoria ocorra em termos de um conceito geral para um grupo de regras. Consequentemente, possibilita que o aprendiz seja modelado quanto à aquisição da base conceitual que permeia determinado conjunto de regras.

Todas as relações genéticas prescritas entre regras também podem ser generalizadas ao escopo de ilhas. Por exemplo, assim como pode existir a analogia entre duas regras, também se faz possível a analogia entre ilhas de regras.

A ampliação de escopo se corrobora porquanto se apropriar das regras contidas em uma ilha é uma tarefa local, ou seja, um episódio natural de aprendizado. Todas as regras de uma ilha específica seguem um mesmo conceito. Entretanto, mover-se até uma ilha vizinha requer uma nova base conceitual.

A perspectiva dessas tarefas implícitas, da aquisição de regras ou de ilhas inteiras, podem ser resumidas, respectivamente, aos planos micro e macro. Nesse âmbito, as ilhas representam perícias cujas componentes são as subperícias.

Embora não tenha sido proposto por Goldstein, aponta-se que o conceito de ilhas pode ser uma definição recursiva. Assim sendo, a classificação partonômica de perícias em mais de um nível é permitida. Garante-se, então, uma granularidade mais apurada que diversos domínios de conhecimento podem exigir.

Outra possibilidade, para citar, seria que as ilhas admitissem interseção de regras. Abre-se, desse modo, espaço para uma classificação partonômica mais detalhada. Em ambos os casos, tratam-se de conceitos simples que demandam peculiaridades de implementação para que sejam trazidos à prática.

Como exemplo, a Figura 4.3 mostra a delimitação das ilhas chamadas *Expressões* e *Estruturas de Controle*. Nesta segunda, são definidas as ilhas *Estruturas Condicionais* e *Estruturas de Repetição*. Merece atenção que a perícia de *projetar (conjuntos de) expressões condicionais coesas e concisas para estruturas de controle* integra igualmente as ilhas *Expressões* e *Estruturas de Controle*. Foi, portanto, alocada na interseção de ambas.

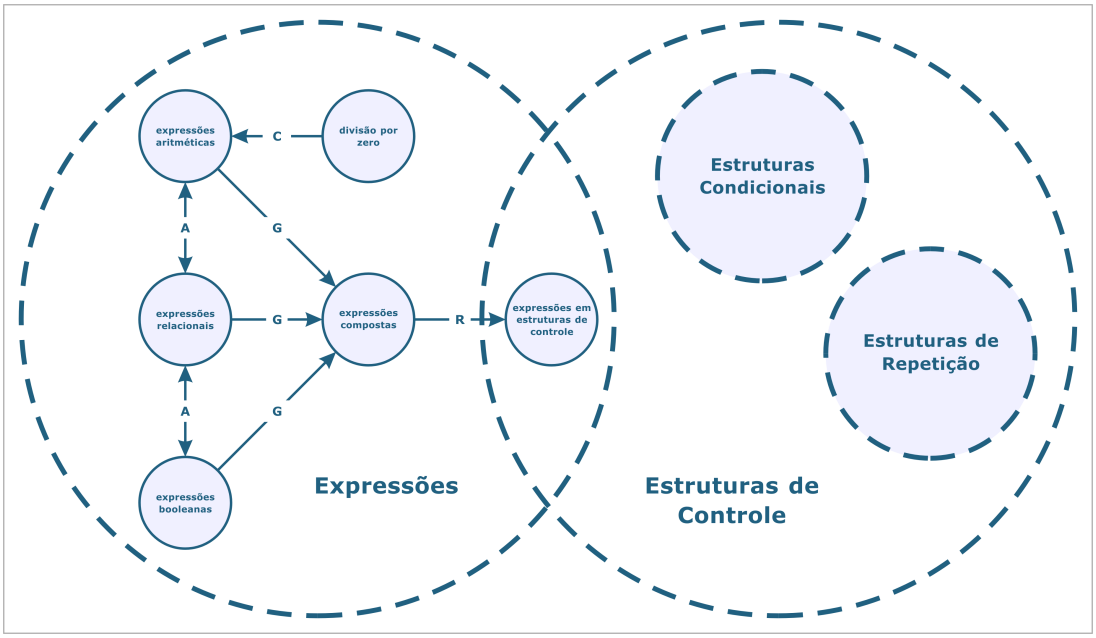


Figura 4.3: Exemplo de agrupamento em ilhas

4.1.5.2 Representação de Conhecimento Declarativo

As regras por si não descrevem o conhecimento declarativo (princípios, no contexto desta tese) que as explicam e justificam. Contudo, tal extensão do grafo genético é facilmente alcançável. As relações evolucionárias que interligam regras procedimentais podem desempenhar o mesmo papel em sentenças declarativas. Logo, uma sentença pode ser a generalização de outra, uma analogia ou ainda um refinamento.

A partir disso, o tutor passa a promover tanto o conhecimento declarativo quanto o procedimental. A alternância entre ambos, diante das dificuldades do aprendiz, potencializa o processo de ensino.

Como possibilidade de representação, pode-se admitir que as sentenças declarativas sejam expressas por uma figura geométrica distinta das regras. Tendo em vista que as regras são significadas por elipses, uma sentença declarativa poderia ser indicada por um retângulo. Mesmo a mera alternância de cores, ao invés da forma geométrica, já poderia cumprir com a finalidade de distinção em ambas.

Em critério de exemplo, a Figura 4.4 agrupa em uma ilha independente o conhecimento declarativo que classifica e define os operadores aritméticos, relacionais e booleanos supor-

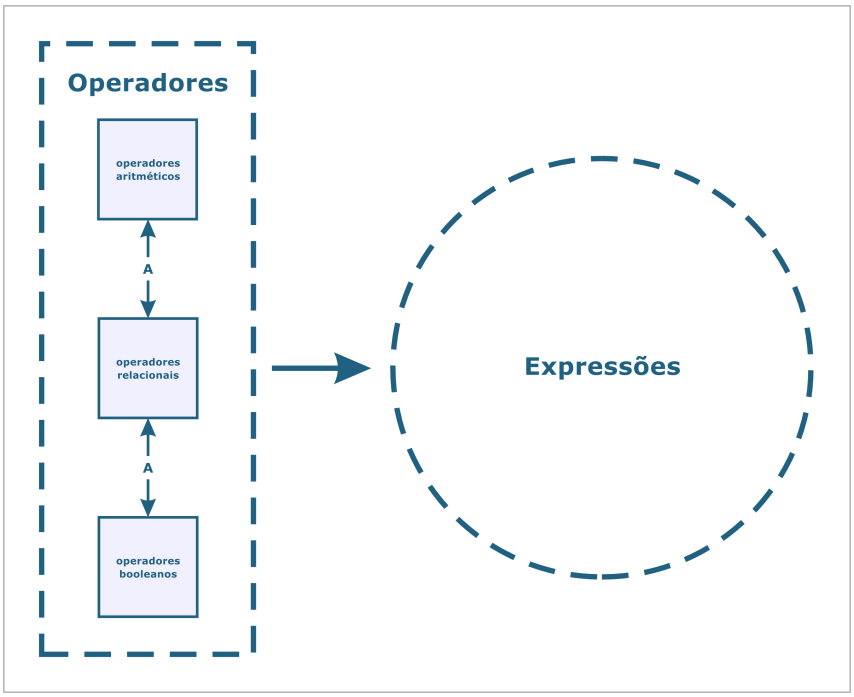


Figura 4.4: Exemplo da representação de conhecimento declarativo

tados. Assim como ocorre com as perícias, foram estabelecidas conexões evolucionárias entre as sentenças declarativas dos operadores. Todo esse conhecimento é vinculado à ilha de perícias que se remete (no caso, *Expressões*).

4.1.5.3 Representação da Ordem

Nem todo conhecimento sobre regras se presta para descrever relações evolutivas. Uma vez que conjuntos de regras constituem roteiros (ou componentes) para resolução de problemas, espera-se que o conhecimento sobre a ordem de aplicação (e de aprendizado) das regras seja representado.

Embora aparente, não há dificuldade em estender o grafo genético nesse sentido. Basta que relações apropriadas sejam definidas. Para isso, diz-se que R é **pré-requisito** de R', se o aprendizado anterior de R é necessário à compreensão e ao desenvolvimento de R'. A relação inversa é denominada **pós-requisito**.

Evidencia-se que as relações de pré e de pós-requisito fornecem apenas as restrições básicas de ordem. Entretanto, o grafo genético estendido passa a incorporar conhecimento explícito de controle a fim de planejar e sequencializar ações pedagógicas. E, assim como

qualquer adição de conhecimento útil, tende a beneficiar o processo de ensino.

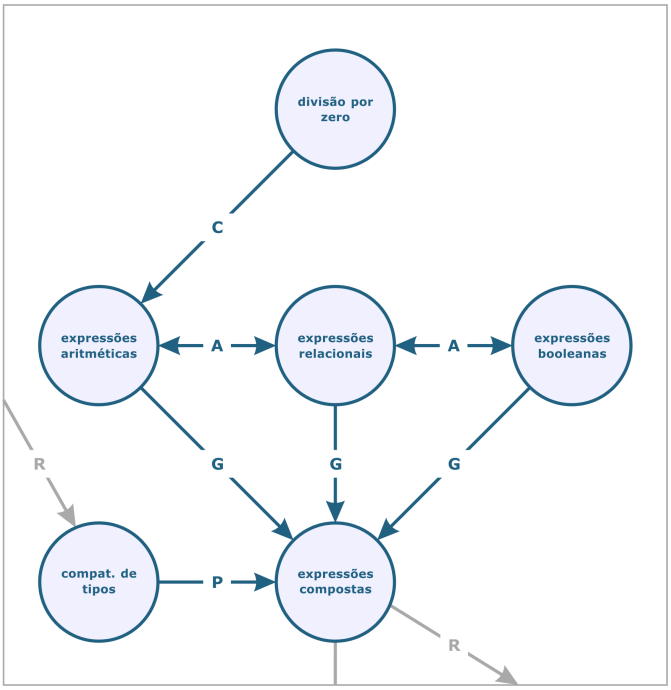


Figura 4.5: Exemplo da relação de pré-requisito

A Figura 4.5 evolui o exemplo anterior (ilustrado pela Figura 4.2) para que admita a relação de pré-requisito. Ocorre que cautela com a compatibilidade de tipos é pré-requisitada para a perícia de projetar (conjuntos de) expressões condicionais coesas e concisas para estruturas de controle. Expressado isso, a primeira perícia deve ser provida em ordem anterior na sequência de ensino.

4.1.5.4 Comparação com o Grafo Genético Básico

Em conclusão, destaca-se que o grafo genético estendido não se difere, em natureza, do grafo genético básico (ou original), pois continua sendo um grafo composto por vértices de conhecimento interconectados por várias relações genéticas.

Houve aumento das possibilidades de representação em consequência:

1. Da estrutura hierárquica das regras agrupadas (ilhas);

2. Da extensão dos vértices individuas, que significavam apenas regras, para representarem tanto regras quanto fatos;

3. Da ampliação do conjunto de relações a fim de incluir também o conhecimento de controle (pré e pós-requisito).

Implicitamente, a estrutura do grafo genético representa a perspectiva de aprendizado em que novas regras são adquiridas a partir daquelas previamente conhecidas. A evolução é guiada pelos processos correspondentes às conexões individuais estabelecidas.

Convém destacar que o grafo não descreve um caminho evolutivo único. Levando isso em conta, um aprendiz, por exemplo, pode rapidamente adquirir uma generalização, enquanto outro pode construir várias especializações antes de consolidá-la. Ademais, um terceiro aprendiz pode sequer conseguir se apropriar da mesma generalização sem o devido auxílio tutorial.

Isso posto, considera-se que o tutor deva encorajar aquilo que Goldstein determina como construção idiossincrática de novos conhecimentos. Ou seja, a maneira própria e pessoal com que cada aprendiz se desenvolve. Para tanto, deve-se fornecer assistência adequada ao estágio atual de conhecimento do aprendiz (posição corrente no grafo genético), bem como ao respectivo estilo de aprendizado (preferência por tipos de relações genéticas específicas na apropriação de conhecimento).

As próximas seções discutem meios de orientar a tutoria para que se aproxime do comportamento de suporte recém-descrito.

4.1.6 Base para a Tutoria

Essencialmente, existem duas maneiras a partir das quais o grafo genético pode guiar o Modelo Pedagógico de um ambiente de inteligente. Primeiro, sugerindo qual perícia deve ser tomada como tópico de estudo pelo aprendiz, dentre aquelas pertencentes à fronteira da posição atual no grafo. A segunda maneira consiste em explicar uma determinada perícia, então escolhida como tópico de estudo, por mais de uma via, através de cada relação evolutiva com outras perícias previamente adquiridas. Ambas são detalhadas na sequência.

4.1.6.1 Sugestão do Tópico Tutorado

Na IAC baseada em *scripts*, os tópicos são introduzidos em uma ordem predefinida. O aprendiz procede à próxima pergunta, fornecida pelo autor, somente após ter respondido corretamente a pergunta atual. Existe a vantagem de o autor controlar a oferta de material mediante a própria compreensão do domínio. Entretanto, tamanha rigidez na ordem de apresentação é também em uma desvantagem na perspectiva do aprendiz.

Em resposta, a IAC baseada no especialista apresenta uma menor rigidez, porquanto permite que o aprendiz explore um problema à própria maneira e que as respostas sejam analisadas de acordo com um conjunto de perícias subjacentes. A tutoria se estabelece em torno de prover suporte nas situações em que o aprendiz escolhe uma opção aquém da aceitável. Contudo, tutores baseados no especialista não predizem quando a introdução de uma perícia é prematura no contexto daquelas anteriormente adquiridas pelo aprendiz.

O grafo genético transpõe parte dessa limitação. Goldstein aceita a heurística educacional de que se facilita o aprendizado pela capacidade de explicar uma nova perícia através de outras anteriormente adquiridas. Logo, as perícias com maior prioridade de serem ensinadas são aquelas localizadas na fronteira do subgrafo correspondente ao Modelo do Aprendiz.

Portanto, o grafo genético não resolve o problema de escolher o tópico a ser ensinado. Ainda assim, oferece uma fronteira de perícias tida como um subconjunto de possíveis conteúdos. Cabe ao tutor eleger um tópico dentre as perícias do subconjunto ou mesmo rejeitá-lo inteiro. Logo, concerne ao tutor a aplicação de heurísticas de ensino (e.g. variar os tópicos escolhidos), como também de estratégias específicas relacionadas ao aprendiz (e.g. do quanto a seleção de tópicos pode variar). Nesse sentido, o grafo genético contribui apenas para disponibilizar, às heurísticas de ensino, as relações epistemológicas entre as perícias do conteúdo programático. Consequentemente, não arbitra sobre decisões de caráter pedagógico.

4.1.6.2 Suporte a Múltiplas Explicações

Tendo selecionado o tópico de estudo, conforme seção anterior, uma importante técnica de ensino consiste na habilidade de explicá-lo por mais de uma maneira. Os ambientes de IAC baseados em *script* conquistaram isso através das linguagens de autoria, propiciando que boas explicações fossem escritas por tutores humanos. Depois, os ambientes baseados no especialista, por suprimirem os *scripts*, perderam a mesma capacidade. Em contrapartida, conseguem responder a um grande número de situações, ainda que por meio de poucos formatos de explicação gerados dinamicamente.

Os ambientes de IIAC, após introduzidos, ao usarem grafos genéticos, passaram a manter a habilidade de resposta a várias situações, mas também somaram a capacidade de explicar uma determinada perícia de diversas maneiras. Consegue-se isso porque o grafo genético proporciona que uma nova regra seja explicada através das próprias conexões. Ou seja, para cada tipo de relacionamento, pode-se adotar uma estratégia diferente de explicação.

A Figura 4.6, por exemplo, mostra múltiplas explicações para a perícia de avaliar e construir expressões booleanas. Cada explicação é suportada pelas diferentes conexões que a perícia mantém.

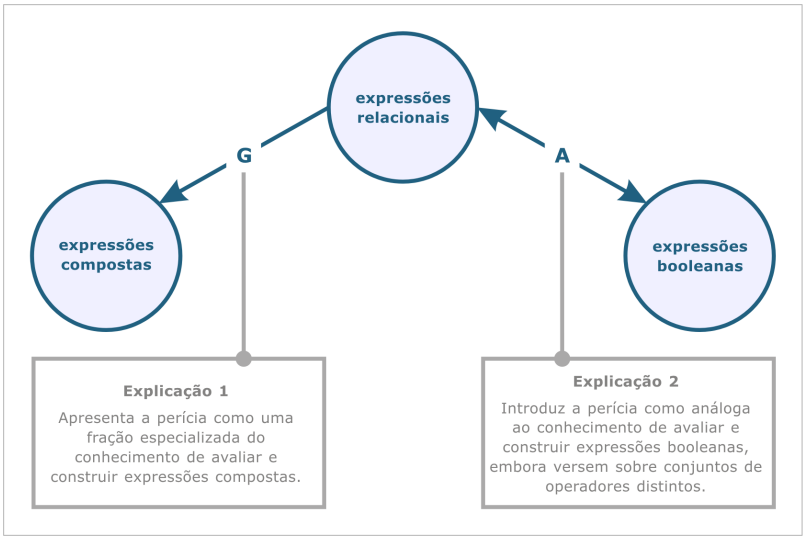


Figura 4.6: Variação e alternância entre explicações

Entretanto, assim como na seleção do tópico a ser ensinado, a escolha da melhor ex-

plicação para uma regra não é determinada pelo grafo genético. A opção deve ser sensível ao aprendiz e ao respectivo estágio naquele momento. Por conseguinte, também depende da aplicação de heurísticas de ensino (e.g. variar as explicações), além de estratégias específicas relacionadas às particularidades do aprendiz (e.g. evitar categorias de explicações anteriormente mal sucedidas). Apesar disso, ainda há benefício visto que o grafo genético aumenta a quantidade de opções disponíveis para que a tutoria possa proceder.

4.1.6.3 Representação Estendida do Conteúdo Programático

A escala de estratégias pedagógicas é incrementada ainda mais com emprego do grafo genético estendido, anteriormente descrito na Seção 4.1.5. O agrupamento de regras por meio de ilhas, por exemplo, e as consequentes relações genéticas trazidas ao escopo do conjunto conseguem abranger um grupo todo de regras em uma única explicação.

Analogamente, o conhecimento declarativo proporciona outra oportunidade para a geração de suporte a conjuntos inteiros de regras. Um relacionamento bastante comum, expressado pela Figura 4.4, consiste na conexão de um conjunto de fatos com a ilha de regras onde se aplica. Isto é, ao discutir a fundamentação teórica, o aprendiz pode deduzir sozinho as regras relacionadas. Tem-se, em consequência, uma estratégia pedagógica potencialmente útil.

4.1.6.4 Discussão sobre a Tutoria

A tutoria consiste em uma tarefa complexa e o ambiente deve decidir:

1. Se deve intervir;
2. O tópico a ser discutido; e
3. O quanto dizer sobre o referido tópico.

Convém ressaltar que apenas o grafo genético não decide qualquer uma dessas três questões. Presta-se, ao invés, para restringir o conjunto de tópicos pela definição de uma fronteira e, depois disso, para ampliar a variedade de explicações disponíveis ao tópico escolhido.

Ferramentas adicionais da Inteligência Artificial podem ser empregadas a fim de auxiliar na resolução das questões citadas.

4.1.7 Base para a Modelagem

A fim de oferecer orientação tutorial adequada, deve-se modelar o aprendiz acuradamente. O grafo genético facilita o processo de modelagem basicamente a partir de três contribuições:

1. Os vértices do grafo fornecem uma estrutura mais refinada de modelagem do que os conjuntos e subconjuntos de perícias dos primeiros ambientes de IIAC;

2. A organização do grafo proporciona uma métrica sobre quais perícias esperar que o aprendiz adquira na sequência; e

3. As conexões do grafo servem como uma estrutura complementar para que se modele o comportamento da aprendizagem de um indivíduo.

4.1.7.1 Modelo de Sobreposição (*Overlay*)

A modelagem do aprendiz, nos ambientes de IAC baseados em *scripts*, ocorre por meio do armazenamento de estatísticas sobre a exatidão das respostas fornecidas. A validade da modelagem é severamente limitada pela capacidade de tais ambientes julgarem a correção, pois geralmente se baseiam apenas em uma lista de respostas predeterminadas.

Depois, a IAC baseada no especialista transpôs a limitação mencionada, determinando a modelagem baseada na hipótese de que o estágio atual do desenvolvimento de perícias do aprendiz consiste em um subconjunto do conhecimento de um experto. Em consequência, as supostas perícias de um experto são formalizadas pelo conhecimento (ou modelo) do domínio.

Nesse contexto, o termo **sobreposição**[8] enfatiza que a estrutura correspondente ao Modelo do Aprendiz é derivada do conhecimento subjacente do domínio [22, 52]. Mais especificamente, tendo o conhecimento do domínio como um grafo genético, modela-se o

[8]Também utilizado o termo *overlay*.

aprendiz ante a perspectiva de um subgrafo, conforme a Figura 4.7. Portanto, considera-se que um aprendiz domina um subconjunto das perícias que perfazem o estereótipo de um experto.

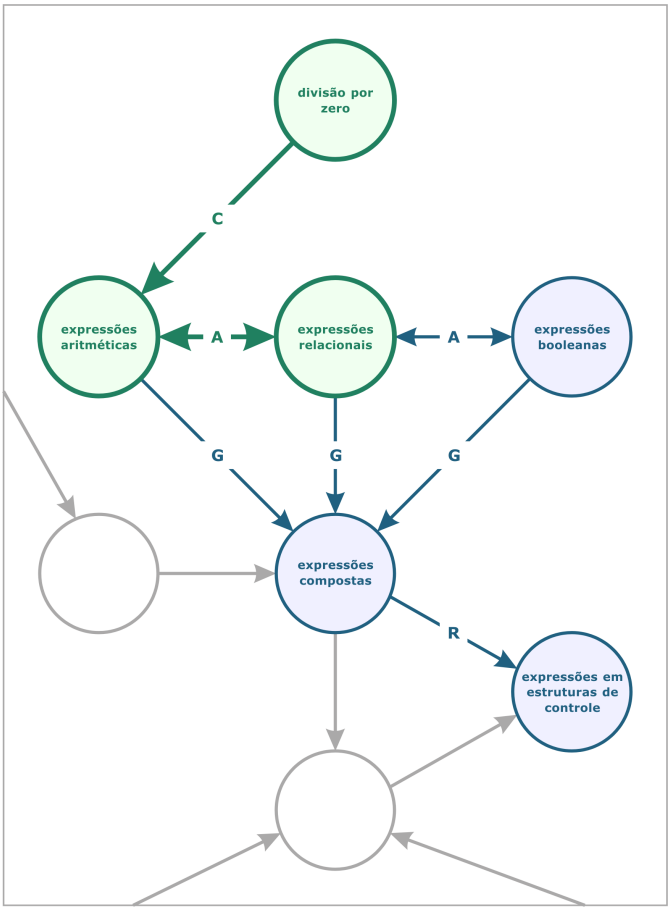


Figura 4.7: Modelo do Aprendiz em aspecto de sobreposição

Diante das possibilidades trazidas pelos grafos genéticos, a modelagem dos ambientes baseados no especialista possuem outra limitação. Há domínios em que o aprendiz não emprega simplesmente um subconjunto das perícias de um experto. Ao invés disso, faz uso de simplificações, desvios, ou mesmo de outros predecessores evolucionários das perícias objetivadas.

Existem, todavia, domínios suficientemente restritos em que se modela o aprendiz apenas como um subconjunto das perícias de um experto. Ou ainda, em alguns casos, pode-se assumir que as perícias sejam compartidas em subperícias e, com isso, tornar mais apropriada a modelagem em termos de presença ou ausência dessas capacidades.

Em situações assim, os ambientes fundamentados em grafos genéticos são reduzidos à IAC baseada no especialista, se forem compostos meramente por conjuntos de perícias.

Por fim, destaca-se que, com o uso dos grafos genéticos, o Modelo do Aprendiz continua sendo construído como uma sobreposição. Entretanto, a modelagem evolutiva contribui para que a sobreposição não se restrinja ao conjunto de perícias. Pode ocorrer em termos de todos os componentes do grafo genético, ou seja, subperícias, simplificações, desvios, ou quaisquer outros.

4.1.7.2 Modelagem do Aprendiz

Considerando a modelagem como a atribuição (ou conquista) de regiões do grafo genético ao aprendiz, convém examinar de que forma isso se infere. Goldstein, primeiramente, descreve os métodos básicos empregados por ambientes de IAC baseados no especialista. Logo após, propõe melhorias subsidiadas em métricas de aprendizagem implícitas na distância entre perícias de um grafo genético.

Os ambientes de IAC baseados no especialista constroem o Modelo do Aprendiz pelo nível das respostas fornecidas. A hipótese consiste em que o aprendiz não possui determinada perícia se a resposta para uma dada situação for pior do que aquela alcançada por um experto com deduções fundamentadas na mesma perícia. Em contrapartida, caso o aprendiz e o experto tenham respostas suficientemente próximas, assume-se que o primeiro está familiarizado com as perícias empregadas para determinar a resposta correta.

Na prática, o processo de modelagem descrito torna-se mais sutil. Para cada situação analisada, os dados brutos são registrados como incrementos para duas variáveis associadas a cada perícia, sendo chamadas de **apropriada** e **usada**. A primeira registra quantas vezes a perícia foi empregada corretamente. Em contraste, a outra armazena quantas vezes o aprendiz acreditou que a perícia foi empregada corretamente.

A proporção das respectivas variáveis constitui a **frequência** de uso relacionada a cada perícia. O Módulo Pedagógico interpreta que o aprendiz tem domínio sobre uma perícia quando a proporção excede um limite estabelecido. A complexidade de manter uma modelagem com tais características foi anteriormente discutida em [22].

O método de comparar o desempenho do aprendiz com o que se espera de um experto continua sendo essencial a um ambiente inteligente que utilize grafo genético. Entretanto, Goldstein propôs a melhoria de predefinir um conjunto de aprendizes simulados, de níveis gradativos, correspondentes a platôs intermediários de habilidades no grafo genético.

Diante do contexto de Programação de Computadores abordado por esta tese, níveis exequíveis poderiam ser:

1. Conceitos iniciais. Análise e abstração de informações na resolução de problemas;
2. Composição e efetivação de algoritmos em linguagem natural;
3. Declarações de variáveis e constantes. Instruções de entrada, saída e atribuição;
4. Expressões aritméticas, relacionais e booleanas;
5. Estruturas condicionais;
6. Estruturas de repetição.

Cada um dos perfis de referência examina a retroalimentação do aprendiz e sugere quais perícias aparentam terem sido utilizadas. Tais hipóteses são anexadas aos vértices do grafo. A crença geral de que o aprendiz possui uma perícia específica provém do somatório sobre as hipóteses de cada perfil individual de referência.

No caso em que o aprendiz possua perícias de regiões esparsas do grafo com igual probabilidade, ocorre de todos os perfis terem o mesmo peso na formulação da hipótese geral. Todavia, o grafo genético incorpora a teoria de que o conhecimento evolui por meio de relações genéticas, ou seja, da simplificação à elaboração, do desvio à correção, da abstração ao refinamento e da especialização à generalização. Por isso, as hipóteses geradas por perfis cada vez mais distantes da situação atual do aprendiz passam a ter, progressivamente, menor peso.

Como resultado, tem-se um conservadorismo desejável no processo de modelagem. Trata-se de algo razoável, pois consente com o senso comum de que uma melhoria profunda no desempenho de um aprendiz deve-se, mais provavelmente, à sorte do que a um salto descontínuo de habilidade. Igualmente, uma deficiência profunda, em um tópico

aprendido, tem mais chances de provir da falta de cuidado do que da necessidade de retroceder a um estágio anterior de conhecimento.

Mesmo assim, tal conservadorismo não previne o Módulo Pedagógico de acreditar em saltos descontínuos no conhecimento do aprendiz. Os perfis de referência distantes da condição atual do aprendiz ainda recebem peso na ponderação. Com isso, aceita-se eventualmente uma mudança radical no conhecimento do aprendiz.

