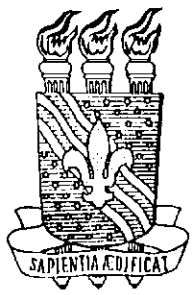

UNIVERSIDADE FEDERAL DA PARAÍBA

Centro de Ciências e Tecnologia
Coordenação de Pós-Graduação em Informática

Gilson Oliveira dos Santos

Uma Metodologia de Resolução de Problemas via Refinamento da Especificação

Dissertação de Mestrado

Campina Grande-1996

Gilson Oliveira dos Santos

Uma Metodologia de Resolução de Problemas via Refinamento da Especificação

Dissertação submetida à Coordenação de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba como requisito parcial para a obtenção do grau de Mestre em Informática.

Prof. Manoel Agamemnon Lopes
(Orientador)

Campina Grande, Paraíba, Brasil

UMA METODOLOGIA DE RESOLUÇÃO DE PROBLEMAS VIA REFINAMENTO DA ESPECIFICAÇÃO

GILSON OLIVEIRA DOS SANTOS

DISSERTAÇÃO APROVADA EM 30.12.96

PROF. MANOEL AGAMEMNON LOPES, Dr.
Presidente

PROF. ANGELO PERKUSICH, D.Sc
Examinador

PROF. BENJAMÍN RENÉ CALLEJAS BEDREGAL, D.Sc
Examinador

PROFª. MÁRCIA DE PAIVA BASTOS GOTTGTROY, D.Sc
Examinador

CAMPINA GRANDE - PB

Dedico:

À Deus,

aos meus pais e avós,

aos meus irmãos,

e à Carmen.

Agradecimentos

Na jornada que envolve um trabalho deste tipo torna-se uma tarefa difícil fazer os agradecimentos sem esquecer de ninguém, pois são inumeráveis as pessoas que deram sua contribuição e me ajudaram a percorrer este caminho.

Agradeço aos meus pais José Muniz dos Santos e Helena Oliveira dos Santos e aos meus irmãos pelo incentivo, apoio e confiança durante todo o período deste trabalho.

À Carmen pela paciência, confiança e carinho com que me apoiou ao longo desta jornada.

Ao Prof. Manoel Agamemnon Lopes pela confiança em ter me permitido trabalhar o tema desta dissertação, pela orientação acadêmico-científica e pela paciência com que tratou minhas limitações no decorrer deste trabalho.

Ao Prof. Benjamín Callejas Bedregal pela atenção, paciência e inestimáveis sugestões que foram fundamentais para a forma final deste trabalho.

Ao Prof. Mário Toyotaro Hattori pela confiança que me foi depositada no início do mestrado.

À Álvaro e Ismênia pela calorosa acolhida, incentivo, apoio e amizade que sempre dedicaram a minha pessoa.

À Robson e Cida pela amizade, incentivo e apoio que sempre me deram no decorrer deste trabalho.

À Maria Elizabeth e demais amigos do NIES/UFAL pelo apoio e incentivo para trilhar esta caminhada. Igualmente, aos companheiros e amigos Adriano, Zan, Luciana e Luiz Lima.

Aos colegas e amigos do DIMAp/UFRN, especialmente a *Profª*. Márcia Gottgtroy pelo incentivo, apoio e sugestões dadas, aos Profs. Ivan e Galileu pela amizade e incentivo, e a Márcia Rodrigues pelo incentivo e amizade.

Estendo meus agradecimentos, a todos os professores do DSC, em especial ao Prof. Edilson, e aos funcionários Aninha, Vera e Marcelo da COPIN, a Alberto e Liliam

do LABCOM, Zeneide, Manuela e Arnaldo da MINIBLIO, Josenilda da Secretaria e demais funcionários do DSC que sempre tiveram boa vontade e muita cordialidade em me atender.

Finalmente, a todos os amigos e companheiros que me deram apoio e confiança. Especialmente, Kissia, Vera, Sônia, Washington, Roberta, Artur, Milena, Ricardo, Norma, Michelle, Adeilton, Cenidalva, Vitor, ...

Resumo

Este trabalho apresenta uma metodologia de resolução de problemas. Nesta metodologia o mecanismo de resolução é composto de diversos níveis, nos quais se processa o refinamento da especificação do problema. Este mecanismo atua por intermédio do aperfeiçoamento da linguagem na qual o problema está sendo especificado. Dependendo do problema, a solução surge na medida que a especificação passa pelas várias etapas de refinamentos sucessivos. O desenvolvimento de tal metodologia é feita dentro do ambiente teórico da Teoria Geral de Problemas. Finalmente, para ilustrar sua aplicabilidade, a metodologia aqui proposta é utilizada na resolução de alguns problemas considerados clássicos na literatura de computação.

Palavras chaves: problemas, solução de problemas, teoria de problemas, decomposição de problemas, refinamento da especificação.

Abstract

This work intends to show a problem solving methodology by refinement of specification. Such a process of resolution is composed of some levels of specification. The refinement of specification of problem solving process consists of a series of language transformations by improving the language used for specification of solution. We use the General Theory of Problems and other formal components to provide some necessary concepts to develop refinements of specifications. The proposed approach is used to solve some classical problems in the field of Computer Science.

Key words: problems, problem solution, theory of problems, problem decomposition, refinement of specification.

Sumário

Lista de Figuras

Capítulo 1

Introdução

A method of solution is perfect if we can foresee from the start, and even prove, that following that method we shall attain our aim.
LEIBNITZ: Opuscules, p. 161

1.1 Motivação

O homem busca incessantemente solucionar problemas que surgem em sua vida cotidiana. Muitas atividades humanas, tais como resolver quebra-cabeças, brincar com jogos, trabalhar com matemática e dirigir um automóvel demandam "inteligência".

Um dos enfoques do comportamento inteligente é considerar o processo de resolução de problemas. Alguns pensadores como Papus, Leibnitz, Bolzano e mais recentemente G. Polya, estudaram a "arte de resolução de problemas".

O professor de matemática George Polya, em seu livro "*How to Solve It*"[Pol78], enfoca a resolução de problemas matemáticos. No entanto, muitas das regras que ele descreve têm aplicação genérica. A questão é que as regras lançadas por Polya não foram colocadas numa forma operacionalizável, mas sim de maneira intuitiva. Alguns trabalhos mais recentes como os de M. A. Lopes [Lop81] e de P. A. S. Veloso [Vel82], introduziram as bases para uma especificação formal de problema.

Polya, no trabalho citado acima, apresenta um conjunto de regras que quando aplicadas podem auxiliar no processo de entendimento e resolução de problemas. Polya [Pol78] argumenta que a resolução de um problema envolve a ação de seguir uma seqüência de passos que permitem apresentar como é a solução. Mas as regras sugeridas por ele não são adequadas e nem estão apresentadas de forma a captar aspectos que são inerentes à resolução de problemas de maneira operacionalizável. Levando-se em conta que um problema é resolvido seguindo uma série de passos, pode-se, também, argumentar que é possível estabelecer um mecanismo definido ou baseado em uma seqüência de passos que podem auxiliar nessa resolução computacionalmente [Lop81] [LCB88]. Um mecanismo com esse objetivo deve ser efetuado da maneira mais natural possível.

Tendo-se em vista o exposto acima e a inexistência de uma análise do processo de refinamento de especificação no contexto da Teoria Geral de Problemas [Lop81], o presente trabalho tem a finalidade de expor uma metodologia de resolução de problemas via refinamento da especificação. Intenciona-se, também, tornar a *representação abstrata do problema* adequada e mais próxima possível de uma representação implementável, de modo que a solução de um determinado problema possa ser tratada computacionalmente.

Normalmente, entende-se por resolver um problema computacionalmente (programar) a codificação em alguma linguagem de programação, sem conexão da situação problema que se quer resolver com o programa codificado. A seguir, tenta-se mostrar que a compreensão da situação tida como problema é um dos passos envolvidos na resolução de um problema com o auxílio de um computador e que devem ser tratados, com o mesmo cuidado dedicado à programação propriamente dita, as etapas que vão desde a descrição da situação problema a ser resolvida até a especificação de todos os elementos essenciais para a resolução do problema.

Um problema raramente é resolvido como é expresso inicialmente. O habitual é utilizar um conjunto de convenções para representar a informação e descrever o problema por meio dessas convenções. Não se resolve um problema real sem uma representação idealizada do mesmo, a partir da qual se obtém uma solução que será

interpretada e traduzida numa solução do problema real.

Na resolução de um problema, com ou sem o auxílio do computador, é necessário escolher uma abstração da realidade, definindo um conjunto de dados que irá representar uma situação real. Essa escolha deve ser feita de acordo com as características do problema a ser resolvido. A etapa seguinte consiste em encontrar uma maneira de representar esses dados. Essa representação dependerá do instrumento ou ferramenta que será utilizada na resolução do problema.

No desenvolvimento do esquema de resolução de muitos problemas, as pessoas são conduzidas a realizar, de maneira sucessiva, uma apresentação das partes que o envolve por meio da aplicação da técnica usualmente conhecida como "refinamentos sucessivos dos detalhes". Processo idêntico é realizado quando no processo de desenvolvimento de programas [Wir89]. Evidentemente, a primeira decisão desta cadeia é influenciada principalmente pelo enunciado do problema, e as últimas dependem progressivamente da ferramenta empregada e da respectiva tecnologia.

A informação contida no enunciado de um problema geralmente consiste de um conjunto selecionado de dados, a partir dos quais é possível se obter os resultados desejados. Os dados representam uma abstração da realidade, no sentido de que certas propriedades e características do objeto real são desprezadas, por serem inexpressivas ou irrelevantes para a resolução do problema abordado. Dessa maneira, a abstração pode ser, também, visualizada como uma simplificação dos fatos que envolvem a situação problema tratada.

Na busca de uma especificação de problema manipulável pela máquina, uma das técnicas mais utilizadas é o chamado "refinamento passo-a-passo" (*stepwise refinement*). Callejas Bedregal [Bed87], inspirado nos trabalhos de [Lop81] e de [OJ85], formalizou algumas estratégias de decomposição de problemas. Ele tratou a decomposição como uma técnica geral e efetiva de resolver problemas. Deve-se considerar que as técnicas de "divisão-e-conquista", "refinamento passo-a-passo", "projeto de cima para baixo (*top down*)", etc., são diferentes nomes utilizados para se referir a decomposição. No trabalho citado, Callejas Bedregal estuda a especificação dos componentes F (domínio fonte), S (domínio dos resultados) e d (restrição ao domínio de dados), dos

problemas solúveis por decomposição via análise da intencionalidade.

O trabalho de especificação é necessário para definições formais em computação, como por exemplo para descrever de forma adequada quais os elementos que irão compor uma determinada linguagem.

Num sentido geral, a representação envolve a relação que se estabelece entre as diferentes maneiras de formular um problema e a eficiência com que se pode encontrar uma solução desse problema.

Na resolução de problemas, o tratamento dado por alguns sistemas especialistas, por exemplo, é resolvê-lo de forma "invertida", pois em geral considera-se o processo de resolução retroativo (*botton up*). Por exemplo, para resolver o "problema da palavra" (*word problem*), inicia-se pela análise *léxica* da mesma, a fim de verificar se os símbolos estão no alfabeto de uma determinada linguagem específica; depois observa-se a *sintaxe*, em que se analisa se o modo como os símbolos estão dispostos obedecem as regras estabelecidas para aquela determinada linguagem; em seguida vê-se a *semântica*, ou seja, analisam-se os diversos significados que a palavra possui naquela linguagem; no final, analisa-se, o que não é trivial, qual dentre os diversos significados da palavra adequa-se ao contexto no qual ela está inserida, que é a *pragmática* (ver esquema apresentado na Figura 1.1). Diversos problemas são tratados e resolvidos seguindo-se esquema idêntico, analisando-se o problema de maneira retroativa.

No entanto, intuitivamente não se inicia o processo de resolução de um determinado problema da maneira como foi exposta acima, mas sim usando o tratamento inverso: primeiro, trabalha-se o entendimento do problema, analisando o que é o problema - quais os dados, o que se busca (pragmática); depois busca-se compreender cada "objeto" identificado no entendimento do problema (semântica), analisando, assim, que informações eles fornecem a fim de que o solucionador busque a solução do problema; em seguida, analisa-se quais os procedimentos e/ou operações devem ser realizadas para resolver o problema (estabelecimento da linguagem); sendo assim, estabelecem-se os elementos suficientes que irão resolver aquele determinado problema (ver esquema Figura 1.2).

É apresentado neste trabalho um mecanismo de resolução por meio de refinamentos,

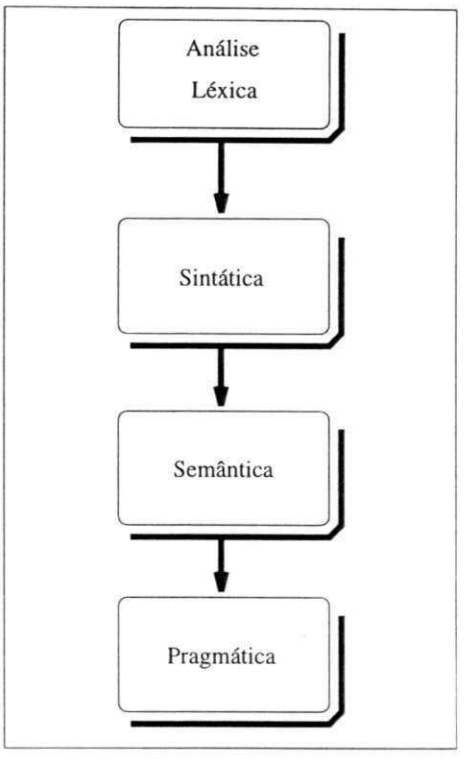

Figura 1.1: *Esquema usualmente utilizado para a resolução do "problema da palavra"*

que vai desde o entendimento do problema até chegar a estabelecer como é a solução (e a respectiva linguagem envolvida). Deve-se deixar claro que esse mecanismo de refinamento, aqui exposto, pressupõe que o problema já esteja "formulado", isto é, que o enunciado já esteja estabelecido e que ele descreva de certa maneira a "situação problema".

Neste trabalho propõe-se que o mecanismo descrito solucione problemas, inicialmente, por meio do aperfeiçoamento do entendimento e refinamento da descrição do mesmo, estabelecendo a linguagem para a resolução do problema. Feito isso, parte-se para apresentar a solução, usando-se como método auxiliar a "decomposição". A técnica de decomposição utilizada para encontrar a solução consiste em decompor o domínio de dados, no qual particiona-se os dados envolvidos em subconjuntos de dados até eles estarem suficientemente simples para se encontrar o resultado. Feito isso, determina-se a solução do problema tratado. Deve está claro que isso compõe todo

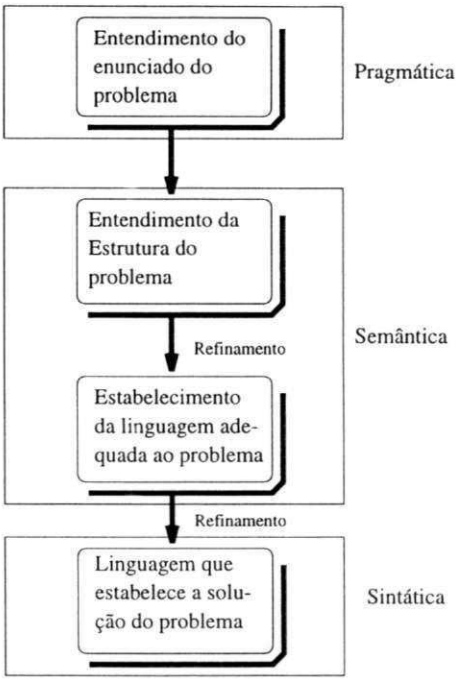

Figura 1.2: *Esquema intuitivo do processo de resolução de problemas*

um mecanismo, o qual é tratado e exposto no decorrer deste trabalho, através de formalizações e exemplos que o explicitam.

1.2 Apresentação

A seguir, é descrito como está dividido este trabalho, apresentando o conteúdo de cada capítulo.

O presente trabalho está dividido em cinco capítulos, dos quais este é o primeiro.

A descrição sintética do conteúdo de cada capítulo é a seguinte:

Neste capítulo apresenta-se a motivação para a leitura do presente trabalho, destacando-se sua importância e contextualizando-o na teoria de problemas.

No Capítulo 2, são apresentados alguns conceitos preliminares que dão garantias ao mecanismo de resolução aqui exposto. Nesse capítulo são incluídos algumas noções

básicas de linguagem, e alguns conceitos sobre estrutura. Apresenta-se, também, alguns elementos da Teoria Geral de Problemas que serão úteis no entendimento das etapas envolvidas no mecanismo de resolução aqui exposto, facilitando e garantindo alguns passos descritos na resolução de problemas, exemplificados neste trabalho. Nesse capítulo é incluído, também, uma axiomatização para o processo de especificação.

No Capítulo 3, é caracterizada a especificação de problemas. Nesse capítulo, é destacado a importância de se modelar (especificar) bem o problema por meio de uma linguagem de representação que o expresse de maneira adequada.

No Capítulo 4, é apresentado a solução de problemas por meio do refinamento da especificação. Buscando com esse refinamento caracterizar como deve ser a solução do problema. A caracterização da solução pode ser dada como sendo um esquema recursivo (utilizando como método a *decomposição n-ária*). É incluído, também, exemplos que ilustram a aplicação do mecanismo exposto para solucionar problemas.

Finalmente, no Capítulo 5 são apresentadas as conclusões, fazendo-se algumas considerações e uma pequena discussão acerca do mecanismo de resolução exposto neste trabalho.

Capítulo 2

Preliminares

2.1 Introdução

Neste capítulo são fornecidos elementos importantes e necessários para o desenvolvimento deste trabalho, como algumas noções básicas sobre linguagem, alguns conceitos básicos sobre estrutura matemática, e alguns elementos da Teoria Geral de Problemas. Tais ingredientes objetivam fornecer embasamento teórico ao trabalho e assegurar o mecanismo de resolução de problemas aqui exposto.

2.2 Noções Básicas sobre Linguagem

Ao abordar a resolução de problemas por meio do refinamento de especificação, há um aperfeiçoamento da linguagem na qual o problema está sendo descrito.

Uma das providências a ser tomada ao trabalhar com linguagens é estabelecer uma notação uniforme e um conjunto de convenções para tais linguagens.

Para formalizar uma linguagem L, há necessidade dos seguintes símbolos lógicos:

parênteses: (,);

variáveis: x, y, ... ;

conectivos: $\wedge$(e), $\vee$(ou), $\neg$(negação), $\rightarrow$(implicação), $\leftrightarrow$(biimplicação);

quantificadores: $\forall$(para todo), $\exists$(existe);

e um símbolo relacional binário: =(igualdade).

Muitas vezes será usado os símbolos "$\in$" por "pertence", "|" por "tal que" como parte da linguagem matemática coloquial.

Os símbolos descritos acima são comuns a todas linguagens aqui consideradas.

Será considerado, também, os conjuntos de símbolos:

- $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ (símbolos constantes)
- $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \ldots\}$ (símbolos funcionais)
- $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots\}$ (símbolos relacionais)

onde para cada símbolo de $\mathcal{F}$ e $\mathcal{R}$ é associado (via uma função $aridade_{\mathcal{F}} : \mathcal{F} \longrightarrow IN$[1]) um número $n \in IN$, $n \geq 1$, chamado *aridade*. A aridade dos símbolos constantes é 0 (zero).

Definição 2.1 [Lop81]: *Uma linguagem L de 1^a ordem é descrita por um terno $\langle \mathcal{R}_L, \mathcal{F}_L, c_L \rangle$, em que $\mathcal{R}_L \subseteq \mathcal{R}$, $\mathcal{F}_L \subseteq \mathcal{F}$ e $\mathcal{C}_L \subseteq \mathcal{C}$.* □

Definição 2.2 [Lop81]: *O conjunto de fórmulas bem formadas (wff) de uma linguagem L, é o menor conjunto de expressões de L definidas como se segue.*

i) Um termo $t \in L$ é uma variável, ou uma constante, ou $f(t_1, t_2, \ldots, t_n)$ em que $f \in \mathcal{F}_L$ com aridade n e t_i, $1 \leq i \leq n$, são termos de L;

ii) Uma fórmula atômica de L é uma expressão da forma $t_1 = t_2$ ou $r(t_1, t_2, \ldots, t_n)$, sendo $r \in \mathcal{R}_L$ com aridade n e t_i, $1 \leq i \leq n$, são termos de L;

[1] IN : conjunto dos números naturais.

iii) Uma fórmula de L é uma fórmula atômica, ou uma fórmula obtida pelas seguintes regras:

iii-a) Se $\alpha, \beta \in wff$ então $\neg\alpha, \alpha \wedge \beta, \alpha \vee \beta, \alpha \rightarrow \beta$ e $\alpha \leftrightarrow \beta$ são fórmulas bem formadas;

iii-b) Se $x \in V$ [2] e $\alpha \in wff$ então $\forall x\alpha$ e $\exists x\alpha$ são fórmulas bem formadas;

iv) Só as expressões obtidas como acima são fórmulas bem formadas. □

Serão usados os símbolos abaixo para expressar linguagens diferentes

$$L_1, L_2, L_3, \text{etc.}$$

Usando numa linguagem símbolos como, por exemplo, $+$ para adição, $\leq$ para uma relação de ordem, etc., simplesmente escreve-se

$$L = \langle \leq \rangle, L = \langle \leq, +, \cdot, 0 \rangle, L = \langle +, \cdot, -, 0, 1 \rangle, \ldots,$$

para tais linguagens.

Ocasionalmente passa-se de uma dada linguagem L para outra linguagem L' que tem todos os símbolos de L mais alguns símbolos adicionais. Em tais casos usa-se a notação $L \subseteq L'$ e diz-se que a linguagem L' é uma *expansão* de L, e que L é uma *redução* de L'. Desde que L e L' são conjuntos de elementos (objetos), a *espansão* L' pode ser escrita como $L' = L \cup \{X\}$, em que X é o conjunto dos novos elementos.

2.3 Conceitos Básicos sobre Estrutura

Neste trabalho será dado um tratamento estrutural à questão "problema". Assim, a seguir são apresentadas algumas definições sobre estrutura (matemática).

[2] V: é o conjunto das variáveis $\{x_1, x_2, \ldots\}$.

As primeiras idéias sobre a noção de "estrutura" são encontradas em trabalhos de Galois e Riemann. No entanto, o conceito geral de estrutura (matematicamente falando), veio a surgir num trabalho de Dedekind, por volta de 1897 [Oli75].

É apresentada a seguir uma definição de *estrutura matemática.*

Definição 2.3 [Oli75] *: Uma estrutura representada por*

$$E = \langle A, R, F \rangle \tag{2.1}$$

é denominada uma **estrutura matemática**, *em que*

- $A \neq 0$ *é o domínio da estrutura,*
- R *é um sistema de relações e operações definidas entre os elementos de* A,
- F *é um fator de auto-regulação da estrutura (morfismos em matemática).* □

Na linguagem matemática ocorrem, em geral, variáveis (quantificáveis) x, y, etc., em expressões tais como $(\forall x)P(x)$ e $(\exists y)Q(y)$, ou em expressões mais complexas como $(\forall x)(\forall y)(x = y \rightarrow R(x,y))$, em que P, Q, R e $=$ são predicados que se referem a propriedades de x e y.

O domínio ontológico dessas expressões, i.e. os objetos (matemáticos) aos quais se referem as variáveis utilizadas, forma um conjunto chamado aqui de *domínio* ou *universo do discurso*, e os predicados mencionados expressam certas relações entre os elementos do domínio. Isso dá origem a noção de *estrutura relacional* que será definida a seguir.

Nota: As definições que seguem foram retiradas de [Alm90].

Definição 2.4 *(Estrutura Relacional): Uma* **estrutura relacional** *é uma quádrupla* $\mathcal{A} = \langle A, \{r_i\}, \{f_j\}, \{c_k\} \rangle$, $i \in I$, $j \in J$, $k \in K$, *com as funções associadas* $\lambda : I \longrightarrow IN^{+3}$, $\mu : J \longrightarrow IN^{+}$ *tal que:*

[3] IN^{+} : conjunto dos números naturais positivos.

(a) A, o domínio de $\mathcal{A}$, é um conjunto não vazio;

(b) I é um conjunto (podendo ser vazio) tal que, para cada $i \in I$, $\lambda(i) \in IN^+$ e r_i é uma $\lambda(i)$-ária relação em A;

(c) J é um conjunto (podendo ser vazio) tal que, para cada $j \in J$, $\mu(j) \in IN^+$ e f_j é uma $\mu(j)$-ária função em A;

(d) K é um conjunto (podendo ser vazio) tal que, para cada $k \in K$, c_k, um elemento distingüido (ou constante), é um elemento de A □

É definido a seguir, um importante conceito envolvendo estruturas relacionais. Esse conceito será bastante utilizado daqui para frente.

Definição 2.5 : *Dada uma estrutura relacional $\mathcal{A} = \langle A, \{r_i\}, \{f_j\}, \{c_k\}\rangle$, $i \in I$, $j \in J$, $k \in K$, o* **tipo** *de $\mathcal{A}$, denotado $\tau_{\mathcal{A}}$, é o terno $\langle k, \lambda, \mu\rangle$, em que $\lambda : I \longrightarrow IN^+$ e $\mu : J \longrightarrow IN^+$, são funções que codificam a aridade das relações e funções, respectivamente* □

De acordo com a definição acima, pode-se concluir que o *tipo* de uma estrutura fica caracterizado pela quantidade de relações, funções (juntamente com suas respectivas aridades) e constantes na estrutura.

As definições colocadas acima são apenas para dar idéia de algumas relações existentes entre estruturas, e que auxiliam no entendimento da conceituação dada a *problema* (forma de estrutura) e das relações existentes entre problemas.

2.4 Elementos da Teoria de Problemas

Na atualidade, um grupo de pesquisadores vem trabalhando num ramo da Inteligência Artificial chamado *Teoria Geral de Problemas*. No Brasil alguns pesquisadores têm se dedicado a esse ramo de pesquisa. Nesta seção serão dados alguns elementos e conceitos básicos da Teoria Geral de Problemas [Lop81] que servem de base a outros capítulos deste trabalho.

A Teoria Geral de Problemas elaborada por M. A. Lopes [Lop81] fornece alguns elementos importantes para o mecanismo de refinamento de especificação exposto neste trabalho. Nesta seção serão apresentados alguns desses elementos que servirão como noções preliminares que auxiliarão no entendimento de certas etapas do mecanismo que será aqui exposto. M. A. Lopes [Lop81] trabalhou o enfoque estrutural de "problema", elaborando uma formalização exibindo as propriedades e os elementos observados na noção intuitiva de "problema". Assim, com o propósito de captar os elementos considerados essenciais à abstração "problema", foi estabelecido uma representação estrutural de problema definindo inicialmente o que se denomina *pré-problema*.

Definição 2.6 [Lop81] : *Um* **pré-problema** *é uma estrutura do tipo* $\mathcal{P} = \langle F, S, Cd \rangle$ *em que* F *e* S *são conjuntos não vazios, chamados de domínio fonte e co-domínio de resultados, respectivamente; e* Cd *é um predicado intencional cuja interpretação é um subconjunto de* $F \times S$ *(ver figura 2.1).* □

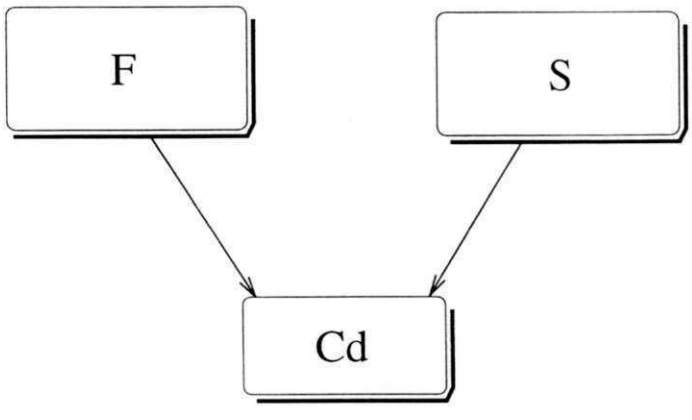

Figura 2.1: *Representação esquemática da estrutura de "pré-problema".*

Segue abaixo um exemplo ilustrando a aplicação da definição de *pré-problema*.

Exemplo 2.1 : *Média de um conjunto de notas*

Uma estruturação para esta situação problema pode ser feita por meio da seguinte interpretação:

$$\mathcal{P} = \langle F, S, Cd \rangle,$$

onde

* $F = P^{FIN}$ (sendo que P^{FIN} representa uma parte finita do conjunto de notas)
* $S = NOTAS$ (representa o co-domínio de resultados)
* $Cd : (X, n) \in Cd \leftrightarrow n = (\sum X / cardinalidade(X))$ (representa a intenção do problema - condicionante) □

A expressão "pré-problema" para a estrutura $\mathcal{P} = \langle F, S, Cd \rangle$, é devido a parte intencional do problema está sendo expressa simplesmente por meio de Cd, a qual não declara explicitamente a situação problema tratada. Torna-se, então, necessário uma representação estrutural que descreva de maneira mais adequada a situação problema.

Será apresentada a seguir uma definição estrutural de "problema" que atende as necessidades expostas no parágrafo acima.

Definição 2.7 citeLopes81 : *Um* **problema** *é uma estrutura do tipo* $\mathcal{P} = \langle F, S, d, Int, Adm \rangle$ *em que* F *e* S *são conjuntos não vazios, denominados domínio de dados e domínio de resultados, respectivamente;* $d \subseteq F$ *é uma restrição sobre os dados,* $Int \subseteq F \times S$ *é o predicado intencional, e* $Adm \subseteq S^{F}$[4] *é a condição de admissibilidade* □

Os conjuntos F e S representam a parte substantiva do problema $\mathcal{P}$; enquanto d, Int e Adm representam a parte intencional e são denominados respectivamente *predicado fonte, predicado intencional* e *predicado de admissibilidade. "A idéia é que Int reflita a condição principal ou intencionalidade principal segundo a qual os pares (f,s) são*

[4] S^F é o conjunto de funções de F em S.

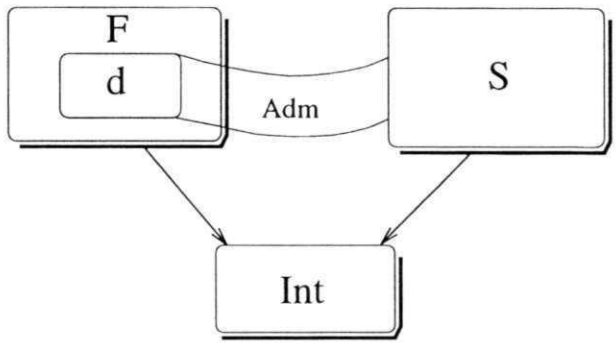

Figura 2.2: *Representação gráfica da Estrutura de "Problema".*

formados; d reflita a parte de F relevante ao problema segundo Int; e Adm é a restrição às muito possíveis funções de F em S que satisfazem Int ou que não serão consideradas como admissíveis." [Lop81]

Segue um exemplo ilustrando a aplicação da definição de *problema* apresentada acima.

Exemplo 2.2 : *Ordenação de Seqüências*

Uma estruturação para a situação problema de ordenar uma seqüência de naturais pode ser feita por meio da seguinte interpretação:

$$\mathcal{P} = \langle F, S, d, Int, Adm \rangle,$$

onde

* $F = S = Seq$ (sendo que Seq representa o conjunto das seqüências de dados)

* $d = Seq'$ (sendo que Seq' representa uma seqüência finita de dados)

* $Int = Ordenar$ ($Ordenar$ representa a intenção do problema)

Seja $s, s' \in Seq$, então

$(s, s') \in Ordenar \leftrightarrow mesma(s, s') \wedge ordenada(s')$, em que

$mesma(s, s')$ indica que s e s' possui os mesmos elementos (não necessariamente na mesma ordem) e $ordenada(s')$ indica que s está ordenada sobre os dados, em que a ordem exprime o tipo de dado.

* $Adm = Alg$ (em que Alg indica que a possível solução é um algoritmo).□

Quando é utilizado um elemento específico do domínio de dados relevante ao problema (d) vem a noção de *instância de problema*, que tem a seguinte definição.

Definição 2.8 [Lop81]:*Uma* **instância** *de um problema* $\mathcal{P}$ *é um problema* $\mathcal{P} = \langle F, S, \{x\}, Int, Adm\rangle$, *tal que* $x \in d$ □

Uma vez que problema foi definido como uma estrutura, pode-se fazer de uma maneira mais natural a avaliação sobre problemas. Assim, uma ferramenta (matemática) interessante para trabalhar problemas é *homomorfismo*.

Um *homomorfismo* entre dois problemas é uma relação obtida por um par de funções definidas entre os conjuntos bases dos correspondentes problemas, preserva a parte intencional do problema (ver figura 2.3).

Definição 2.9 [Lop81] : *Sejam* $\mathcal{P} = \langle F, S, d, Int, Adm\rangle$ *e* $\mathcal{P}' = \langle F', S', d', Int', Adm'\rangle$ *dois problemas. Um* **Homomorfismo** (φ, ψ) *de* $\mathcal{P}$ *em* $\mathcal{P}'$, *denotado por* $\mathcal{P} \tilde{\rightarrow} \mathcal{P}'$, *consiste de duas funções* $\varphi : F \longrightarrow F'$ *e* $\psi : S \longrightarrow S'$, *tal que:*

(i) $\forall x \in F,\ x \in d \rightarrow \varphi(x) \in d'$,

(ii) $\forall x \in F, \forall y \in S,\ (x, y) \in Int \rightarrow (\varphi(x), \psi(y)) \in Int'$,

(iii) $\forall a \in S^F,\ \forall b \in S'^{F'}, (\ b \circ \varphi = \psi \circ a \rightarrow (a \in Adm \rightarrow b \in Adm'))$. □

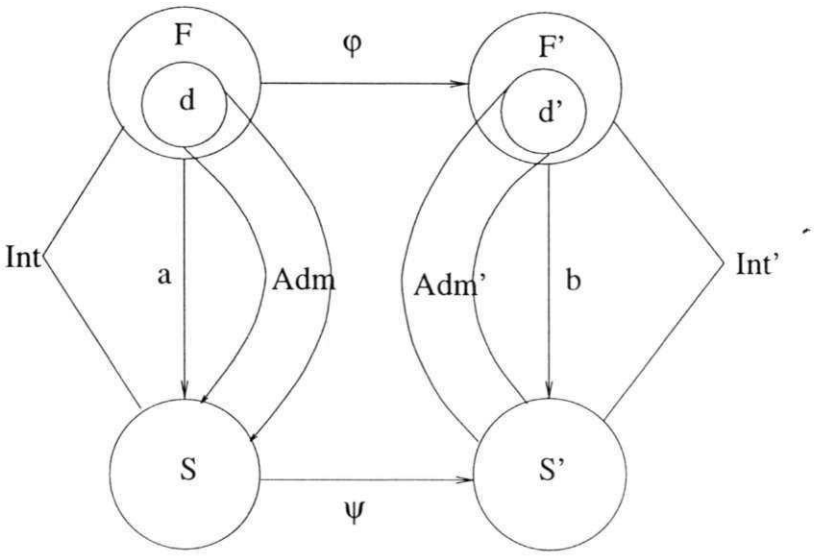

Figura 2.3: *Homomorfismo entre problemas*

O conceito de *homomorfismo entre problemas* é uma noção interessante, pois permite relacionar problemas diferentes, mas que *preservam a parte intencional.* Isso permite relacionar um determinado problema a uma "classe" de outros problemas que possuem a propriedade de preservar a parte intencional.

Será dada a seguir uma definição de subproblema.

Definição 2.10 [Lop81] : *Um problema* $\mathcal{P}' = \langle F', S', d', Int', Adm' \rangle$ *é um* **subproblema** *de um problema* $\mathcal{P} = \langle F, S, d, Int, Adm \rangle$, *que se denota por* $\mathcal{P}' \leq \mathcal{P}$, *sse*[5] $F' \subseteq F$, $S' \subseteq S$ *e as inserções* $i{:}F' \longrightarrow F$ *e* $j{:}S' \longrightarrow S$, *constituem um homomorfismo de* $\mathcal{P}'$ *em* $\mathcal{P}$ □

As garantias dadas para verificar se um problema $\mathcal{P}'$ é um subproblema de um problema $\mathcal{P}$, é dada pela seguinte proposição.

Proposição 2.1 [Lop81] : *Sejam* $\mathcal{P}'$ *e* $\mathcal{P}$ *dois problemas.* $P \leq P'$ *sse* $F' \subseteq F$, $S' \subseteq S$, $d' \subseteq d$, $Int' \subseteq Int$ *e* $\{b \in S^F \mid b(F') \subseteq S' \text{ e } b \angle F'^{[6]} \in Adm'\} \subseteq Adm$ □

[5] será utilizado "sse" por "se e somente se".