Salienta-se que, desprovido do conservadorismo, o Módulo Pedagógico perde a capacidade de observar um palpite de sorte ou um descuido ocasional. Assim, a métrica de aprendizagem definida pelos grafos genéticos proporcionam parte da uma estabilidade que faltava nos ambientes de IAC baseados no especialista.

4.1.7.3 Sobreposição das Conexões

Existe ainda o benefício inerente de que as relações do grafo genético fornecem a estrutura para um modelo de aprendizagem. Anteriormente se discutiu sobre a habilidade de explicar uma mesma perícia de várias maneiras, por meio das respectivas relações genéticas. Agora, remetendo-se a um modelo de aprendizagem, é possível observar o efeito de cada categoria de explicação. Pode-se então determinar se o aprendiz empregou a perícia ensinada em um problema subsequente.

Diante disso, se uma estratégia de explicação conduz consistentemente ao desenvolvimento de perícias, pode-se considerá-la efetiva para um aprendiz em particular. Caso contrário, deduz-se que aquela estratégia não se mostra proveitosa ao mesmo aprendiz.

Por conseguinte, uma sobreposição da aprendizagem[9] pode ser gerada sobre o conjunto de relações genéticas suportado. Mantem-se registro sobre a efetividade da estratégia de explicação associada a cada tipo de relação genética.

Trata-se de um modelo simples que orienta a escolha de estratégias de explicação personalizadas para cada aprendiz. A opção é feita dentre aquelas que se provaram bem sucedidas em situações passadas.

[9] *Learning overlay.*

4.1.7.4 Discussão sobre a Modelagem

Goldstein ressalta a complexidade de modelar o conhecimento do aprendiz, bem como as propriedades da aprendizagem. Aponta como, certamente, a atividade mais difícil de se desempenhar em um ambiente inteligente de ensino.

Considerando isso, resume-se que o grafo genético apenas proporciona um arcabouço para tais modelagens: o conhecimento do aprendiz é descrito sobre os vértices do grafo; o comportamento do aprendizado em termos das relações; o progresso, por sua vez, na forma de caminhos no grafo.

Embora o grafo genético proporcione uma fundamentação sólida à modelagem, não resolve a questão por completo. A critério de exemplo, nenhuma resposta dada pelo aprendiz pode ser tomada como evidência segura. O aprendiz pode ter entendido mal o enunciado, perdido interesse de formular uma resposta, ou ainda mudado bruscamente de objetivo.

Permanece ainda, obviamente, uma severa desvantagem entre ambientes computacionais e tutores humanos. Os primeiros são inaptos na observação de fatores subjetivos, como interpretar expressões faciais, compreender a linguagem natural e mesmo saber se o aprendiz continua pensando ou se interrompeu a sessão de ensino. Mesmo para profissionais experientes, a modelagem do aprendiz é tida como uma das atividades mais difíceis de serem realizadas na área.

De modo oportuno, Goldstein introduz o problema da largura de banda, em que, livremente de como se representa a aquisição de conhecimento, a modelagem também depende de se observar e capturar informações ao longo do processo decorrido. Diante disso, métodos que aperfeiçoem a observação do aprendiz, ou seja, ampliem a largura de banda, passam a ser importantes suplementos à modelagem através de grafos genéticos. Portanto, a contribuição dos grafos genéticos reside em prover uma estrutura de dados que seja o destino de evidências trazidas por métodos diversos.

Outra limitação advém do pensamento de que só se pode modelar aquilo que se compreende. Assim sendo, não é fácil que algum tutor ou experto humano consiga representar

o conteúdo programático de maneira tão detalhada e explícita. No entanto, espera-se que os profissionais envolvidos aprimorem tal entendimento em resposta à observação do comportamento do aprendiz, entre outros fatores.

Remetendo-se a isso, faz-se necessário que os ambientes busquem incorporar determinada capacidade de aprendizado (de máquina) também na tutoria inteligente. A próxima seção discute uma formulação preliminar da teoria de aprendizagem requerida para tanto.

4.1.8 Base para a Aprendizagem

Conforme Goldstein explica, implicitamente ao grafo genético, existe uma teoria de aprendizagem. A corrente seção explora essa teoria e considera respectivas implicações no projeto de tutores inteligentes.

Primeiramente, Goldstein propõe uma intricada simulação dos comportamentos do aprendiz, como um agente, como apresentado pela Figura 4.8. Os processos do aprendiz são internamente divididos entre dois outros papéis, sendo um especialista na resolução de problemas e outro na aprendizagem. O grafo genético serve a ambos como se fosse a estrutura básica de memória do aprendiz para o conhecimento declarativo e procedimental do domínio.

O agente responsável pela resolução de problemas aplica os processos definidos pela fronteira do grafo genético ao enunciado sugerido. O outro, referente à aprendizagem, estende o grafo genético em resposta às novas tarefas, orientação tutorial e dificuldades apresentadas nos processos correntes.

Goldstein usa, ao invés de "agente", o termo *homunculus*[10], proveniente do latim que equivale ao "homúnculo" da língua portuguesa. Com isso, desejava enfatizar que os componentes de aprendizagem e de resolução de problemas foram imaginados como máquinas com o exato mesmo alcance. A diferença consiste nos processos, ou programas, que são particulares de cada qual.

O agente de aprendizagem consiste em um conjunto de estratégias correspondentes às várias relações do grafo, ocupando-se da apropriação de novas perícias, bem como

[10]No plural, *homunculi*.

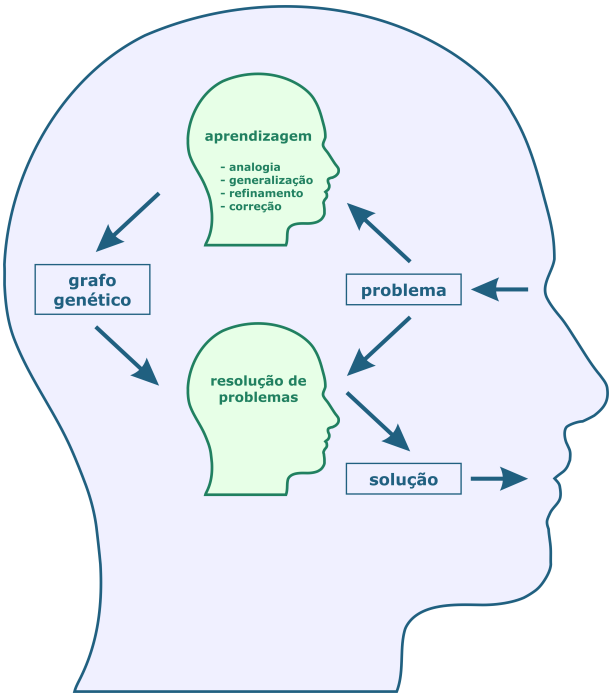


Figura 4.8: Agente para a simulação dos comportamentos do aprendiz

registrando as devidas conexões com os predecessores evolutivos. As relações são rotuladas com a estratégia de aprendizagem responsável pela aquisição e os registros são úteis como parâmetros para aprimorar o Módulo Pedagógico.

Cita-se também que existe uma simplificação ao se supor que todas as perícias desenvolvidas continuam disponíveis ao aprendiz, pois ocorrem também processos de esquecimento e desuso. Trata-se de um recurso útil para eliminar o conhecimento desatualizado que desviaria o percurso de aprendizagem. Portanto, perfaz-se uma consideração importante para uma teoria geral de ensino-aprendizagem, mas tal detalhamento excede a proposta dos estudos de Goldstein.

Assim sendo, o grafo genético apenas oferece a estrutura para uma teoria de aprendizagem, sugerindo que os processos de aprendizagem consistem de procedimentos que geram relações diversas, sem pormenorizar como isso ocorre. O grafo também não fornece detalhes sobre quais critérios são usados para fazer uma analogia, reconhecer desvios, induzir generalizações, ou construir refinamentos conceituais. Pesquisas que abordaram detalhes adicionais sobre a analogia podem ser encontradas em [115, 40, 80, 17]. A depuração foi discutida por [50, 100].

Entretanto, o grafo genético permite que se concentrem esforços em algumas questões envolvidas na interação dos processos de ensino e de aprendizagem. Goldstein transcorre sobre quatro dessas questões, a saber:

1. Consideração do aprendiz como um agente ativo;
2. Modelagem de processos metacognitivos;
3. Definição de uma métrica de crença; e
4. Reconhecimento de métricas para a complexidade de aprendizado.

4.1.8.1 Consideração do Aprendiz como um Agente Ativo

A simulação recém apresentada pela Figura 4.8 enfatiza a visão do aprendiz como um agente ativo, envolvido no processo construtivo da geração de novos conhecimentos. A partir dessa perspectiva, o objetivo do tutor é justamente motivar a sustentação do processo pelo aprendiz.

Assim, tem-se a intervenção e o suporte a explicações detalhadas como apenas algumas das finalidades providas pela tutoria. No outro extremo, reside o "ensinar sem falar"[11] que consiste em, ao invés de fornecer informações cada vez mais detalhadas ao aprendiz, alterar o domínio do problema no intuito de facilitar o processo de aprendizagem.

Entre os dois extremos existe também toda uma escala intermediária de possíveis intervenções. O tutor poderia sugerir, por exemplo, sobre uma nova perícia, passível de ser aplicada à situação corrente, que fosse análoga a outras perícias já adquiridas. Todavia, sem especificar a nova perícia ou indicar a analogia.

Diante disso, os tutores evoluem ao propiciar orientação ao longo dos extremos apresentados. É conveniente que a natureza e o alcance da intervenção sejam relacionados ao estágio atual do aprendiz.

[11] Denominado por Goldstein como *tutoring without talking*.

4.1.8.2 Modelagem de Processos Metacognitivos

O agente que representa o aprendiz foi subdividido em dois papéis, sendo um especialista na resolução de problemas e outro na aprendizagem. Supõe-se, em adição, que as habilidades de estudo[12], ou seja, metacognitivas [74], do segundo especialista também poderiam ser representadas em um grafo genético. Considerando que exista uma coleção de regras que definem os próprios processos de analogia, generalização, depuração e refinamento correspondentes às relações genéticas, então explicar esse suposto grafo se torna uma importante meta tanto da Inteligência Artificial quanto da Psicologia Cognitiva.

Goldstein discute que uma hipótese concorrente enuncia que os processos de aprendizagem não se relacionam e sequer possuem simplificações das quais evoluem. São, portanto, uma coleção desestruturada de heurísticas, adquiridas de maneira isolada. Embora essa segunda hipótese seja improvável diante das pesquisas de Goldstein, pode também não ser simples chegar a um grafo genético que modele o aprendizado em si.

Nesse sentido, seria igualmente necessário coordenar a aquisição das perícias de aprendizagem. E, considerando-as representadas em um grafo genético, tal como ocorre com o conhecimento do domínio, o especialista na aprendizagem seria potencialmente apto a operar sobre a modelagem de si mesmo. Logo, haveria recursão e, antes ainda, um conjunto de estratégias inatas que iniciariam o processo. Como resultado específico, uma mesma teoria de aprendizagem seria suficiente tanto para a aquisição de conhecimento do domínio quanto para a melhoria recursiva das capacidades de ensino e aprendizado do ambiente.

Outro importante avanço consiste na possibilidade de tutorar as habilidades de estudo, isto é, os próprios processos de aprendizagem. Com isso, as explicações poderiam ser orientadas à metacognição, ou seja, ao conhecimento que o indivíduo tem, e constrói, sobre o próprio conhecimento [74]. Demarca-se uma importante direção para pesquisas futuras, uma vez que ensinar perícias de um domínio específico é menos importante do que ensinar os processos que conduzem à evolução desse conhecimento.

[12]Não se trata de conhecimento do domínio.

4.1.8.3 Definição de uma Métrica de Crença

Ainda diante da perspectiva da simulação do aprendiz pelos dois agentes especialistas, conjetura-se sobre quando uma nova perícia, adicionada ao grafo genético, torna-se inerente à prática do especialista em resolução de problemas. Considerar que o especialista sempre atua na fronteira do grafo genético pode ser uma suposição simplista, enquanto a eminência de uma nova perícia pode representar um equívoco ou ainda estar em estágio bastante incompleto. Assim, alguma inércia torna-se desejável, mesmo no dinamismo do ambiente, a fim de evitar oscilações abruptas e modificações prematuras.

Pode-se também observar que um aprendiz nem sempre aplica uma perícia recém-abordada. Enquanto o aprendiz não tiver condições de repetir a explicação, ou mesmo descrever as implicações do novo conhecimento, acredita-se que não esteja apto a utilizá-lo na resolução de problemas. Os professores desenvolvem a heurística de reconhecer essa situação e, em resposta, fornecem exemplos e explicações adicionais.

Uma representação formal desse conservadorismo pode ser adicionada ao modelo de aprendizagem através da introdução de uma métrica de crença. É possível restringir o emprego, na resolução de problemas, de uma perícia pertencente à fronteira da sobreposição do aprendiz. A restrição é mantida até que a crença naquela perícia ultrapasse um limite predefinido. Determina-se, nesse caso, a **crença** como uma função do número, da natureza e da recência, tanto de explicações quanto de exemplos que foram providos.

Em se tratando do grafo genético, diz-se que uma nova perícia não pode ser empregada até que as respectivas conexões com o grafo sejam suficientemente fortes. A crença em uma nova perícia é definida em termos do número e da natureza das relações que a vinculam ao grafo, excedendo determinado limite.

A citada métrica também pode melhorar a expectativa que o tutor mantém sobre o uso de uma nova perícia pelo aprendiz. No momento em que a crença em uma perícia estiver abaixo do limite, explicações e exemplos suplementares serão necessários, assim como o aprendiz pode estar apto a descrever aquela perícia, mas provavelmente não a empregá-la corretamente. O limite, inclusive, pode ser ajustado com base no desempenho

do aprendiz.

Mediante o refinamento da modelagem descrito, pode-se realizar uma análise mais minuciosa dos critérios de crença em aprendizes. Para alguns casos, muitos exemplos de poucas conexões genéticas podem gerar uma crença mais consistente do que poucos exemplos de muitas conexões. Na análise do desempenho de cada aprendiz, com relação aos limites adotados, pode-se explorar a alternância entre diversidade, repetição e recência de materiais. Com a utilização do que foi citado, é possível obter quais são os limites que conduzem a uma taxa de aprendizado razoável, como também aqueles que se mostram muito instáveis ou conservadores.

4.1.8.4 Reconhecimento de Métricas para a Complexidade de Aprendizagem

Tendo o grafo genético como um registro do processo de aprendizagem, é possível estabelecer uma correlação entre conceitos da Teoria dos Grafos e a complexidade de aprendizagem. A partir disso, são providos meios de orientar o tutor sobre regiões do conteúdo programático que requerem atenção, bem como sobre perícias potencialmente difíceis de serem adquiridas pelo aprendiz. Assim, o subsídio de algoritmos próprios da área e o reconhecimento de características topológicas do grafo revelam-se instrumentos especialmente úteis.

Como analogia, na perspectiva do aprendiz, o grafo genético que representa o domínio de conhecimento assemelha-se a um mapa rodoviário. Uma perícia que possui várias relações oferece múltiplos caminhos para ser alcançada e, com isso, propõe uma menor dificuldade ao aprendiz. Em contrapartida, a disposição mais isolada de outra sugere a necessidade de orientação tutorial mais intensa.

Portanto, a métrica de crença sugere que agrupamentos de perícias são mais fáceis de se adquirir do que regiões esparsamente conectadas do grafo. Consequentemente, para a tutoria, a repetição deve ser pouco esperada em regiões densas e bastante requerida em áreas dispersas.

Torna-se concebível que a análise formal de um conteúdo programático representado

por um grafo genético possa desempenhar uma função útil no ensino, pela previsão da complexidade de aprendizagem do material. Em se tratando de um grafo cuja maior parte das perícias constitua meramente uma cadeia, o suporte tutorial reside inteiramente na repetição de um único método explicativo. Ao contrário, outro grafo genético, composto por várias ilhas, pontes e agrupamentos, pode exigir pouco esforço tutorial devido à conectividade rica em conhecimento naquele domínio, pois oferece múltiplos caminhos (ou explicações) para que as perícias sejam desenvolvidas.

Por fim, evidencia-se que a topologia do conteúdo programático, modelado em um grafo, sugere uma teoria da complexidade de aprendizagem. Embora experimentos adicionais precisem ser feitos para que se corrobore, refere-se a uma possibilidade teórica importante para a Educação, independentemente do uso de computadores.

4.1.8.5 Simulação de Aprendizes como Metodologia de Pesquisa

Outra potencialidade reside em explorar questões relativas ao ambiente para simulação de aprendizes. À proporção da autonomia que os perfis de simulação atinjam, pode-se experimentar o efeito de diferentes métricas de crença na estabilidade do ambiente, bem como de várias estratégias de aprendizagem na conquista do grafo genético.

Em aspecto avançado, ainda é possível incorporar a capacidade de aprendizado ao próprio tutor. Um impacto que se antecipa incide na construção do grafo genético. A intenção é eliminar a exigência do grafo genético finalizado para se iniciar o ensino, admitindo uma variante incompleta que possa evoluir, por aprendizagem de máquina, quando necessário.

4.1.8.6 Discussão sobre o Aprendizado

Goldstein finalmente justifica que, embora as questões levantadas pela introdução do grafo genético no meio científico sejam provocativas, não se trata de uma teoria de aprendizagem completa. Apontam-se questões importantes que ainda precisam ser estudadas, como:

1. Quando uma estratégia de aprendizagem deve ser aplicada?

2. Como analogias, generalizações e refinamentos vantajosos são detectados?

3. Quais sãos os critérios do esquecimento?

Além disso, como discutido, é necessário que um amplo esforço em pesquisa experimental seja despendido. Entretanto, acredita-se na potencialização que os ambientes de aprendizagem possam ter com a incorporação de uma teoria explícita sobre o aprendiz.

4.1.9 Considerações Finais

Por fim, Goldstein reforça que as próprias pesquisas sobre como o conhecimento do aprendiz evolui foram motivadas por Piaget, autodenominado um epistemologista genético. Ademais, a introdução dos grafos genéticos à comunidade científica explora como ocorre a construção de novo conhecimento pelo aprendiz. Como demonstração da efetividade dessa teoria, apresentaram-se diretrizes que podem aperfeiçoar a tutoria e a modelagem dos ambientes de aprendizagem. Também foram propostos mecanismos para modelar e operar no metaconhecimento sobre a aprendizagem.

Destaca-se, como contribuição secundária, a análise refinada da aprendizagem que um ambiente, modelado conforme se discutiu, pode proporcionar. Com isso, pretende-se explorar várias questões piagetianas relativas ao processo de aprendizagem. Trata-se, portanto, de uma metodologia proveitosa para esclarecer questões fundamentais tanto na Psicologia Cognitiva quanto na Inteligência Artificial.

4.2 Modelo de Sobreposição para Programação de Computadores

Conforme explicitado nos objetivos desta pesquisa, parte do mérito do corrente trabalho reside na criação de um Modelo de Sobreposição (*Overlay*) que seja preciso ao representar perícias no domínio de Programação de Computadores. Cabe, portanto, ao modelo denotar a incidência do **conhecimento do aprendiz** sobre o **conhecimento do domínio**.

Inicialmente, partiu-se do conjunto de perícias identificadas por [92], conforme Seção 2.2.2, retomado abaixo:

1. precisão sintática;
2. precisão semântica;
3. identificação de estruturas principais no programa fonte (busca por palavra-chave);
4. simulação mental dos estados do computador durante a execução;
5. catálogo de erros;
6. mapeamento mental das estruturas do programa;
7. checagem de pré-condições;
8. análise do problema;
9. integração dos subproblemas;
10. generalização da solução;
11. reutilização de soluções já conhecidas; e
12. catálogo de soluções.

Dessa forma, considerando-se doze perícias, o conhecimento do domínio de Programação de Computadores pode ser visto como um espaço **12-dimensional**. Cada dimensão representa a escala de aprendizado em uma perícia específica. Assim, uma perícia/dimensão poderia ser valorada dentro desta escala de, por exemplo, **0 a 100**. O estado

atual do conhecimento do aprendiz consistiria em uma **12-upla** cujo cada elemento pertenceria à referida escala.

Embora se trate de uma abstração bastante objetiva, representar o conhecimento do aprendiz apenas como uma coordenada no espaço do conhecimento do domínio mostra-se insuficiente para subsidiar os processos de um STI alimentado por tais modelos. Também, indicar uma perícia meramente por um fator numérico careceria de informações sobre quais parcelas desta perícia foram aprendidas e quais estão pendentes.

Além disto, dimensões independentes desconsideram possíveis interconexões e intersecções entre perícias. Na prática, embora cada perícia procure ter características bem definidas, existe entrelaçamentos entre duas, ou mais, perícias que configuram determinadas habilidades. Por exemplo, existem habilidades que se estabelecem na intersecção das perícias de *precisão sintática* e *precisão semântica*. As quatro últimas perícias elencadas, a saber, *integração dos subproblemas*, *generalização da solução*, *reutilização de soluções já conhecidas* e *catálogo de soluções*, também se mostram bastante atreladas para serem representadas por dimensões independentes.

Os entraves descritos estimularam a busca por um meio alternativo de representação que discretizasse cada perícia em subperícias componentes e que admitisse interconexões. Tal pormenorização facilitaria o mapeamento das habilidades que compõem o perfil de um programador perito.

Diante da revisão bibliográfica, o **grafo genético** [51] foi adotado como formalismo para a presente tese (vide Seção 4.1). Consiste em um arcabouço, com ênfase epistêmica, para representação de conhecimento. Nesse tipo específico de grafo, o **conhecimento do domínio** (ou de um perito) é constituído de um conjunto de fatos ou regras (vértices do grafo) interconectados por arestas que expressam relações como:

- generalização / especialização;
- analogia;
- simplificação / refinamento;
- desvio / correção; e

- pré-requisito / pós-requisito.

Considerado a modelagem do grafo genético, o conhecimento do aprendiz é tido como um subconjunto (ou subgrafo) do conhecimento de um perito. Todas as (sub)perícias pendentes ao aprendiz são evidenciadas pela sobreposição (*overlay*) do segundo grafo no primeiro.

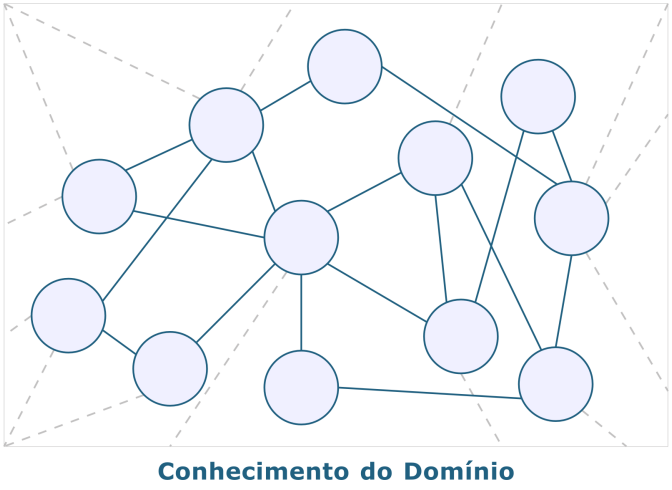


Figura 4.9: Representação simplificada de um grafo genético

A Figura 4.9 ilustra um exemplo simplificado de representação, em um grafo genético, das perícias que constituem um domínio de conhecimento. Convém lembrar que cada perícia pode ser descrita em subperícias componentes.

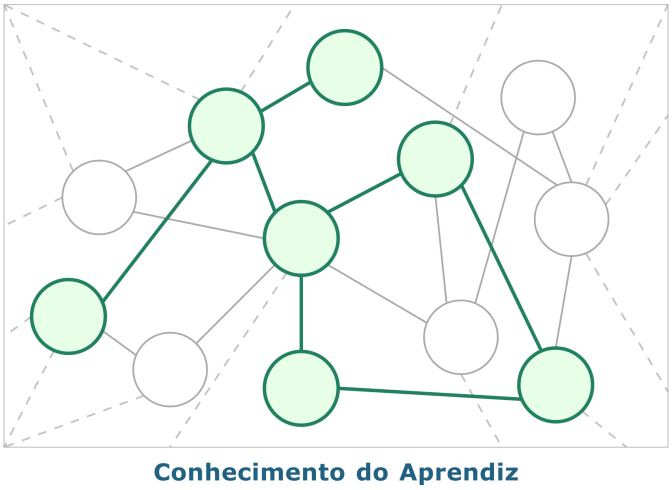


Figura 4.10: Sobreposição do Modelo do Aprendiz frente ao conhecimento do domínio

Através dos mesmos aspectos de representação externa, a Figura 4.10 mostra o estágio

atual do aprendiz frente ao conhecimento do domínio, ou seja, a sobreposição propriamente dita. Assim, o conhecimento do aprendiz é modelado como um subconjunto das perícias mapeadas no domínio. Ressalta-se o aspecto de sobreposição por meio da Figura 4.11.

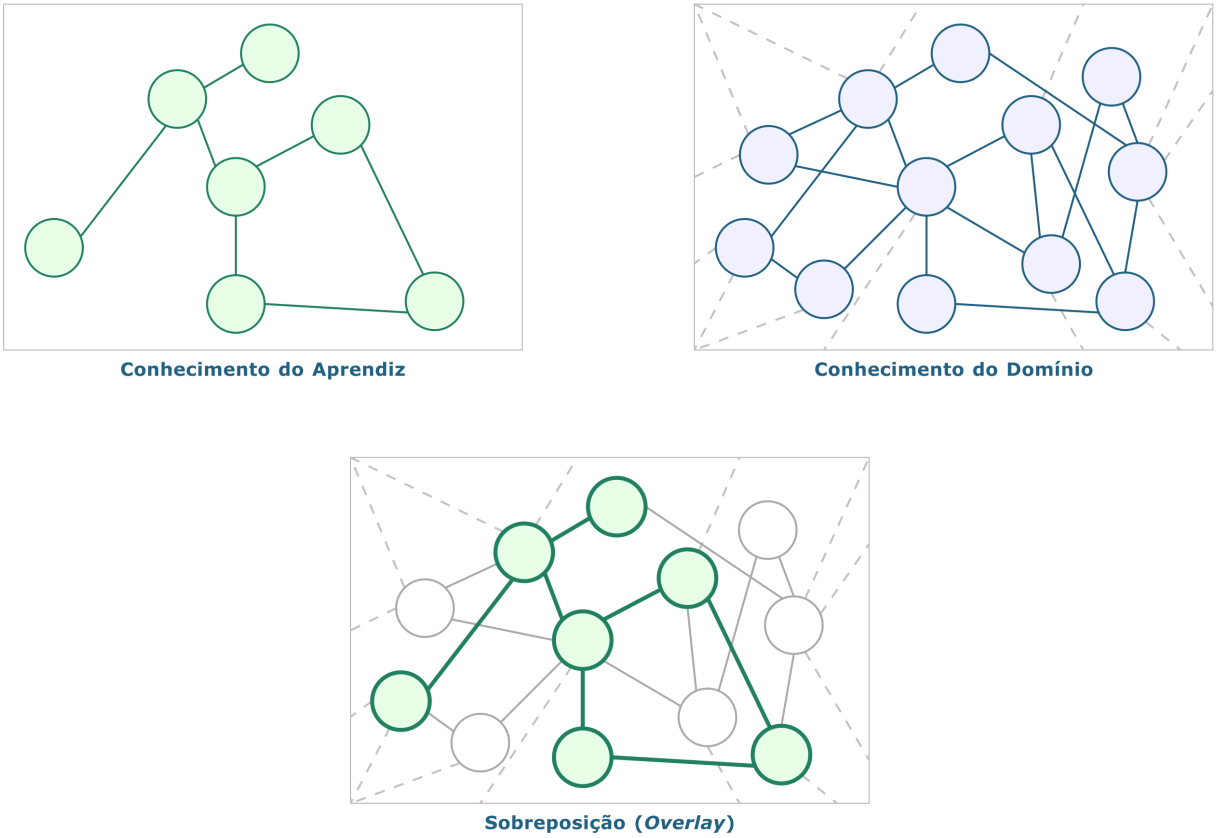


Figura 4.11: Aspecto de sobreposição realçado

A partir dos dois modelos, do aprendiz e do domínio, existe como prover um processo de busca heurística que oriente regiões do grafo a serem exploradas pelo aprendiz. Torna-se praticável, portanto, a indicação de perícias a serem prioritariamente cumpridas pelo aprendiz, conforme o estágio atual de conhecimento (Figura 4.12).