[6] é escrito "$b \angle F'$" por "b restrito F'"

Uma vez que foi definido o que é problema e como podem ser analisada as relacões entre eles, deve-se também definir o que é uma solução para problema. Uma solução h para um problema deve ser admissível e sempre que um elemento x satisfaz a condição de dados o par $(x, h(x))$ deve satisfazer a intenção do problema. Formalmente, a definição de solução para problema é dada a seguir.

Definição 2.11 [Lop81] : *Seja $\mathcal{P} = \langle F, S, d, Int, Adm \rangle$ um problema. Uma* **solução** *para $\mathcal{P}$ é uma função h de F em S (h:$F \longrightarrow S$) tal que $h \in Adm$ (h é admissível) e $\forall f \in d$ (h satisfaz a condição de dados) (f, $h(f)$) $\in Int$ (h satisfaz a condição do problema). Indica-se que "h resolve $\mathcal{P}$" ou "h é solução de $\mathcal{P}$" por $h \mid\vdash \mathcal{P}$.* □

Será apresentada a seguir a definição de *espaço de soluções*.

Definição 2.12 [Lop81] : *O* **Espaço de Soluções** *de um Problema $\mathcal{P}$ denotado por $\Sigma(\mathcal{P})$, é fornecido pelo seguinte conjunto: $\Sigma(\mathcal{P}) = \{h \in S^F \mid h \mid\vdash P\}$* □

O conceito mencionado acima é extremamente importante. Tal conceito tem muitas aplicações, uma destas está em que muitos sistemas especialistas são implementados fazendo-se uso da idéia de explorar o espaço de solução do problema, como exemplo cita-se sistemas como o "Soar"[7].

Uma forma utilizada no estudo dos métodos de determinação de soluções de problemas é a *semelhança*. Formalmente, pode-se definir *semelhança* entre problemas como se segue.

Definição 2.13 [Lop81] : *Sejam $\mathcal{P}$ e $\mathcal{P}'$ dois problemas. Seja $\beta : Int \longrightarrow Int'$ uma função. Diz-se que β é uma* **semelhança** *de $\mathcal{P}$ em $\mathcal{P}'$, denotado por $\mathcal{P} \xrightarrow{s} \mathcal{P}'$, se, e somente se, existirem funções $\phi : F \longrightarrow F'$ e $\psi : S \longrightarrow S'$, tais que, a seguinte propriedade é satisfeita: $\forall x \in d, \forall y \in S, (x, y) \in Int \rightarrow \beta(x, y) = (\phi(x), \psi(y)) \in Int'$*
□

[7] "Soar" tem sido desenvolvido para ser uma arquitetura cognitva geral. Trata-se de um ambiente para geração de sistemas especialistas baseados em regras, que além de gerar regras apresenta toda árvore semântica da geração automática do conhecimento. Para maiores informações a respeito do Soar recomenda-se os artigos: *The Soar Papers: Readings on Integrated Intelligence*, Rosenbloom, Laird, e Newell (1993), e *Unified Theories of Cognition*, Newell (1990).

2.4.1 Métodos de Resolução de Problemas

Decomposição de Problemas

Decomposição é um método bastante utilizado em várias áreas do conhecimento para resolver problemas das mais variadas naturezas, sendo um método seguro para abordar problemas resolvíveis, como foi mostrado em [Bed87]. Deve-se levar em conta que as estratégias de "divisão-e-conquista", "refinamento passo-a-passo", "projeto de cima para baixo", etc., são diferentes nomes utilizados para se referir à *decomposição.*

Um problema pode ser decomposto de duas maneiras:

(a) decomposição estrutural;

(b) decomposição n-ária.

Freqüentemente, um problema complexo se apresenta como uma junção de problemas que são subestruturas ou etapas deste. Na *decomposição estrutural* um problema é decomposto em problemas mais "*fáceis*". Uma vez encontrada a solução de cada um desses problemas mais *fáceis*, elas são tratadas de alguma maneira para apresentar a solução do problema original. Várias formas de tratar a decomposição estrutural são encontradas no trabalho de [OJ85].

A *decomposição n-ária* se inspira na idéia de recursividade, entendendo-se por recursividade a propriedade que tem alguns objetos de estarem compostos de diversas partes com propriedades de se converterem em totalidades, isto é, em objetos independentes. Um problema tem propriedades recursivas se for possível ser decomposto em partes. Mas para que tal problema seja assim resolvido, será preciso determinar um limite na recursividade, para a qual é necessário distingüir um subconjunto das possíveis partes do problema, que sejam suficientemente *simples* para aplicar diretamente uma solução. Por fim, os resultados são recombinados para obter o resultado do problema inicial; tudo isso tendo o cuidado que a condição de admissibilidade das soluções seja satisfeita.

A decomposição utilizada aqui é a *decomposição n-ária*, que se aplica da seguinte maneira: os dados (elementos *predicado fonte*) são particionados repetidamente por meio de algumas função de partição (*part*) até encontrarem-se numa forma *simples*. Quando isso ocorrer, encontra-se soluções *diretas* sobre esses elementos *simples*, encontrando-se os elementos correspondentes no co-domínio de resultados, e constróem-se, através de uma função de recombinação (*recomb*), os elementos do co-domínio de resultados correspondentes aos elementos originais do predicado fonte, tomando-se o cuidado de que a função resultante satisfaça o predicado de admissibilidade (Adm) do problema. A seguir uma definição que exprime isso é apresentada.

Definição 2.14 [Lop81] : *Seja* $\mathcal{P} = \langle F, S, d, Int, Adm \rangle$ *um problema. Uma* **decomposição n-ária** Δ_n *sobre* $\mathcal{P}$ *consiste de:*

- n funções unárias $part_i{:}F \longrightarrow F$, $i = 1 \ldots n$,
- uma função n-ária $recomb{:}S^n \longrightarrow S$
- uma função unária $direta{:}F \longrightarrow S$
- um predicado unário $simples \subseteq F$

tais que a função $h : F \longrightarrow S$ definida como segue:

$$h(f) = \begin{cases} direta(f), & \text{se } simples(f) \\ recomb(h(part_1(f)), h(part_2(f)), \ldots, h(part_n(f))) & \text{caso contrário} \end{cases}$$

é solução de $\mathcal{P}$, quando $h \in Adm$.□

A definição de decomposição acima envolve a função h que é, neste caso, *recursiva primitiva* se $part_i (1 \leq i \leq n), recomb, direta$ o forem e $simples$ for recursiva primitiva. [BL74].

Será dado a seguir um lema que associa uma ordem bem-fundada a um problema solúvel por decomposição.

Lema 2.1 [OJ85] : *Seja $\mathcal{P}$ um problema solúvel por decomposição. A relação $\preceq$ sobre F definida como abaixo*

$$x \preceq y \leftrightarrow (\exists k > 0 \mid part_k(y) = x \wedge y \notin simples). \qquad (2.2)$$

é uma ordem bem-fundada. □

Será estabelecido agora as condições necessárias e suficientes para um problema ser solúvel por decomposição, o que caracteriza tal classe de problemas.

Teorema 2.1 [OJ85] : *Seja $\mathcal{P}$ um problema solúvel por decomposição e $\preceq$ uma relação binária sobre F. Então F é bem-fundado[8] e Δ satisfaz as seguintes propriedades:*

(i) $(\forall x \in F),\ x \in simples \rightarrow (x, direta(x)) \in Int,$

(ii) $(\forall x \in F),\ x \notin simples \rightarrow part_i(x) \preceq x,\ i = 1, \ldots, n,$

(iii) $(\exists h \in Adm),$

$$h(x) = \begin{cases} direta(x), & \text{se } simples(x) \\ recomb(h(part_1(x)), h(part_2(x), \ldots, h(part_n(x))) & \text{caso contrário} \end{cases}$$

(iv) $(\forall x \in F),\ x \in d \wedge x \notin simples \rightarrow (\forall i = 1 \ldots n)[(part_i(x), h(part_i(x))) \in Int]$
$\rightarrow [(x, recomb(h{-}part(x)))^9 \in Int].$

Prova: Vê-se inicialmente que F é bem-fundado. De fato, a relação $\preceq$ do Lema 2.1 é uma ordem bem-fundada. Por definição (iii) é verificado. Ainda da definição de h tem-se que se $simples(x)$, $h(x) = direta(x)$ logo $(x, direta(x) \in Int$. Se $x \notin simples$

[8] Um conjunto F é bem-fundado se existe uma relação binária $\preceq$ (chamada *bem-fundada*) tal que não existe encadeamento decrescentemente infinito sobre os elementos de F (garante a finitude do processo).

[9] $h{-}part(x) \stackrel{def}{=} (h(part_1(x)), h(part_2(x)), \ldots, h(part_n(x)))$

então $part(x) \preceq x$, pela definição de $\preceq$. Obtém-se assim (i) e (ii). Será dada agora a prova do item (iv). Supondo então não, isto é:

$$x \in d, \tag{2.3}$$

$$x \notin simples, \tag{2.4}$$

$$y = h(part(x)), \tag{2.5}$$

$$(part(x), y) \in Int, \tag{2.6}$$

$$(x, recomb(y)) \notin Int. \tag{2.7}$$

Como $h \mid\vdash \mathcal{P}$ e 2.3 tem-se que

$$(x, h(x)) \in Int \tag{2.8}$$

De 2.7 e 2.8 tem-se

$$h(x) \neq recomb(y) \tag{2.9}$$

Da definição de h, de 2.4 e 2.9 obtem-se

$$recomb(h(part(x))) \neq recomb(y) \tag{2.10}$$

Assim, de 2.5 e 2.9 chega-se a

$$recomb(h(part(x))) \neq recomb(h(part(x))) \tag{2.11}$$

o que é um absurdo. Logo (iv) deve ser satisfeita. □

Corolário 2.1 [OJ85] (Teorema da Decomposição): *Seja $\mathcal{P}$ um problema, então Δ é uma decomposição sse valem as seguintes propriedades:*

(i) F é bem-fundado,

(ii) $(\forall x \in F),\ x \in simples \rightarrow (x, direta(x)) \in Int$,

(iii) $(\forall x \in F),\ x \notin simples \rightarrow part_i(x) \preceq x,\ i = 1, \ldots, n$.

(iv) $(\exists h \in Adm)$,

$$h(f) = \begin{cases} direta(f), & \text{se } f \in simples \\ recomb(h(part_1(f)), h(part_2(f)), \ldots, h(part_n(f))) & \text{caso contrário} \end{cases}$$

(v) $(\forall x \in F),\ x \in d \wedge x \notin simples \rightarrow (\forall i = i \ldots n)[(part_i(x), h(part_i(x))) \in Int] \rightarrow [(x, recomb(h{-}part(x))) \in Int]$

Prova: Direto do Teorema 2.1 e do Teorema da Decomposição □.

De acordo com os resultados apresentados, um problema é solúvel por decomposição se $\Delta = \langle direta, simples, part_1, \ldots, part_n, recomb, h \rangle$ satisfaz as propriedades (i)-(v) do corolário 2.1. Assim, a especificação de um problema solúvel por decomposição tem os seguintes passos:

1. especificar o *problema abstrato.*
2. estender o *problema abstrato* por acrescimos de pelo menos os símbolos
   * $direta$, símbolo relacional unário de F em S,
   * $simples$, símbolo predicado unário sobre F,
   * $part_i$, símbolo funcional unário sobre F,
   * $recomb$, símbolo funcional n-ário sobre S,
   * h, símbolo funcional de F em S,

* $\preceq$, símbolo predicado binário sobre F,

e por acréscimo dos axiomas (i)-(v) do corolário 2.1. Assim, pelos resultados apresentados aqui, o problema concreto que realiza este problema abstrato é solúvel por decomposição. □

Redução de Problemas

Uma maneira de abordar a resolução de problemas é a "redução de problemas". A idéia de redução de um problema para outro é expressa da seguinte maneira [Lop81]: "*quando não for possível resolver um problema, faz-se uma transformação do seu domínio fonte para o domínio fonte de outro problema que se pode resolver; determina-se uma solução deste e realiza-se uma transformação do co-domínio de resultados do segundo problema para o co-domínio de resultados do problema original. Esperando-se com isso encontrar a solução do problema dado*" (ver figura 2.4).

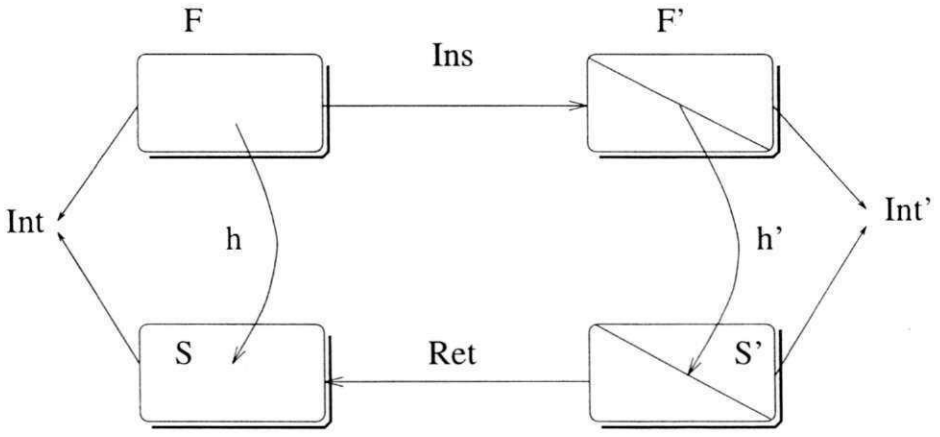

Figura 2.4: *Representação do processo de redução, tendo-se dois problemas $\mathcal{P}$ e $\mathcal{P}'$*

Definição 2.15 [Lop81] : *Uma* **redução** Γ *de um problema $\mathcal{P}$ para um problema $\mathcal{P}'$, consiste de duas funções* **Ins** *e* **Ret**, *com Ins:$F \longrightarrow F'$ e Ret:$S' \longrightarrow S$, tais que* $\forall f \in F, \forall s' \in S'\ (Ins(f), s') \in Int' \rightarrow (f, Ret(s')) \in Int$. □

Descrições de problemas podem assumir uma variedade de formas. Portanto, o problema de descrição significa *mudar de uma configuração para outra configuração.* Assim, dado um problema $\mathcal{P}$, pode existir um outro problema $\mathcal{P}'$, inclusive tendo domínios diferentes, que tenha funções admissíveis com a mesma propriedade do problema original.

Definição 2.16 [LCB88] : *Seja $\wp$ uma propriedade qualquer, e sejam os problemas $\mathcal{P} = \langle F, S, d, Int, Adm\rangle$ e $\mathcal{P}' = \langle F', S', d', Int', Adm'\rangle$, em que $Adm = (a \in S^F \mid \wp(a))$. Diz-se que Adm' herda a propriedade $\wp$ de Adm, denotado por $Adm' = Her(Adm, S'^{F'})$, sse $Adm' = (a' \in S'^{F'} \mid \wp(a'))$, isto é, se Adm' é definido de maneira análoga a Adm, mas em domínios diferentes* □

Para ilustrar o que diz a definição acima considere o seguinte exemplo: Seja $\mathcal{P}$ o problema do fluxo d'água (vazão) e $\mathcal{P}'$ o problema de corrente elétrica. Sabe-se que embora o domínio de dados e resultados desses problemas sejam diferentes, mas a maneira como eles são tratados é análoga, ou seja, a admissibilidade é preservada (ver figura 2.5). De acordo com a definição acima, Adm' herda a propriedade $\wp$ de Adm.

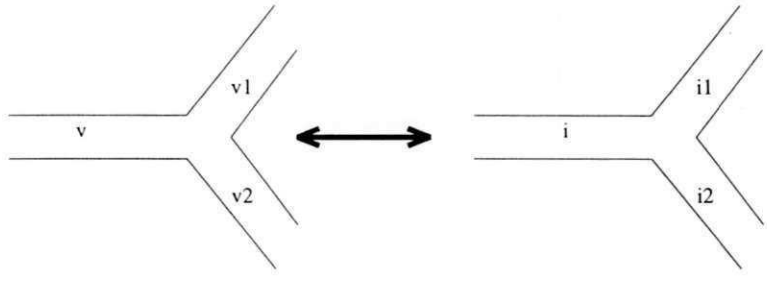

Figura 2.5: *fluxo d'água versus corrente elétrica*

Um *operador* de redução de problema transforma uma descrição de um problema em um conjunto de descrições do problema reduzido. A transformação será tal que as soluções de todos os problemas reduzidos implica numa solução do problema original. □

2.5 Uma Abordagem Axiomática para o Processo de Especificação de Problemas

O aprimoramento da descrição e o refinamento da linguagem do problema, requer o aperfeiçoamento da especificação do problema. Procura-se, então, assegurar formalmente o processo de especificação. Para isso, é usado como base o trabalho de Mahr e Makowsky [MM83], no qual se faz algumas adaptações para adequar ao enfoque da especificação de problemas - uma vez que esse não é o objetivo original do trabalho citado.

Tendo em vista que parte do processo de especificação de problemas dedica-se a manipular as várias interpretações contidas na situação problema, ou seja, em trabalhar a semântica que cada etapa envolvida na especificação da linguagem carrega em si, a seguir serão apresentadas definições envolvendo *Sistemas Semânticos*. É definido a seguir um sistema semântico utilizando tipos abstratos, adaptado de [MM83].

Definição 2.17 *(Sistema Semântico): Um* **sistema semântico** *consiste de um par* $(\mathcal{D}, \Phi)$, *em que*

- $\mathcal{D}$ *uma classe de descrições finitas;*
- $\Phi = (\mathcal{C}_i)_{i \in I}$ *uma família de classes de tipo(i)-estruturas, em que* $tipo : I \longrightarrow \mathcal{D}$ *associa cada índice* $i \in I$ *satisfazendo uma descrição* $tipo(i)$, *assegurado pelos quatro axiomas seguintes.* □

Axioma 2.1 *(Fechado sobre Isomorfismo): Dadas estruturas* $\mathcal{A}$, $\mathcal{B}$, *e uma classe* $\mathcal{C}_i$ *de* Φ, *então* $\mathcal{A} \approx \mathcal{B}$ *implica que* $\mathcal{A} \in \mathcal{C}_i$ *sse* $\mathcal{B} \in \mathcal{C}_i$ □

Este axioma diz meramente que as estruturas estão somente nos tipos isomórficos e não em sua representação particular.

Axioma 2.2 *(Fechado sobre Renomeação): Dada uma renomeação* $\rho : \tau \longrightarrow \mathcal{M}$ *e uma classe* $\mathcal{C}_i$ *com* $tipo(i) = \tau$, *então existe* $j \in I$ *com* $tipo(j) = \mathcal{M}$ *tal que* $\mathcal{A} \in \mathcal{C}_i$ *sse* $\mathcal{A}^\rho \in \mathcal{C}_j$ *para todas as* τ*-estruturas* $\mathcal{A}$ □

Este axioma diz que se podem mudar os nomes de relações e funções sem afetar a estrutura. Por exemplo, pode-se mudar da notação aditiva para a multiplicativa quando se procede com um grupo (estrutura) sem afetar o próprio grupo.

Axioma 2.3 *(Fechado sobre Interseção): Para todos os índices* $i, j \in I$ *existe um índice* $k \in I$ *tal que* $\mathcal{C}_k = \mathcal{C}_i \cap \mathcal{C}_j$ □

Este axioma garante que a interseção de duas especificações é novamente uma especificação.

Axioma 2.4 *(Classe Vazia): Para cada* $\tau \in \mathcal{D}$ *existe* $i \in \mathcal{D}$ *tal que o tipo*$(i) = \tau$ *e* $\mathcal{C}_i = \oslash$ □

Este axioma garante meramente que se pode especificar a classe vazia das τ-estruturas.

É preciso destacar que a definição de um sistema semântico corresponde exatamente à noção de *definibilidade* para linguagens de especificação, que é dado em [MM83]. Explicitamente: *"Para cada sistema semântico* $(\mathcal{D}, \Phi)$ *existe uma linguagem de especificação* L *tal que* k *está em* Φ *sse* k *for* L*-definível; e reciprocamente, a família das classes* L*-definíveis, indexada por especificações da linguagem de especificação* L*, forma um sistema semântico"*. Note-se que os axiomas de um sistema semântico não prevêm qualquer formalismo sintático para linguagens usadas nas especificações, mas possuem significação correta. Eles dizem que um sistema semântico:

1. Não é diferenciável entre estruturas isomórficas.
2. É fechado sobre renomeação de nomes de conjuntos, constantes, operações e relações.
3. É fechado sobre refinamento por junções ou uniões de especificações.
4. Possui proteção contra especificações inconsistentes.

O que se está fazendo aqui é apenas apresentando uma definição de uma lógica abstrata. As classes $\mathcal{C}_i$ são as classes definidas pelos "conjuntos de especificações" (os quais podem ser chamados simplesmente de *especificações*).

As definições e axiomas apresentados nesta seção proporcionam uma formalização sobre o processo de especificação envolvendo estruturas. As garantias dadas nessa formalização são refletidas no tratamento estrutural dado a questão "problema", bem como na linguagem de representação que envolve o problema.

As noções apresentadas neste capítulo serão usadas no desenvolvimento do tema explorado neste trabalho em capítulos posteriores.

Capítulo 3

Especificação de Problemas

3.1 Introdução

Foi apresentado no capítulo anterior conceitos e definições que serão úteis para garantir o mecanismo aqui exposto. Entretanto, uma vez que é tratado problema via refinamento de especificação, torna-se necessário abordar a questão da especificação.

Neste capítulo será dada uma noção geral de especificação, e também o que é a especificação de problema. Será apresentado, também, como se deve proceder para especificar um problema.

Para solucionar problemas torna-se necessário que todos os elementos essenciais estejam presentes na especificação do problema. Assim, neste capítulo será conduzido a questão da especificação, envolvendo a *Teoria Geral de Problemas* [Lop81], subdividindo a tarefa da especificação em etapas. A primeira dessas etapas consiste em trabalhar o enunciado da situação problema, procurando identificar que informações estão contidas nesse enunciado. Com isso, busca-se entender o problema para estabelecer uma estratégia de resolução. Numa segunda etapa, o problema é representado na forma de uma estrutura matemática. Nessa segunda etapa, é apresentada uma primeira linguagem para o problema. Numa terceira etapa, procurando embutir na especificação elementos que permitam uma maior percepção a respeito da questão problema, aperfeiçoa-se a linguagem até que todos os elementos necessários para especificar

o problema estejam presentes. Ao final dessas etapas o problema estará suficientemente compreendido e com uma especificação considerada *adequada* (ver figura 3.1).

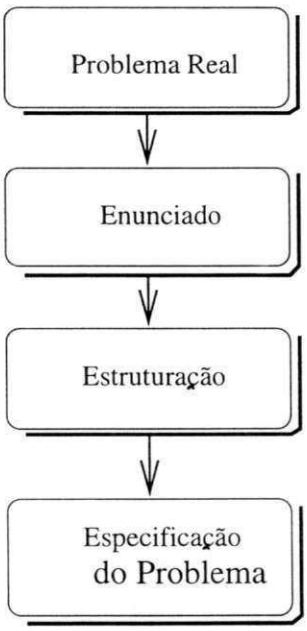

Figura 3.1: *Diagrama das etapas envolvidas na especificação do problema.*

3.2 O Enunciado de Problema

Um problema muitas vezes é enunciado de maneira pouco precisa se uma linguagem natural, não formal, foi usada. Especificar um problema é transformar essa descrição informal em uma descrição formal. Essa é uma tarefa freqüentemente difícil. Descrever um problema é uma tarefa que requer do especificador a capacidade de identificar se a descrição apresenta *informação suficiente*, *informação irrelevante* e *informação insuficiente* .

É notadamente sabido que o problema de representação é um dos complicadores da resolução de problemas. A elaboração da descrição de um problema provê uma medida relativamente direta de compreensão da estrutura do problema, pois o especificador

só poderá detectar qual informação é irrelevante ou está faltando no enunciado de um problema se ele souber quais as informações necessárias para resolvê-lo.

Assim, no processo de resolução de um problema o primeiro passo é compreender o problema a ser resolvido. Em notando que o primeiro passo para a resolução é a compreensão do problema, Polya [Pol78] enfatiza a necessidade de conhecer as principais partes do problema (a incógnita, os dados, as condições) antes de o "solucionador" se engajar na especificação do problema.

As pessoas que falham em detectar a informação irrelevante no enunciado de um problema muitas vezes incorporam na resolução do problema essa informação. Por exemplo, Low e Over [LO89] relatam uma experiência em que estudantes tentam resolver o seguinte problema: *"Amanda é 4 anos mais velha que Brenda, enquanto Brenda é 2 anos mais velha que David. Se as idades combinadas de Amanda e Brenda totalizam 28 anos, qual é a idade de Brenda?"*. A maior parte dos estudantes obteve como resposta para o problema 14 anos ou 18 anos de idade, presumivelmente influenciada pela proposição irrelevante *"Brenda é 2 anos mais velha que David"*. Os estudantes aparentemente cometeram erro na resolução porque faltou-lhes conhecimento da estrutura do problema.

Será dado a seguir exemplos que ilustram os diversos tipos de informação citados [LODM94] :

1. O comprimento de uma janela de vidro retangular é duas vezes a sua largura. A área da janela é de $98cm^2$. Quais são as dimensões da janela? [informação **suficiente**]

2. Carol deseja revestir uma bancada com azulejos. Ela deseja usar azulejos retangulares medindo $10cm$ por $5cm$. Quantos azulejos ela necessitará para revestir a bancada completamente? [informação **faltando**]

3. Uma piscina retangular de $30m$ por $20m$ tem uma borda de concreto de $3m$ de largura circulando toda a piscina. Qual é a área da borda se a piscina tem $5m$ de profundidade? [informação **irrelevante**]

Esta última situação problema é passível de ser resolvida, no entanto, seu enunciado contém informação irrelevante.

Algums aspectos comuns que devem ser analisados no enunciado de problemas são:

- Presença de elementos no enunciado que não dão informações importantes a qualquer característica do problema (contém informações irrelevantes, desnecessárias à resolução).
- Existência de uma (ou mais) característica do problema que não foi mencionada pelo enunciado (falta de informação).
- Presença de elementos no enunciado que possibilitam determinar uma característica do problema de mais de uma maneira distinta (*ambigüidade*).
- Presença de elementos no enunciado que necessitam usar características do problema ainda não definidas (informações faltando).
- Presença de elementos no enunciado que correspondem não a uma característica do problema, mas sim a uma possível solução.

Sweller menciona em seu trabalho [Swe88] que as bases para um conhecimento esquemático para a resolução de um problema é a familiaridade com as etapas envolvidas na resolução do problema e também com os elementos essenciais para resolvê-lo, e aponta [Swe89] que outro modo de instrução que melhora o problema da resolução é habilitada por focar a atenção na estrutura do problema.

Para compreender a estrutura do problema, é preciso que inicialmente se faça uma análise do enunciado do problema, em que se realiza uma compreensão dessa primeira verbalização, que se pode chamar de "aperfeiçoar o entendimento do problema", trabalhando os níveis de abstração que eventualmente envolvem a situação problema. A esse trabalho é dado o nome de "processo de abstração do problema" ou "estruturação do problema".

Enfocando-se a atenção na estrutura de problema, será apresentado como se pode realizar o processo de refinamento da descrição de problemas (estruturalmente), saindo

desde uma descrição pouco representativa até chegar a uma descrição estrutural com uma representação considerada *adequada.*

3.3 Especificação

Especificação pode ser conceituada como sendo uma descrição univoca ou conjunto de elementos que validam determinado objeto ou situação.

Sobre a descrição feita em linguagem natural, pode-se identificar os elementos que devem ser especificados.

Na especificação deve-se descrever todos os aspectos de um objeto de um modo finito, formal e não ambigüo.

Toda descrição formal deverá ser acompanhada por uma discussão da interpretação real do modelo matemático.

Será apresentado agora algumas das vantagens da especificação formal.

A vantagem primária da especificação formal sobre uma especificação informal é a possibilidade de se fazer uma avaliação a respeito das propriedades da especificação. Isso é feito por duas razões:

- Desde que as declarações sobre o "*comportamento do objeto*" (ações) podem ser descritas em uma linguagem formal, esse "*comportamento*" pode ser determinado se as declarações acerca dele são consistentes.
- O "*comportamento do objeto*" pode ser determinado quando uma especificação é completa (quando o modelo tiver sido completamente definido). Nesse ponto, deve ser possível responder todas questões com relação ao *comportamento* pertinente.

Uma outra importante característica do uso de especificações formais é a habilidade para escrever especificações de objetos em níveis crescentes de detalhes, e o raciocínio

sobre o relacionamento entre tais especificações. O processo de construção de uma especificação mais detalhada e mais abstrata é chamada "refinamento".

Especificações em computação tendem a ser usadas para três propósitos principais:

- O primeiro é para analisar os requisitos necessários para descrever úm *"sistema"*[1] pela construção de um modelo abstrato.
- O segundo propósito é para fornecer descrições concretas, não ambigüas, para *sistemas* abertos a revisões datalhadas.
- O terceiro propósito é para servir como guia para os desenvolvedores de *"sistemas"* descreverem as propriedades (elementos) necessárias para seus programas.

3.4 O que é Especificação de Problemas

Especificar um problema é descrever univocamente todos os elementos essenciais para compreender o problema. O especificador é quem deve, *a priori*, identificar e representar todos os elementos necessários para resolver o problema. Tornando-se, assim, importante a especificação formal pois a mesma ajudará ao especificador a não cometer possíveis enganos, como também irá complementar a descrição informal do problema. As técnicas de especificação geralmente se valem da experiência do especificador, de sua visão macroscópica sobre aquilo que ele quer especificar, tentando identificar comportamentos que possam conduzir a interpretações factuais de estruturas matemáticas (modelos) conhecidas.

O uso de uma notação formal não impede o uso da linguagem natural. De fato, especificar formalmente um problema, usualmente, torna a descrição informal do problema e a linguagem natural, mais clara e de melhor entendimento. Antes de começar a especificar um problema, deve-se ter claro qual é a situação problema, e quais são as restrições que o contexto impõe a situação. Um primeiro passo neste sentido é responder de maneira informal as questões sugeridas por Polya [Pol78]:

[1] o termo "*sistema*" representa aqui um conjunto de programas

- Quais são os dados? (descrever F)
- Quais são os resultados? (descrever S)
- Qual é a relação entre dados e resultados? (descrever Int)

A seguir, segundo [Lop81], se deve responder, de acordo com o contexto no qual se situa o problema, as seguintes perguntas:

- Quais são os dados relevantes ao problema? (descrever d)
- Quais são as formas de associar dados e resultados admissíveis? (descrever Adm)

A partir desse entendimento do problema, se inicia o processo de especificação propriamente dito.

Um problema $\mathcal{P}$ fica especificado por meio das componentes básicas F, S, d, Int, Adm, o que é realizado em três etapas:

- Concretização ou Modelagem do Problema.
- Especificação do Problema Concreto.
- Representação Usando uma Linguagem Formal.

3.4.1 Concretização ou Modelagem do Problema

A concretização ou modelagem do problema é feita estabelecendo uma interpretação factual da estrutura abstrata de um problema ($\mathcal{P} = \langle F, S, d, Int, Adm \rangle$). Uma tal interpretação é realizada aplicando uma função $\zeta : \mathcal{P} \longrightarrow \mathcal{M}$, em que $\mathcal{M}$ é um modelo de $\mathcal{P}$. $\mathcal{M}$ é chamado de *problema concreto.*

Nesse nível pode-se dizer, que de um modo geral a passagem da estrutura abstrata $\mathcal{P}$ para uma específica $\mathcal{M}$, modelo para $\mathcal{P}$, obedece ao esquema definido em [Bun76] (ver figura 3.2).

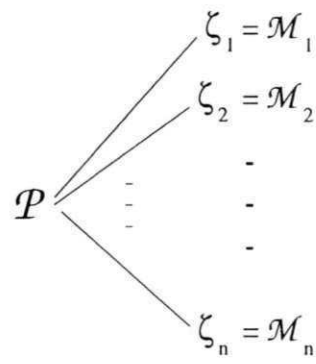

Figura 3.2: *Representação do processo de modelagem da estrutura abstrata de um problema.*

Cada $\mathcal{M}_i$ é um modelo para $\mathcal{P}$, ou seja, $\mathcal{M}_i$ é a imagem de alguma interpretação ζ_i que satisfaz a definição de problema.

Toda vez que se elabora ou enuncia um problema, na realidade se está dando um modelo ou uma interpretação do problema original que se busca resolver.

Definição 3.1 : *Seja $\mathcal{P}$ um problema e ζ uma interpretação. $\mathcal{M}$ é um modelo para $\mathcal{P}$, se por alguma interpretação ζ, $\mathcal{M}$ satisfizer a $\mathcal{P}$ (denotado por $\mathcal{M} \models \mathcal{P}(\zeta)$, indicando que "$\mathcal{P}$ é satisfeito pela interpretação ζ em $\mathcal{M}$")* □

Por exemplo, seja o problema de **ordenação** (Exemplo 2.2). É definida a seguinte interpretação:

Seja $\mathcal{P} = \langle F, S, d, Int, Adm \rangle$.

$\zeta : \mathcal{P} \longrightarrow$ ORD ($\mathcal{M}$ = ORD), em que:

- $\zeta(F) = \zeta(S) = Seq$ (sendo que *seq* representa uma seqüência de dados)
- $\zeta(d) = Seq'$ (sendo que *seq'* representa uma seqüência de dados finita)
- $\zeta(Int) = Ordenar$
- $\zeta(Adm) = Alg$ (Alg indica que a possível solução é um algoritmo).

Assim,

$$\text{ORD} = \langle Seq, Seq, Seq', Ordenar, Alg \rangle.$$

□

A parte intencional do problema abstrato (d, Int, Adm) deve ser descrita utilizando a linguagem da parte substantiva (F, S). Observe que F e S têm suas linguagens subjacentes.

3.4.2 A Especificação de um Problema Concreto $\mathcal{M}_i$

A partir da interpretação factual da estrutura abstrata de um problema, vem a especificação do problema concreto $\mathcal{M}$. A especificação do problema se constitue na especificação do modelo $\mathcal{M}$, e é dada pela especificação de suas componentes.

A escolha da estratégia de especificação via descrição algébrica, axiomática, usando linguagem de especificação (por exemplo, Z e ForMooZ), etc., é uma escolha do especificador segundo os seus recursos técnicos e teóricos e adequação da descrição ao problema específico.

Nesta etapa, as componentes F e S podem se apresentar como domínios de estruturas complexas, cujas componentes são transparentes a nível de estrutura abstrata Problema.

Numa segunda fase passa-se para a especificação do problema concreto $\mathcal{M}$, utilizando aqui para ilustrar as técnicas de especificação de tipos abstratos. Isto é, será especificado um tipo abstrato de dados T tal que $\mathcal{M}$ seja um modelo para T.

Definição 3.2 [OJ85] : *Um* **problema abstrato** $\mathcal{P}$ *é um tipo abstrato de dados, cuja linguagem tem os seguintes símbolos:*

- F : para o *sorte* de dados;
- S : para o *sorte* de resultados;

- d : símbolo predicado unário sobre o *sorte* F;
- Int : símbolo predicado binário sobre o *sorte* F e o *sorte* S;
- Adm : símbolo predicado unário de segunda ordem sobre $F \longrightarrow S$. □

As realizações de um problema abstrato $\mathcal{P}$ são os problemas concretos ($\mathcal{M}$) correspondentes.

Isso corresponde a interpretações factuais ζ da estrutura $\mathcal{P} = \langle F, S, d, Int, Adm \rangle$, obedecendo ao esquema ilustrado pela figura 3.2.

A parte intencional do problema abstrato (d, Int, Adm) deve ser descrita utilizando a linguagem da parte substantiva (F, S). Deve-se observar que F e S, por sua vez, são tipos abstratos de dados e como tais têm suas linguagens subjacentes. Ainda nesta fase a especificação dos tipos F e S podem ser estendidos (refinados) afim de facilitar a tarefa de especificar a parte intencional do problema abstrato.

Portanto, uma vez especificada a parte substantiva do problema utiliza-se a linguagem gerada para especificar a parte intencional. Porém, as vezes é necessário fazer alguma extensão nos tipos especificados afim de "melhor" especificar a situação problema. Feito isso, a especificação da situação problema fica então completa.

Existem três maneiras naturais de se estender um tipo abstrato de dados: por acrescentar definições de novos *sortes*, por acrescentar novas definições procedurais (por exemplo programas), e por estender definições não-procedurais (por exemplo fórmulas lógicas). Nestas condições pode-se obter uma especificação (parcial ou não) da situação problema.

Pode ser citado como referência de trabalhos que apresentam especificações utilizando tipos abstratos de dados, entre outros, os de [Lop81], [OJ85] e [Bed87].

Será apresentada a seguir uma definição de *especificação de problema*.