Em adição às possibilidades de suporte ao ensino providas pelos grafos genéticos e antecipadas por Goldstein (Seção 4.1), o presente doutorado contribui com a perspectiva de empregar o mesmo mapeamento de perícias (conhecimento do domínio) como um gabarito para os enunciados a serem propostos ao aprendiz. Cada enunciado coopera para o desenvolvimento de perícias específicas do aprendiz, podendo também ser visto como um subgrafo do conhecimento do domínio, conforme a Figura 4.13. Assumindo isso, os enunciados podem ser catalogados de acordo com o gabarito (Figura 4.14).

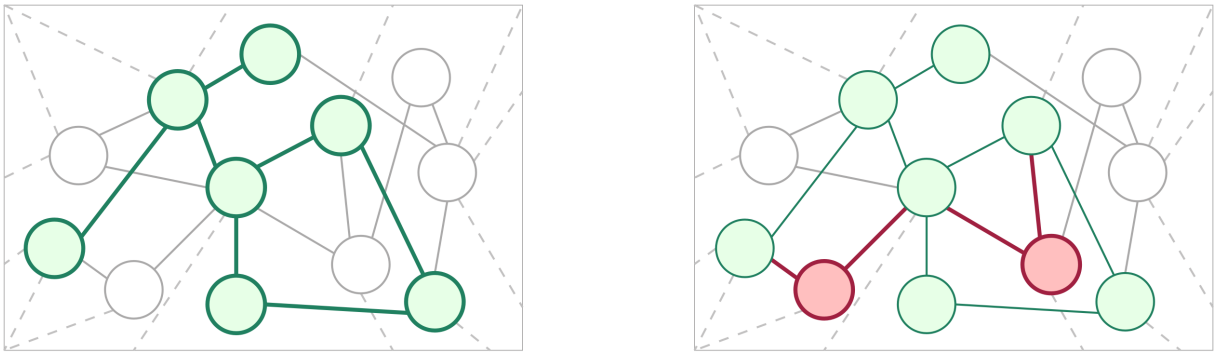


Figura 4.12: Indicação de regiões do grafo a serem exploradas

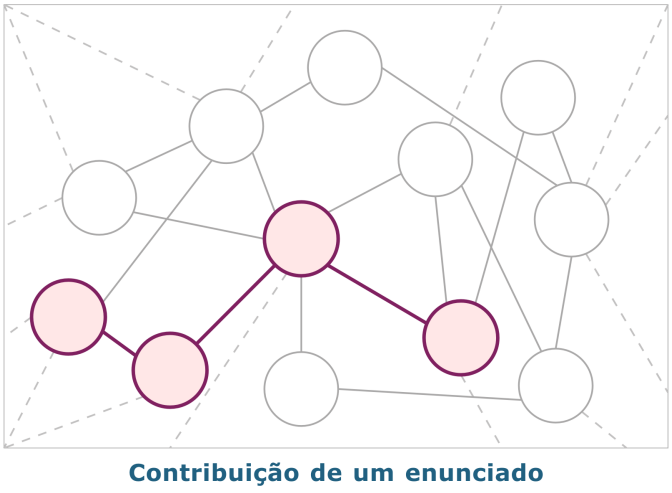


Figura 4.13: Sobreposição de um enunciado frente ao conhecimento do domínio

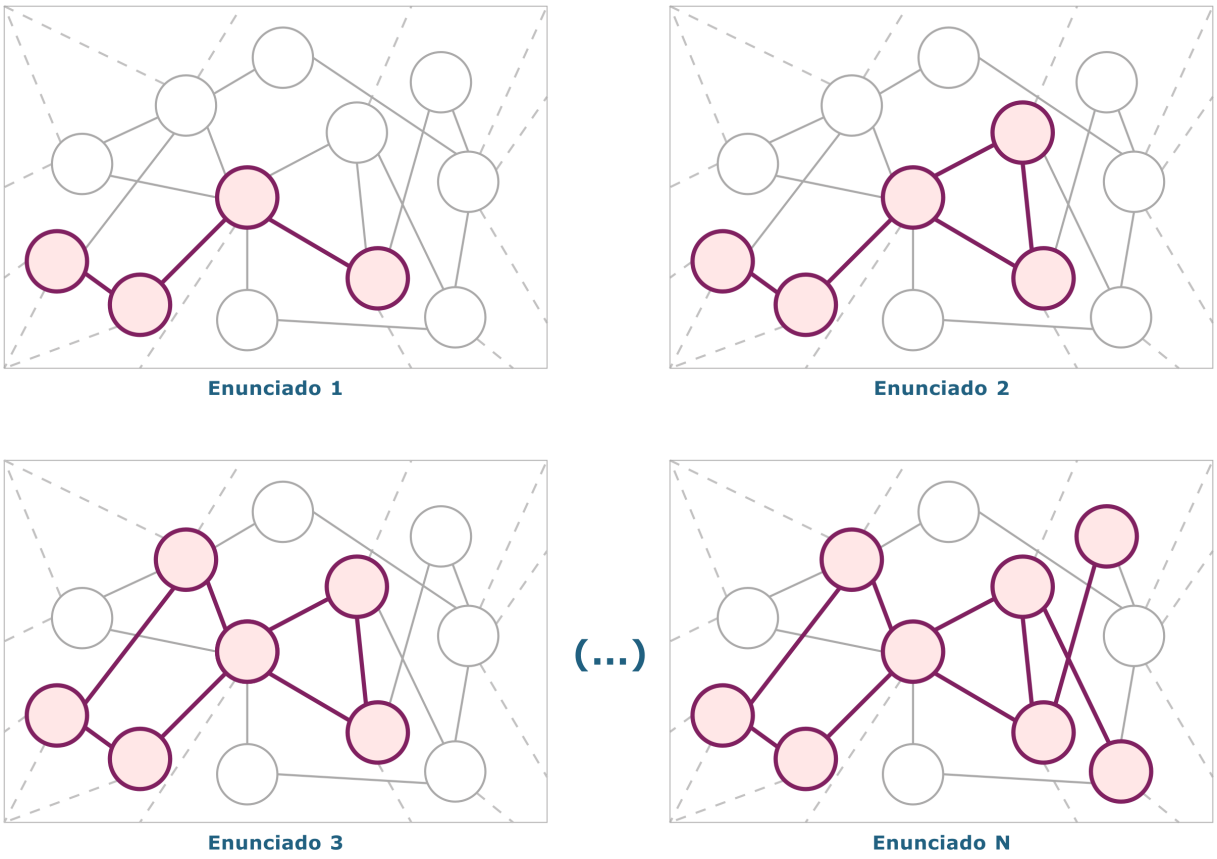


Figura 4.14: Catálogo de enunciados conforme o gabarito de conhecimento do domínio

Isto posto, o Modelo do Aprendiz adquire caráter dinâmico por ser alimentado com a avaliação do cumprimento dos enunciados propostos. Supõe-se a avaliação das resoluções por um especialista externo (humano), diante do gabarito de perícias fornecido pelo enunciado (Figura 4.15). Tal procedimento documenta o progresso do aprendiz e consequente evolução na estrutura do grafo genético. O especialista externo pode dar lugar a uma ferramenta de diagnóstico (conforme resenhado na Seção 2.3.2).

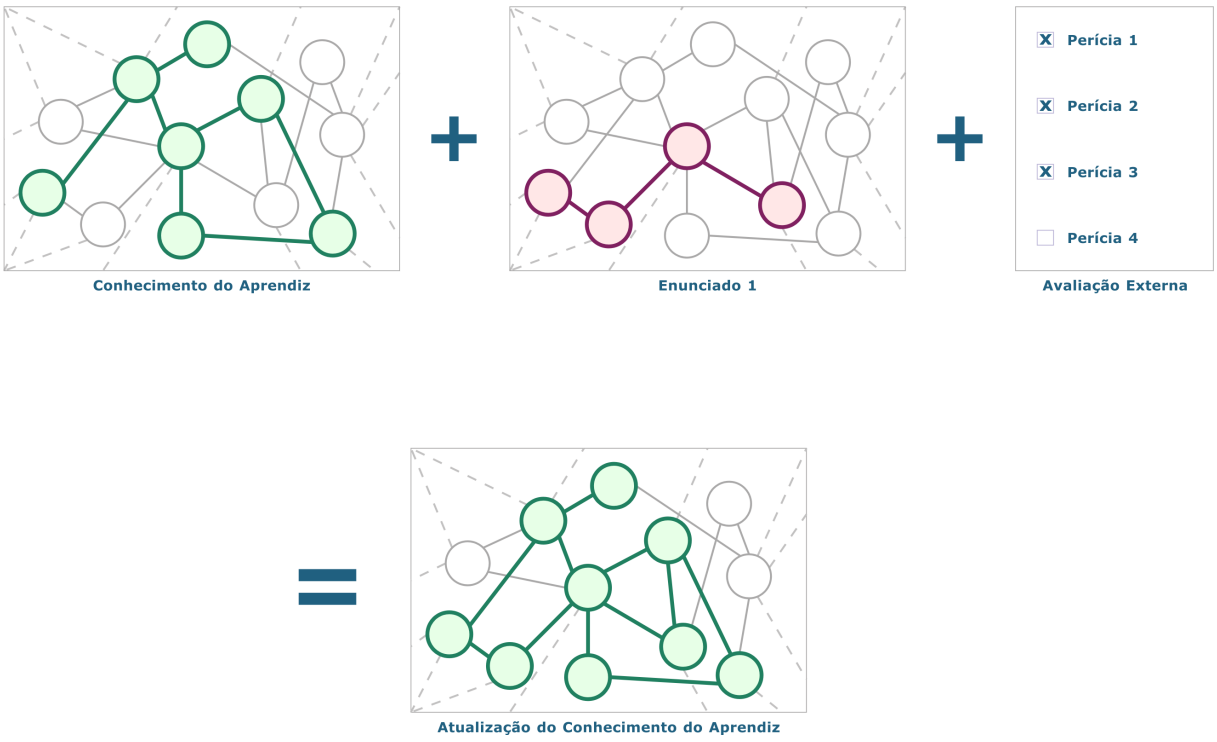


Figura 4.15: Atualização do Modelo do Aprendiz

Outra possibilidade, alcançada pela pesquisa e garantida pela representação em grafo, consiste em sintetizar o Modelo do Aprendiz de um grupo (sala de aula) e assim ter um espelho da ênfase com que as perícias são ensinadas pelo instrutor (Figura 4.16). Com isso, pode-se notar que um instrutor se sobressai no ensino de determinadas perícias, ou que um segundo negligencia algumas outras. De maneira semelhante, a sintetização destaca eventuais dificuldades do grupo frente à apropriação das perícias objetivadas.

Convém salientar que, nos exemplos dados, as perícias são sinalizadas como desenvolvidas ou não-desenvolvidas. Entretanto, é factível a implementação do desenvolvimento gradativo das perícias pelo aprendiz através de uma escala entre tais extremos, conforme

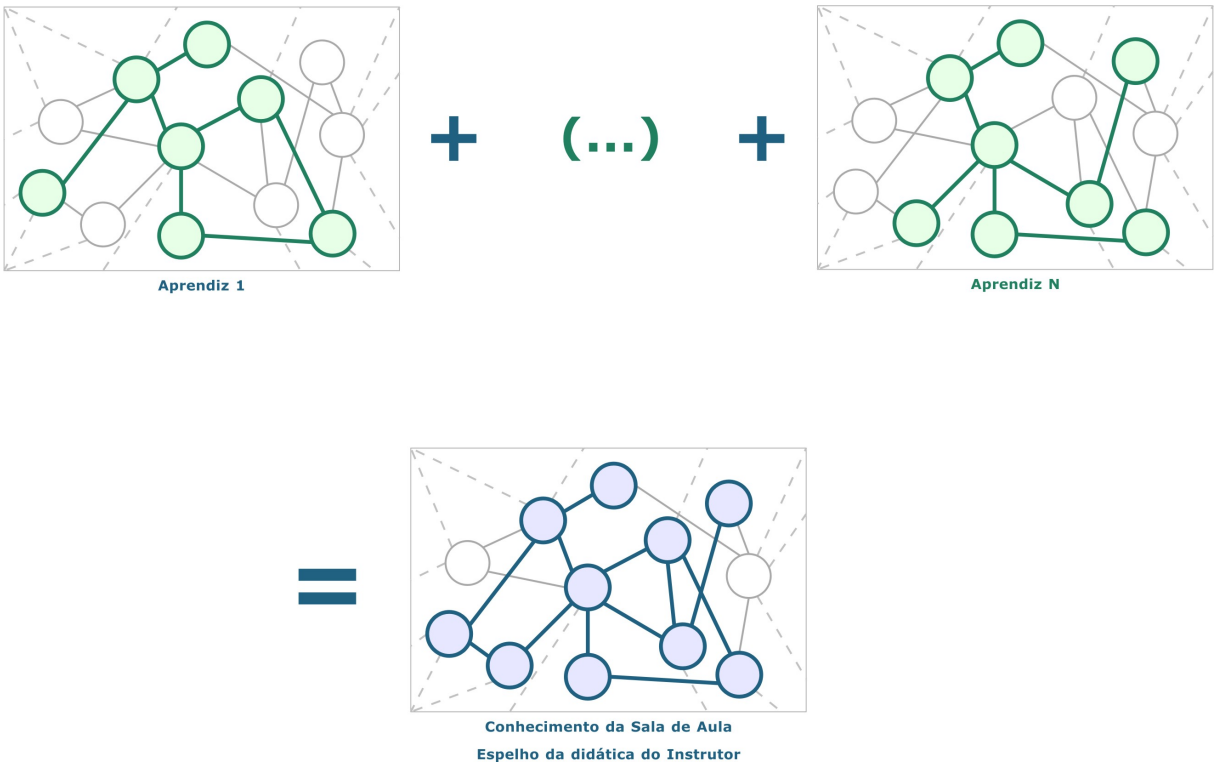


Figura 4.16: Sintetização do Modelo do Aprendiz de um grupo

a Figura 4.17. Dessa forma, a progressão seria gradual e mais fiel ao denotar o percentual aprendido/cumprido de cada perícia.

Figura 4.17: Perícias vistas em escala de progressão gradual

Com isso, pode-se fornecer detalhes da dimensão temporal do progresso do aprendiz, ou seja, apresentar um histórico do desenvolvimento das perícias pelo aprendiz (Figura 4.18). Remetendo-se à escala de progressão gradual, recém citada, há âmbito para uma representação externa bastante informativa, como exibir os vértices sendo (mais) coloridos

à proporção do tempo, em uma animação.

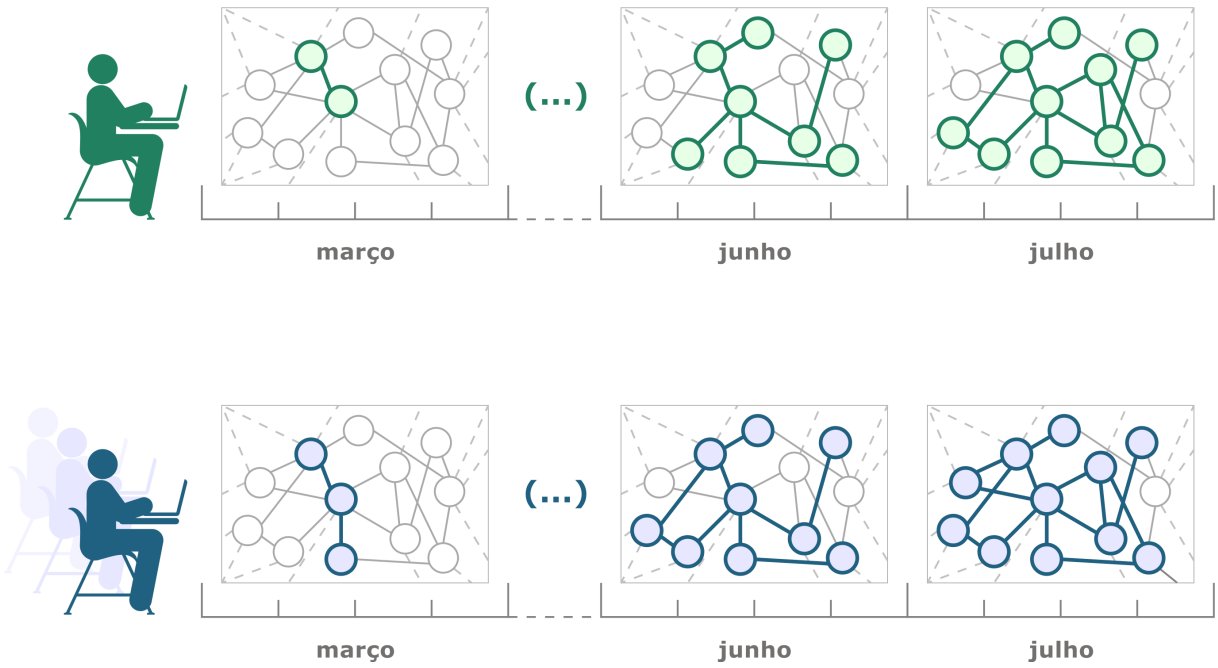


Figura 4.18: Dimensão temporal do progresso do aprendiz

O mesmo recurso é também aplicável à sintetização do Modelo do Aprendiz de um grupo. Logo, pode-se visualizar a progressão do aprendizado de uma sala inteira, ou de um curso inteiro ao longo de determinado tempo.

Sublinha-se, por fim, que a modelagem do aprendiz e do conhecimento do domínio através da sobreposição abre uma vasta gama de possibilidades de aplicação na tutoria inteligente. Nenhuma das ideias apresentadas nesta seção é complexa em si, entretanto se mostram ferramentas úteis para guiar o Modelo Pedagógico ou orientar instrutores (humanos) em sessões de ensino.

4.2.1 Subconjunto Modelado do Domínio

Considerações iniciais do reconhecimento das perícias que compõem o estereótipo de um programador experto provieram de [92] e foram retomadas neste capítulo. Como continuidade, somou-se outra pesquisa realizada pelo proponente enquanto professor do Departamento de Ciência da Computação na Universidade Estadual do Centro-Oeste, também credenciada junto à Pró-Reitoria de Pesquisa e Pós-Graduação segundo aprovação do de-

partamento mencionado. Os resultados foram documentados por meio de um relatório técnico final daquela pró-reitoria [76] e disseminados em [77].

O material produzido pelos alunos das disciplinas da área de Programação de Computadores (relacionadas no Anexo E), ministradas pelo doutorando, foi instrumento de observação e análise para o levantamento de perícias componentes do subconjunto modelado. Embora unidades curriculares como Estrutura de Dados e Paradigmas em Linguagem de Programação não atuem diretamente no desenvolvimento das perícias objetivadas por este estudo, são situações cujas lacunas e deficiências de aprendizado na área se tornam evidentes. Por conseguinte, também contribuíram para a determinação do subconjunto de perícias pretendido.

Constatou-se, primeiramente, a necessidade de se definir perícias de mais baixo nível de abstração que correspondessem à prática em Programação de Computadores. Tal percepção adveio da dificuldade de remetimento, na consideração de enunciados, a perícias isoladas do conjunto definido por [92]. Em critério de exemplo, a *precisão semântica* é uma perícia demandada praticamente por qualquer exercício da área. Como outro exemplo, a perícia denominada *catálogo de erros* acaba permeando todo o conjunto de instruções fornecido pela linguagem utilizada.

Considerando isso, assumiu-se que o conjunto proposto por [92] se caracteriza pelo alto nível de abstração e pela sobrejacência a outras perícias mais próximas dos elementos instrucionais da programação. Todavia, são justamente as últimas perícias que se mostram pertinentes à modelagem e evolução do aprendiz frente ao conhecimento do domínio, como também à catalogação e ao sequenciamento de enunciados.

Portanto, investiu-se na definição de um segundo conjunto de perícias, subjacente em nível de abstração, àquele de referência. A abordagem consistiu em reconhecer frações mais restritas e discretizadas de habilidades em correspondência às instruções e princípios de programação.

O principal instrumento para observação e análise do conjunto perícias que se determinava foi a sondagem de avaliações escritas, trabalhos, exercícios e dúvidas das disciplinas ministradas. Mediante a sondagem do material produzido pelos alunos, revisou-se suces-

sivamente o conjunto de perícias. O contraste, entre o material produzido e o conjunto de perícias suposto até o mesmo momento, norteava a admissão de novas perícias e a omissão de outras.

Uma primeira tentativa resultou em um mapeamento detalhado que identificou 335 perícias. O subconjunto de Programação de Computadores considerado foi:

1. Conceitos iniciais. Análise e abstração de informações na resolução de problemas;
2. Composição e efetivação de algoritmos em linguagem natural;
3. Declarações de variáveis e constantes. Instruções de entrada, saída e atribuição;
4. Expressões aritméticas, relacionais e booleanas;
5. Estruturas condicionais;
6. Estruturas de repetição.

Tal pormenorização teve resultado esclarecedor sobre o domínio de conhecimento naquela etapa da pesquisa. O conjunto obtido atingiu o refinamento de ser representado por uma lista hierarquizada com 5 níveis. No entanto, ponderando sobre a provável quantidade de interconexões das 335 perícias, inferiu-se a alta densidade e os consequentes cruzamentos de arestas no grafo genético resultante.

Também foi considerada a carga cognitiva exigida no processo de autoria para gerenciar tamanho conjunto de perícias, além do impacto negativo de uma representação que se tornaria inextricável como referência. Com isso, a elicitação de um enunciado, por exemplo, seria feita frente à alternativa de contribuição para o desenvolvimento de 335 perícias, ou seja, um detalhamento excessivo para ser contemplado produtivamente pela autoria.

Como refinamento subsequente, o conjunto de perícias foi reduzido em termos daquelas pertencentes aos primeiros níveis hierárquicos. Depois, ainda aprimorado continuamente por meio da sondagem do material fornecido pelos alunos. Considerou-se o conjunto estável quando, por mais de um semestre, não demandou ajustes.

Em conclusão, o subconjunto do domínio de Programação de Computadores consolidado por esta pesquisa foi modelado, em um grafo genético, pela ferramenta de autoria prototipada (descrita posteriormente em 5.1). O referido grafo, composto por 41 perícias e 60 relações, foi acrescentado juntamente com a descrição de cada perícia no Anexo F. Adverte-se, por fim, que o mapeamento é uma versão experimental, proposto para conduzir a validação da pesquisa e, assim sendo, sujeito a alterações pela comunidade científica.

4.2.2 Sensibilidade da Modelagem à Perspectiva do Autor

Convém destacar que a formalização das perícias e respectivas relações, principalmente em contextos mais abstratos do que a aplicação direta de regras lógicas, torna-se sensível à perspectiva de quem a realiza. Assim, diferentes autores, ao assumirem a própria perspectiva sobre o conhecimento do domínio, consequentemente terão modelagens desiguais refletindo isso em um grafo genético.

Em Programação de Computadores, as perícias constituem regras implícitas e abstratas que subsidiam o conhecimento procedimental da área. Assim, abre-se espaço para discussão ainda nas perícias (regras) que alguém formalize. Além disso, há igualmente margem para controvérsia quanto à definição de relações evolutivas apropriadas. Por exemplo, consideradas duas perícias, um autor pode interpretar que a primeira é uma especialização da segunda, por abordar uma fração particularizada do conhecimento que a segunda acumula. Outro autor, por sua vez, pode concluir que a primeira é um refinamento da segunda, pois evolui a perícia original para considerar um conjunto de distinções mais peculiares.

Acredita-se, portanto, que não haja formalização unívoca na maioria dos domínios de conhecimento que sejam modelados. Contudo, é praticável se atingir versões progressivamente mais precisas e estáveis de tais modelagens. Torna-se inerente pensar que a modelagem do conhecimento de um domínio convenha ser realizada de maneira colaborativa por um grupo de autores. Isso, independentemente do subsídio de software quanto à colaboração.

4.2.3 Considerações sobre Princípios e Perícias

Conforme descrito nas seções 4.1.2 e 4.1.5, o grafo genético fornece subsídio para a representação de conhecimento tanto procedimental (perícias) quanto declarativo (princípios) de um domínio. Entretanto, nesta pesquisa, durante a atividade de reconhecimento das perícias que compõem o estereótipo de um programador experto, percebeu-se que em Programação de Computadores os princípios se tratam de conhecimentos bastante pontuais cuja simples articulação ou contraste já caracterizam perícia. E isso, inclusive, é possível de ocorrer na especificação de outros domínios de conhecimento.

Em tais termos, por exemplo, o *conceito de variável* consiste em um princípio. Contudo, o *reconhecimento de variáveis na análise e resolução do problema*, a *determinação dos tipos de dados correspondentes*, bem como a *distinção conceitual e pragmática entre variáveis e constantes*, logo denotam perícias. Ou seja, embora os princípios se constituam de fundamentos isolados, o simples domínio sobre esses conceitos em se tratando de coordenação e contraste a fim de elaborar uma solução (algoritmo) passa a consolidar perícia na área.

Portanto se esclarece que, no subconjunto de Programação de Computadores modelado, o domínio de cada aspecto, em termos da articulação ou do contraste de fundamentos, já foi considerado como perícia a ser desenvolvida pelo aprendiz.

4.2.4 Perícias Sobrejacentes de Alto Nível

Diante da oportunidade, revisou-se o conjunto de características do estereótipo de um programador perito estabelecido por [92]. No contexto desta pesquisa, tais características são denominadas perícias sobrejacentes de alto nível. Em acréscimo às 12 perícias identificadas, tem-se:

13. **Velocidade de resolução**

    Corresponde à rapidez com que uma solução correta é atingida pelo aprendiz. Embora a velocidade de resolução, em determinados momentos do aprendizado, possa não figurar em proporção crescente à experiência, trata-se de uma das características

em que mais facilmente se nota a disparidade de progresso entre aprendizes.

14. **Legibilidade do código escrito**

    Consiste na gradual qualidade de escrever códigos legíveis, com indentação apropriada, nomenclatura clara para identificadores e uso de elementos que reduzam a ambiguidade de interpretação (por exemplo, parênteses em expressões). Como perícia, influencia diretamente a capacidade de escrita do aprendiz, a confiabilidade do programa resultante e o custo da reparação de erros.

15. **Otimização de soluções**

    Diz respeito ao aperfeiçoamento da solução desenvolvida diante da fronteira de conhecimento atual do aprendiz. Não se trata de alcançar uma solução ótima para o problema, mas de buscar e explorar a alternativa mais eficiente possível de resolvê-lo, instrumentada pela perícia do aprendiz naquele momento. Uma solução pode evoluir no sentido de ser mais rápida, exigir menos recursos computacionais ou ainda se mostrar mais manutenível em consequência da própria legibilidade.

16. **Capacidade de depuração**

    Abrange o processo de encontrar e extinguir possíveis erros na solução, além de sanar comportamentos inesperados que possam ocorrer durante a execução do programa resultante.

17. **Definição de casos básicos de teste**

    Resume-se à elaboração de casos básicos de teste para identificar erros de *precisão semântica* e garantir o atendimento aos requisitos do problema determinado.

18. **Construção de diálogo adequado de interface**

    Compreende o emprego dos elementos da linguagem utilizada no intuito de prover um diálogo de interação minimamente adequado à solução proposta.

19. **Autoconhecimento sobre habilidades metacognitivas**

    Remete, conforme sinalizado por Goldstein [51], bem como descrito nas seções 2.1 e 4.1.8, ao conhecimento que o indivíduo detém, e constrói, sobre o próprio processo de

aprendizagem. Ajuda o estudante a identificar e utilizar abordagens individuais de aprendizado mais eficientes, como fazer anotações ao longo do conteúdo, sintetizar em esquemas gráficos, encontrar exemplos externos, realizar experimentos, discutir soluções, entre outras.

Convém ressaltar que as características das últimas perícias descritas, dependendo do aprofundamento que sejam tratadas, começam a extrapolar o domínio de Programação de Computadores e a adentrar áreas como Engenharia de Software, Interação Humano-Computador (IHC) e, mesmo, Psicologia Cognitiva.