Definição 3.3 : *Uma* **especificação de problema** *é dada pela quádrupla*

$$Esp(\mathcal{P}) = \langle \mathcal{P}, \zeta(\mathcal{P}), L(\mathcal{P}), \Lambda \rangle, \tag{3.1}$$

em que

- $\mathcal{P} = \langle F, S, d, Int, Adm \rangle$ (Problema)
- $\zeta(\mathcal{P}) = \langle \zeta(F), \zeta(S), \zeta(d), \zeta(Int), \zeta(Adm) \rangle$ (Interpretação Problema)
- $L(\mathcal{P}) = L(F \cup S \cup d \cup Int \cup Adm)$ (Linguagem Problema)
- $\Lambda = \Lambda_F \cup \Lambda_S \cup \Lambda_d \cup \Lambda_{Int} \cup \Lambda_{Adm}$[2] (Axiomática). □

3.4.3 Representação Usando uma Linguagem Formal

Uma vez tendo especificado o problema, passa-se a representá-lo de maneira implementável, isto é, usando uma linguagem de 1ª ordem, ou uma linguagem de programação (Pascal, C, Fortran, etc.).

Embora essa etapa seja importante para a resolução do problema, não será abordada na sua totalidade, apenas se verá uma idéia geral do caminho a percorrer para tornar implementável a situação problema. Isso porque, na realidade não se faria nenhuma contribuição nesse domínio da implementação (não é o objetivo deste trabalho). Para implementar basta recorrer a qualquer das literaturas já existentes a esse respeito.

3.5 Primeiro Nível da Especificação de Problema

A primeira providência a ser tomada para especificar um problema é identificar na situação problema as componentes básicas da estrutura

$$\mathcal{P} = \langle F, S, Int \rangle. \tag{3.2}$$

[2] $\Lambda_F, \Lambda_S, \Lambda_d, \Lambda_{Int}$ e Λ_{Adm} são as axiomáticas do domínio de dados, domínio de resultados e da parte intencional do problema, respectivamente.

Assim, nesse primeiro nível de especificação busca-se captar os elementos considerados essenciais à "abstração problema".

Deve-se observar que no enunciado da situação problema são dadas algumas informações. Tais informações relatam quais os dados (domínio fonte), quais os resultados esperados, e sob que condições esses resultados têm que ser obtidos.

Num primeiro nível de especificação busca-se o entendimento do problema. Sugere-se, numa primeira etapa de especificação, a compreenção do problema como um todo, para em seguida ir aos detalhes. Para tal utiliza-se a relação (3.2).

Analisando os domínios fonte F e de resultados S, busca-se apresentar quais os elementos dos domínios do problema, de acordo com a intenção do mapeamento entre dados e resultados Cd do problema.

$$L_1(\mathcal{P}) = L(F \cup S).$$

A descrição apresentada na primeira etapa é ampla, e capta elementos irrelevantes ao problema. Por exemplo, será interessante considerar todo o "universo" do domínio fonte? Será que qualquer "solução" serve? Logo, torna-se necessário enriquecer mais a parte intencional do problema que é responsável pelo próprio sentido do problema.

Assim, numa segunda etapa, se faz necessário uma descrição mais elaborada.

Uma providência a ser tomada nesse sentido é restringir o domínio fonte. Devendo-se, portanto, especificar qual o domínio de dados para o problema tratado, em particular.

A restrição ao domínio fonte se reflete também sobre as intenções com que o problema será resolvido. Tem-se, então, que expressar as condições intencionais Int inerentes à situação problema tratada. Deve-se especificar também a restrição as funções de F em S que satisfazem Int e que não serão consideradas como admissíveis. Assim, especifica-se a parte intencional do problema, ou seja, d, Int e Adm.

Essa nova etapa de especificação deve ser realizada utilizando-se os elementos da linguagem de F e S. As vezes essa linguagem não é suficiente para descrever a parte

intencional do problema, sendo então necessário estender essa linguagem adicionando novos elementos a ela. Com isso, todos os componentes da estrutura

$$\mathcal{P} = \langle F, S, d, Int, Adm \rangle,$$

ficam especificados. O que significa que o problema está suficientemente entendido.

Para se chegar a especificar todos os componentes da estrutura do problema, pode ser preciso reespecificar a linguagem obtida inicialmente ou adicionar novos elementos a ela, até a linguagem descrever todos os elementos necessários para que o problema fique especificado, ou seja, obtém-se uma nova linguagem de tal modo que [3]

$$L_{i+1}(\mathcal{P}) = L_i(\mathcal{P}) \cup \{X\}, i \geq 1,$$

e

$$L_1(\mathcal{P}) \subseteq L_2(\mathcal{P}) \subseteq \ldots \subseteq L_n(\mathcal{P}),$$

em que $L_n(\mathcal{P})$ é a linguagem com todos os elementos necessários para que o problema fique especificado.

Mostrando como se deve proceder neste nível de especificação, é apresentado exemplo ilustrando a especificação de uma situação tida como problema.

Uma das situações tidas como problemas mais comuns na computação é a de ordenar objetos. Algumas situações problemas podem ser consideradas como situações particulares dessa. Uma forma geral de enunciar essas situações problemas pode ser dada como a seguir.

Exemplo 3.1 : Ordenação.

[3] $\{X\}$: representa os novos elementos da linguagem.

"*ordenar uma seqüência de itens*".

Analisando o enunciado da situação problema, pode-se identificar nele algumas informações respondendo, de maneira informal, as seguintes questões:

- Quais são os dados?

  - seqüência de itens

- Que resultados são esperados?

  - seqüência de itens

- Qual é a intenção de resolução?

  - ordenar uma seqüência.

Uma estruturação para a situação problema acima (concretização), pode ser dada pela relação $\mathcal{P} = \langle F, S, Int \rangle$, em que

$F = S$: $SEQITEM$ ($SEQITEM$ representando uma "seqüência de itens")[4].

Int: $Ordenar$ ($Ordenar$ representa a intenção de "ordenar")

A primeira etapa a ser realizada é a especificação dos elementos de F e S.

Inicialmente, supõe-se que um tipo $ITEM$ tenha sido especificado.

Especificando $SEQITEM$, tem-se:

$\lambda : \emptyset \longrightarrow SEQITEM$ (seqüência vazia)

$* \ \lambda \in SEQITEM.$

$inserir : ITEM \times SEQITEM \longrightarrow SEQITEM$

[4] $SEQITEM$ é um tipo SEQ parametrizado.

* $inserir(item, seq) \neq seq$

* $inserir(item, seq) = inserir(item, seq') \rightarrow seq = seq'$

* $inserir(item, inserir(item', seq)) = inserir(item', inserir(item, seq)) \rightarrow item = item'$.

$vazio : SEQITEM \longrightarrow Bool$[5]

* $vazio(\lambda)$

* $seq \in SEQITEM \wedge item \in ITEM \rightarrow \neg vazio(inserir(item, seq))$

Assim, a linguagem do problema até o momento fica do seguinte modo:

$$L_1(\mathcal{P}) = \langle \lambda, inserir, vazio, verd, falso \rangle. \tag{3.3}$$

Uma vez especificado como se obtém os elementos dos domínios F e S ($SEQITEM$), deve-se especificar a intenção do problema, que é ordenar os elementos da seqüência ($Ordenar$). Para isso, precisa-se de uma ordem $\leq$ (Total) de $ITEM's$.

Inicialmente $\leq$ deve obedecer as propriedades:

* $x \in ITEM, x \leq x$ (reflexiva)

* $x, y \in ITEM, x \leq y \wedge y \leq z \rightarrow x \leq z$ (transitiva)

* $x, y \in ITEM, x \leq y \wedge y \leq x \rightarrow x = y$ (antisimétrica)

* $x, y \in ITEM, x \leq y \vee y \leq x$ (totalidade)

Segue então, numa segunda etapa, a especificação de $Ordenar$, ou seja,

$boa : SEQITEM \longrightarrow Bool$

[5] $verd, falso \in Bool$, em que $Bool$ é um tipo booleano.

* $boa(\lambda)$

* $[seq \in SEQITEM \wedge x, y \in ITEM] \mid boa(inserir(x, seq)) \wedge x \leq y \rightarrow boa(inserir(y, inserir(x, seq))$

* $x \in ITEM \mid boa(inserir(x, seq)) \rightarrow boa(seq)$.

Entretanto, essa especificação ainda não é suficiente para satisfazer a intenção do problema. Torna-se necessário ainda especificar condições que venham a garantir:

(a) que uma seqüência seja finita;

(b) que dadas duas seqüências, elas possuem os mesmos elementos (não necessariamente na mesma ordem, considerando elementos repetidos);

(c) se uma possível solução para o problema é admissível.

A especificação dos itens acima é feita acrescentando-se novos elementos à especificação de $SEQITEM$ (refinando $SEQITEM$).

A especificação da garantia sugerida pelo item (a) será dada da seguinte forma:

Seja $seq, seq' \in SEQITEM$

$finita : SEQITEM \longrightarrow Bool$

* $finita(\lambda)$

* $finita(seq) \wedge x \in ITEM \rightarrow finita(inserir(x, seq))$

* $x \in ITEM \mid seq = inserir(x, seq') \wedge finita(seq) \rightarrow finita(seq')$.

A especificação da garantia sugerida no item (b), não é possível de ser realizada apenas com os elementos presentes até o momento. Assim, tem-se que especificar novos elementos a SEQITEM.

Seja $seq, seq', seq'' \in SEQITEM$

$concat : SEQITEM \times SEQITEM \longrightarrow SEQITEM$

* $concat(\lambda, seq) = seq$

* $concat(seq, \lambda) = seq$

* $concat(seq, concat(seq', seq'')) = concat(concat(seq, seq'), seq'')$

* $concat(seq, inserir(item, \lambda)) = inserir(item, seq)$

* $concat(seq, inserir(item, seq')) = inserir(item, concat(seq, seq'))$

Especificando o que foi sugerido no item (b), tem-se o seguinte:

$mesma : SEQITEM \times SEQITEM \longrightarrow Bool$

* $(seq, seq' \in SEQITEM), seq = seq' \rightarrow mesma(seq, seq)$

* $mesma(\lambda, seq) = vazio(seq)$

* $mesma(seq, seq') = mesma(seq', seq)$

* $mesma(seq, seq') \wedge mesma(seq', seq'') \rightarrow mesma(seq, seq'').$

* $mesma(inserir(x, seq), inserir(x, seq')) \rightarrow mesma(seq, seq')$

* $mesma(concat(inserir(x, seq), seq'), concat(inserir(x, t), t')) \rightarrow$
  $mesma(concat(seq, seq'), concat(t, t'))$

* $mesma(inserir(x, seq), inserir(y, seq)) \rightarrow x = y.$

Portanto, o problema de "ordenação de seqüência de itens" fica especificado completamente, a menos do item (c) que será dado como um predicado de segunda ordem Alg. A especificação de Alg é a sua própria concretização: Algoritmos que representam funções de $SEQITEM$ em $SEQITEM$ ($Alg : SEQITEM \longrightarrow SEQITEM$).

Então, nessa etapa de especificação o problema de "ordenação de uma seqüência de itens" possue como elementos de sua linguagem:

$$L_2(\mathcal{P}) = L_1(\mathcal{P}) \cup \{\leq, boa, finita, concat, mesma, Alg\}.$$

Nota-se, também, que a representação para o problema acima é ampla e capta elementos irrelevantes ao problema tratado. Assim, como próxima etapa da especificação vem a descrição da parte intencional, descrevendo as componentes d, Int e Adm (Definição 2.7). Para isso, será utilizado os elementos que compõem a linguagem $L_2(\mathcal{P})$. Desse modo, tem-se que

- $(seq \in SEQITEM), seq \in d$ sse $finita(seq)$.
- $(seq, seq' \in SEQITEM), (seq, seq') \in Int$ sse $mesma(seq, seq') \wedge boa(seq')$.
- $\forall h \in S^F$, $h \in Adm$ sse $Alg(h)$. □

Com isso, todos os componentes da estrutura

$$\mathcal{P} = \langle F, S, d, Int, Adm \rangle,$$

foram descritos, o que significa que o problema está devidamente compreendido.

Nos exemplos acima, o que se fez foi a "*modelagem*" ou "*concretização*" do problema, que consiste em estabelecer uma interpretação factual da estrutura abstrata do problema $\mathcal{P}$. Por exemplo, uma interpretação para o problema de "Ordenação" foi dada da seguinte maneira:

$\zeta : \mathcal{P} \longrightarrow \mathcal{M}$[6], em que

$\zeta(F) = \zeta(S) = SEQITEM$ (seqüência de itens);

$\zeta(d) = SEQ'$ (seqüência finita de itens);

$\zeta(Int) = Ordenar$ (seqüência ordenada);

$\zeta(Adm) = Alg$ (Algoritmo),

[6]Lembrando que "$\zeta : \mathcal{P} \longrightarrow \mathcal{M}$" significa que $\mathcal{M}$ é modelo para $\mathcal{P}$ segundo uma interpretação ζ (ver definição 3.1).

obedecendo ao esquema apresentado na figura 3.2.

Com esse exemplo acima, fica encerrado o primeiro nível da especificação de problema.

Entretanto, fica a seguinte indagação: "Esta especificação é suficiente para especificar uma solução?"

Capítulo 4

A Solução de Problemas via Refinamento da Especificação

4.1 Introdução

No primeiro nível foi especificado elementos da linguagem de problema que permitiram descrever as componentes estruturais de problema (Definição 2.7).

Mas, será que apenas com a linguagem apresentada no primeiro nível de especificação (Capítulo 3) é possível apresentar uma solução para o problema tratado? Pode ser preciso reespecificar a linguagem ou adicionar novos elementos a ela, até a linguagem descrever todos os elementos necessários para resolver o problema (ver Figura 4.1).

Além dos elementos estabelecidos pela linguagem do primeiro nível de especificação, será visto neste capítulo que os novos elementos necessários para apresentar uma solução devem ser obtidos da maneira mais natural possível. Contudo, pode-se também especificar os elementos para uma solução de tal maneira que venham a concretizar uma estratégia de solução estabelecida pelo solucionador, e que eventualmente também pode ser feita utilizando um método de resolução auxiliar (como por exemplo, decomposição *n-ária*).

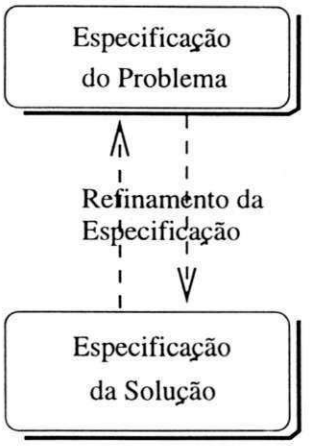

Figura 4.1: *Completando o diagrama para a resolução do problema.*

4.1.1 Segundo Nível da Especificação de Problema

Para responder a última indagação formulada no capítulo anterior, tem-se que analisar o que é preciso para a solucionar o problema.

Pensando-se na intensão de resolução da situação problema tratada no capítulo anterior de "ordenar uma seqüência", vem de maneira natural a idéia de comparar os itens da seqüência e de fazer uma permutação (troca) entre os itens da seqüência de tal modo a deixá-la ordenada (isso independe da estratégia de solução).

É trivial se pensar em começar comparando os dois primeiros itens da seqüência.

Primeiramente, torna-se necessário, antes de comparar, especificar como obter os dois primeiros intes de uma seqüência qualquer. Feito isso, passa-se então a compará-los.

Para obter os dois primeiros itens de uma seqüência é suficiente especificar acrescentar a linguagem um elemento que apresente o primeiro item de uma seqüência ($prim$) e um elemento que apresente uma seqüência sem o primeiro item ($resto$), pois o segundo item é o primeiro do resto da seqüência.

Seja $x, y \in ITEM$ e $seq \in SEQITEM$.

$prim : SEQITEM \longrightarrow ITEM$

* $\neg prim(\lambda) = x$[1]

* $seq \neq \lambda \rightarrow prim(inserir(seq, y)) = prim(seq)$

* $prim(inserir(\lambda, y)) = y$

* $seq \neq \lambda \rightarrow prim(concat(seq, seq')) = prim(seq)$.

$resto : SEQITEM \longrightarrow SEQITEM$

* $\neg resto(\lambda) = seq$

* $resto(inserir(\lambda, y)) = \lambda$

* $seq \neq \lambda \rightarrow resto(inserir(seq, y)) = inserir(resto(seq), y)$

Para comparar dois itens será usado $\leq$ (ordem entre itens) especificado no Capítulo 3. Entretanto, dependendo do resultado da comparação, será preciso que se faça permuta (troca) com os itens. Para facilitar o processo de permuta, se procederá trocando apenas o primeiro com o segundo item da seqüência. Para isso, deve-se especificar um novo elemento que será denominado de "troca".

$troca : SEQITEM \longrightarrow SEQITEM$

* $\neg resto(seq) \neq \lambda \rightarrow troca(seq) = concat(inserir(inserir(\lambda, prim(resto(seq))),$-
  $prim(seq), resto(seq)))$

Será que com esses novos elementos já é possível pensar num esquema de solução para este problema?

Pensando-se numa solução apenas utilizando os elementos contidos atualmente na linguagem do problema, pode-se apresentar o seguinte esquema:

* Inicialmente, apresenta-se uma seqüência. Em seguida, se faz uma análise para saber se a seqüência está ordenada ou não.

[1]será usada esta notação daqui em diante para indicar ação não válida.

* Se a seqüência estiver ordenada o resultado é apresentado.

* Se a seqüência não estiver ordenada, analisa-se, para facilitar, os dois primeiros itens da seqüência. Caso estejam ordenados entre si, passa-se a fazer novamente a análise da seqüência, mas agora sem olhar para o primeiro item.

* Se os dois primeiros itens da seqüência não estiverem ordenados, então se faz uma permuta com os dois itens. Feito isto, passa-se a novamente analisar a seqüência, retirando-se o primeiro item.

Este processo segue até analisar todos os itens da seqüência.

Entretanto, pode ser que ao final de simplesmente realizar os passos acima, a seqüência ainda não esteja ordenada. Tornando-se necessário, então, que o processo acima descrito seja realizado até que a seqüência em algum momento figue ordenada.

Para ilustrar o esquema de solução acima, veja o exemplo dado a seguir:

Seja uma seqüência $seq = \{8, 10, 7, 5, 9\}$

1^o passo: $\{8, 10, 7, 5, 9\}$, seqüência não ordenada.

2^o passo: os dois itens ordenados,

$seq = \{10, 7, 5, 9\}$

3^o passo: $\{10, 7, 5, 9\}$, seqüência não ordenada.

4^o passo: os dois primeiros itens não estão ordenados.

fazer permuta de 10 com 7 $\rightarrow \{7, 10, 5, 9\}$

$seq = \{10, 5, 9\}$

5^o passo: $\{10, 5, 9\}$, seqüência não ordenada.