CAPÍTULO 5

FERRAMENTAS DE AUTORIA PARA OS FORMALISMOS

O processo de autoria, de acordo com o formalismo adotado e descrito no capítulo anterior (4), é suportado por duas ferramentas implementadas como protótipos. A primeira se concentra na definição do conhecimento do domínio por meio do detalhamento em grafo genético das perícias componentes. A segunda ferramenta, usando a mesma representação gráfica, destina-se à catalogação e elicitação de enunciados descritos como subgrafos daquele modelado pela primeira.

As ferramentas prototipadas foram concebidas como uma extensão do ambiente MÚLTIPLA. Para fins de referência, deu-se o nome de **Dandelion**[1] ao conjunto dos dois protótipos. A linguagem **Java** foi utilizada para a implementação de ambos, conforme critérios previamente considerados no desenvolvimento do MÚLTIPLA [74].

Toda a representação visual do grafo genético foi fundamentada sobre a biblioteca de componentes **JGraph**[2], também anteriormente usada pelo MÚLTIPLA. Trata-se de uma ferramenta madura, rica em funcionalidades, com código-aberto e integralmente escrita em Java. Possui compatibilidade com Swing, tanto em aspectos visuais quanto no que se refere à arquitetura.

Sendo própria para a manipulação de grafos, a biblioteca JGraph alicerça-se na teoria matemática correlata (Teoria dos Grafos). Desse modo, o pacote descrito oferece tanto mecanismos para visualização, interação, desenho e disposição de grafos, como também para o tratamento de operações que cabem à teoria que os envolve (e.g. determinação do caminho mínimo entre dois vértices).

Somadas tais características à familiaridade do proponente com a biblioteca, foi consequente a escolha do JGraph para modelar, visual e internamente, muitos dos aspectos

[1] Significa, em português, "dente-de-leão". Remete à semelhança que a flor tem, em determinada fase, com uma componente conexa de um grafo.

[2] Disponível em *http://www.jgraph.com*.

que concernem à manipulação dos grafos genéticos. Salienta-se que, em vários momentos, as funcionalidades da biblioteca precisaram ser estendidas a fim de que suportassem peculiaridades da modelagem de grafos genéticos.

A persistência dos dados em memória secundária, outra característica de implementação, foi inicialmente idealizada através da serialização de objetos nativa provida pela linguagem Java. Entretanto, o formato do arquivo gerado não se mostra transparente para interoperação com futuras ferramentas, bem como o método é bastante sensível à consistência entre as classes de origem e os objetos persistentes.

Como alternativa, foi pesquisada e empregada a biblioteca **XStream**[3] que fornece componentes de uso facilitado para a serialização de objetos no formato XML[4]. Corresponde a uma possibilidade interessante visto que a solidez da padronização do formato permite que diferentes pesquisas compartilhem arquivos inteligíveis.

Considerando as escolhas supracitadas, as seções imediatas descrevem separadamente as duas ferramentas de autoria prototipadas.

5.1 Ferramenta de Definição de Capacidades do Domínio

A primeira ferramenta de autoria implementada destina-se à definição de capacidades em domínios de natureza prática. Mediante o formalismo proposto nesta tese, incide na formalização do conhecimento do domínio por meio da particularização das perícias componentes em um grafo genético. Trata-se de uma ferramenta que utiliza visualmente a representação de grafo genético e cuja interação é, em essência, guiada por *menus* de contexto (acessíveis pelo botão contrário do *mouse*), conforme ilustra a Figura 5.1.

Na memória primária, os vértices do grafo são alocados em uma tabela de dispersão (*hashing*) cuja chave é um identificador interno exclusivo e predefinido. Armazenam-se as arestas em listas de adjacências associadas aos respectivos vértices.

A ferramenta basicamente permite: **(a)** a descrição de informações textuais do domínio de conhecimento; **(b)** a inserção, edição e remoção de perícias; **(c)** o estabelecimento

[3]Disponível em *http://xstream.codehaus.org*.
[4]Do inglês, *Extensible Markup Language* (Linguagem de Marcação Extensível).

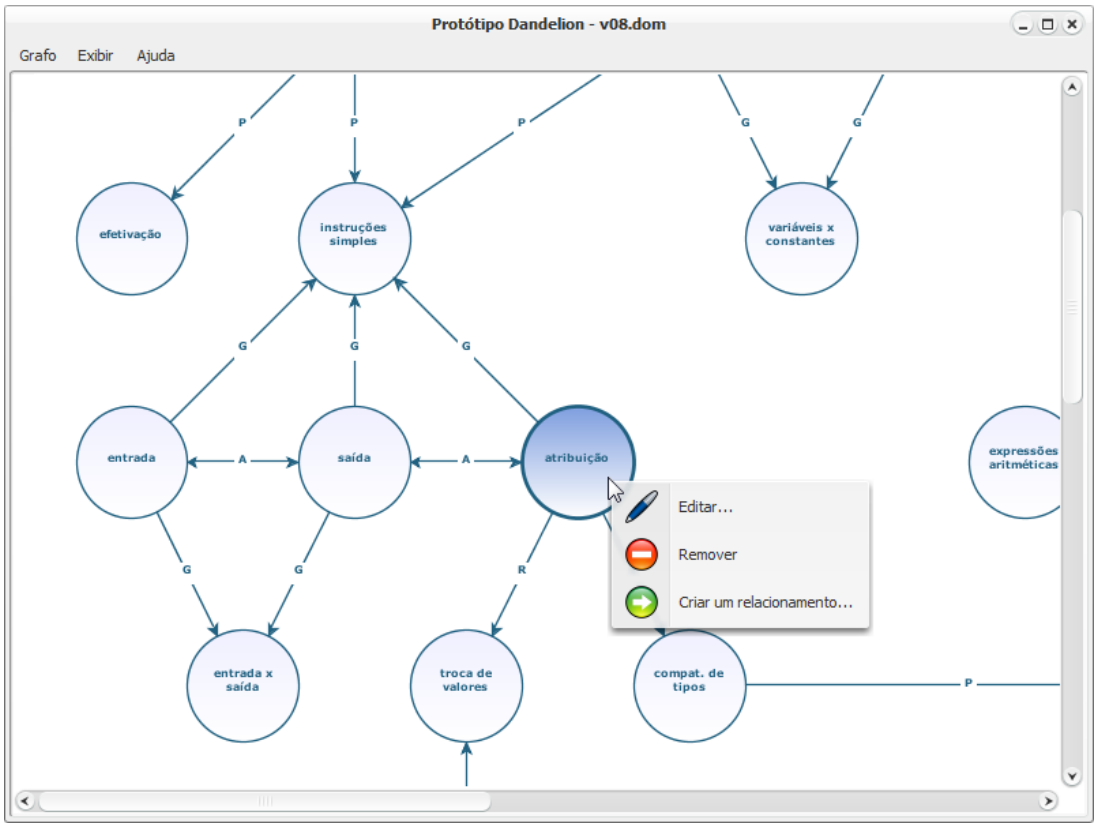


Figura 5.1: Interface da ferramenta de definição de capacidades do domínio

das relações evolucionárias entre as perícias; **(d)** a definição de uma perícia inicial; **(e)** a validação do modelo descrito quanto à alcançabilidade das demais perícias a partir daquela inicial. Cada conjunto de opções mencionado é estendido na sequência. O grafo genético constante nos exemplos foi introduzido na Seção 4.2.1 e perfaz o Anexo F.

5.1.1 Descrição de Informações Textuais das Capacidades do Domínio

A opção é fornecida no *menu* de contexto associado a espaços vazios na área do grafo genético, vide Figura 5.2. Em uma caixa de diálogo, apresentada pela Figura 5.3, há campos para que sejam informados o nome e uma descrição para o grafo genético que formaliza as capacidades do domínio. A intenção foi possibilitar um espaço textual para que se vinculassem informações anexas pertinentes ao modelo.

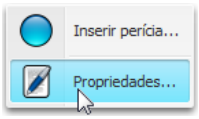


Figura 5.2: Acesso às informações textuais das capacidades do domínio

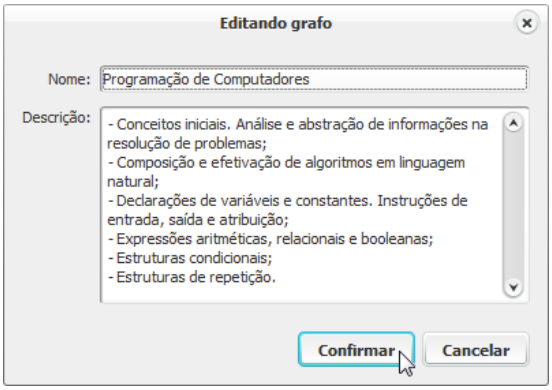


Figura 5.3: Edição das informações textuais das capacidades do domínio

5.1.2 Inserção, Edição e Remoção de Perícias

A inserção de uma perícia também é feita por meio do *menu* de contexto associado a espaços vazios na área do grafo genético, conforme mostrado pela Figura 5.4. O vértice que representa a perícia é adicionado na posição corrente do ponteiro do *mouse*.

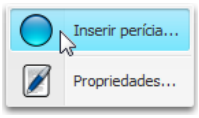


Figura 5.4: Inserção de uma perícia

As opções de edição e remoção de perícias estão acessíveis no *menu* de contextos referente a cada vértice correlato (Figura 5.5). A edição é realizada em uma caixa de diálogo onde existem campos para que se informe um identificador e uma descrição para aquela perícia (Figura 5.6). Tal identificador consiste em um mnemônico para a perícia e, conforme validação associada, deve ser único no grafo. A descrição, por sua vez, provê maior detalhamento para a mesma perícia. Distinguem-se porquanto os identificadores são usados como rótulos dos vértices na representação visual do grafo e, em contraste, as descrições permanecem ocultas da tela principal.

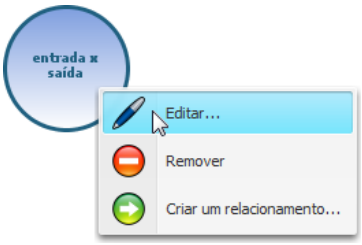


Figura 5.5: Acesso à edição e remoção de uma perícia

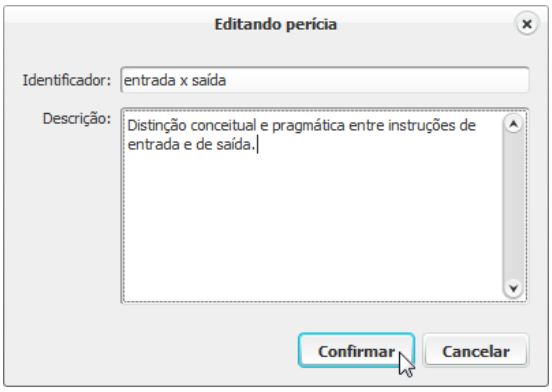


Figura 5.6: Edição das informações relacionadas à perícia

A remoção, convém destacar, implica o apagamento de todas as relações que envolvem a referida perícia. Ou seja, remove-se também quaisquer arestas que tenham a perícia como extremidade.

5.1.3 Estabelecimento de Relações Evolucionárias

A conexão entre duas perícias é disponibilizada no *menu* de contexto daquela que origina o relacionamento, como indicado pela Figura 5.7. Mediante o acesso citado, abre-se uma caixa de diálogo para que a relação seja estabelecida (Figura 5.8). São requeridos a perícia de destino e o tipo de relacionamento, ambos informados em caixas de seleção.

Na implementação da ferramenta, foram consideradas as relações de:

- generalização / especialização;
- analogia;
- simplificação / refinamento;

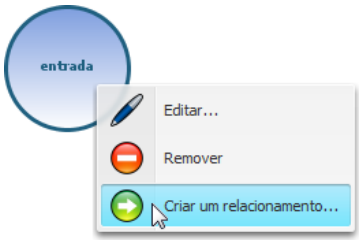


Figura 5.7: Acesso à criação de um relacionamento entre duas perícias

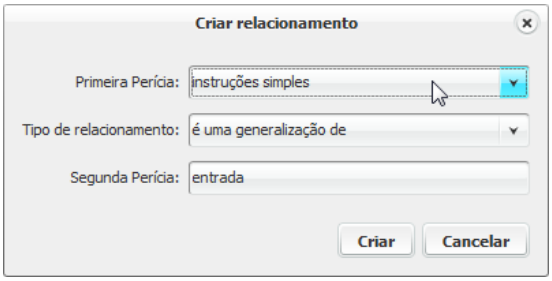


Figura 5.8: Criação de um relacionamento entre duas perícias

- desvio / correção; e
- pré-requisito / pós-requisito.

Em caso de equívoco, a criação pode ser cancelada pelo autor. Além disso, a aresta que denota o relacionamento oferece um *menu* de contexto (Figura 5.9) com as opções de edição e remoção, bem como a possibilidade de inverter o próprio sentido (a origem passa a ser o destino e vice-versa)

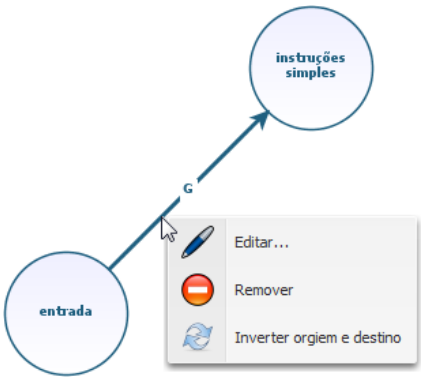


Figura 5.9: Acesso à edição, remoção e inversão do sentido de um relacionamento

5.1.4 Definição de uma Perícia Inicial

No mapeamento das capacidades de um domínio, é necessário determinar uma perícia que seja assumida como estado inicial para que as demais se desenvolvam. A opção é provida no *menu* denominado *Grafo* da tela principal. Então, a perícia deve ser selecionada em uma caixa de diálogo específica, apresentada pela Figura 5.10.

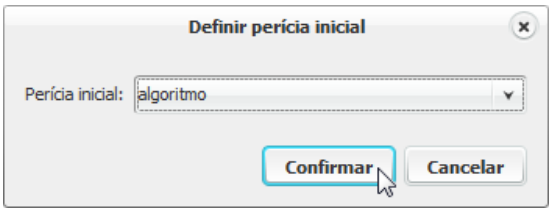


Figura 5.10: Definição de uma perícia inicial para o domínio

5.1.5 Validação do Modelo Descrito

A validação do modelo de capacidades descrito é realizada em termos da alcançabilidade das demais perícias a partir daquela definida como inicial. Portanto, a determinação de uma perícia inicial, conforme detalhado na seção anterior, é condição necessária para que o modelo seja válido. A próxima etapa reside na constatação de que o grafo genético construído possui uma única componente conexa e, com isso, admite que todas as perícias sejam alcançáveis a partir daquela inicial.

A constatação é realizada por uma única busca em profundidade [102], feita a partir da perícia inicial do grafo. A busca progride até que não haja vértices a serem expandidos. Depois, caso restem vértices que não foram visitados pela busca, significa que não pertencem à mesma componente conexa da perícia inicial e, em consequência, são inalcançáveis através do percurso. O modelo, portanto, é considerado inválido e requer alterações. Com a finalidade de orientação, a ferramenta aponta as perícias que não puderam ser alcançadas, de acordo com a Figura 5.11.

Logo, evidencia-se que a conexidade do grafo foi associada à consistência do modelo. Embora não se possa dizer que qualquer grafo com mais de uma componente conexa seja

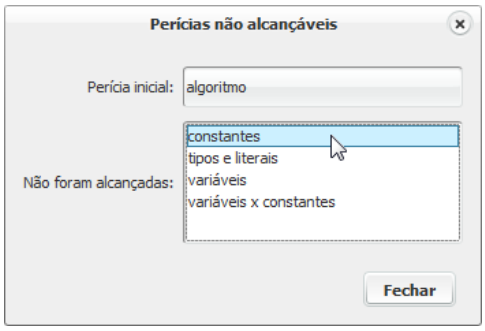


Figura 5.11: Apontamento de perícias não alcançáveis na verificação do modelo

um modelo inconsistente, o protótipo se restringe em assumir a conexidade como critério mínimo para validação.

Ademais, convém registrar que o direcionamento no grafo genético informa o sentido de leitura das relações (semântica, pode-se dizer) e não diz respeito à orientação do grafo. Por conseguinte, o direcionamento torna-se irrelevante na alcançabilidade dos vértices, sendo o grafo considerado não-direcionado. Assim, a direção expressa o sentido mais epistemologicamente preciso para a evolução da perícia, mas não impede que o percurso contrário aconteça.

5.1.6 Funcionalidades Adicionais

As funcionalidades adicionais da ferramenta estão dispostas na barra de *menus* da tela principal, subdividindo-se em:

Grafo

Comporta opções de caráter mais geral (já abordadas neste capítulo) sobre os modelos que descrevem capacidades de domínio, de acordo com a Figura 5.12, sendo: **(a)** editar propriedades textuais, **(b)** definir uma perícia inicial e **(c)** validar o modelo formalizado; bem como opções referentes à manipulação dos respectivos arquivos: **(d)** novo, **(e)** abrir, **(f)** salvar, **(g)** salvar como, **(h)** exportar como imagem, **(i)** exportar como PDF[5] e **(j)** sair do protótipo;

[5] *Portable Document Format* (Formato de Documento Portátil).

Exibir

Oferece um painel, mostrado pela Figura 5.13, com uma escala de *zoom* para a visualização do modelo;

Ajuda

Reduz-se a uma única opção, vide Figura 5.14, que exibe uma caixa de diálogo contendo informações sobre o protótipo. São compreendidos o logotipo da UFPR, o título da tese, assim como os nomes do orientador, do proponente e *e-mail* para contato (Figura 5.15).

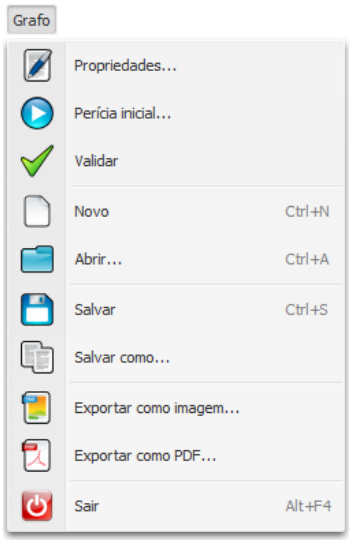


Figura 5.12: *Menu* Grafo

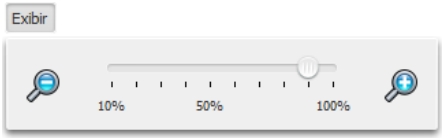


Figura 5.13: *Menu* Exibir

Figura 5.14: *Menu* Ajuda

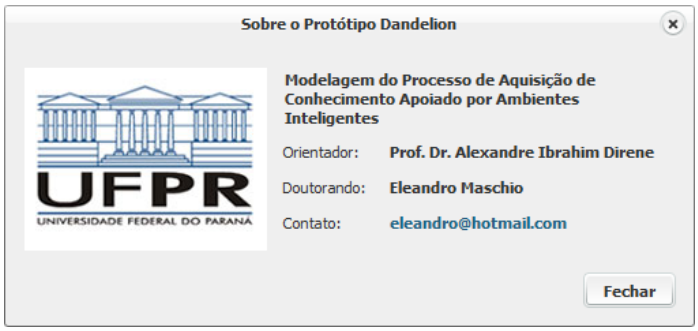


Figura 5.15: Tela de informações sobre o protótipo

5.2 Ferramenta de Elicitação de Enunciados

A segunda ferramenta, usando a mesma representação visual de grafo genético, destina-se à elicitação e catalogação de enunciados descritos como subgrafos do conhecimento do domínio formalizado. Isto é, permite que sejam demarcadas as perícias específicas para as quais cada enunciado contribui. A sobreposição do enunciado no grafo enfatiza a fração do conhecimento exercitada, de acordo com a Figura 5.16.

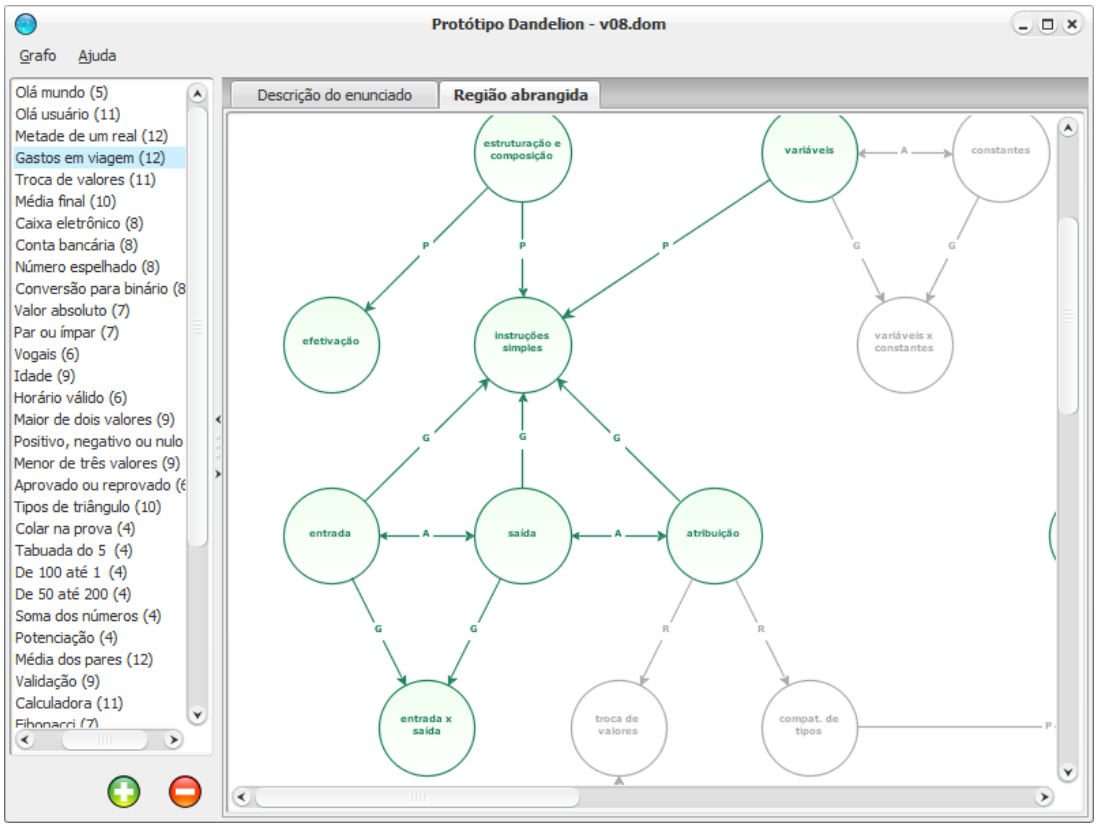


Figura 5.16: Interface da ferramenta de elicitação de enunciados

A representação do catálogo, na memória primária, é feita por uma tabela de dispersão cuja chave de cada enunciado consiste em um identificador interno exclusivo e predefinido. As perícias que um enunciado contempla são armazenadas em uma lista anexada.

Em essência, possibilita-se: **(a)** a inserção, edição e remoção de enunciados; **(b)** a inspeção da quantidade de enunciados que exercitam cada perícia; e **(c)** a recomendação de enunciados pela ferramenta. As três interações são detalhadas na sequência e requerem a abertura do modelo de capacidades do domínio descrito pela primeira ferramenta. As duas últimas funcionalidades são oferecidas no *menu* intitulado *Grafo* da tela principal (Figura 5.17). A propósito de conhecimento, o catálogo de enunciados utilizado para exemplificação compõe o Anexo G. A modelagem foi feita em sobreposição às capacidades detalhadas no Anexo F e que foram introduzidas na Seção 4.2.1.

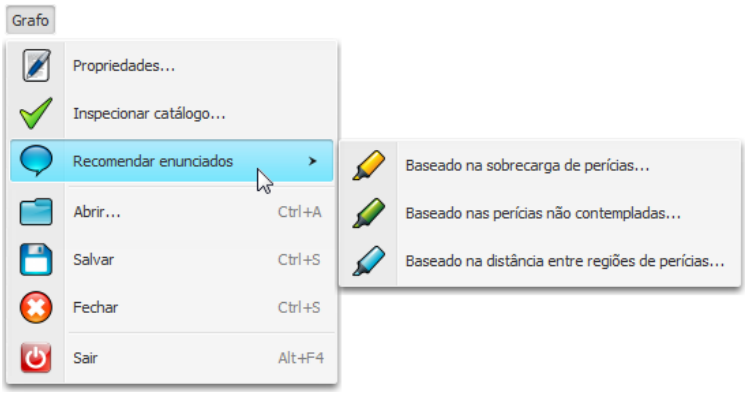


Figura 5.17: *Menu* Grafo

5.2.1 Inserção, Edição e Remoção de Enunciados

Realiza-se a inserção e a remoção de enunciados por meio dos botões correspondentes, localizados na parte inferior da lista que indexa o catálogo. Um enunciado, recém-inserido, assume automaticamente um identificador sequencial designado pela ferramenta. A edição é provida no painel direito da mesma tela e também habilitada por um botão (vide Figura 5.18).

As informações requeridas pela edição foram compartimentadas em duas guias do mesmo painel. Na primeira, há campos para serem informados um identificador e o texto

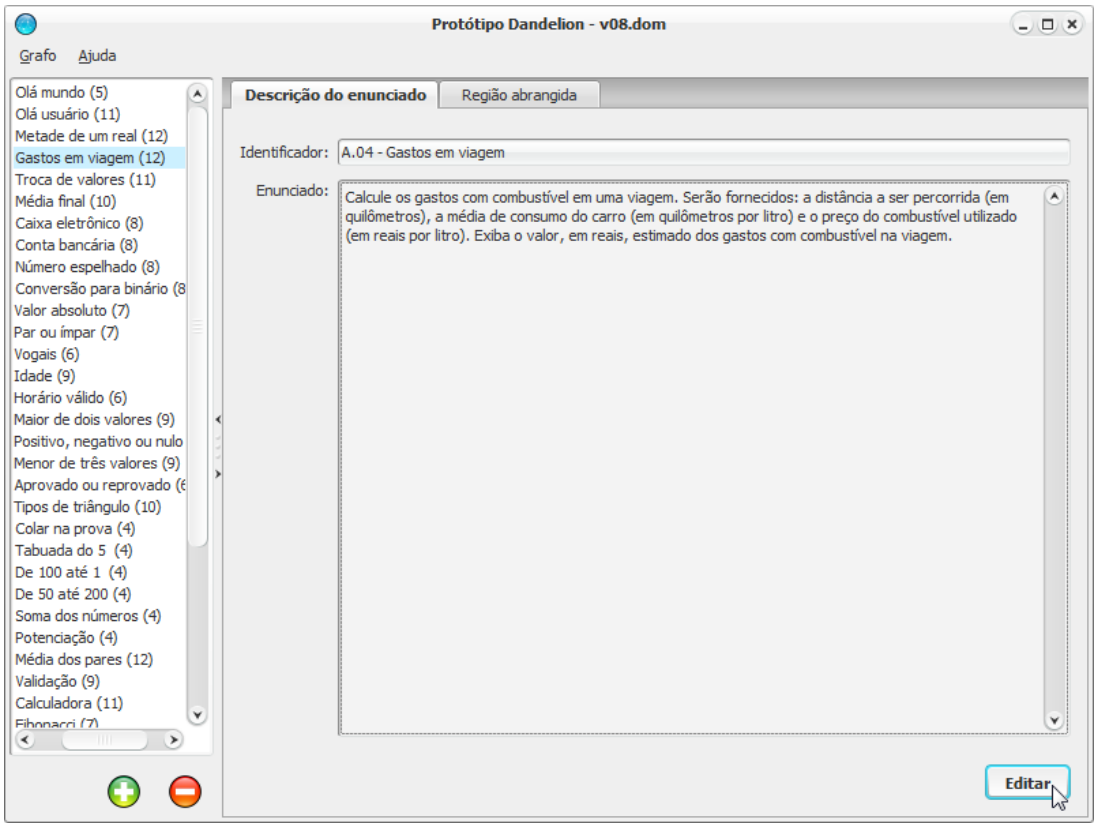


Figura 5.18: Descrição do enunciado

de enunciado propriamente dito. O identificador, assim como ocorre na descrição de perícias, deve ser um mnemônico exclusivo no catálogo. Portanto, na lista que indexa o catálogo, os enunciados são relacionados por meio dos respectivos identificadores sucedidos, entre parênteses, pelo número de perícias contempladas.