6^o passo: os dois primeiros itens não estão ordenados.

fazer permuta de 10 com 5 $\rightarrow \{5, 10, 9\}$

$seq = \{10, 9\}$

7º passo: $\{10, 9\}$, seqüência não ordenada.

8º passo: os dois primeiros itens não estão ordenados.

fazer permuta de 10 com 9 $\rightarrow \{9, 10\}$

$seq = \{10\}$

9º passo: $\{10\}$, seqüência ordenada.

Resultado destes passos é $seq = \{8, 7, 5, 9, 10\}$.

A seqüência ainda não está ordenada. Assim, é necessário novamente aplicar o esquema de solução, agora sobre a seqüência

$seq = \{8, 7, 5, 9, 10\}$

Desse modo,

1º passo: $\{8, 7, 5, 9, 10\}$, seqüência não ordenada.

2º passo: os dois primeiros itens não estão ordenados.

fazer permuta de 8 com 7 $\rightarrow \{7, 8, 5, 9, 10\}$

$seq = \{8, 5, 9, 10\}$

3º passo: $\{8, 5, 9, 10\}$, seqüência não ordenada.

4º passo: os dois primeiros itens não estão ordenados.

fazer permuta de 8 com 5 $\rightarrow \{5, 8, 9, 10\}$

$seq = \{8, 9, 10\}$.

5^o passo: $\{8, 9, 10\}$, seqüência ordenada.

Resultado destes passos é $seq = \{7, 5, 8, 9, 10\}$

Como dá para notar, a seqüência ainda não está ordenada. Assim, o esquema de solução estabelecido tem que ser aplicado novamente, agora sobre a seqüência

$seq = \{7, 5, 8, 9, 10\}$.

Ou seja,

1^o passo: $\{7, 5, 8, 9, 10\}$, seqüência não ordenada.

2^o passo: os dois primeiros itens não estão ordenados.

fazer permuta de 7 com 5 $\rightarrow \{5, 7, 8, 9, 10\}$

$seq = \{7, 8, 9, 10\}$

3^o passo: $\{7, 8, 9, 10\}$, seqüência ordenada.

Com o resultado destes passos obtém-se $seq = \{5, 7, 8, 9, 10\}$, que é o resultado desejado.

Como não se sabe quantos passos serão necessários para apresentar o resultado desejado para uma seqüência qualquer, será interessante utilizar neste esquema de solução a idéia de recursividade.

Assim, uma formulação para o esquema de solução apresentado acima para utilizar apenas os elementos especificados até o momento, pode ser dado da seguinte maneira[2]:

$$h(seq) = \begin{cases} seq, & \text{se } boa(seq) \\ h(prim(seq), h(resto(seq)) & \text{se } prim(seq) \leq prim(resto(seq)) \\ h(prim(resto(seq)), h(resto(troca(seq)))) & \text{se } prim(seq) > prim(resto(seq)) \end{cases}$$

[2] "$x > y$" abrevia "$\neg x \leq y$".

Isto mostra como se pode obter a resolução de uma situação problema utilizando o refinamento como base para obter e especificar os elementos necessários para solucioná-lo.

Percebe-se que nesta especificação o caminho da solução seguiu de maneira natural até o elemento "*troca*", a partir daí tornou-se necessário a interferência do solucionador, entrando em ação a sua macrovisão sobre o problema, que o possibilitou perceber que apenas com elementos presentes até o momento na linguagem já se poderia chegar a uma solução.

Deve-se frisar também que o esquema estabelecido para essa solução foi obtido sem nenhuma estratégia de solução pré-estabelecida. Devido a isso, a solução apresentada não é a mais eficiente para essa situação problema.

Apesar da solução estabelecida acima ter sido obtida sem estabelecer previamente uma estratégia de solução, uma outra meneira de apresentar a solução de uma determinada situação problema é seguindo-se uma estratégia de solução específica e/ou método de solução. Observe que fazendo isso, o processo de refinamento fica direcionado e a solução obtida pode ser mais eficiente. Assim, a especificação dos elementos será realizada buscando dar subsídio para que a solução do problema seja dada de acordo com aquela estratégia e/ou método escolhido.

Para ilustrar isso, será apresentado a seguir especificações de soluções, seguindo ainda para deixar mais explícito a situação problema de "ordenar uma seqüência".

Por exemplo, se pensar na estratégia do algoritmo seqüêncial para resolver o problema de "*ordenar uma seqüência de itens*" é claro que se deve acrescentar uma função

$$menor : SEQITEM \longrightarrow ITEM$$

que dê o menor elemento da seqüência. Logo, deve-se "refinar" a especificação do problema incorporando essa função.

Mas, antes de especificar o elemento acima, primeiro é necessário especificar alguns outros elementos adicionais como *primeiro* e *resto*, necessários para especificar "*menor*".

Seja $x, y \in ITEM$ e $seq \in SEQITEM$.

$primeiro : SEQITEM \longrightarrow ITEM$

* $\neg primeiro(\lambda) = x$

* $seq \neq \lambda \rightarrow primeiro(inserir(seq, y)) = primeiro(seq)$

* $primeiro(inserir(\lambda, y)) = y$

* $seq \neq \lambda \rightarrow primeiro(concat(seq, seq')) = primeiro(seq)$.

$resto : SEQITEM \longrightarrow SEQITEM$

* $\neg resto(\lambda) = seq$

* $resto(inserir(\lambda, y)) = \lambda$

* $seq \neq \lambda \rightarrow resto(inserir(seq, y)) = inserir(resto(seq), y)$

Especificando a função *menor*:

* $\neg menor(\lambda) = x$

* $primeiro(resto(seq)) \leq primeiro(seq) \rightarrow menor(seq) = menor(resto(seq))$

* $primeiro(seq) \leq primeiro(resto(seq)) \rightarrow menor(seq) =$
  $menor(inserir(primeiro(seq), resto(resto(seq))))$.

Assim, tem-se agora as ferramentas suficientes para usar o algoritmo seqüêncial $(AS)^3$:

$$AS(seq) = \begin{cases} seq, & \text{se } seq = \lambda \\ concat(SEQUI, AS(concat(seq', seq'')) & \text{se } seq = concat(SEQUI2, seq'') \end{cases}$$

[3] "$SEQUI$" abrevia "$inserir(menor(seq), \lambda)$", "$SEQUI2$" abrevia "$inserir(menor(seq), seq')$".

Poderia-se, também, ter usado um método de solução, como por exemplo a desomposição, para resolver esta situação problema. Neste caso se deveria complementar a linguagem com os elementos necessários para a aplicação do método.

O caminho sugerido aqui para solucionar o problema de "ordenação" é verificar a possibilidade de quebrá-lo.

Analisando a intenção do problema, que é "ordenar", e a estratégia estabelecida, pode-se apresentar o seguinte esquema de solução, utilizando *decomposição n-ária*:

- Se uma seqüência $seq \in F$ fornecida for *boa* então obtém-se de maneira *direta* uma solução, encontrando-se elemento correspondente no co-domínio de resultados,
- Se a seqüência não for *boa*, a seqüência *seq* deve ser quebrada. Isso pode ser realizado, repetidamente, por meio de funções de partição ($part_i : F \longrightarrow F$) até que fiquem reduzidas a seqüências consideradas *boa*. Quando isso ocorrer, obtém-se soluções *diretas* sobre esses elementos, encontrando-se os elementos correspondentes no co-domínio de resultados.
- Uma vez encontrado os elementos correspondentes das partições dos dados no co-domínio de resultados, constróem-se, por meio de uma função de recombinação ($recomb : S^n \longrightarrow S$), os elementos do co-domínio de resultados correspondentes aos elementos do predicado fonte, tomando-se o cuidado de que a função resultante satisfaça ao predicado de admissibilidade (Adm) do problema.

Assim, uma solução $h : F \longrightarrow S$ (Definição 2.11), aplicando o que foi descrito acima, pode ser apresentada da seguinte forma[4]:

$$h(seq) = \begin{cases} direta(seq), & \text{se } boa(seq) \\ recomb(h(part_1(seq)), h(part_2(seq)), \ldots, h(part_n(seq))) & \text{caso contrário.} \end{cases}$$

[4] *boa* é equivalente a *simples* da definição 2.14.

As condições para que o método funcione são dadas pelo Teorema 2.1.

Concretizando uma estratégia de resolução, como por exemplo *quicksort*, utilizando os elementos da linguagem acima, observa-se o seguinte:

Na estratégia *quicksort* se faz o particionamento tomando o primeiro elemento do vetor e, por comparação e trocas sucessivas fazendo com que os menores passem para a sua esquerda e os maiores para sua direita. Ou então por transferência direta para vetores novos $part_1, part_2$ e $part_3$.

Seja $seq = (f_1, f_2, \ldots, f_{i-1}, f_i, f_{i+1}, f_{i+2}, \ldots, f_n)$ um vetor. Pode-se particioná-lo em três partes de tal forma que todos os elementos em $part_1$ sejam menores ou iguais a f_i, $part_2$ seja igual a f_i, e todos os elementos em $part_3$ maiores do que f_i, ou seja,

$part_1 = \{f_j : f_j \leq f_i \wedge j \neq i\}$, em que $i, j = 1, \ldots, n$

$part_2 = f_i$, para algum $i = 1, \ldots, n$

$part_3 = \{f_i : f_i < f_j \wedge j \neq i\}$, em que $i, j = 1, \ldots, n$.

A classificação (ordenação) de $part_2$ e $part_3$ será realizada repetidamente até que todos os elementos das "partições" estejam ordenados. Quando essas partições ordenadas forem obtidas, recombina-se os três segmentos $part_1, part_2$ e $part_3$ para apresentar o vetor seq ordenado.

Para a especificação do exposto acima, torna-se necessário estender (refinar) a linguagem do problema, acrescentando novos elementos a especificação de $SEQITEM$. Assim, fazendo uma especificação da estratégia com os elementos estabelecidos para a linguagem do problema até o momento, tem-se o seguinte:

Seja $seq \in SEQITEM$,

$part_1 : SEQITEM \longrightarrow SEQITEM$

* $primeiro(seq) \leq primeiro(resto(seq)) \rightarrow$

  $part_1(seq) = part_1(concat(inserir(primeiro(seq), \lambda), resto(resto(seq))))$

* $primeiro(resto(seq)) \leq primeiro(seq) \rightarrow$

$part_1 = concat(inserir(primeiro(resto(seq)), \lambda), part_1(concat(inserir(primeiro(seq),$
$resto(resto(seq)))))$

$part_2 : SEQITEM \longrightarrow SEQITEM$

$* \; part_2(seq) = inserir(primeiro(seq), \lambda)$

$part_3 : SEQITEM \longrightarrow SEQITEM$

$* \; primeiro(seq) > primeiro(resto(seq)) \rightarrow$

$part_3(seq) = part_3(concat(inserir(primeiro(seq), \lambda), resto(resto(seq)))$

$* \; primeiro(resto(seq)) > primeiro(seq) \rightarrow$

$part_3(seq) = concat(inserir(primeiro(resto(seq)), \lambda), part_3(concat(inserir(primeiro($
$\lambda), resto(resto(seq)))))$

$direta : SEQITEM \longrightarrow SEQITEM$

$* \; direta(seq) = seq$

$recomb : SEQITEM^n \longrightarrow SEQITEM$

$* \; recomb(part_1(seq), part_2(seq), part_3(seq)) =$

$concat(concat(part_1(seq), part_2(seq)), part_3(seq)).$

Com isso, a linguagem para a resolução desta situação problema fica estabelecida da seguinte maneira:

$$L_3(\mathcal{P}) = L_2(\mathcal{P}) \cup \{primeiro, resto, menor, direta, part_1, part_2, part_3, recomb\}. \quad (4.1)$$

Uma vez que a liguagem $L_3(\mathcal{P})$ possue todos os elementos necessários para resolver o problema, então pode-se considerar essa linguagem suficiente para especificar o problema.

□

$part_1 = concat(inserir(primeiro(resto(seq)), \lambda), part_1(concat(inserir(primeiro(seq),$
$resto(resto(seq)))))$

$part_2 : SEQITEM \longrightarrow SEQITEM$

$* \; part_2(seq) = inserir(primeiro(seq), \lambda)$

$part_3 : SEQITEM \longrightarrow SEQITEM$

$* \; primeiro(seq) > primeiro(resto(seq)) \rightarrow$

$part_3(seq) = part_3(concat(inserir(primeiro(seq), \lambda), resto(resto(seq)))$

$* \; primeiro(resto(seq)) > primeiro(seq) \rightarrow$

$part_3(seq) = concat(inserir(primeiro(resto(seq)), \lambda), part_3(concat(inserir(primeiro($
$\lambda), resto(resto(seq)))))$

$direta : SEQITEM \longrightarrow SEQITEM$

$* \; direta(seq) = seq$

$recomb : SEQITEM^n \longrightarrow SEQITEM$

$* \; recomb(part_1(seq), part_2(seq), part_3(seq)) =$

$concat(concat(part_1(seq), part_2(seq)), part_3(seq)).$

Com isso, a linguagem para a resolução desta situação problema fica estabelecida da seguinte maneira:

$$L_3(\mathcal{P}) = L_2(\mathcal{P}) \cup \{primeiro, resto, menor, direta, part_1, part_2, part_3, recomb\}. \quad (4.1)$$

Uma vez que a liguagem $L_3(\mathcal{P})$ possue todos os elementos necessários para resolver o problema, então pode-se considerar essa linguagem suficiente para especificar o problema.

□

4.2 Exemplo

Deve-se frisar que esta metodologia não foi estabelecida apenas solucionar a situação problema de "ordenar um seqüência", mas também para outras situações problemas, como por exemplo a que vem a seguir.

A situação problema que a partir de agora será trabalhada usando refinamento da especificação trata-se do já bem conhecido problema da "Torre de Hanoi". Uma versão para este quebra-cabeça pode ser declarada como segue.

Exemplo 4.1 *:* **Torre de Hanoi**

O problema da Torre de Hanoi é baseado em um jogo consistindo de três pinos e um conjunto de discos graduados por tamanho. O jogo inicia com os discos empilhados em tamanho decrescente no primeiro pino (configuração "inicial"). O objetivo é mover os discos para o terceiro pino mantendo a ordem (configuração "meta"), sujeito as condições: (1) somente um disco pode ser movido por vez; (2) em nenhum momento um disco pode ser colocado sobre (no topo) um disco menor. Todos os três pinos podem ser usados, e um disco pode ser movido diretamente de qualquer pino para qualquer outro (ver figura 4.2).

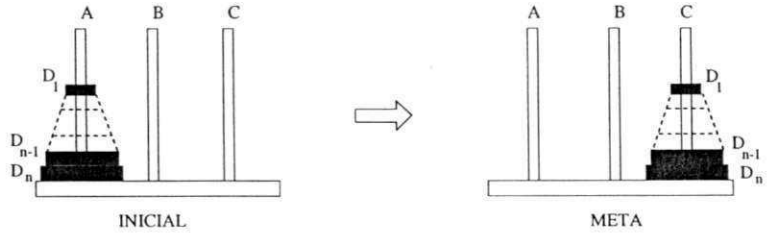

Figura 4.2: *Quebra-cabeça "Torre de Hanoi", configuração "inicial" e "meta".*

Num primeiro nível de especificação busca-se o entendimento do problema. Sugere-se inicialmente a compreensão do problema de uma maneira geral, para em seguida ir aos detalhes. Realizando uma análise do enunciado da situação problema acima, identifica-se, de maneira informal, as seguintes informações:

- Quais são os dados?

  - conjunto de configurações de três pinos

- Quais são os resultados?

  - conjunto de configurações de três pinos

- Qual é a intenção de resolução?

  - transformar uma configuração "inicial" numa configuração "meta".

O enunciado acima descreve uma situação problema que é uma instância do problema *"Torre de Hanoi"*. Para tratar a especificação desse problema, num primeiro nível, utiliza-se a relação $\mathcal{P} = \langle F, S, Cd \rangle$ (Definição 2.6). Assim, a estruturação para a situação problema acima (concretização), pode ser representada como segue:

- $F = S : TORRE3$[5] ("$TORRE$" representa "configurações de itens(discos) numa torre (pino)")
- $Cd : HANOI$ ("$HANOI$" representa a intenção de "transformar uma conf. 'inicial' numa conf. 'meta' ".)

A primeira etapa a ser realizada é a especificação dos elementos de F e S. Inicialmente supõe-se que um tipo $ITEM$ tenha sido especificado.

A especificação de $TORRE$ será dada, inicialmente, por meio da especificação de alguns elementos, o primeiro será o elemento $inserir$, que tem como ação inserir um 'item' numa 'torre'. Assim, segue a especificação.

Seja $torre, torre' \in TORRE$ e $item, item' \in ITEM$.

$inserir : TORRE \times ITEM \longrightarrow TORRE$

$* \neg inserir(torre, item) = torre$

[5] "$TORRE3$" abrevia "$TORRE \times TORRE \times TORRE$".

* $inserir(torre, item) = inserir(torre', item) \rightarrow torre = torre'$

* $inserir(inserir(torre, item'), item) = inserir(inserir(torre, item), item') \rightarrow item = item'$

A especificação de "*inserir*" acima não deixa claro onde será inserido o 'item' em 'torre' (se no topo na torre, se na base da torre, etc.). Portanto, a especificação de *inserir*, exposta acima, não está clara e permite ambigüidade quanto a ação a ser realizada. Entretanto, para dirimir isso torna-se necessário que antes seja acrescentada a especificação de novos elementos que são: *retira_topo*, que retira um 'item' do topo de uma 'torre'; e *topo* que apresenta o 'item' que está no topo de uma determinada 'torre'.

$retira\_topo : TORRE \longrightarrow TORRE$

* $\neg retira\_topo(\lambda) = torre$

* $retira\_topo(inserir(torre, item)) = torre$

$topo : TORRE \longrightarrow ITEM$

* $\neg topo(\lambda) = item$

* $topo(inserir(torre, item)) = item$

Agora sim, com a especificação dos elementos acima, pode-se especificar "*inserir*" de maneira completa sem ambigüidade (a equação abaixo esclarece que o 'item' tem que ser colocado no topo da 'torre').

* $inserir(retira\_topo(torre), topo(torre)) = torre$

Precisa-se agora definir como se pode realizar movimentos de itens entre as diversas torres. Para isso, especifica-se agora um novo elemento para a especificação do tipo

$TORRE$ (refinando $TORRE$), que é o elemento *move* (que executa a ação de mover um 'item' do topo de uma 'torre' para uma outra 'torre').