A segunda guia se remete à sinalização das perícias abrangidas pelo enunciado. Tais perícias são marcadas e desmarcadas pelo duplo-clique do *mouse* no grafo genético que descreve as capacidades do domínio. Com isso, destaca-se o aspecto de sobreposição da área exercitada pelo enunciado no grafo. Trata-se, convém evidenciar, das perícias cujo enunciado realmente se presta a contribuir com o desenvolvimento, não daquelas meramente abordadas na resolução.

5.2.2 Inspeção do Catálogo de Enunciados

Cabe ao recurso de inspeção, sintetizar no mesmo grafo genético a sobreposição acumulada de todo o catálogo de enunciados (Figura 5.20). Assim, ao receber o ponteiro do

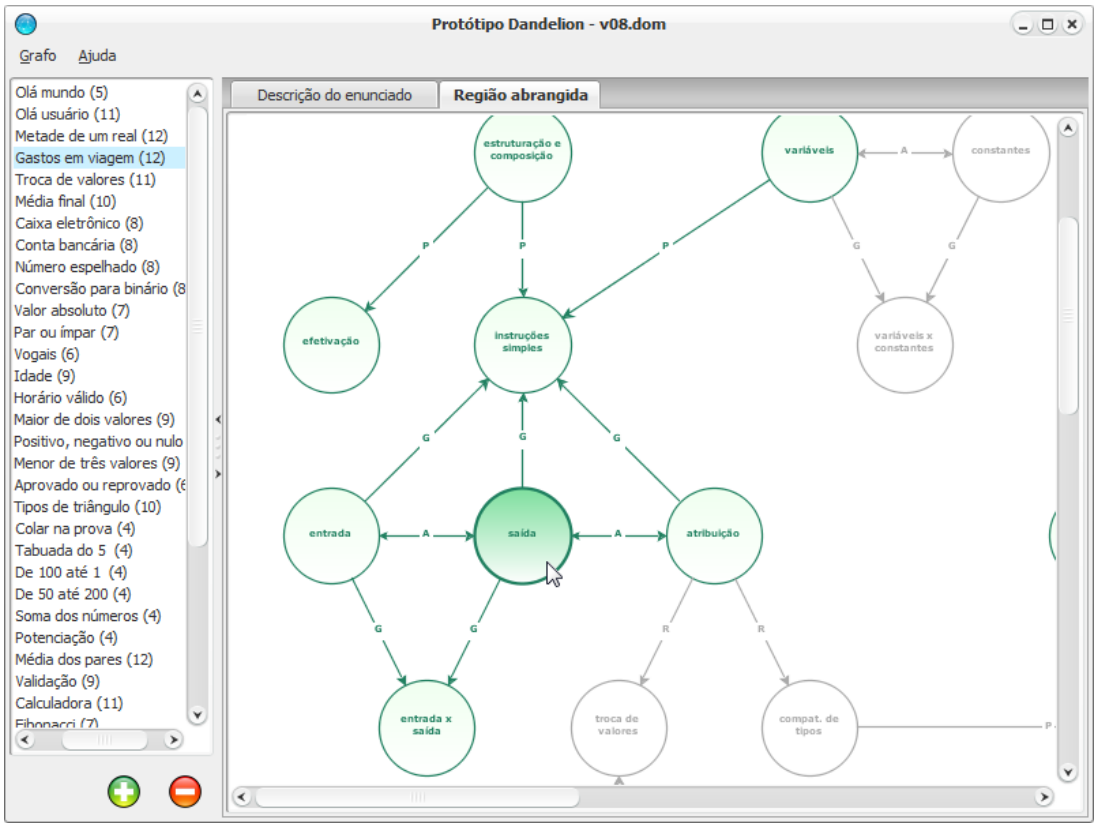


Figura 5.19: Sinalização de perícias abrangidas pelo enunciado

mouse, cada perícia exibe o número de enunciados que a exercitam. Adicionalmente, são sinalizadas em vermelho as perícias não contempladas, ou seja, negligenciadas pelo catálogo.

A inspeção, no catálogo de exemplo, constatou que não há exercícios que incidam sobre as perícias de: *reconhecimento de constantes na análise e na resolução do problema*; *distinção conceitual e pragmática entre variáveis e constantes*; e *encadeamento de estruturas de repetição*. Outro fato constatado foi que a maioria dos enunciados em estruturas de repetição se resume a laços contados e, em sensibilidade à abordagem de ensino utilizada, pode não exercitar os laços condicionais.

5.2.3 Recomendação de Enunciados

Com a intenção de construir metaconhecimento sobre o catálogo de enunciados, foram providos três níveis de recomendação, descritos na continuidade.

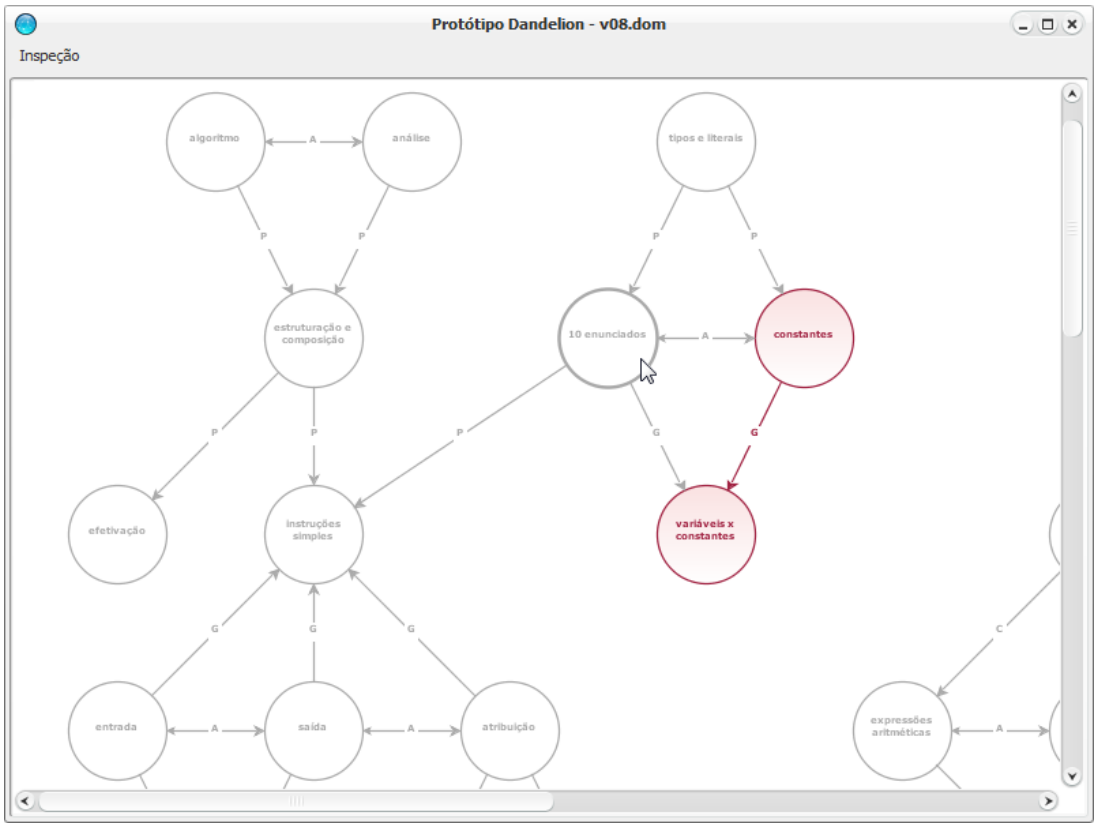


Figura 5.20: Inspeção do Catálogo de Enunciados

5.2.3.1 Baseada na Sobrecarga de Perícias

O primeiro nível de recomendação identifica os enunciados que contemplam mais do que um determinado número de perícias (Figura 5.21). O parâmetro é configurável e sugere a proporção de um terço da quantidade de perícias descritas no domínio. Na sequência, de acordo com a tela mostrada na Figura 5.22, exibe-se a lista dos enunciados que excedem o número de perícias prescrito, justaposta com o aspecto de sobreposição no grafo.

Na perspectiva desta recomendação, considera-se ineficiente um enunciado que procure exercitar muitas perícias e, por isso, sobrecarregue cognitivamente o aprendiz. Indica-se que cada um dos enunciados expostos seja desmembrado em outros menores que, consequentemente, incidam sobre conjuntos reduzidos de perícias.

No catálogo de exemplo, o parâmetro sugerido é de 13 perícias, considerando as 41 perícias da descrição do domínio. Como inexiste, no catálogo, enunciados com tal abrangência, a Figura 5.22 apresenta a recomendação daqueles que superam 11 perícias.

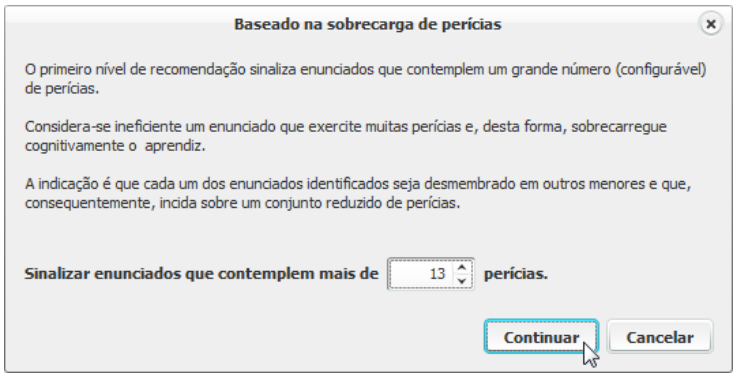


Figura 5.21: Caixa de diálogo da recomendação baseada na sobrecarga de perícias

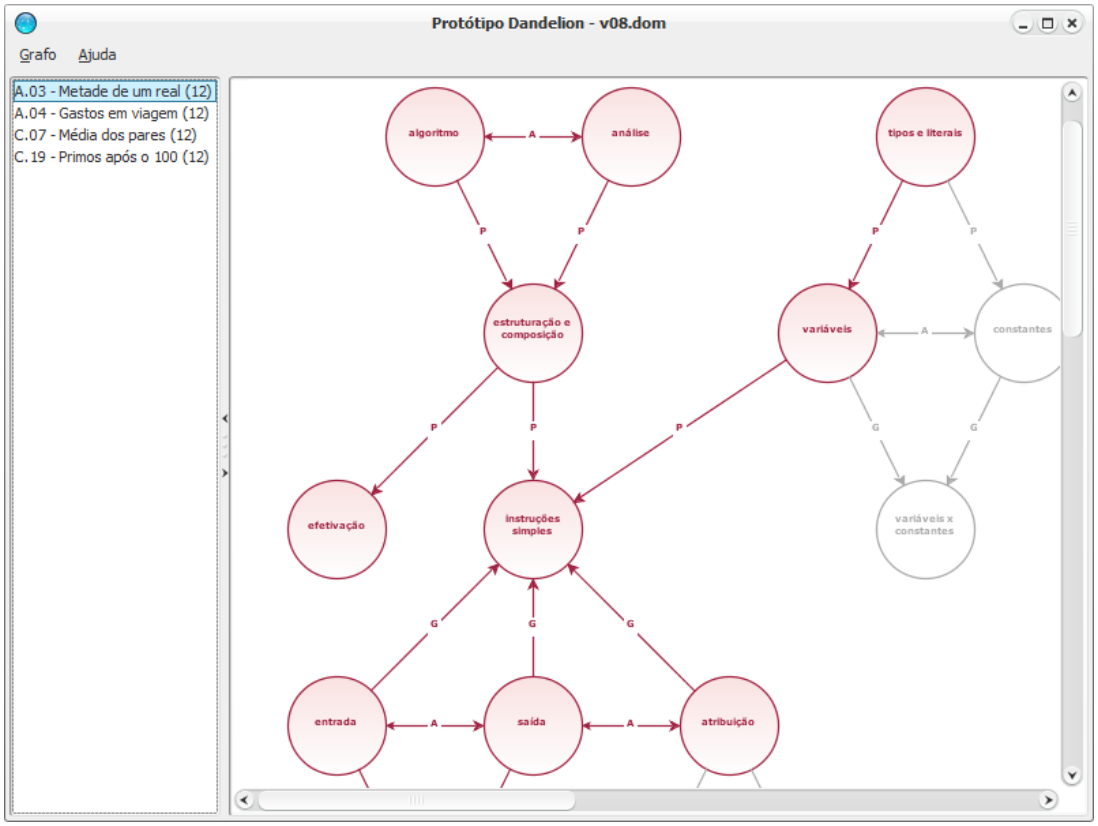


Figura 5.22: Recomendação baseada na sobrecarga de perícias

5.2.3.2 Baseada nas Perícias Não Contempladas

O segundo nível de recomendação (Figura 5.23) indica as perícias ainda não contempladas pelo catálogo, bem como a correspondência com enunciados que exercitam perícias vizinhas. Conforme a tela ilustrada na Figura 5.24, apresenta-se, à esquerda, a lista de perícias que o catálogo não abrange e, logo abaixo, outra lista que correlaciona a perícia selecionada com enunciados que contemplam regiões vizinhas. A sobreposição de cada enunciado que se seleciona e o relacionamento com a perícia pendente são evidenciados no grafo.

O objetivo desta recomendação é incitar a extensão de enunciados específicos (na forma de novos enunciados) para que promovam adicionalmente perícias omissas. Como exemplo, no catálogo instanciado, pode-se propor um enunciado que exercite o *reconhecimento de constantes na análise e na resolução do problema*. Sugere-se, como percebido na Figura 5.24, que o novo enunciado se baseie naquele intitulado *Gastos com viagem*.

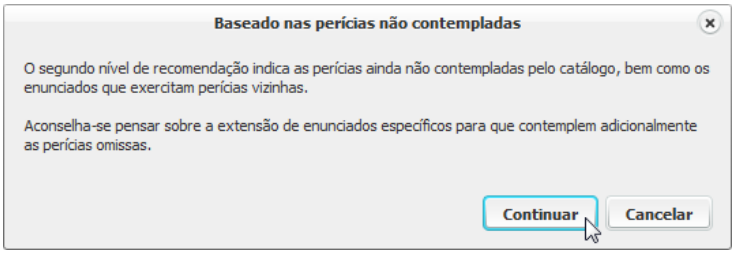


Figura 5.23: Caixa de diálogo da recomendação baseada nas perícias não contempladas

5.2.3.3 Baseada na Distância Entre Regiões de Perícias

O terceiro nível de recomendação detecta enunciados que contemplam regiões de perícias muito distantes no domínio de conhecimento. Sinalizam-se os enunciados que exercitam regiões com distância maior que um determinado número configurável de relacionamentos (ou seja, arestas, como mostra a Figura 5.25). Na tela subsequente, vide Figura 5.26, expõe-se a lista de enunciados e a respectiva sobreposição no grafo.

O metaconhecimento considerado é que perícias distantes potencialmente possuem diferentes níveis de dificuldade. Recomenda-se que os enunciados sejam mais objetivos e

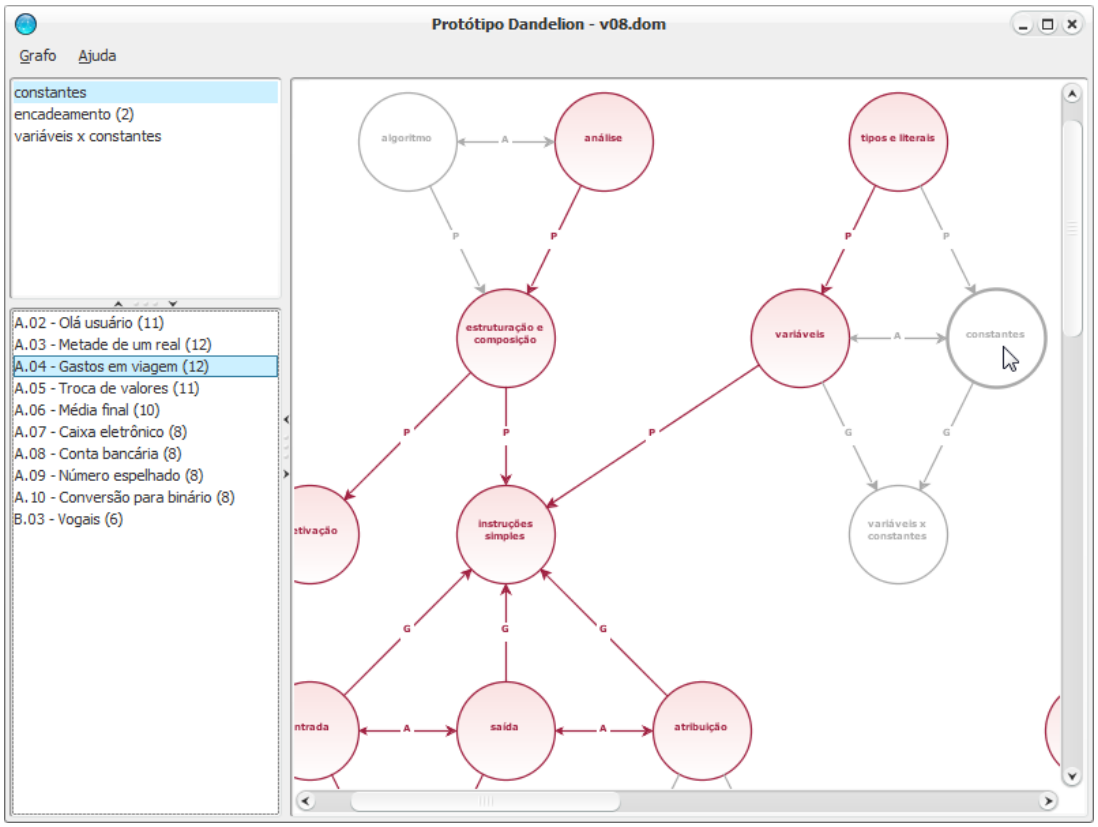


Figura 5.24: Recomendação baseada nas perícias não contempladas

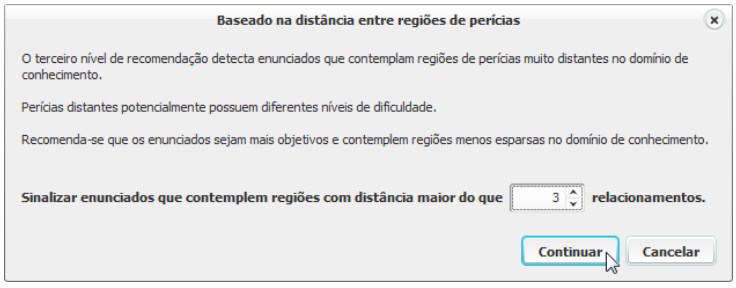


Figura 5.25: Caixa de diálogo da recomendação na distância entre regiões de perícias

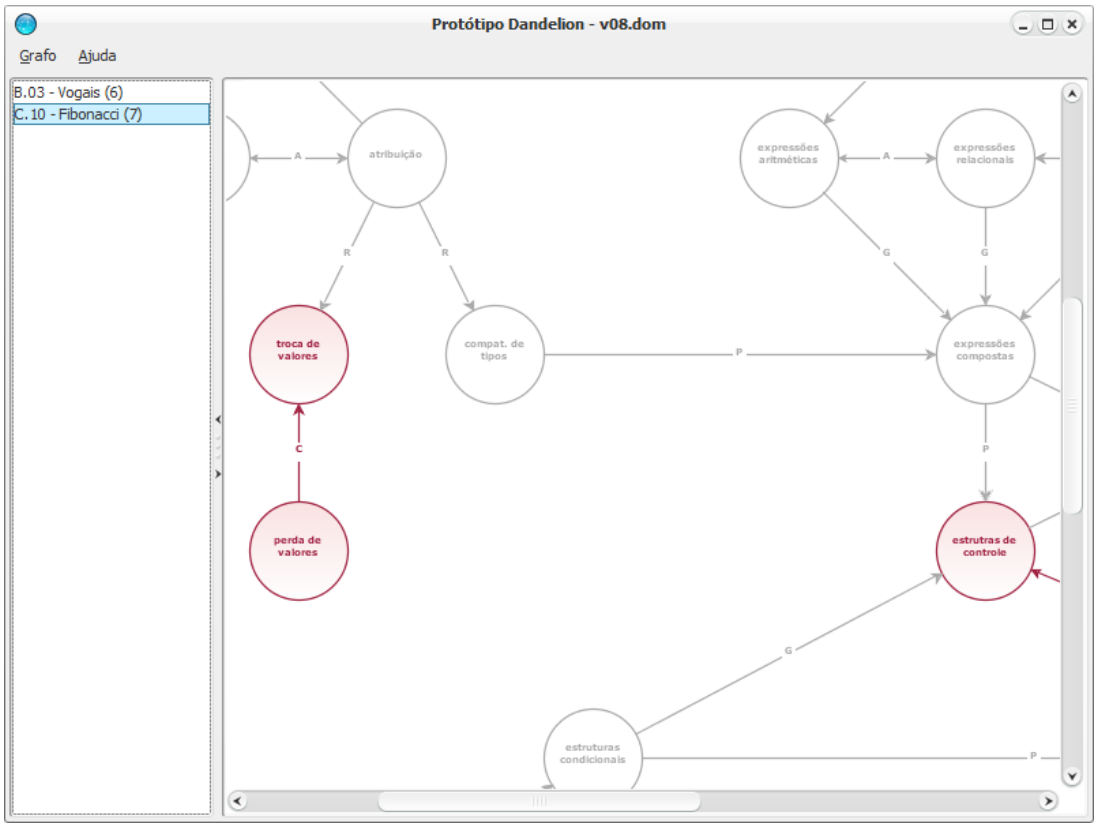


Figura 5.26: Recomendação baseada na distância entre regiões de perícias

contemplem regiões menos esparsas no domínio de conhecimento.

O cálculo das distâncias é realizado mediante a identificação das regiões como componentes conexas dos subgrafos. Em uma estrutura de dados secundária, substitui-se cada componente, por um vértice auxiliar (virtual) que a representa. Depois se calcula a distância entre cada um dos vértices auxiliares, considerando a menor distância de uma região relativa às demais que compõem o enunciado. Foi considerada uma adaptação da busca em amplitude para o cálculo das distâncias [102]. A validade de se registrar a técnica descrita é justificada porquanto a ideia intuitiva de confrontar a distância entre perícias (vértices) isoladas de regiões distintas tem pouca eficiência

Em critério de exemplo, no catálogo citado, o clássico exercício da série de Fibonacci foi proposto para contemplar a troca, e possível perda, de valores em variáveis[6]. Entretanto, embora o enunciado aborde tais perícias, a contribuição específica reside no conteúdo de

[6]Ou seja, as perícias respectivas de *troca de valores entre duas variáveis com o uso de uma auxiliar*, bem como *sobreposição e consequente perda de valores armazenados.*

Estruturas de Controle (Repetição). Assim, considerando que a distância tomada de dois relacionamentos entre as regiões foi excedida, o exercício recomenda atenção autoral.

5.2.4 Funcionalidades Adicionais

As maioria das funcionalidades adicionais do protótipo já foram apresentadas pela Figura 5.17 e são análogas àquelas da primeira ferramenta. Complementarmente, aponta-se que, em qualquer interação com o grafo genético, é disponibilizada uma escala de *zoom* para aprimorar a visualização (acessível pelo clique contrário do *mouse*). Também, todas as divisões entre listas, e destas com o grafo, podem ser redimensionadas.

CAPÍTULO 6

REPERCUSSÕES ESPERADAS

Acredita-se que a corrente pesquisa abra espaço para que sejam retomados os estudos de [32, 92], a fim de revisar e consolidar um elenco de medidas cognitivas no aprendizado de Programação de Computadores. Trata-se de uma linha de pesquisa promissora visto que os resultados obtidos podem potencializar o tempo e o esforço de instrutores e aprendizes na área. A contribuição estende-se também no sentido de propiciar o uso de estratégias pedagógicas mais eficientes, tanto se embutidas em ambientes de aprendizagem quanto se utilizadas em sessões convencionais de ensino em sala de aula.

Igualmente se presume que o refinamento da descrição das capacidades do domínio, assim como da elicitação de enunciados e de outras iniciativas relacionadas pela pesquisa tenham evolução contínua ao longo do tempo. Tão logo, é considerável que o amparo de abordagens colaborativas reforce e acelere o progresso desejado. Com isso, a constante troca e avaliação do (meta)conhecimento construído, por instrutores e aprendizes, tende a elevar a qualidade do ensino no âmbito de Programação de Computadores.

Tais resultados também serão ferramentas ricas para aumentar o grau de abstração dos instrutores no que tange à escolha de enunciados para construir uma sequência de ensino no citado domínio. A seleção de um enunciado em detrimento de outro deixará de ser uma ação exclusivamente empírica. Assim, um instrutor poderá prescrever exercícios com propósitos específicos projetados nos campos de aquisição de princípios e de desenvolvimento de perícias na área. Isso seria uma intensa contraposição à prática corriqueira de delegar exercícios como mero prolongamento de uma sessão de ensino, atitude que visa mais suprir falhas docentes do que cumprir com algum objetivo pedagógico.

Estudos com a mesma perspectiva poderão ainda incrementar a qualidade dos livros didáticos da área, cuja precariedade da maioria dos títulos é destacada por docentes. Nesse sentido, duas falhas relevantes observadas pelo proponente deste projeto, enquanto

professor de disciplinas introdutórias de Programação de Computadores, foram: **(a)** bibliografias que consistem em simples manuais sintáticos das linguagens de programação e que pouco remetem ao desenvolvimento de perícias na área; e **(b)** exercícios descontextualizados, com objetivos vagos e que compõem uma sequência de ensino de baixa coesão diante do conteúdo programático.

Portanto, um conjunto consistente de perícias pode permitir que as explicações abordem habilidades específicas em uma progressão favorável ao aprendizado. Além disso, subsidiaria a composição de enunciados cognitivamente densos, providos de contexto, com objetivos precisos e que perfizessem uma continuidade factível à maioria dos aprendizes.

Isso posto, admite-se a possibilidade de que seja identificado um corpo de perícias comuns a diferentes paradigmas de Programação de Computadores. Antecipa-se ainda que, além da intersecção, sejam reconhecidas subperícias (ou nuances daquelas que perfazem o corpo central) próprias de cada paradigma em particular.

Como investida seguinte, é possível observar e mapear a progressão típica do desenvolvimento de perícias em Programação de Computadores (inicialmente, não se atendo a paradigmas). Adianta-se que as citadas perícias, principalmente o conjunto determinado como sobrejacente de alto nível nesta pesquisa, não se desenvolvem de forma sucessivamente encadeada, na ordem de uma perícia após a outra. O incitamento de cada uma dá-se de maneira intercalada (e, às vezes, paralelo) com o de outras, à proporção da prática do aprendiz.

Esforços consequentes podem se concentrar no detalhamento dos picos de incitação de perícias no decorrer da aprendizagem. Abre-se, assim, espaço para que metodologias apropriadas privilegiem o ensino de perícias manifestadas em tais momentos específicos.

Outro resultado praticável consiste em demarcar a composição de um elenco de exercícios canônicos que instancie uma progressão típica do desenvolvimento de perícias em cada paradigma de programação abordado. Tais catálogos de exercícios teriam utilidade como modelos (*templates*) que aceitassem ajustes, feitos sob o gabarito das perícias mapeadas, com o propósito de melhor atender ao perfil de (grupos de) aprendizes específicos.

Embora todas as possibilidades (e oportunidades) relacionadas tiveram a área de Pro-

gramação de Computadores como instância, a maioria pode ser remetida a outros domínios de natureza prática e complexa. Idealmente, sobre a mesma expectativa, um portal que se convergisse como repositório de tamanho (meta)conhecimento epistemológico consolidado, em uma variedade de domínios, pode constituir uma linha de projeto para ambientes de ensino inteligentes promissora.

CAPÍTULO 7

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Propôs-se, diante da hipótese assumida para a pesquisa, a investigação sobre como ocorre o desenvolvimento de perícias em Programação de Computadores. A intenção consequente residiu em elevar os resultados obtidos à generalização para outros domínios de natureza prática e complexa. Nesse contexto, o trabalho teve como objetivo explorar e construir modelos que atuem na especificação do processo de desenvolvimento pericial em Sistemas Tutores Inteligentes.