Contudo, para especificar o elemento "*move*" torna-se necessário uma ordem $<$ de $ITEM$. Tem-se que $<$ deve obedecer as propriedades:

* $x, y \in ITEM, x < y \wedge y < z \rightarrow x < z$ (transitiva)

* $x, y \in ITEM, x < y \rightarrow \neg y < x$ (antisimétrica)

* $\neg(x < y \wedge y < x)$ (antireflexiva)

Assim, a especificação do elemento "*move*" é dada como segue:

$move : TORRE \times TORRE \longrightarrow TORRE$

* $\neg move(\lambda, torre) = torre'$

* $torre \neq \lambda \rightarrow move(torre, \lambda) = inserir(topo(torre), \lambda)$

* $(torre, torre' \neq \lambda) \wedge (topo(torre) < topo(torre')) \rightarrow move(torre, torre') =$ $inserir(torre', topo(torre))$

Uma vez apresentado como se procede para movimentar 'itens' entre 'torres' (por meio "*move*"), torna-se necessário também expressar quando é que uma torre pode ser considerada adequada.

$Boa : TORRE \longrightarrow Bool$

* $Boa(\lambda)$

* $x \in ITEM,\ Boa(inserir(\lambda, x))$

* $[torre \in TORRE \wedge y \in ITEM \wedge torre \neq \lambda] \mid Boa(torre) \wedge (y < topo(torre)) \rightarrow$ $Boa(inserir(torre, y)$

* $torre \neq \lambda \wedge Boa(torre) \rightarrow Boa(retira\_topo(torre))$.

Assim, nessa etapa de especificação, a linguagem para esta situação problema é apresentada como possuindo os seguintes elementos:

$$L_1(\mathcal{P}) = \langle inserir, retira\_topo, topo, <, move, Boa \rangle$$

Entretanto, essa especificação ainda não é suficiente para satisfazer de maneira plausível a intenção do problema. Nota-se também que a especificação acima é ampla e está sujeita a ações que são irrelevantes para o problema tratado.

Assim, como próxima etapa da especificação vem a descrição da parte intencional Cd, descrevendo as componentes d, Int, e Adm (Definição 2.7). Esses componentes serão representados do seguinte modo:

- d : $TORRE3'$[6] ("$TORRE'$" é uma "$TORRE$" com uma quantidade finita de 'itens' de diferentes tamanhos ordenados de maneira decrescente);
- Int : $HANOI$ (representa restrições aos movimentos a serem realizados entre as diversas torres de forma a transformar uma configuração "inicial" numa configuração "meta");
- Adm : $Prog$ (representa que uma possível solução para o problema é considerada admissível se ela é programável).

A especificação dos componentes acima será feita utilizando os elementos da linguagem de F e S, ou seja, elementos especificados em $TORRE$.

Entretanto, a especificação de "$TORRE3'$" não é possível de realizada apenas com os elementos presentes até o momento na linguagem. Assim torna-se necessário estender (refinar) a especificação de "$TORRE$" acresentando um novo elemento. Trata-se de

[6] "$TORRE3'$" abrevia "$TORRE' \times TORRE' \times TORRE'$".

um elemento que verifique se a quantidade de elementos numa torre é finita ou não. Assim, será especificado agora um elemento denominado "$finita$".

Seja $torre, torre' \in TORRE$

$finita : TORRE \longrightarrow Bool$

* $finita(\lambda)$

* $finita(torre) \wedge x \in ITEM \rightarrow finita(inserir(torre, x))$

* $torre \neq \lambda \wedge finita(torre) \rightarrow finita(retira\_topo(torre))$.

Segue, então a especificação de "$TORRE3'$".

Seja $t_1, t_2, t_3, p_1, p_2, p_3 \in TORRE$

- $(t_1, t_2, t_3) \in TORRE3'$ sse $finita(t_1) \wedge finita(t_2) \wedge finita(t_3) \wedge Boa(t_1) \wedge Boa(t_2) \wedge Boa(t_3)$

Vem agora a especificação de $HANOI$. Do mesmo modo que na especificação de $TORRE3'$ foi necessário acrescentar um novo elemento à linguagem, também para a especificação de $HANOI$ é necessário que se acrescente a linguagem um novo elemento indicando quando um conjunto de movimentos é considerado "válido" para passar uma certa quantidade de discos de uma configuração para outra. Assim, segue a especificação de um elemento com esse objetivo.

$move\_val : IN \times TORRE3 \times TORRE3 \longrightarrow Bool$

* $mov\_val(0, (p_1, p_2, p_3), (t_1, t_2, t_3))$ sse $(p_1, p_2, p_3) = (t_1, t_2, t_3)$

* $mov\_val(1, (p_1, p_2, p_3), (t_1, t_2, t_3))$ sse

$[(move(p_1, p_2) = t_2 \wedge t_1 = retira\_topo(p_1) \wedge p_3 = t_3)] \vee$

$[(move(p_1, p_3) = t_3 \wedge t_1 = retira\_topo(p_1) \wedge p_2 = t_2)] \vee$

$[(move(p_2, p_3) = t_3 \wedge t_2 = retira\_topo(p_2) \wedge p_1 = t_1)]$

* $move\_val(n+1, (p_1, p_2, p_3), (t_1, t_2, t_3))$ sse

$move\_val(n, (p_1, p_2, p_3), (p'_1, p'_2, p'_3)) \wedge move\_val(1, (p'_1, p'_2, p'_3), (t_1, t_2, t_3))$

Agora segue a especificação de $HANOI$.

- $((p_1, p_2, p_3), (t_1, t_2, t_3)) \in HANOI$ sse $[t_3 = p_1 \wedge t_1 = p_3 \wedge t_2 = p_2] \wedge move\_val(n, (p_1, p_2, p_3), (t_1, t_2, t_3))$, para algum $n \in IN$.

$Prog$ será dada como um predicado de segunda ordem, e especificação dele é a sua própria concretização. Assim,

- $\forall h \in F^S$, $h \in Prog$ sse $Prog(h)$. □

Com isso, todos os componentes da estrutura

$$\mathcal{P} = \langle F, S, d, Int, Adm \rangle$$

foram descritos (Definição 2.7).

Assim, o problema está suficientemente compreendido, o que significa que se sabe exatamente o que se está querendo resolver.

Obedecendo ao esquema estabelecido na figura 3.2., uma interpretação para o problema "Torre de Hanoi" foi dada da seguinte maneira:

$\zeta : \mathcal{P} \longrightarrow \mathcal{M}$[7], em que

$\zeta(F) = \zeta(S) = TORRE$ (configurações de torres);

$\zeta(d) = TORRE'$ (restrição as configurações entre torres);

$\zeta(Int) = HANOI$ (configurações que satisfazem a intensão do problema);

$\zeta(Adm) = Prog$ (soluções admissíveis).

[7]Lembrando que "$\zeta : \mathcal{P} \longrightarrow \mathcal{M}$" significa que $\mathcal{M}$ é modelo para $\mathcal{P}$ segundo uma interpretação ζ (ver definição 3.1).

Com isso, fica encerrado o primeiro nível da especificação dessa situação problema.

Entretanto, fica a seguinte indagação: "Esta especificação é suficiente para especificar uma solução?"

Deve-se ter em mente que para determinar uma solução geral para este problema de mover **n** discos do pino A para o pino C, é interessante analisar alguns exemplos com somente uns poucos discos. O caso base é quando existe somente um disco. Neste caso, o disco pode ser movido diretamente do pino A para o pino C (ver figura 4.3).

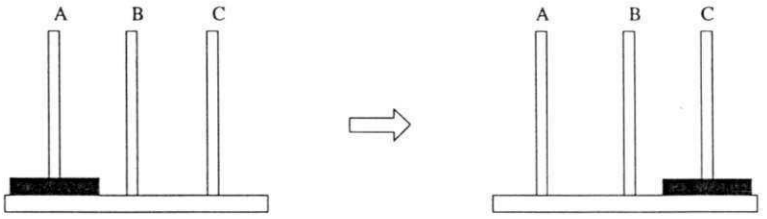

Figura 4.3: *mover disco do pino A para o pino C, caso básico com somente um disco.*

O caso de dois discos também não é complicado. Primeiro, o disco 1 é movido para o pino B, então o pino 2 é movido para o pino C. No último passo, coloca-se o disco 1 no topo do disco 2 - pino C (ver figura 4.4).

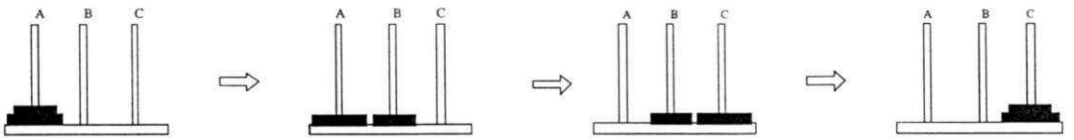

Figura 4.4: *mover disco do pino A para o pino C, caso com dois discos.*

Com três discos o problema torna-se mais complexo. Para resolver o problema, é necessário encontrar um padrão na solução.

Examinando-se os casos de dois e três discos, o seguinte é visto: para mover o disco maior de A para C, deve-se mover os discos menores que estão no topo do pino A para o pino B. Somente o disco maior tem de ser movido. Depois, move-se todos os discos menores para o pino C.

Conhecendo-se a estrutura que envolve o problema, deve-se adotar uma estratégia

de solução que busque os resultados esperados. O caminho sugerido aqui é verificar a possibilidade de quebrá-lo. Para isso será utilizado a *decomposição n-ária.*

Analisando a intenção do problema, pode-se apresentar o seguinte esquema: para mover **n** discos de A para C primeiro verifica se **n** é 1. Se for, a solução é encontrada de forma *direta* (apenas um movimento). Caso contrário, decompõe-se o problema em três partes e as resolve em seqüência.

É possível, então, descrever recursivamente um esquema com os passos necessários para mover **n** discos do pino A para o pino C usando o pino B.

Seja **n** o número de discos a ser movido,

(*i*) Se **n=1** então
Mover o disco de A para C

(*ii*) Se **n>1** então
Mover **n-1** discos menores do pino A para o pino B
Mover o disco maior do pino A para o pino C
Mover os **n-1** discos menores do pino B para o pino C.

No entanto, para trabalhar com a "solucionática" acima seria necessário reformular a concretização do problema adicionando a quantidade de discos a ser movida. Para evitar isso e trabalhar com a própria configuração dos pinos, torna-se necessário trabalhar com outra solucionática.

Será proposta a seguinte:

- mover um disco do pino 1 para o pino 3, ou vice-versa;
- mover um disco do pino 1 para o pino 2, ou vice-versa;
- mover um disco do pino 2 para o pino 3, ou vice-versa.

As ações acima são realizadas dependendo de quais movimentos são válidos. Este processo acima é repetido até se obter a configuração "meta".

A especificação para os movimentos acima será dada por $transf_1, transf_2$ e $transf_3$, da seguinte maneira:

$transf_1 : TORRE3 \longrightarrow TORRE3$[8]

$$transf_1(conf) = \begin{cases} (retira\_topo(p_1), p_2, move(p_1, p_3)) & \text{se } (topo(p_1) < topo(p_3)) \vee \\ & (p_1 \neq \lambda \wedge p_3 = \lambda) \\ (move(p_3, p_1), p_2, retira\_topo(p_3)) & \text{se } topo(p_3) < topo(p_1) \vee \\ & (p_3 \neq \lambda \wedge p_1 = \lambda) \end{cases}$$

$transf_2 : TORRE3 \longrightarrow TORRE3$

$$transf_2(conf) = \begin{cases} (retira\_topo(p_1), move(p_1, p_2), p_3) & \text{se } (topo(p_1) < topo(p_2)) \vee \\ & (p_1 \neq \lambda \wedge p_2 = \lambda) \\ (move(p_2, p_1), retira\_topo(p_2), p_3) & \text{se } (topo(p_2) < topo(p_1)) \vee \\ & (p_2 \neq \lambda \wedge p_1 = \lambda) \end{cases}$$

$transf_3 : TORRE3 \longrightarrow TORRE3$

$$transf_3(conf) = \begin{cases} (p_1, retira\_topo(p_2), move(p_2, p_3)) & \text{se } (topo(p_2) < topo(p_3)) \vee \\ & (p_2 \neq \lambda \wedge p_3 = \lambda) \\ (p_1, move(p_3, p_2), retira\_topo(p_3)) & \text{se } (topo(p_3) < topo(p_2)) \vee \\ & (p_3 \neq \lambda \wedge p_2 = \lambda) \end{cases}$$

Uma formulação da solução para este problema, usando *decomposição n-ária* (em que $n = 1$), pode ser dada como segue:

Seja $conf \in TORRE3$

[8] "$conf$" abrevia "(p_1, p_2, p_3)".

$$h(conf) = \begin{cases} direta(conf) & \text{se } simples(conf) \\ recomb(h(part(conf))) & \text{caso contrário} \end{cases}$$

onde,

$simples : TORRE3 \longrightarrow Bool$

* $simples(conf)$ sse $Boa(move(p_1, p_3)) \wedge p_2 = \lambda \wedge retira\_topo(p_1) = \lambda$.

$direta : TORRE3 \longrightarrow TORRE3$

* $direta(conf) = (\lambda, \lambda, move(p_1, p_3))$

$part : TORRE3 \longrightarrow TORRE3$

* $part(conf) = transf_3(transf_2(transf_1(conf)))$

$recomb : TORRE3 \longrightarrow TORRE3$

* $recomb(conf) = conf$.

A linguagem para o problema acima é apresentada, nesse nível de especificação, da seguinte meneira:

$$L_2(\mathcal{P}) = L_1(\mathcal{P}) \cup \{finita, move\_val, transf_1, transf_2, transf_3, direta, simples, part, recomb, h\}.$$

Uma vez que a liguagem $L_2(\mathcal{P})$ possue todos os elementos necessários para resolver o problema, então essa linguagem é considerada suficiente para especificar a resolução do problema.

Note que para tornar os elementos estabelecidos para a linguagem desse problema implementáveis, bastaria escolher uma nova linguagem que possua, por exemplo, indexação. Por exemplo, uma linguagem de programação como "Fortran", "C", "Pascal", etc.

□

4.3 Descrição da Metodologia

O mecanismo exposto neste trabalho consiste de duas fases. A primeira delas consiste em fazer uma análise sobre o entendimento da situação problema. Na segunda fase, é tratado a solubilidade.

Na primeira fase - a do entendimento - trabalha-se o refinamento da descrição, visando aperfeiçoar essa descrição até se poder expressar todos os componentes estruturais do problema (Definição 2.7).

Na segunda fase trabalha-se a solubilidade do problema.

O mecanismo aqui exposto, trata-se na realidade de uma "heurística de resolução de problemas". O que se faz numa heurística de resolução de problemas é elucidar os passos que devem ser dados para resolver situações tidas como problemas.

Pode-se dizer que heurísticas são critérios, métodos, ou príncipios para decidir dentre muitas alternativas de ação, os mais indicados a serem seguidos obedecendo uma determinada ordem para realizar com êxito um determinado objetivo.

Apresenta-se a seguir os passos (ou ações) que devem ser dados, de acordo com o mecanismo aqui exposto, para solucionar problemas.

1. Inicialmente, parte-se do pressuposto que o enunciado da situação problema já esteja

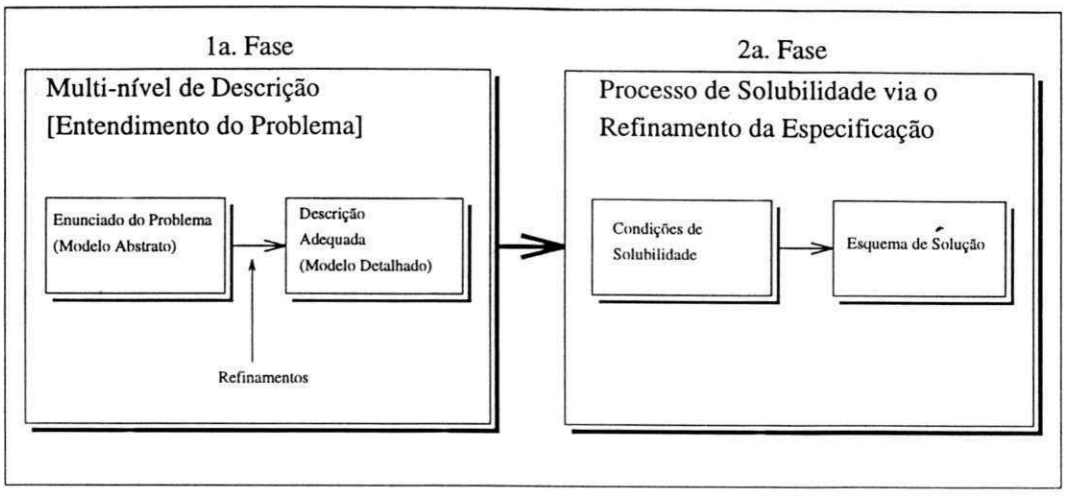

Figura 4.5: *Representação esquemática das fases envolvidas no mecanismo de resolução de problemas*

declarado. A primeira providência a ser tomada na especificação de um problema é identificar no seu enunciado as informações nele contidas. Assim sendo, deve-se responder as seguintes indagações:

- Qual é o domínio fonte (dados)? [representado por F]
- Qual é o domínio dos resultados? [representado por S]
- Quais as condicionantes do problema? [representado por Int]

Mas, numa primeira percepção deve-se olhar para o problema de uma maneira generalizada, sem se preocupar com detalhamentos específicos. A linguagem utilizada nesse nível de descrição (especificação) não oferece muito poder de percepção a cerca da situação tratada. O processo de especificação pode ser auxiliado mais facilmente se uma estrutura formal está presente. Assim, as indagações colocadas acima podem ser representadas por meio de uma estrutura matemática da seguinte forma:

$$\mathcal{P} = \langle F, S, Int \rangle,$$

simplificando a notação representacional das questões abordadas.

2. No nível anterior podem ser captados elementos irrelevantes ao problema. Numa segunda etapa de especificação, o entendimento do problema deve ser dado num nível mais específico e particularizado para o problema tratado.

Deve-se ter em mente que a especificação de um problema numa linguagem natural pode causar ambigüidades, desde que o problema pode ser interpretado diferentemente por pessoas diferentes. Assim, para eliminar os problemas de ambigüidades numa especificação, um sistema de especificação deve ser expresso em uma notação adequada com única interpretação. Isso irá reduzir a probabilidade de ambigüidades e equívocos. Para tal, utiliza-se uma linguagem que consiga ter uma percepção dos elementos envolvidos numa visão minunciosa do domínio de dados.