Em particular, enfocou-se a revisão bibliográfica em Múltiplas Representações Externas, abordagens sobre capacidades da perícia, ambientes de apoio ao ensino e à aprendizagem de Programação de Computadores, bem como modelagem de aprendizes em Sistemas Tutores Inteligentes. Conforme o que se destacou por meio da resenha literária, percebia-se carência de pesquisas recentes que tratassem do desenvolvimento de perícias em Programação de Computadores e, principalmente, sob perspectiva epistemológica. Ademais, a elicitação de um catálogo de enunciados frente a uma descrição detalhada do domínio também não havia sido promovida.

Em contexto mais amplo, tratando-se de tutoria inteligente, o problema de prover uma epistemologia para a descrição de conhecimento sobre enunciados e estágios de conhecimento de aprendizes foi negligenciado até então. Ao encontro disso, o experimento realizado com o ambiente MÚLTIPLA, na presente pesquisa, atestou que o formalismo subjacente potencializa o estudo na área e mostra-se benéfico por empregar múltiplas representações externas no suporte a uma abordagem metacognitiva de ensino. Em consonância, investiu-se na extensão do formalismo, e consequentes benefícios, para o processo de autoria. Houve especial enfoque na influência de representações adequadas para a elicitação e catalogamento de enunciados em domínios de natureza prática.

Como formalismo adicional proposto para a validação da hipótese desta pesquisa, foi

apresentado um arcabouço conceitual delineado diante de grafos genéticos e de um modelo de sobreposição para Programação de Computadores. Em paralelo, sinalizaram-se muitas das expectativas depositadas na contribuição disso para com o estado da arte em foco.

O processo de autoria, de acordo com o formalismo expresso, foi suportado pela prototipagem de duas ferramentas, sendo uma destinada à definição do conhecimento do domínio por meio do detalhamento em grafo genético das perícias componentes e a outra, usando a mesma representação gráfica, à elicitação de enunciados descritos como subgrafos daquele modelado pela primeira. A avaliação do método e das ferramentas propostos foi realizada por estudos empíricos focados na generalidade dos grafos genéticos como linguagem de autoria com o objetivo de prover um arcabouço unificado para o desenvolvimento de perícias.

Diante dos resultados positivos no cumprimento dos objetivos propostos, há evidências suficientes para validar a hipótese assumida. Em adição, algumas possibilidades descritas pelo documento foram retomadas e acrescidas no traçado das repercussões esperadas da pesquisa desenvolvida. Acredita-se que, com isso, frente às lacunas constatadas na interseção de áreas apresentadas, tenha-se atingido uma contribuição relevante para a Ciência da Computação.

7.1 Outros Resultados Alcançados

Em alinhamento aos objetivos expostos na Seção 1.4, faz-se agora uma sumarização dos resultados secundários alcançados pela pesquisa, a saber:

1. Estabelecimento de um teste para avaliação de habilidades pré-algorítmicas (detalhado na Seção 3.1.4.1 e incluído no Anexo A);

2. Definição de um subconjunto das capacidades do domínio de Programação de Computadores, sendo, mais precisamente, uma parcela das perícias que compõem o estereótipo de um programador experto (vide Seção 4.2.1 e Anexo F);

3. Modelagem, na representação de grafo genético, do subconjunto de perícias recém-mencionado (Anexo F);

4. Validação de modelos, representados em grafos genéticos, quanto à alcançabilidade das demais perícias a partir da inicial (Seção 5.1);

5. Revisão do conjunto das características de um programador perito estabelecido por [92]. Tais perícias foram, então, denominadas sobrejacentes de alto nível. Sete perícias adicionais foram identificadas (Seção 4.2.4);

6. Extensão do grafo genético pela perspectiva de empregar a sobreposição do Modelo do Domínio também como gabarito para elicitar e catalogar enunciados (Seção 4.2);

7. Requisição inteligente da autoria de enunciados, oferecida em três níveis, baseada: **(a)** na sobrecarga de perícias; **(b)** nas perícias não contempladas e **(c)** na distância entre regiões de perícias (Seção 5.2);

8. Projeção da possibilidade de sintetizar o Modelo do Aprendiz de um grupo (sala de aula) em uma única sobreposição, a fim de representar o estágio de conhecimento dos indivíduos integrantes em um dado momento (Seção 4.2);

9. Proposta de se representar externamente o desenvolvimento gradativo das perícias, pelo aprendiz, através de uma escala visual de progressão. Intenciona-se denotar o percentual cumprido de cada perícia pela intensificação da cor do vértice correspondente (Seção 4.2);

10. Associação das duas últimas representações citadas para compor uma terceira, dinâmica, que remete ao progresso ou histórico de desenvolvimento pericial de um aprendiz (ou grupo) em determinado período (Seção 4.2).

7.2 Trabalhos Futuros

Concerne ao final deste capítulo, apresentar direções para trabalhos futuros que venham a estender e suprir limitações da pesquisa realizada.

7.2.1 Avaliar Formalmente o Processo de Autoria

Embora tenha sido feita uma avaliação formal do ambiente MÚLTIPLA, houve apenas observações empíricas sobre o processo de autoria suportado pelo mesmo formalismo. A partir de uma avaliação formal aplicada ao processo de autoria, algumas potencialidades do arcabouço proposto podem ser aprimoradas. Além disso, é conveniente a validação de duas hipóteses, a saber:

1. O grafo genético tem caráter geral suficiente, como linguagem, para o desenvolvimento de uma ampla gama de ambientes destinados ao processo de ensino e aprendizagem;

2. As ferramentas interativas que suportam o processo de autoria mostram-se apropriadas no sentido de proporcionar maior rapidez e de também prover um arcabouço unificado para domínios de natureza prática e complexa.

7.2.2 Implementação da Modelagem Dinâmica do Aprendiz

Diante do modelo de sobreposição apresentado como formalismo proposto para a solução do problema, como se enfocou o processo de autoria, não foram implementadas ferramentas que abrangessem o progresso do aprendiz frente ao conhecimento do domínio. Conforme descrito na Seção 4.2, pode-se implementar:

1. A modelagem dinâmica do conhecimento do aprendiz como um subconjunto das capacidades do domínio descritas;

2. Um processo de busca heurística que oriente regiões do grafo a serem exploradas pelo aprendiz;

3. A alimentação do Modelo do Aprendiz conforme a avaliação de desempenho nas soluções dos enunciados propostos;

4. A sintetização do Modelo do Aprendiz de um grupo (sala de aula) para que se tenha um espelho da ênfase com que as perícias são ensinadas pelo instrutor;

5. O fornecimento de detalhes da dimensão temporal de progresso do aprendiz (ou da sala de aula). Isto é, a apresentação de um histórico de desenvolvimento do aprendiz.

7.2.3 Adição de Inteligência Artificial ao Ambiente MÚLTIPLA

Conforme anteriormente mencionado, a Programação de Computadores demanda alta carga cognitiva para iniciantes, devido à falta de conhecimento de princípios e à perícia pouco desenvolvida. Os ambientes destinados a iniciantes de programação, segundo [92], procuram lidar com essas carências por meio da combinação de **(1)** técnicas de Inteligência Artificial, **(2)** ambientação construtivista por exploração livre e **(3)** recursos de visualização científica para, no final, criar interfaces de razoável versatilidade para o aprendiz.

O MÚLTIPLA, nesses aspectos, trabalha de forma intensa para propiciar um ambiente semanticamente rico para exploração, bem como mecanismos diversos para visualização de informações. Contudo, há necessidade de se investir em técnicas de Inteligência Artificial que propiciem certo grau de automatização na progressão de enunciados que devam ser explorados.

Também há interesse em prover o MÚLTIPLA de uma galeria de enunciados catalogados diante da perspectiva das capacidades do domínio definidas. Assim, com base no conhecimento sobre o aprendiz, podem ser utilizadas estratégias pedagógicas eficientes e personalizadas para a proposição de enunciados que perfaçam uma sequência favorável a um perfil específico.

Nesse espaço, outra perspectiva que merece ser documentada diz respeito aos aprendizes de desempenho acima da média. Geralmente estudos do gênero enfocam perfis de aprendizes que possuem dificuldades diversas na área de Programação de Computadores. No entanto, descuidam-se absurdamente daqueles que apresentam facilidade de aprendizado e grande potencial a ser desenvolvido. Logo, não raro excelentes perfis são desmotivados por progressões de ensino enfadonhas que subutilizam a capacidade dos aprendizes.

Portanto, com o emprego de Inteligência Artificial para guiar o sequenciamento de

tais enunciados, acredita-se que a atenção dispensada aos aprendizes seja mais uniforme. Além disso, essa funcionalidade deve impactar em fatores motivacionais que impulsionem a dedicação do aprendiz para investir em tempo de estudo extra-classe.

BIBLIOGRAFIA

[1] Sam A. Abolrous. *Learn Pascal in Three Days*. Wordware Publishing, 3 ed., 2001.

[2] Anne Adam e Jean-Pierre H. Laurent. LAURA, a system to debug student programs. *Artificial Intelligence*, páginas 75–122, 1980.

[3] Francisco Meira Aguiar, Alexandre Ibrahin Direne, Luis de Bona, Fabiano Silva, André Guedes, Marcos Castilho, Marcos Sunyé, e Laura García. Ferramentas e métodos para apoiar o ensino de Xadrez na fronteira entre os fundamentos e a perícia. *Anais do XXVII Congresso da SBC. XIII Workshop de Informática na Escola - WIE 2007*, 13:380–387, 2007.

[4] A.V. Aho, J.D. Ullman, e R. Sethi. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley series in Computer Science. Pearson/Addison Wesley, 2007.

[5] Shaaron Ainsworth. A functional taxonomy of multiple representations. *Computers and Education*, 33(2/3):131–152, 1999.

[6] Shaaron Ainsworth. DeFT: A conceptual framework for considering learning with multiple representations. *Learning and Instruction*, 2006. (Em publicação).

[7] Shaaron Ainsworth e Nicolas Van Labeke. Using a multi-representational design framework to develop and evaluate a dynamic simulation environment. Dynamic Information and Visualisation Workshop, julho de 2002.

[8] John R. Anderson, Robert Farrell, e Ron Sauers. Learning to program in LISP. *Cognitive Science*, 8:87–129, 1984.

[9] Lorin W. Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Raths, e Merlin C. Wittrock. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Completed Edition*. Allyn & Bacon, 2001.

[10] David J. Barnes e Michael Kölling. *Objects First with Java: A Practical Introduction using BlueJ*. Pearson Education, Edinburgh Gate, UK, 2003.

[11] Shawkat Bhuiyan, Jim E. Greer, Gordon I. Mccalla, e Jim Greer. Supporting the learning of recursive problem solving. *Interactive Learning Environments*, 4(2):115–139, 1994.

[12] A. Brown. *Metacognition, Motivation, and Understanding*, capítulo Metacognition, executive control, self-regulation, and other more mysterious mechanisms, páginas 65–116. LEA, Hillsdale, NJ, 1987.

[13] J.S. Brown, R. Burton, e F. Zdybel. A model-driven question-answering system for mixed-initiative computer-assisted instruction. *IEEE Transactions on Systems, Man, and Cybernetics*, volume SMC-3, páginas 248–257. Institute of Electrical and Electronics Engineers, maio de 1973.

[14] Marc H. Brown. Exploring algorithms using BALSA-II. *Computer*, 21:14–36, maio de 1988.

[15] Marc H. Brown e Robert Sedgewick. A system for algorithm animation. *SIGGRAPH Comput. Graph.*, 18:177–186, janeiro de 1984.

[16] Marc H. Brown e Robert Sedgewick. Techniques for algorithm animation. *IEEE Softw.*, 2:28–39, janeiro de 1985.

[17] R. Brown. Use of analogy to achieve new expertise. MIT AI Technical Report 403, Massachusetts Institute of Technology, abril de 1977.

[18] R. Burton e J.S. Brown. A tutoring and student modelling paradigm for gaming environments. R. Coleman e Jr. P. Lorton, editores, *Computer Science and Education*, volume 8 of *ACM SIGCSE Bulletin*, páginas 236–247, 1976.

[19] R. Burton e J.S. Brown. Semantic grammar: A techinique for constructing natural language interfaces to instructional systems. BBN Report 3857 (ICAI Report 5), Bolt, Beranek and Newman Inc., maio de 1977.

[20] Rogers Cadenhead. *Sams Teach Yourself Java in 24 Hours - Covering Java 7 and Android*. Sams, 6 ed., 2011.

[21] J. Carbonell. AI in CAI: An artificial-intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, volume MMS-11. Institute of Electrical and Electronics Engineers, dezembro de 1970.

[22] B. Carr e Ira P. Goldstein. Overlays: A theory of modelling for computer aided instruction. MIT AI Memo 406 (LOGO Memo 40), Massachusetts Institute of Technology, fevereiro de 1977.

[23] B. Carr e Ira P. Goldstein. Wusor II: A computer aided instruction program with student modelling capabilities. MIT AI Memo 417 (LOGO Memo 45), Massachusetts Institute of Technology, maio de 1977.

[24] Jennifer Case e Richard Gunstone. Metacognitive development as a shift in approach to learning: an in-depth study. *Studies in Higher Education*, 27(4):459–470, 2002.

[25] A. Collins, E. Warnock, e J. Passafiume. *The Psycology of Learning and Motivation*, volume 9, capítulo Analysis and Synthesis of Tutorial Dialogues. Academic Press, New York, NY, 1975.

[26] Stephen Cooper, Wanda Dann, e Randy Pausch. Alice: a 3-D tool for introductory programming concepts. *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges*, CCSC '00, páginas 107–116, USA, 2000. Consortium for Computing Sciences in Colleges.

[27] Flávio R. S. Coutinho, Jussara Almeida, Raquel O. Prates, e Luiz Chaimowicz. Belesminha: Um jogo educacional para apoio ao aprendizado de recursividade. Sociedade Brasileira de Computação (SBC), editor, *Proceedings of SBGames'08: Game & Culture Track*, páginas 171–175, Belo Horizonte, Minas Gerais, novembro de 2008.

[28] Richard Cox e Paul Brna. Supporting the use of external representations in problem solving: the need for flexible learning environments. *Journal of Artificial Intelligence in Education*, 6(2/3):239–302, 1995.

[29] Andreia de Jesus. Sistemas Tutores Inteligentes uma visão geral. *Revista Eletrônica de Sistemas de Informação*, 2(2), 2003.

[30] Elieser Ademir de Jesus e André Luis Alice Raabe. Interpretações da TAXONOMIA de Bloom no contexto da Programação Introdutória. *Anais do XX Simpósio Brasileiro de Informática na Educação. SBIE 2009*, 20, 2009.

[31] Alexandre Ibrahim Direne. Methodology and tools for designing concept tutoring systems. P. Brna, S. Ohlsson, e H. Pain, editores, *Proceedings of World Conference on Artificial Intelligence in Education (AI-ED 93)*, páginas 58–65, Edinburgh, Scotland, 1993. Association for the Advancement of Computing in Education (AACE).

[32] Alexandre Ibrahim Direne. Intelligent training shells for the operation of digital telephony stations. B. du Boulay e R. Mizoguchi, editores, *Frontiers in Artificial Intelligence and Applications: Proceedings of World Conference on Artificial Intelligence in Education (AI-ED 97)*, páginas 71–78. IOS Press, 1997.

[33] Alexandre Ibrahim Direne e Dona Scott. Identifying the component features of expertise in domains of complex visual recognition. Information Technology Research Institute Technical Report Series, University of Brighton, Lewes Road, Brighton, UK, 2001.

[34] Gabriel dos Santos. Autoria e interpretação tutorial de soluções alternativas para promover o ensino de Programação de Computadores. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Paraná, Brasil, 2003. Alexandre Ibrahim Direne (orientador).

[35] Bennedict du Boulay. *Artificial Intelligence Tools in Education*, capítulo Intelligent Systems for Teaching Programming, páginas 5–15. Elsevier Science, 1988.

[36] Marc Eisenstadt e Mike Brayshaw. Graphical debugging with the Transparent Prolog Machine (TPM). *Proceedings of the 10th international joint conference on Artificial Intelligence - Volume 1*, páginas 83–86, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[37] Marc Eisenstadt e Mike Brayshaw. The Transparent Prolog Machine TPM: An execution model and graphical debugger for logic programming. *The Journal of Logic Programming*, 5(4):277–342, 1988.

[38] Max D. Engelhart, Edward J. Furst, Walker H. Hill, e David R. Krathwohl. *Taxonomy of educational objectives. The classification of educational goals. Handbook 1: Cognitive domain, by a Committee of College and University Examiners*. Longmans Green, 1956.

[39] K. Anders Ericsson, Ralf Th. Krampe, e Clemens Tesch-Romer. The role of deliberate practice in the acquisiton of expert performance. American Psychological Association, editor, *Psycological Review*, volume 100, páginas 363–406, 1993.

[40] T. Evans. *Semantic Information Processing*, capítulo A Program for the Solution of Geometry-Analogy Intelligence Test Questions, páginas 271–353. The MIT Press, Cambridge, MA, 1968.

[41] Márcia Valéria Rodrigues Ferreira. Estudo empírico e aspectos pré-computacionais para a detecção automática de capacidades da perícia em aprendizes de Xadrez. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Paraná, Brasil, 2007. Alexandre Ibrahim Direne (orientador).

[42] Ana Paula Mendes Correia Couceiro Figueira. Metacognição e seus contornos. *Revista Iberoamericana de Educación*, junho de 2003. http://www.campus-oei.org/revista/deloslectores/446Couceiro.pdf.

[43] Sally Fincher, Stephen Cooper, Michael Kölling, e John Maloney. Comparing Alice, Greenfoot & Scratch. *SIGCSE '10: Proceedings of the 41st ACM Technical Sympo-*

sium on Computer Science Education, páginas 192–193, New York, NY, USA, 2010. ACM.

[44] J. H. Flavell. *The nature of intelligence*, capítulo Metacognitive aspects of problem solving, páginas 231–235. Lawrence Erlbaum Associates, Hillsdale, N.Y., 1976.

[45] Mary Forehand. *Emerging Perspectives on Learning, Teaching, and Technology*, capítulo Bloom's Taxonomy: Original and Revised. Association for Educational Communications and Technology, Athens, Georgia, EUA, 2005.

[46] Ursula Fuller, Colin G. Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L. Lewis, Donna McGee Thompson, Charles Riedesel, e Errol Thompson. Developing a Computer Science-specific learning taxonomy. *Working group reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '07, páginas 152–170, New York, NY, USA, 2007. ACM.

[47] Claudia Gama. Metacognition and reflection in ITS: increasing awareness to improve learning. *Proceedings of World Conference on Artificial Intelligence in Education (AI-ED 2001)*, páginas 492–495, San Antonio, Texas, EUA, maio de 2001. IOS Press.

[48] Lucia Maria Martins Giraffa. Fundamentos de sistemas tutores inteligentes. Relatório técnico, Pontifícia Universidade Católica do Rio Grande do Sul, Pós-Graduação em Ciência da Computação. Inteligência Artificial Aplicada à Educação.

[49] Malcolm Gladwell. *Outliers: The Story of Success*. Little, Brown and Company, New York, 1 ed., 2008.

[50] Ira P. Goldstein. Summary of MYCROFT: A system for understanding simple picture programs. *Artificial Intelligence*, 6(3):249–288, 1975.

[51] Ira P. Goldstein. The genetic graph: a representation for the evolution of procedural knowledge. *International Journal of Man-Machine Studies*, 11(1):51–77, janeiro de 1979.

[52] Ira P. Goldstein e B. Carr. The computer as coach: An athletic paradigm of intellectual education. *Proceedings of* 1977 *Annual Conference*, páginas 227–233, 1977.

[53] Daniel C. Halbert. *Watch what I do: programming by demonstration*, capítulo SmallStar: Programming by Demonstration in the Desktop Metaphor. MIT Press, Cambridge, MA, USA, 1993.

[54] T. Hasemer, Open University. Human Cognition Research Laboratory, e University of Open. *An empirically-based debugging system for novice programmers*. HCRL Technical report. n.p., 1983.

[55] Allan Heydon e Greg Nelson. The Juno-2 constraint-based drawing editor. *Technical Report 131a, Digital Systems Research*, 1994.

[56] Colin G. Johnson e Ursula Fuller. Is Bloom's Taxonomy appropriate for Computer Science? *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, Baltic Sea '06, páginas 120–123, New York, NY, USA, 2006. ACM.

[57] W. L. Johnson. Understanding and debugging novice programs. *Artificial Intelligence*, 42(1):51–97, fevereiro de 1990.

[58] Bradley L. Jones e Peter Aitken. *Sams Teach Yourself C in 21 Days*. Sams, 6 ed., 2002.

[59] K. Larkin J.S. Brown, R. Burton. Representing and using procedural bugs for educational purposes. *Proceedings of* 1977 *Annual Conference*, páginas 247–255, 1977.

[60] Johan De Kleer. Local methods for localizing faults in electronic circuits. MIT AI Memo 394, Massachusetts Institute of Technology, novembro de 1976.

[61] Michael Kölling. The Greenfoot programming environment. *Trans. Comput. Educ.*, 10:1–21, novembro de 2010.

[62] David R. Krathwohl. A revision of Bloom's Taxonomy: an overview. *Theory Into Practice*, 41(4):212–218, Autumn de 2002.

[63] Jean Lave. *Cognition in Practice: Mind, Mathematics and Culture in Everyday Life*. Cambridge University Press, New Cambridge, 1988.

[64] A. Lesgold. *Toward a theory of curriculum for use in designing intelligent instructional systems*, páginas 114–137. Springer-Verlag New York, Inc., New York, NY, USA, 1988.

[65] A. Lesgold, H. Rubinson, P. Feltovitch, R. Glaser, D. Klopfer, e Y. Wang. *The nature of expertise*, capítulo Expertise in a Complex Skill: Diagnosing X-ray Pictures, páginas 311–342. Lawrence Erlbaum Associates, Inc, Hillsdale, NJ, England, 1988.

[66] Henry Lieberman. Tinker, a programming by demonstration system for Lisp. http://web.media.mit.edu/~lieber/Lieberary/Tinker/Tinker.html.

[67] Henry Lieberman. Seeing what your programs are doing. *International Journal Of Man-Machine Studies*, 21(4):311–331, 1984.

[68] Henry Lieberman. *Your wish is my command: programming by example*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[69] Henry Lieberman. *HCI Remixed*, capítulo A Creative Programming Environment, páginas 37–42. MIT Press, Cambridge, MA, USA, 2008.

[70] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, e Lynda Thomas. A multi-national study of reading and tracing skills in novice programmers. *Working group reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '04, páginas 119–150, New York, NY, USA, 2004. ACM.

[71] F. J. Lukey. Understanding and debugging programs. *International Journal on Man Machine Studies*, 12(2):189–202, 1980.

[72] John H. Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, e Natalie Rusk. Programming by choice: urban youth learning programming with Scratch. *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, SIGCSE '08, páginas 367–371, New York, NY, USA, 2008. ACM.

[73] Daniel Martineschen. Alternância entre competição e colaboração para promover o aprendizado por meio de heurísticas de jogos. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Paraná, Brasil, 2006. Alexandre Ibrahim Direne (orientador).

[74] Eleandro Maschio. Abordagem metacognitiva através de múltiplas representações externas para o ensino de Programação de Computadores. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Paraná, Brasil, abril de 2007. Alexandre Ibrahim Direne (orientador).

[75] Eleandro Maschio. Coleta e análise de resultados a partir do uso inicial do ambiente MÚLTIPLA. Relatório técnico final de pesquisa, Universidade Estadual do Centro-Oeste, Guarapuava, Paraná, abril de 2009.

[76] Eleandro Maschio. Medidas cognitivas no aprendizado de Programação de Computadores. Relatório técnico final de pesquisa, Universidade Estadual do Centro-Oeste, Guarapuava, Paraná, março de 2011.

[77] Eleandro Maschio. Medidas cognitivas no aprendizado de Programação de Computadores. *Anais da II SIEPE - Semana de Integração Ensino, Pesquisa e Extensão*, Guarapuava, Paraná, setembro de 2011. Universidade Estadual do Centro-Oeste, Editora Unicentro.

[78] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, volume SE-2, páginas 308–320. Institute of Electrical and Electronics Engineers, dezembro de 1976.

[79] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, e Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *Working group reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '01, páginas 125–180, New York, NY, USA, 2001. ACM.

[80] J. Moore e A. Newell. *Knowledge and Cognition*, capítulo How Can MERLIN Understand? Lawrence Erlbaum Associates, Potomac, MD, 1973.

[81] Paul Mulholland e Marc Eisenstadt. *Using Software to Teach Computer Programming: Past, Present and Future*, capítulo 26, páginas 399–408. MIT Press, 1998.

[82] Tom Murray, Larry Winship, Roger Bellin, e Matt Cornell. Toward glass box educational simulations: reifying models for inspection and design. *External Representations in AIED: Multiple Forms and Multiple Roles - AI-ED 2001 Workshop*, San Antonio, Texas, EUA, maio de 2001. IOS Press.

[83] Greg Nelson. Juno, a constraint-based graphics system. *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, páginas 235–243, New York, NY, USA, 1985. ACM.

[84] Peter Norvig. Teach yourself programming in ten years. http://norvig.com/21-days.html, 2001.

[85] Hyacinth S. Nwana. Intelligent tutoring systems: an overview. *Artificial Intelligence Review*, (4):251–277, 1990.

[86] S. E. Palmer. *Cognition and categorization*, capítulo Fundamental aspects of cognitive representation, páginas 259–303. LEA, 1978.

[87] Seymour Papert. *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc., New York, NY, USA, 1980.

[88] Randy Pausch. Alice: a dying man's passion. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, páginas 1–1, New York, NY, USA, 2008. ACM.

[89] Fillipi Domingos Pelz. Correção automática de algoritmos no ensino introdutório de programação. Dissertação de Mestrado, Universidade do Vale do Itajaí, Itajaí, Santa Catarina, Brasil, junho de 2011. André Luis Alice Raabe (orientador).

[90] Jean Piaget. *Epistemologia Genética*. Martins Fontes, 2002.

[91] Andrey Ricardo Pimentel. Medidas cognitivas para o ensino de conceitos visuais com Sistemas Tutores Inteligentes. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba, Paraná, Brasil, 1997. Alexandre Ibrahim Direne (orientador).

[92] Andrey Ricardo Pimentel e Alexandre Ibrahim Direne. Medidas cognitivas no ensino de programação de computadores com Sistemas Tutores Inteligentes. *Revista Brasileira de Informática na Educação (IE)*, 3:17–24, 1998.

[93] André Luís Alice Raabe. *Uma proposta de arquitetura de sistema tutor inteligente baseada na teoria das experiências de aprendizagem mediadas*. Tese de doutorado, Universidade Federal do Rio Grande do Sul, Porto Alegre, dezembro de 2005.

[94] Warren Rachele. *Learn C++ in Three Days*. Wordware Publishing, 2001.

[95] B. J. Reiser, P. Friedmann, J. Gevins, D. Y. Kimberg, e M. Ranney. A graphical programming language interface for an intelligent LISP tutor. *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '88, páginas 39–44, New York, NY, USA, 1988. ACM.

[96] Steven P. Reiss. Graphical program development with PECAN program development systems. *Proceedings Of The First ACM SIGSOFT/SIGPLAN Software Engineering Symposium On Practical Software Development Environments*, SDE 1, páginas 30–41, New York, NY, USA, 1984. ACM.

[97] Steven P. Reiss. PECAN: Program development systems that support multiple views. *Proceedings Of The 7th International Conference On Software Engineering*, ICSE '84, páginas 324–333, Piscataway, NJ, USA, 1984. IEEE Press.

[98] Pablo Romero, Richard Cox, Benedict du Boulay, e Rudi Lutz. A survey of external representations employed in object-oriented programming environments. *Journal of Visual Languages and Computing*, 14(5):387–419, outubro de 2003.

[99] Stuart J. Russell e Peter Norvig. *Inteligência Artificial: Uma Abordagem Moderna*. Campus, Rio de Janeiro, RJ, 2004.

[100] E. Sacerdoti. The nonlinear nature of plans. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, páginas 206–218, Tbilisi, Georgia, USSR, 1975.

[101] Robert W. Sebesta. *Conceitos de Linguagens de Programação*. Bookman, 5 ed., 2003.

[102] Robert Sedgewick e Kevin Wayne. *Algorithms*. Addison-Wesley, Boston, MA, 4 ed., 2011.

[103] John Self. Student models: what use are they? Paolo Ercoli e Robert Lewis, editores, *Artificial intelligence tools in education*, páginas 73–96. Elsevier Science, Amsterdam, North Holland, 1988.

[104] Ehud Y. Shapiro. *Algorithmic Program DeBugging*. MIT Press, Cambridge, MA, USA, 1983.

[105] D. Sleeman. A problem-solving monitor for a deductive reasoning task. *International Journal of Man-Machine Studies*, volume 7, páginas 183–211, 1975.

[106] David Canfield Smith. *Pygmalion: a creative programming environment*. Tese de Doutorado, Stanford, CA, USA, 1975. AAI7525608.

[107] David Canfield Smith. *Pygmalion: an executable electronic blackboard*, páginas 19–48. MIT Press, Cambridge, MA, USA, 1993.

[108] R.L. Smith, H. Graves, L.H. Blaine, e V.G. Marinov. *Computers in Education Part I*, capítulo Computer-Assisted Axiomatic Mathematics: Informal Rigor. IFIP, Amsterdan, North Holland, 1975.

[109] T. Del Soldado e Bennedict du Boulay. Implementation of motivational tactics in tutoring systems. *Journal of Artificial Intelligence in Education*, 6(4):337–378, 1995.

[110] Errol Thompson, Andrew Luxton-Reilly, Jacqueline L. Whalley, Minjie Hu, e Phil Robbins. Bloom's Taxonomy for CS assessment. *Proceedings of The Tenth conference on Australasian Computing Education - Volume 78*, ACE '08, páginas 155–161, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[111] Richard Wagstaff. *Python In A Day - Learn The Basics, Learn It Quick, Start Coding Fast.* CreateSpace Independent Publishing Platform, 2013.

[112] Etienne Wenger. *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge.* Morgan Kaufmann, 1987.

[113] K. Wescourt, M. Beard, e L. Gould. Knowledge-based adaptive curriculum sequencing for CAI: Application of a network representation. *Proceedings of* 1977 *Annual Conference*, páginas 234–240, 1977.

[114] Jacqueline L. Whalley, Raymond Lister, Errol Thompson, Tony Clear, Phil Robbins, P. K. Ajith Kumar, e Christine Prasad. An australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. *Proceedings of the 8th Austalian conference on Computing Education - Volume 52*, ACE '06, páginas 243–252, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[115] P. Winston. *The Psychology Of Computer Vision*, capítulo Learning Structural Descriptions From Examples, páginas 157–209. McGraw-Hill, New York, NY, 1975.

[116] Adriana Salvador Zanini. Avaliação da influência dos enunciados na resolução de problemas de programação introdutória. Dissertação de Mestrado, Universidade do Vale do Itajaí, Itajaí, Santa Catarina, Brasil, fevereiro de 2013. André Luis Alice Raabe (orientador).

[117] Jiajie Zhang. The nature of external representations in problem solving. *Cognitive Science*, 21(2):179–217, 1997.

ANEXO A

AVALIAÇÃO DE PROBLEMAS PRÉ-ALGORÍTMICOS

As questões foram extraídas do arquivo da Olimpíada Brasileira de Matemática[1] (OBM), considerando as provas de 2003 a 2007.

Questão 1 (2007, Fase 1, Nível 1, Problema 6)

Sílvia pensou que seu relógio estava atrasado 10 min e o acertou, mas na verdade o relógio estava adiantado 5 min. Cristina pensou que seu relógio estava adiantado 10 min e o acertou, mas na verdade o relógio estava atrasado 5 min. Logo depois, as duas se encontraram, quando o relógio de Sílvia marcava 10 horas. Neste momento, que horas o relógio de Cristina indicava?

(a) 9h30min
(b) 9h50min
(c) 10h
(d) 10h05min
(e) 10h15min

Questão 2 (2007, Fase 1, Nível 1, Problema 11)

Uma loja de CDs realizará uma liquidação e, para isso, o gerente pediu para Anderlaine multiplicar todos os preços dos CDs por $0,68$. Nessa liquidação, a loja está oferecendo um desconto de:

(a) 68%
(b) $6,8\%$
(c) $0,68\%$
(d) $3,2\%$
(e) 32%

Questão 3 (2007, Fase 2, Nível 1, Problema 1)

O número $N = 1010010100101\ldots$ contém somente os algarismos 0 e 1, de modo que o número de algarismos 0 entre dois algarismos 1 é um ou dois, alternadamente. O número N tem exatamente 101 algarismos. Qual é a soma de todos os algarismos do número N?

R: _____

[1] Provas e resoluções disponíveis em *http://www.obm.org.br/opencms/provas_gabaritos/*.

Questão 4 (2007, Fase 2, Nível 2, Problema 5)

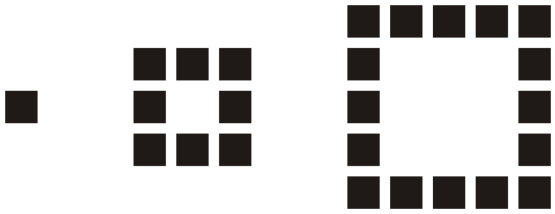
Em 1949 o matemático indiano D. R. Kaprekar, inventou um processo conhecido como *Operação de Kaprekar*. Primeiramente escolha um número de quatro dígitos (não todos iguais), em seguida escreva a diferença entre o maior e o menor número que podem ser formados a partir de uma permutação dos dígitos do número inicial. Repetindo o processo com cada número assim obtido, obtemos uma sequência. Por exemplo, se o primeiro número for 2007, o segundo será $7200 - 0027 = 7173$. O terceiro será $7731 - 1377 = 6354$.

Começando com o número 1998, qual será o 2007-ésimo termo da sequência?

R: _____

Questão 5 (2006, Fase 1, Nível 1, Problema 13)

Usando pastilhas de cerâmica preta na forma de quadradinhos foi composta uma decoração numa parede, mostrada parcialmente abaixo:



Quantas pastilhas foram empregadas em toda a decoração considerando-se que na última peça montada foram utilizadas 40 pastilhas?

(a) 60
(b) 68
(c) 81
(d) 100
(e) 121

Questão 6 (2006, Fase 1, Nível 3, Problema 10)

Uma sequência tem 9 números reais, sendo o primeiro 20 e o último, 6. Cada termo da sequência, a partir do terceiro, é a média aritmética de todos os anteriores. Qual é o segundo termo da sequência?

(a) -8
(b) 0
(c) 4
(d) 14
(e) 2006

Questão 7 (2005, Fase 1, Nível 1, Problema 2)

Numa caixa havia 3 meias vermelhas, 2 brancas e 1 preta. Professor Piraldo retirou 3 meias da caixa. Sabendo-se que nenhuma delas era preta, podemos afirmar sobre as 3 meias retiradas que:

(a) são da mesma cor.
(b) são vermelhas.
(c) uma é vermelha e duas são brancas.
(d) uma é branca e duas são vermelhas.
(e) pelo menos uma é vermelha.

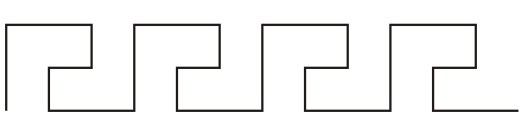
Questão 8 (2005, Fase 1, Nível 2, Problema 16)

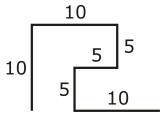
Em um ano, no máximo quantos meses têm cinco domingos?

(a) 3
(b) 4
(c) 5
(d) 6
(e) 7

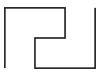
Questão 9 (2005, Fase 1, Nível 2, Problema 15)

Um serralheiro solda varetas de metal para produzir peças iguais que serão juntadas para formar o painel abaixo. O desenho ao lado apresenta as medidas, em centímetros, de uma dessas peças. O serralheiro usa exatamente 20 metros de vareta para fazer o seu trabalho.

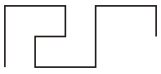


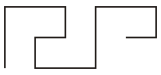


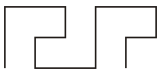
Qual dos desenhos abaixo representa o final do painel?

(a) 

(b) 

(c) 

(d) 

(e) 

Questão 10 (2004, Fase 1, Nível 1, Problema 13)

Um artesão começa a trabalhar às 8h e produz 6 braceletes a cada vinte minutos; seu auxiliar começa a trabalhar uma hora depois e produz 8 braceletes do mesmo tipo a cada meia hora. O artesão para de trabalhar às 12h mas avisa ao seu auxiliar que este deverá continuar trabalhando até produzir o mesmo que ele. A que horas o auxiliar irá parar?

(a) 12h
(b) 12h30min
(c) 13h
(d) 13h30min
(e) 14h30min

Questão 11 (2004, Fase 1, Nível 2, Problema 20)

Sobre uma mesa estão três caixas e três objetos, cada um em uma caixa diferente: uma moeda, um grampo e uma borracha. Sabe-se que:

- A caixa verde está à esquerda da caixa azul;
- A moeda está à esquerda da borracha;
- A caixa vermelha está à direita do grampo;
- A borracha está à direita da caixa vermelha.

Em que caixa está a moeda?

(a) Na caixa vermelha.
(b) Na caixa verde.
(c) Na caixa azul.
(d) As informações fornecidas são insuficientes para se dar uma resposta.
(e) As informações fornecidas são contraditórias.

Questão 12 (2004, Fase 2, Nível 1, Problema 7)

Esmeralda, de olhos vendados, retira cartões de uma urna contendo inicialmente 100 cartões numerados de 1 a 100, cada um com um número diferente. Qual é o número mínimo de cartões que Esmeralda deve retirar para ter certeza de que o número do cartão seja um múltiplo de 4?

R: _____

Questão 13 (2003, Fase 1, Nível 2, Problema 8)

Considere um número inteiro x e faça com ele as seguintes operações sucessivas: multiplique por 2, some 1, multiplique por 3 e subtraia 5. Se o resultado foi 220, o valor de x é:

(a) um número primo.
(b) um número par.
(c) um número entre 40 e 50.
(d) um número múltiplo de 3.
(e) um número cuja soma dos algarismos é 9.

Questão 14 (2003, Fase 1, Nível 2, Problema 15)

Você está em um país estrangeiro, a LUCIÂNIA, e não conhece o idioma, o LUCIANÊS, mas sabe que as palavras "BAK" e "KAB" significam *sim* e *não*, porém não sabe qual é qual. Você encontra uma pessoa que entende português e pergunta: "KAB significa *sim*?" A pessoa responde "KAB". Pode-se deduzir que:

(a) KAB significa *sim*.
(b) KAB significa *não*.
(c) A pessoa que respondeu mentiu.
(d) A pessoa que respondeu disse a verdade.
(e) Não é possível determinar sem um dicionário LUCIANÊS-PORTUGUÊS.

Questão 15 (2003, Fase 1, Nível 3, Problema 18)

Carlinhos pensa num número ímpar positivo menor do que 100. Pedrinho se dispõe a descobrir que número é esse fazendo a seguinte pergunta, quantas vezes forem necessárias: "O número que você pensou é maior, menor ou igual a x?". Note que x é um número que Pedrinho escolhe.

Quantas perguntas desse tipo Pedrinho poderá ter que fazer até descobrir o número pensado por Carlinhos?

(a) 5
(b) 7
(c) 15
(d) 25
(e) 45

Questão 16 (2003, Fase 2, Nível 1, Problema 6)

Anos bissextos são múltiplos de 4, exceto aqueles que são múltiplos de 100 mas não de 400. Quantos anos bissextos houve desde a Proclamação da República, em 1889, até hoje[2]?

R: _____

Gabarito

(1)	a	**(5)**	121	**(9)**	b	**(13)**	a
(2)	e	**(6)**	a	**(10)**	d	**(14)**	d
(3)	41	**(7)**	e	**(11)**	a	**(15)**	a
(4)	6174	**(8)**	c	**(12)**	76	**(16)**	27

[2]Atente-se que a questão é de 2003.

ANEXO B

EXEMPLO: CÁLCULO DA MÉDIA ANUAL

Dadas duas notas semestrais, calcula e exibe a média anual. Apresenta a situação do aluno como: aprovado ($média >= 70$), em exame ($média >= 50$) ou reprovado.

B.1 Código em Pascal

```
1  program media_semestral;
2  var
3     media : real;
4     nota1 : integer;
5     nota2 : integer;
6  begin
7     /*
8      * As variáveis são declaradas através do menu Editar\
9      * Declaração de Variáveis.
10     */
11    writeln('Digite a nota do primeiro semestre:');
12    readln(nota1);
13    writeln('Digite a nota do segundo semestre:');
14    readln(nota2);
15    media := (nota1+nota2)/2;
16    writeln('Sua média é:');
17    writeln(media);
18    if (media >= 70) then
19       begin
20          writeln('Você foi aprovado.');
21       end
22    else
23       begin
24          if (media >= 50) then
25             begin
26                writeln('Você está em exame.');
27             end
28          else
29             begin
30                writeln('Você foi reprovado');
31             end;
32       end;
33 end.
```

B.2 Fluxograma

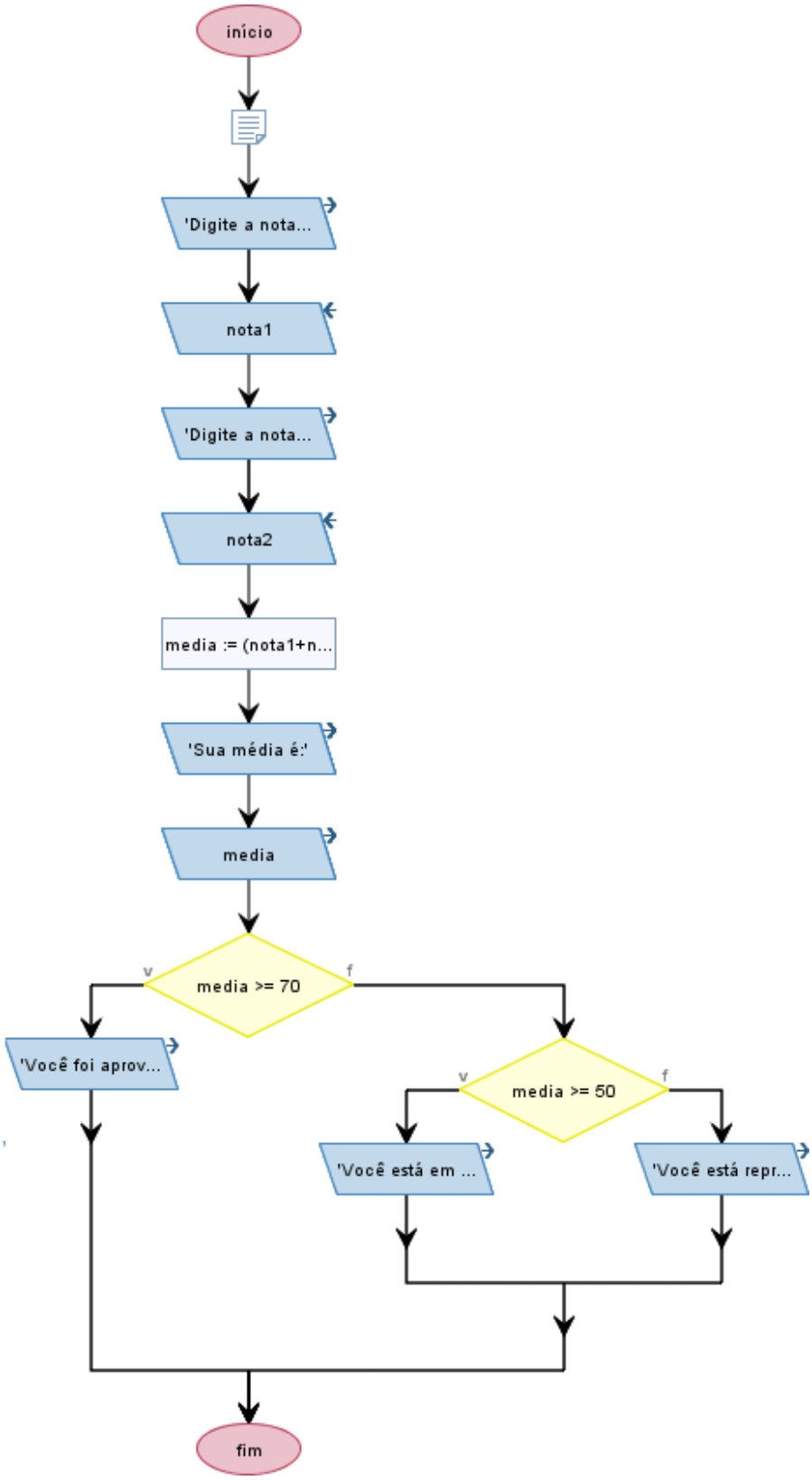


Figura B.1: Cálculo da média anual

ANEXO C

EXEMPLO: CONTAGEM REGRESSIVA

Faz a contagem regressiva de 10 até 1.

C.1 Código em Pascal

```
1  program contagem_regressiva;
2  var
3     i : integer;
4  begin
5     i := 10;
6     writeln('Iniciando contagem regressiva...');
7     while (i > 0) do
8        begin
9           writeln(i);
10          i := i - 1;
11       end;
12    writeln('Fogo!');
13 end.
```

C.2 Fluxograma

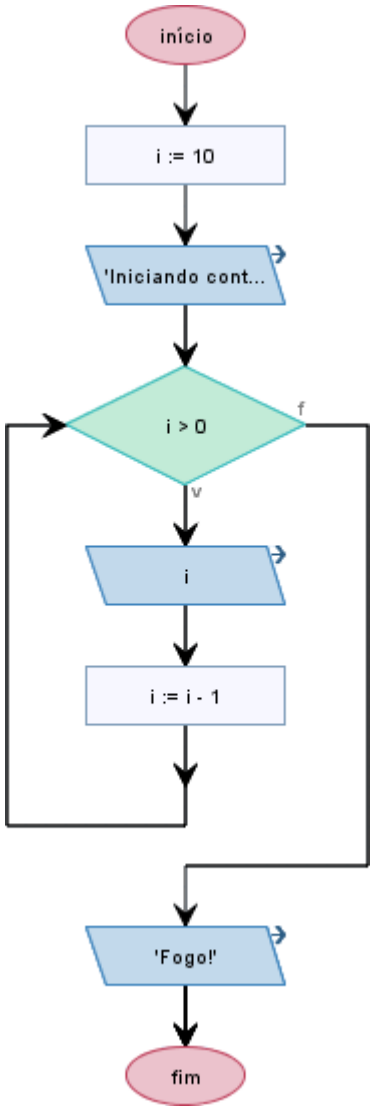


Figura C.1: Contagem regressiva de 10 a 1.

ANEXO D

EXEMPLO: CÁLCULO DE POTÊNCIA

Dados os inteiros *base* e *expoente*, calcula e exibe $base^{expoente}$.

D.1 Código em Pascal

```
1  program calcula_potencia;
2  var
3     base : integer;
4     expoente : integer;
5     potencia : integer;
6  begin
7     writeln('Digite a base:');
8     readln(base);
9     writeln('Digite o expoente:');
10    readln(expoente);
11    if (not(base = 0)) then
12       begin
13          potencia := 1;
14          if (expoente < 0) then
15             begin
16                // Não trabalharemos com números negativos.
17                expoente := -expoente;
18             end;
19          while (expoente > 0) do
20             begin
21                potencia := potencia * base;
22                expoente := expoente - 1;
23             end;
24       end
25    else
26       begin
27          // Evita iteração quando a base for zero. Imagine 0^50
28          potencia := 0;
29       end;
30    writeln('Resultado:');
31    writeln(potencia);
32 end.
```

D.2 Fluxograma

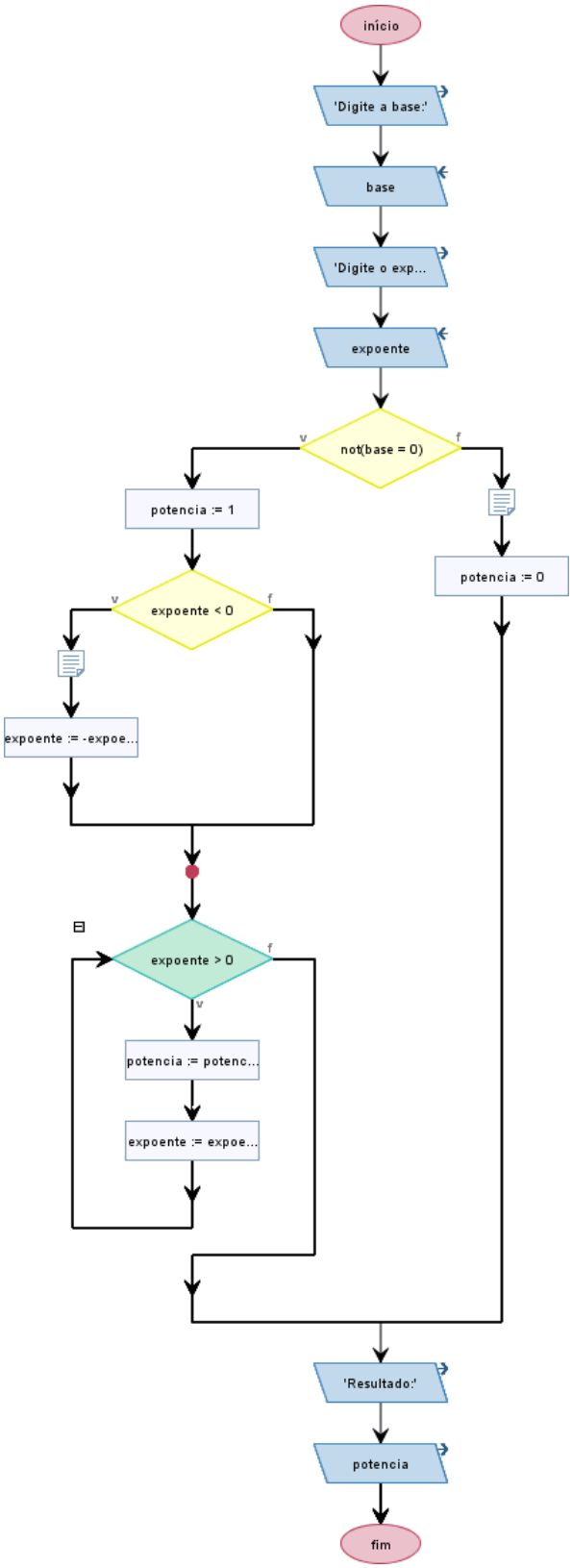


Figura D.1: Cálculo de potência

ANEXO E

DISCIPLINAS LECIONADAS

Universidade Tecnológica Federal do Paraná (UTFPR)

Guarapuava, Paraná

Curso de Tecnologia em Sistemas para Internet

- Algoritmos: 2013/1, 2012/2, 2012/1, 2011/2 e 2011/1;
- Linguagem de Programação: 2012/1;
- Estrutura de Dados 1: 2013/1, 2012/2, 2012/1 e 2011/2;

Universidade Estadual do Centro-Oeste (Unicentro)

Câmpus Santa Cruz, Guarapuava, Paraná

Curso de Ciência da Computação

Regime semestral, conforme nova grade curricular iniciada em 2010/1:

- Programação de Computadores 1: 2011/1;
- Programação de Computadores 2: 2010/2;

Regime anual, vigente até a turma ingressante em 2009:

- Programação de Computadores 1: 2009;
- Estrutura de Dados: 2009;
- Paradigmas em Linguagem de Programação: 2011, 2010, 2009, 2008, 2007.

ANEXO F

SUBCONJUNTO DE PERÍCIAS NO DOMÍNIO DE PROGRAMAÇÃO DE COMPUTADORES

O subconjunto de perícias no domínio de Programação de Computadores foi modelado com a intenção de fornecer linhas gerais para que sejam adaptadas às peculiaridades de cada linguagem de programação. Considerou-se o paradigma Imperativista.

1. Domínio do conceito de algoritmo.
   (a) Conceito de algoritmo. Entendimento do conceito de algoritmo como uma sequência finita de instruções a fim de realizar determinada tarefa.
   (b) Conceito de instrução.
   (c) Entendimento do fluxo sequencial de execução.
2. Análise e abstração de informações trazidas pelo enunciado do problema.
   (a) Leitura atenta do enunciado.
   (b) Identificação do problema.
   (c) Identificação da entrada, do processamento e da saída do problema.
3. Composição e estruturação do algoritmo.
   (a) Domínio da estrutura de um algoritmo.
   (b) Refinamento sucessivo das tarefas às instruções para resolver o problema determinado.
   (c) Transcrição do algoritmo em linguagem natural ou formal.
4. Efetivação do algoritmo.
   (a) Simulação (mental) do algoritmo alcançado.
   (b) Simulação do algoritmo através de testes.
   (c) Retroalimentação, correção e aperfeiçoamento do algoritmo.
5. Distinção entre diferentes tipos de dados e literais.
   (a) Booleano.
   (b) Inteiro.
   (c) Real.
   (d) Caractere.
   (e) Texto.
   (f) Compreensão da Tabela ASCII.

   i. Distinção entre maiúsculas e minúsculas.
   ii. Caracteres especiais.

6. Reconhecimento de variáveis na análise e na resolução do problema.
   (a) Domínio do conceito de variável.
   (b) Sintaxe da declaração de variáveis.
   (c) Identificação do tipo de dado correspondente.
   (d) Definição de nomes apropriados para variáveis.
   (e) Declaração de variáveis.
      i. Declaração de variáveis do tipo booleano.
      ii. Declaração de variáveis do tipo inteiro.
      iii. Declaração de variáveis do tipo real.
      iv. Declaração de variáveis do tipo caractere.
      v. Declaração de variáveis do tipo texto.
7. Reconhecimento de constantes na análise e na resolução do problema.
   (a) Domínio do conceito de constante
   (b) Sintaxe da declaração de constantes.
   (c) Identificação do tipo de dado correspondente.
   (d) Convenções para a denominação de constantes.
   (e) Declaração de constantes.
      i. Declaração de constantes do tipo booleano.
      ii. Declaração de constantes do tipo inteiro.
      iii. Declaração de constantes do tipo real.
      iv. Declaração de constantes do tipo caractere.
      v. Declaração de constantes do tipo texto.
8. Distinção conceitual e pragmática entre variáveis e constantes.
9. Instruções simples.
10. Domínio sobre a instrução de atribuição.
   (a) Conceito de atribuição.
   (b) Sintaxe da instrução de atribuição.
      i. Atribuição de variáveis do tipo inteiro.
      ii. Atribuição de variáveis do tipo real.
      iii. Atribuição de variáveis do tipo caractere.
      iv. Atribuição de variáveis do tipo texto.
      v. Atribuição de variáveis do tipo booleano.
   (c) Conceito de inicialização.
      i. Inicialização de variáveis do tipo inteiro.
      ii. Inicialização de variáveis do tipo real.
      iii. Inicialização de variáveis do tipo caractere.

   iv. Inicialização de variáveis do tipo texto.
   v. Inicialização de variáveis do tipo booleano.

11. Troca de valores entre duas variáveis com o uso de uma auxiliar.
12. Sobreposição e consequente perda de valores armazenados.
13. Cautela com a compatibilidade de tipos.
    (a) Incorrência de tipos incompatíveis.
    (b) Resultado da divisão de dois inteiros.
    (c) Conversão de tipos.
14. Domínio sobre a instrução de entrada.
    (a) Conceito de entrada.
    (b) Sintaxe da instrução de entrada.
       i. Entrada de uma variável do tipo inteiro.
       ii. Entrada de uma variável do tipo real.
       iii. Entrada de uma variável do tipo texto.
       iv. Entrada de uma variável do tipo caractere.
       v. Entrada de uma variável do tipo booleano.
    (c) Entrada de múltiplos valores.
    (d) Captura de teclas (especiais).
15. Domínio sobre a instrução de saída.
    (a) Conceito de saída.
    (b) Sintaxe da instrução de saída.
       i. Saída de literais.
       ii. Saída de caracteres especiais.
       iii. Saída de variáveis.
       iv. Saídas compostas.
       v. Sequências de escape.
       vi. Formatação da saída.
          A. Alinhamento.
          B. Separador de milhares.
          C. Decimais.
16. Distinção conceitual e pragmática entre instruções de entrada e de saída.
17. Avaliação e construção de expressões compostas.
    (a) Precedência de operadores aritméticos, relacionais e booleanos.
    (b) Parênteses.
       i. Alteração da precedência de operadores através de parênteses.
       ii. Desambiguação de expressões através de parênteses.
18. Avaliação e construção de expressões aritméticas.

(a) Operadores aritméticos.
    i. Adição.
    ii. Subtração.
    iii. Multiplicação.
    iv. Divisão.
    v. Divisão inteira.
    vi. Resto.
(b) Precedência dos operadores aritméticos.
(c) Avaliação de expressões aritméticas.
(d) Pragmática destacada:
    i. Quadrado de um número.
    ii. Média aritmética.
        A. Média aritmética simples.
            • A soma precede a divisão.
        B. Média aritmética composta.
            • A soma dos produtos precede a divisão.
    iii. Porcentagem.
        A. Acréscimo percentual.
        B. Decréscimo percentual.
    iv. Regra de três.
        A. Regra de três simples.
        B. Regra de três composta.
    v. Ordem dos algarismos de um inteiro.
        A. Decomposição dos dígitos de um inteiro através dos operadores DIV e MOD.
        B. Recomposição dos dígitos de um inteiro através do operador de multiplicação.

19. Incorrência da divisão por zero.

20. Avaliação e construção de expressões relacionais.

(a) Resultado de expressões relacionais.
(b) Operadores relacionais.
    i. Igual.
    ii. Diferente.
    iii. Maior que.
    iv. Menor que.
    v. Maior ou igual.
    vi. Menor ou igual.
(c) Precedência dos operadores relacionais.
(d) Avaliação de expressões relacionais.

21. Avaliação e construção de expressões booleanas.

(a) Resultado de expressões booleanas.

(b) Operadores booleanos e respectivas tabelas verdade.
    i. Negação.
    ii. Conjunção.
    iii. Disjunção.
    iv. Disjunção exclusiva.
        A. Emulação da disjunção exclusiva através dos operadores booleanos fundamentais.
(c) Precedência dos operadores booleanos.
(d) Avaliação de expressões booleanas.

22. Projeto de (conjuntos de) expressões condicionais coesas e concisas para estruturas de controle.
    (a) Entendimento de que expressões condicionais precisam ter resultado booleano.
    (b) Cuidado com a repetição indevida de condições face à exclusividade do fluxo de execução.
    (c) Distinção entre maiúsculas e minúsculas na construção de expressões condicionais, envolvendo os tipos caractere ou texto.
23. Domínio sobre estruturas de controle.
    (a) Composição de subcontextos de instruções coesos e concisos para estruturas de controle.
24. Domínio sobre as estruturas condicionais.
    (a) Conceito de desvio condicional progressivo.
        i. Entendimento da necessidade de controle do fluxo de execução.
    (b) Pragmática destacada:
        i. Determinação de datas válidas.
        ii. Reconhecimento de vogais e consoantes.
    (c) Composição de subcontextos de instruções coesos e concisos para estruturas condicionais.
        i. Evitamento da repetição indevida de instruções.
25. Domínio sobre as estruturas condicionais simples e composta.
    (a) Semântica das estruturas condicionais simples e composta.
        i. Compreensão da existência de dois caminhos possíveis e exclusivos.
    (b) Sintaxe da estruturas condicionais simples e composta.
        i. Indentação.
        ii. Blocos com uma única instrução.
    (c) Pragmática destacada:
        i. Classificação de um número como nulo ou não-nulo.
        ii. Classificação de um número como positivo, negativo ou nulo.
            A. Módulo ou valor absoluto de um número.
        iii. Reconhecimento de multiplicidade numérica.

        A. Reconhecimento de um número par (ou ímpar).
    iv. Determinação de que uma data é posterior (ou anterior) a outra.
    v. Verificação de anos bissextos.
    vi. Validação de entradas.
        A. Determinar se uma data é válida.
    vii. Determinação do maior (ou menor) valor.
        A. Entre dois valores.
        B. Entre três valores.
        C. Entre quatro valores.
    viii. Ordenação.
        A. De dois valores.
        B. De três valores.

26. Domínio sobre a estrutura condicional de seleção múltipla.

    (a) Semântica da estrutura condicional de seleção múltipla.
        i. Restrição de tratar somente tipos contáveis.
            A. Inteiro.
            B. Caractere.
            C. Booleano.
        ii. Impossibilidade de tratar tipos incontáveis.
            A. Real.
            B. Texto.
        iii. Uso do caso-contrário.
    (b) Sintaxe da estrutura condicional seleção múltipla.
        i. Seleção de um valor específico.
        ii. Seleção de uma escala de valores.
        iii. Seleção de uma lista de valores.
        iv. Seleção composta.
        v. Indentação.
    (c) Reconhecimento de problemas resolvíveis pela estrutura condicional de seleção múltipla.

27. Distinção conceitual e pragmática entre as estruturas condicionais da linguagem utilizada.

    (a) Entendimento da generalidade das estruturas condicionais simples e composta.
    (b) Entendimento da especificidade da estrutura condicional de seleção múltipla.
    (c) Reconhecimento de situações que privilegiem uma ou outra estrutura condicional.

28. Distinção e concomitância de aninhamento e encadeamento de estruturas condicionais.

    (a) Importância da indentação.

29. Aninhamento de estruturas condicionais para classificações exclusivas.

30. Encadeamento de estruturas condicionais para classificações não-exclusivas.

31. Domínio sobre as estruturas de repetição.

(a) Conceito de laço ou desvio regressivo.
    i. Entendimento da necessidade de controle do fluxo de execução.
(b) Conceito de subcontextos ciclados.
(c) Composição de subcontextos de instruções coesos e concisos para estruturas de repetição.
    i. Evitamento da repetição indevida de instruções.
    ii. Noções básicas de complexidade temporal de algoritmos.
(d) Pragmática destacada:
    i. Validação e repetição de entradas.
    ii. Repetição de entradas.
        A. Leitura de uma quantidade determinada de valores.
        B. Leitura de uma quantidade indeterminada de valores.
    iii. Potenciação.
    iv. Fatorial.
    v. Séries numéricas.
        A. Séries numéricas com um único sinal.
        B. Séries numéricas com intercalação de sinais.
    vi. Conjuntos finitos.
        A. Determinação do maior (ou menor) valor de um conjunto finito.
            - Atentamento à inicialização da variável que armazena o maior (ou menor) valor.
            - Desconsideração do valor de escape quando se tem uma quantidade de valores previamente indeterminada.
        B. Média aritmética simples de um conjunto numérico finito.
            - Entender a necessidade de dividir a somatória somente depois de ter acumulado todos os valores do conjunto.
        C. Determinação de elementos com propriedades específicas (pares, ímpares, positivos, negativos).
            - Número de elementos com propriedades específicas.
            - Percentual de elementos com propriedades específicas.
    vii. Verificação de números primos.
        A. Verificação suficiente de números primos.
        B. Verificação eficiente de números primos.

32. Domínio sobre os laços condicionais.

33. Domínio sobre a estrutura de repetição pré-avaliada.

(a) Semântica da estrutura de repetição pré-avaliada.
    i. Compreensão do pré-teste.
    ii. Entendimento de que a ciclagem ocorre incondicionalmente ao final do bloco de instruções do subcontexto.
(b) Sintaxe da estrutura de repetição pré-avaliada.
    i. Blocos com uma única instrução.
    ii. Indentação.

34. Domínio sobre a estrutura de repetição pós-avaliada.
    (a) Semântica da estrutura de repetição pós-avaliada.
        i. Compreensão do pós-teste.
        ii. Entendimento de que primeira repetição do bloco de instruções do subcontexto ocorre incondicionalmente à avaliação do condicional.
    (b) Sintaxe da estrutura de repetição pós-avaliada.
        i. Blocos com uma única instrução.
        ii. Indentação.
    (c) Otimização do uso de estruturas de repetição.
        i. Interromper a ciclagem tão logo o objetivo seja satisfeito.
        ii. Identificar e evitar passos e ciclagens desnecessários.
            A. Composição de expressões condicionais eficientes.
        iii. Percepção de que subcontextos aninhados aumentam exponencialmente a complexidade de tempo em um trecho de código.

35. Domínio sobre laços contados.
    (a) Estrutura de repetição com variável de controle.
        i. Semântica da estrutura com variável de controle.
            A. Conceito de variável de controle.
                - Risco de se manipular a variável de controle no subcontexto da estrutura de repetição.
            B. Contagem progressiva.
            C. Contagem regressiva.
            D. Incremento (ou passo).
        ii. Sintaxe da estrutura de repetição específica de laços contados.
            A. Parâmetros.
                - Variável de controle.
                - Valor inicial.
                - Valor final.
                - Valor do incremento (ou passo).
                - Indicador de contagem regressiva.
            B. Blocos com uma única instrução.
            C. Indentação.

36. Domínio conceitual e pragmático de contadores e acumuladores.
    (a) Contadores.
        i. Conceito de contador.
        ii. Inicialização de um contador.
            A. Inicialização de um contador para uso em laços condicionais.
            B. Inicialização de um contador com o uso de laços contados.
        iii. Atualização de um contador.
            A. Atualização de um contador em laços condicionais.
            B. Atualização de um contador em laços contados.

   (b) Acumuladores.
      i. Conceito de somas e produtos acumulativos.
      ii. Inicialização de um acumulador.
      iii. Atualização de um acumulador.

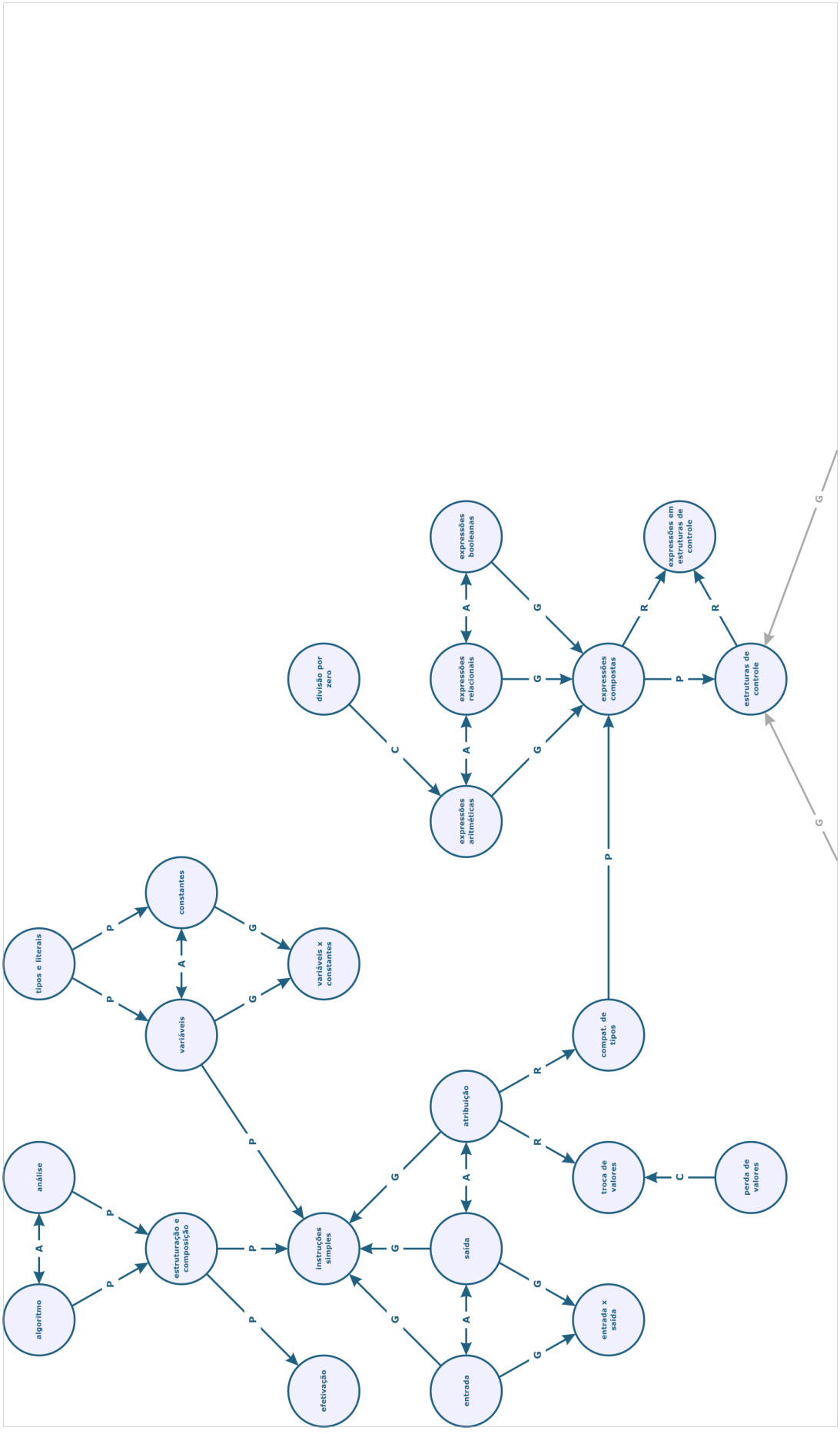
37. Incorrência de laços infinitos.
   (a) Identificação de laços infinitos.

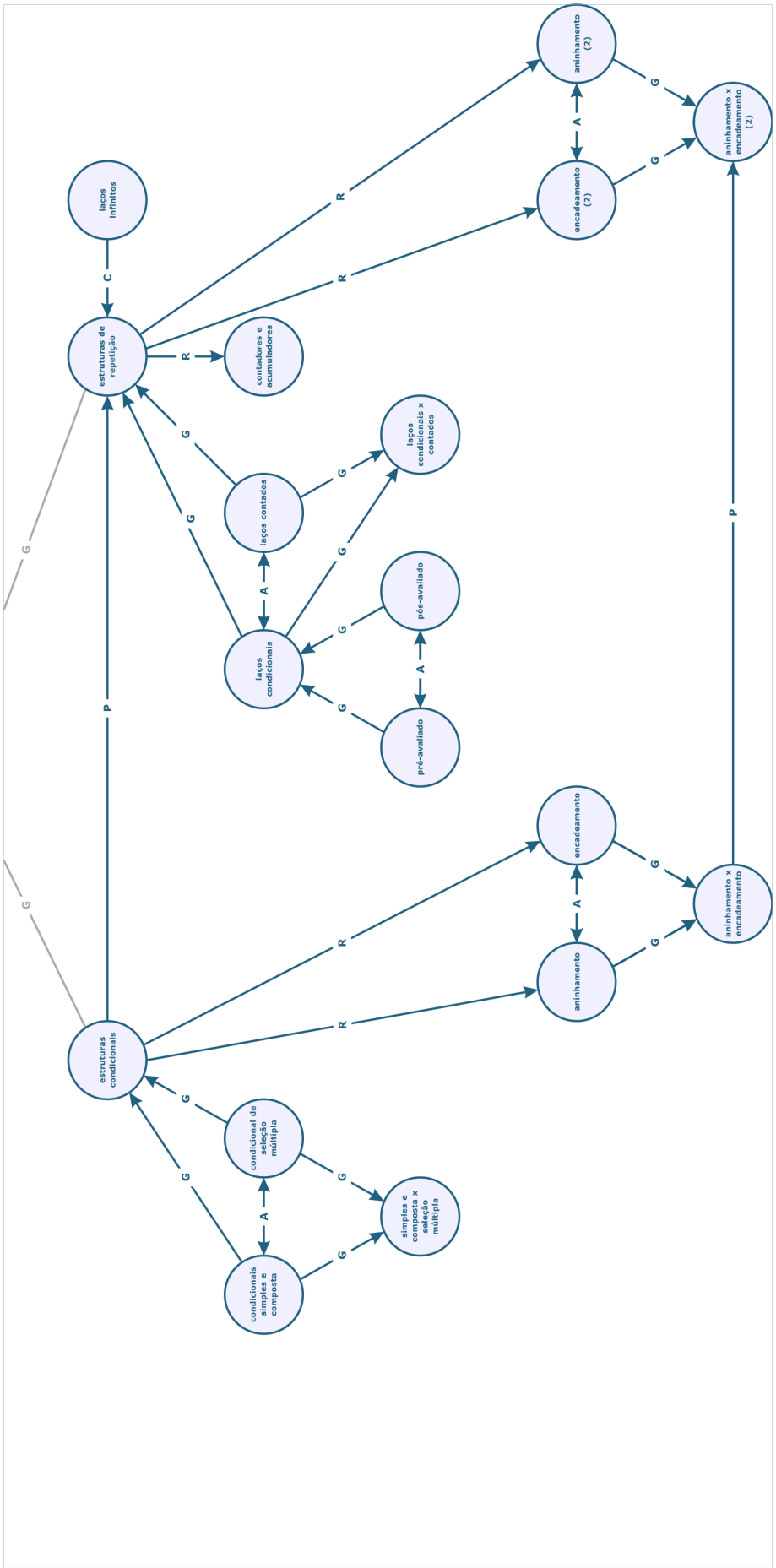
38. Distinção e concomitância de aninhamento e encadeamento de estruturas de repetição.
   (a) Importância da indentação.

39. Aninhamento de estruturas de repetição.

40. Encadeamento de estruturas de repetição.

41. Distinção conceitual, pragmática e intercabiamento entre as estruturas de repetição da linguagem utilizada.
   (a) Problemas de contagem.
      i. Identificação de problemas que envolvam contagem (inclusive séries).
      ii. Favorecimento dos problemas que envolvam contagem, pela estrutura de repetição com variável de controle.
      iii. Restrição da estrutura de repetição com variável de controle aos problemas estritamente de contagem.
   (b) Pré e pós-avaliação do condicional.
      i. Distinção de quando uma resolução privilegia a pré ou a pós-avaliação do condicional em uma estrutura de repetição.
   (c) Intercambiamento entre estruturas de repetição.
      i. Entendimento de que as estruturas pré e pós-avaliada podem ser intercambiáveis e resolver qualquer problema de repetição (inclusive aqueles solucionáveis pela estrutura de repetição com variável de controle).
      ii. Limitação do intercambiamento ao subconjunto dos problemas de contagem à estrutura de repetição com variável de controle.


tipos e literais
constantes
variáveis
variáveis x constantes
análise
algoritmo
estruturação e composição
efetivação
instruções simples
atribuição
compat. de tipos
troca de valores
perda de valores
saída
entrada
entrada x saída
divisão por zero
expressões booleanas
expressões relacionais
expressões aritméticas
expressões compostas
expressões em estruturas de controle
estruturas de controle
P
A
G
R
C


laços infinitos
estruturas de repetição
contadores e acumuladores
laços contados
laços condicionais x contados
laços condicionais
pós-avaliado
pré-avaliado
aninhamento (2)
encadeamento (2)
aninhamento x encadeamento (2)
estruturas condicionais
condicional de seleção múltipla
condicionais simples e composta
simples e composta x seleção múltipla
encadeamento
aninhamento
aninhamento x encadeamento
C
R
G
A
P

ANEXO G

CATÁLOGO DE ENUNCIADOS

O presente catálogo subsidia o desenvolvimento de perícias no subconjunto do domínio de Programação de Computadores considerado pela tese. Consiste em uma adaptação de [89] que propõe um grupo de exercícios para abordar em extensão o conteúdo programático tradicional de disciplinas correspondentes. Observou-se, em acréscimo, resultados parciais que perfazem continuidade à pesquisa recém-citada e, por enquanto não disseminados, foram cordialmente cedidos pelo professor André Raabe (Univali).

Convém a advertência de que cada enunciado pode contemplar regiões ligeiramente diferentes das capacidades do domínio, em consequência das peculiaridades sintáticas e semânticas de cada linguagem de programação Imperativista. Por exemplo, há distinção entre estruturas condicionais de seleção múltipla, quando remetidas às linguagens C ou Pascal. O mesmo ocorre com estruturas de repetição para laços contados.

A fim de orientar a composição de novos enunciados, sugere-se consultar [116] que fez uma avaliação detalhada sobre a estrutura e o contexto usualmente empregados pelos exercícios.

Instruções Simples

1. **Olá, mundo**
   Exiba a mensagem "Olá, mundo!" na tela.
2. **Olá, usuário**
   Leia o nome do usuário e exiba-o, na saída, antecedido pela mensagem "Olá".
3. **Metade de um real**
   Leia um número real e exiba a metade do valor digitado.
4. **Gastos em viagem**
   Calcule os gastos com combustível em uma viagem. Serão fornecidos: a distância a ser percorrida (em quilômetros), a média de consumo do carro (em quilômetros por litro) e o preço do combustível utilizado (em reais por litro). Exiba o valor, em reais, estimado dos gastos com combustível na viagem.
5. **Troca de valores**
   Leia dois números inteiros, a e b, e faça com que eles troquem os valores entre si. Exiba os valores de a e b.
6. **Média final**
   O sistema de avaliação de uma disciplina é composto por três provas com pesos respectivos

de 2, 4 e 6. Leia as notas obtidas por um acadêmico em cada prova e exiba a média final obtida.

7. **Caixa eletrônico**
   Simule o saque em um caixa eletrônico. Leia o valor a ser retirado e apresente a quantidade de cada cédula que será entregue ao usuário. Assuma que o equipamento possui cédulas de 10, 5 e 1 reais. Utilize o menor número possível de cédulas.

8. **Conta bancária**
   Dado o número de uma conta corrente com quatro dígitos, retorne o dígito verificador correspondente, calculado da seguinte maneira:

   - Some os quatro dígitos;
   - Multiplique os quatro dígitos,
   - Subtraia o segundo resultado do primeiro;
   - O dígito verificador será o resto da divisão do último resultado por 9.

9. **Número espelhado**
   Dado um número no formato CDU, apresente-o espelhado: UDC. Por exemplo: 123 resultará em 321. O número deverá ser armazenado em outra variável antes de ser exibido.

10. **Conversão para binário**
    Leia um número decimal, converta-o para binário e exiba o resultado. Considere que a entrada consistirá apenas dos números de 0 a 15.

Estruturas Condicionais

1. **Valor absoluto**
   Leia um número inteiro e exiba-o na saída. Se o número digitado for negativo, transforme-o no equivalente positivo (módulo) antes de apresentá-lo.

2. **Par ou ímpar**
   Leia um número inteiro e exiba se o valor é par ou ímpar.

3. **Vogais**
   Leia uma letra e retorne se esta é, ou não, uma vogal. Atente-se às maiúsculas e minúsculas. Considere que a entrada consistirá apenas de letras sem acentuação ou sinalização.

4. **Idade**
   Leia a data de nascimento de uma pessoa e a data atual. Calcule e exiba a idade do indivíduo em anos. Cada data é representada por três variáveis inteiras distintas e correspondentes ao dia, mês e ano.

5. **Horário válido**
   Leia dois números inteiros que correspondem, respectivamente, às grandezas de horas e minutos de um suposto relógio. Retorne se o horário informado é válido. Considere a hora 24 como inválida.

6. **Maior de dois números**
   Exiba o maior dentre dois números reais lidos. Identifique também e informe caso os números sejam iguais.

7. **Positivo, negativo ou nulo**
   Leia um número inteiro e exiba se o valor é positivo, negativo ou nulo.

8. **Menor de três valores**
   Leia três números inteiros e exiba somente o menor.

9. **Aprovado ou reprovado**
   Leia o nome de um acadêmico e as notas obtidas em três avaliações. Calcule a média semestral (aritmética) e informe se o acadêmico foi aprovado ou reprovado. A aprovação é conquistada com média igual ou superior a 6.

10. **Tipos de triângulos**
    Leia três valores, correspondentes aos lados de um triângulo, e apresente a classificação do referido triângulo, segundo as proporções relativas dos lados:

    - Equilátero: três lados iguais;
    - Isósceles: dois lados iguais;
    - Escaleno: três lados diferentes.

Estruturas de Repetição

1. **Colar na prova**
   Apresente, iterativamente, 30 vezes a mensagem "Não vou colar na prova".

2. **Tabuada do 5**
   Exiba iterativamente a tabuada do número 5, conforme o formato seguinte:

```
5 X 1 = 5
5 X 2 = 10
5 X 3 = 15
(...)
5 X 10 = 50
```

3. **De 100 até 1**
   Exiba iterativamente os números de 100 até 1.

4. **De 50 até 200**
   Exiba iterativamente os números de 50 até 200.

5. **Soma dos números**
   Calcule e apresente a soma de todos os inteiros no intervalo $[100, 200]$. O cálculo deve ser iterativo.

6. **Potenciação**
   Dados a base e o expoente, ambos inteiros, calcule e exiba a potência correspondente, ou seja, a base elevada ao expoente. Assuma que o expoente fornecido nunca será negativo. Não utilize função predefinida para o cálculo da potência.

7. **Média dos pares**
   Leia uma quantidade indeterminada de números inteiros. Calcule e exiba a média aritmética apenas dos números pares. A leitura deve ser interrompida quando o número zero for digitado. Desconsidere o zero no cálculo.

8. **Validação**
   Solicite a entrada de um número real positivo. Exiba uma mensagem de advertência e repita a leitura enquanto o valor digitado seja negativo ou nulo. Saiba que, através do processo recém-descrito, valida-se a entrada e garante-se a consistência dos dados fornecidos.

9. **Calculadora**
   Simule uma calculadora simples que leia dois operandos, permita a escolha da operação a ser realizada e exiba o resultado obtido. As operações possíveis (adição, subtração, multiplicação, divisão e potenciação) são indicadas pelos inteiros de 1 a 5. O funcionamento da calculadora é interrompido quando ambos os operandos informados forem iguais a zero. Exiba uma mensagem de erro, caso uma operação inválida seja escolhida. Atente-se também ao tipo de dado do resultado e aos tratamentos necessários para a realização do cálculo.

10. **Fibonacci**
    Gere e exiba os 20 primeiros termos da série de Fibonacci. Os dois primeiros termos da série são 1 e os termos subsequentes resultam da soma dos dois anteriores imediatos: 1, 1, 2, 3, 5, 8, 13, 21, ...

11. **1/n**
    Dado n, representando o número de termos da série harmônica, calcule e exiba o valor de H, sendo:
    $$H = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}$$

12. **Maioridade**
    Leia a idade de 10 pessoas e apresente quantas atingiram maioridade.

13. **Menor valor**
    Leia 20 números inteiros e apresente o menor valor informado.

14. **Maior e menor peso**
    Leia o peso de 25 pessoas, depois exiba o menor e o maior peso informados.

15. **Pares e ímpares**
    Leia 20 números inteiros, então apresente a quantidade de valores pares e ímpares informados.

16. **Fatorial**
    Calcule e exiba o fatorial de um número inteiro lido. Apresente uma mensagem de erro, caso seja fornecido um valor negativo. Não utilize função predefinida para o cálculo do fatorial.

17. **Número primo**
    Constate e apresente se um número inteiro lido é primo. Um número natural n é primo se, e somente se, tiver exatamente dois divisores naturais: 1 e o próprio n. Por definição, os números 0 e 1 não são primos nem compostos. Realize uma verificação eficiente.

18. **Tabuadas do 1 ao 10**
    Exiba iterativamente todas as tabuadas do 1 até o 10, conforme o formato seguinte:

```
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
```

```
(...)
10 X 8 = 80
10 X 9 = 90
10 X 10 = 100
```

19. **Primos após o 100**
    Encontre e apresente os 20 primeiros números primos após o 100.

20. **Maior média**
    Leia o número de acadêmicos de uma turma e a quantidade de avaliações realizadas. Depois, leia o nome de um acadêmico por vez, seguido da nota obtida em cada uma das avaliações. Exiba, de imediato, a média (aritmética) atingida pelo acadêmico. Por fim, apresente a maior média obtida e o nome do acadêmico que a conquistou.

ELEANDRO MASCHIO KRYNSKI

MODELAGEM DO PROCESSO DE AQUISIÇÃO DE CONHECIMENTO APOIADO POR AMBIENTES INTELIGENTES

Tese apresentada como requisito parcial à obtenção do grau de Doutor. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Alexandre Ibrahim Direne

CURITIBA

2013