Na compreensão estrutural do problema deve-se, portanto, especificar melhor qual o domínio fonte (de dados) para o problema em particular. Haverá, com isso, um aumento do entendimento estrutural da situação problema tratada. Pode-se, assim, expressar melhor as condições de intencionalidade que envolvem o problema. A estrutura representacional de problema fica, nessa etapa de especificação, da seguinte forma:

$$\mathcal{P} = \langle F, S, d, Int \rangle,$$

sendo que

- d representa as restrições ao domínio de dados; e
- Int representa as intenções que estão envolvidas na resolução do problema.

3. Deve-se notar que no enriquecimento, feito na última etapa de especificação, das condicionantes que envolvem o problema, não foi estabelecido formas de associar dados com resultados admissíveis. Assim, surge a necessidade de mencionar quais são as restrições às muito possíveis funções de F em S que não serão consideradas como admissíveis. Dessa forma, deve-se incluir na especificação a representação do que é admissível ao problema, nos termos colocados acima. Para tal, admite-se um predicado de admissibilidade (Adm), o que deixa a estrutura representacional de problema, agora, da seguinte maneira:

$$\mathcal{P} = \langle F, S, d, Int, Adm \rangle.$$

Com esse enriquecimento estrutural e a devida especificação das componentes que compõem essa estrutura, o problema estará devidamente compreendido. A linguagem fornecida nesse nível de espcificação é obtida aperfeiçoando-se a percepção a cerca dos elementos envolvidos na resolução da situação problema.

4. Tendo-se entendido o problema (primeiro nível a ser trabalhado em busca da solução), pode-se então passar para a fase de solubilidade do mesmo, buscando especificar elementos que permitam apresentar a solução para a situação problema tratada.

A obtenção e especificação dos elementos necessários para apresentar a solução da situação problema deve ser feita da maneira mais natural possível. Entretanto, apartir de um determinado momento da especificação torna-se necessário a interferência do solucionador, entrando em ação a sua macrovisão sobre o problema, que o possibilita perceber que apenas com os elementos presentes até o momento na linguagem já se pode chegar ou não a uma solução.

A solução de um problema pode ser apresentada sem que necessariamente tenha sido estabelecido previamente uma estratégia de solução. Entretanto, uma outra meneira

de apresentar a solução de uma determinada situação problema é seguindo-se uma estratégia de solução específica e/ou método de solução. Observe que fazendo isso, o processo de refinamento fica direcionado e a solução obtida pode ser mais eficiente. Neste caso, a especificação dos elementos será realizada buscando dar subsídio para que a solução do problema seja dada de acordo com aquela estratégia e/ou método escolhido.

Para tratar a solubilidade de algumas situações problemas, lançou-se mão de um método de resolução auxiliar, como por exemplo nas situações problemas de "Ordenação" e "Torre de Hanoi". Um método auxiliar utilizado neste trabalho para abordar a solubilidade de problemas foi a *decomposição*, embora se possa escolher outro método dependendo da conveniência para a situação tratada.

A utilização da *decomposição* como método auxiliar para tratar a solução de problemas é bastante conveniente devido aos resultados obtidos em [OJ85]. Os resultados obtidos em [OJ85] indicam que o método de decomposição é um método seguro e muito abrangente (resolve também os problemas solúveis por algoritmos) para abordar os problemas que surgem na programação dos computadores atuais. Embora este fato não seja novo, o trabalho citado acima aponta tal indicação com rigor e formalismo matemático.

A decomposição de um problema pode ser "estrutural" ou "n-ária". Aqui utiliza-se, pela abordagem seguida para resolver problemas, a segunda forma.

A forma de decomposição escolhida para ser utilizada neste trabalho (*n-ária*), pode ser descrita do seguinte modo: os dados (elementos do "domínio fonte") são partidos repetidamente por meio de funções de partição ($part_i$) até os mesmos encontrarem-se numa forma "*simples*". Quando isso ocorrer, operam-se soluções "*diretas*" sobre esses elementos "*simples*", encontrando-se os elementos correspondentes no co-domínio de resultados, e constróem-se, por meio de uma função de recombinação (*recomb*), os elementos do co-domínio de resultados correspondentes aos elementos do predicado fonte, tomando-se o cuidado de que a função resultante satisfaça ao predicado de admissibilidade (Adm) do problema.

Assim, a forma para o esquema de solução para um problema $\mathcal{P}$ abordado por esse

método é dado da seguinte maneira:

$$h(f) = \begin{cases} direta(f), & \text{se } simples(f) \\ recomb(h(part_1(f)), h(part_2(f)), \ldots, h(part_n(f))) & \text{caso contrário} \end{cases}$$

Desse modo, a solução é apresentada na forma de uma função recursiva (ou primitiva recursiva)[BL74].

Assim, precisando-se resolver uma situação problema por esse método acima, é necessário refinar a especificação do problema incorporando as especificações das funções $part_i, direta, recomb$ e $simples$.

Ao se chegar no ponto da especificação em que a solução do problema é mostrada, a heurística de como o problema pode ser solucionado foi obtido, uma vez que se apontando quais os elementos que irão compor a linguagem necessária para resolver o problema. Essa linguagem do problema é formada pela linguagem necessária para especificar F, S, d, Int e Adm acrescida dos elementos fornecidos para especificar a solução do problema.

Deve-se frisar, que esta metodologia pode ser usada para as mais diversas situações problemas. O fato de neste trabalho se ter explorado extensivamente a situação problema de "ordenar uma seqüência", foi simplesmente devido a sua grande aplicação na informática, o que facilita a percepção de como a metodologia pode ser utilizada.

As etapas do processo de resolução via o refinamento da especificação aqui exposto, é resumidamente colocado na Figura 4.6.

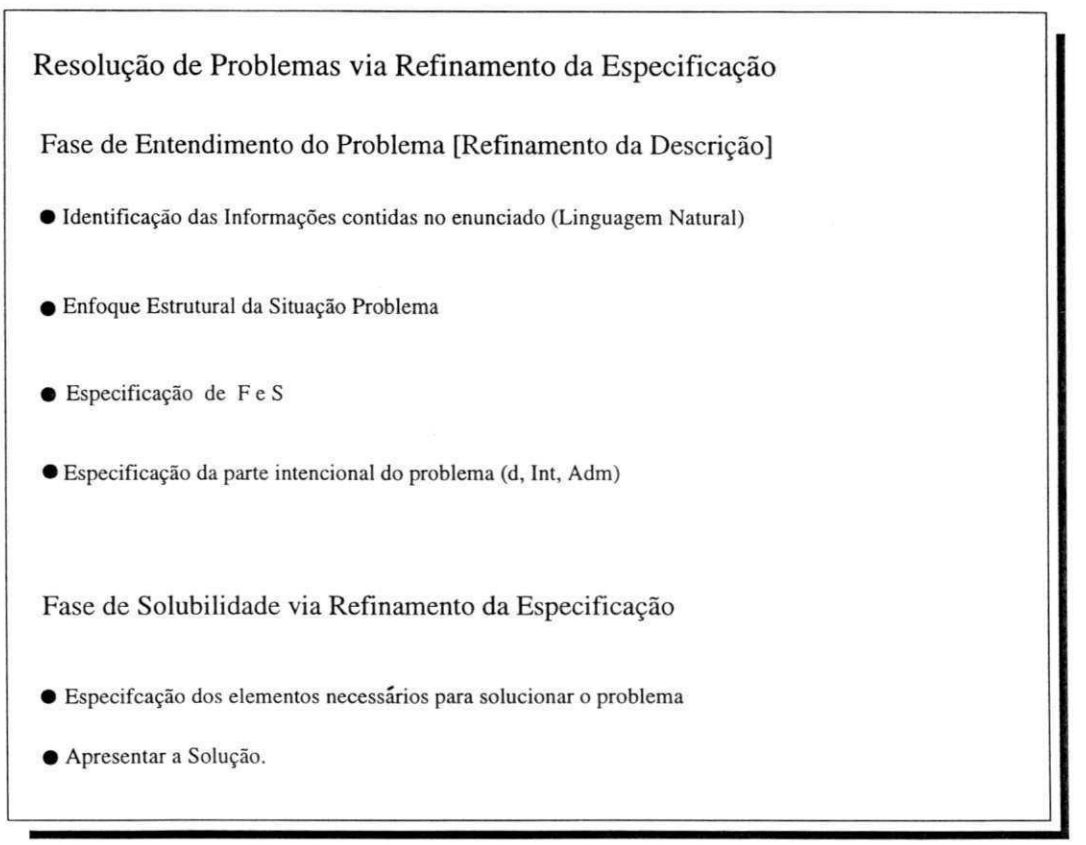

Figura 4.6: *Representação esquemática das fases envolvidas no mecanismo de resolução de problemas via refinamento da especificação.*

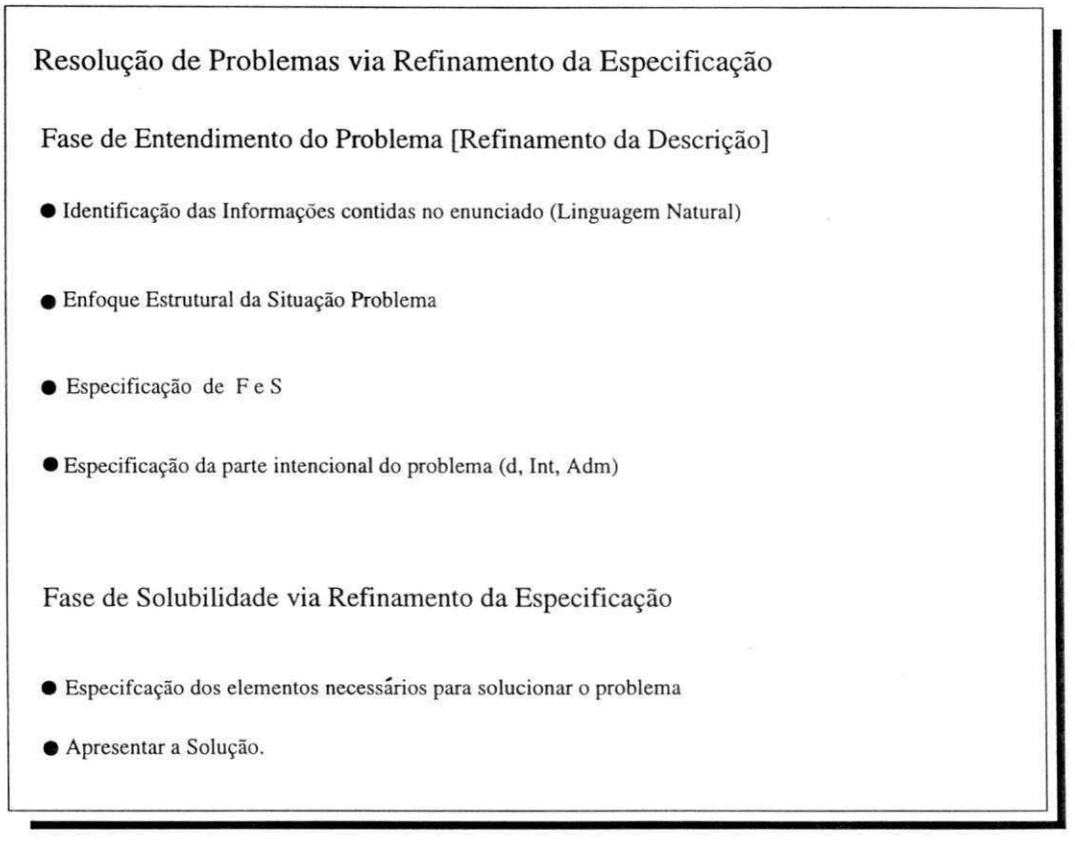

Figura 4.6: *Representação esquemática das fases envolvidas no mecanismo de resolução de problemas via refinamento da especificação.*

Capítulo 5

Conclusão

O estudo desenvolvido por este trabalho abordou a questão da solubilidade de problemas e mostrou como se pode chegar a resolução de um problema, por meio do "*refinamento da especificação*".

A contribuição feita por este trabalho é a de fornecer um mecanismo que permite resolver problemas refinando a especificação até se obter todos os elementos necessários para a resolução da situação problema tratada.

Esta metodologia proporciona ao especificador um modo de declarar os requisitos necessários para solucionar computacionalmente um problema. Conduzindo-o, assim, a explicitar os elementos que estão envolvidos na resolução de uma determinada situação problema, bem como apresentar como esses elementos são especificados. Facilitando, desse modo, a implementação de um sistema computacional que resolva a situação problema tratada.

A metodologia exposta, trata a resolução de uma situação tida como problema desde o entendimento do enunciado até chegar a apresentar a solução, e a respectiva linguagem suficiente para resolver o problema.

Observa-se, seguindo-se esta metodologia, que no primeiro nível de especificação não se consegue, necessariamente, expressar todos os elementos suficientes para resolver o problema. Torna-se preciso, então, adicionar a essa linguagem novos elementos que

sejam necessários para a resolução. Isso deve ser feito até obter uma linguagem com a qual se consiga apresentar os elementos suficientes para a resolução do problema (ver representação abaixo).

$$L_1(\mathcal{P}) \subseteq L_2(\mathcal{P}) \subseteq \ldots \subseteq L_n(\mathcal{P}).$$

Esta medologia apresenta, no final, uma solução para o problema por meio de uma função recursiva. Se o método utilizado para definir a função recursiva for a decomposição n-ária, vê-se que a função h é definida por "recursão primitiva" relativo a *part*, *recomb*, *direta*, e *simples*, como já foi predito em [OJ85]. Considera-se o termo "recursão primitiva" no sentido como abordado em [BL74], em que h sendo definida como *recursiva primitiva* significa ser efetivamente resolvível.

Os exemplos expostos neste trabalho, apresentam de forma clara como se deve proceder para a utilização da metodologia aqui exposta. Estes exemplos serviram para expor, também, de maneira completa a utilização da metodologia para solucionar situações problemas por meio do refinamento de sua especificação, que foi o objeto principal desse estudo. Deve-se deixar claro também, que esta metodologia serve para abordar as mais diversas situações problemas.

Na haurística de resolução de situações problemas, existem elementos que saem de maneira natural por serem as condições mínimas de qualquer eventual solução. Entretanto, deve-se frisar que sair refinando a especificação passo a passo não garante se obter uma solução, pois nada impede que se vá por caminhos que o levem a nenhum resultado. Devido a isto, em algum momento no processo de especificação da solução torna-se fundamental a percepção do solucionador a respeito da situação problema tratada.

Os problemas que se adequam a esta metodologia são problemas sob os quais ocorre a existência de *procedimentos computacionalmente resolvíveis.*

Com esta metodologia, consegue-se mostrar claramente o potencial téorico da Teoria Geral de Problemas [Lop81], em que a utilização de suas difinições, formalmente descritas, permitiram dar o rigor necessário para a metodologia aqui exposta.

Esta metodologia conduz, também, a tornar a representação abstrata do problema adequada e mais próxima possível de uma representação operacional do problema.

Deve-se frisar, também, que não faz parte do escopo do trabalho aqui exposto tratar como pode ser realizada sua implementação, pois isso depende das ferramentas tecnológicas (linguagem de programação) utilizada para a implementação.

Um desdobramento decorrente deste trabalho, é a sua utilização na Modelagem e Representação do Conhecimento, constituindo-se numa metodologia base para que se possa obter os elementos necessários para a resolução de uma determinada situação problema.

Uma sugestão de trabalho futuro decorrente deste método, é de se analisar a possibilidade da utilização da metodologia aqui exposta para a contrução e implementação de um sistema computacional que auxilie na resolução de problemas baseado no refinamento da especificação.

Referências

[Alm90] Adriano P. de Almeida. *Especificação e Solubilidade de Classes de Problemas com Paradígmas e de Classes Nomeáveis.* Diss. de Mestrado, Dept. de Informática/CCEN/UFPE, Recife-PE, 1990.

[Arb69] Michael A. Arbib. *Theories of Abstract Automata.* Prentice-Hall, Englewood Cliffs, 1969.

[Bed87] Benjamín René Callejas Bedregal. *Especificação de Problemas Solúveis por Decomposição via Análise da Intencionalidade.* Diss. de Mestrado, Dept. de Informática/CCEN/UFPE, Recife-PE, 1987.

[BL74] Walter S. Brainerd and Lawrence H. Landweber. *Theory of Computation.* John Wiley & Sons, New York, 1974.

[Bun76] M. Bunge. *Tratado de Filosofia Básica*, volume 2. E.P.U. EDUSP, São Paulo, 1976.

[CK77] C. C. Chang and H. J. Keisler. *Model Theory.* North-Holland Publishing Company, Amsterdam, 1977. 2^a edição.

[GTW87] J. A. Goguen, J. W. Tratcher, and E. G. Wagner. *An Inicial Algebra to The Specification, Correctness and Implementation of Abstract Data Types.* Rc6487, IBM Research Report, 1987.

[LCB88] M. A. Lopes and Benjamín R. Callejas Bedregal. "Decomposição de Problemas: Uma Solução para Um Problema Interessante da Teoria dos

Números". In *Anais do XI Congresso Nacional de Matemática Aplicada e Computacional (SBMAC)*, Ouro Preto-MG, 1988. SBC.

[LO89] Renae Low and Ray Over. "Detection of Missing and Irrelevant Information in Algebraic Story Problems". *British Journal of Educational Psychology*, 59:296–305, 1989.

[LODM94] Renae Low, Ray Over, Lawrence Doolan, and Sue Michell. "Solution of Algebraic Problems Following Training in Identifying Necessary and Sufficient Information within Problems". *American Journal of Psychology*, 107(3):17, 1994.

[Lop81] Manoel Agamemnon Lopes. *Introdução a uma Teoria Geral de Problemas.* Tese de Doutorado, Dept. Informática PUC-RIO, Rio de Janeiro, 1981.

[MM83] B. Mahr and J. A. Makowsky. "An Axiomatic Approach to Semantics of Specification Language". In *Lectures Notes in Computer Science*, volume 145, pages 211–219, Berlin, 1983. Springer-Verlarg.

[OJ85] Wilson Rosa de Oliveira Junior. *Especificação e Caracterização de Problemas Solúveis por Decomposição.* Diss. de Mestrado, Dep. de Informática/CCEN/UFPE, Recife-PE, 1985.

[Oli75] Antonio Marmo de Oliveira. "Estrutura e estruturalismo". Em *Curso de Matemática Moderna*, volume 1. Editora LISA, Brasil, 1975.

[Pea85] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading-Massachusetts, 1985.

[Peq81] T. H. C. Pequeno. *Uma Descrição Formal dos Processos de Especificação e Implementação de Tipos Abstratos de Dados.* Tese de Doutorado, Dep. Informática PUC-RIO, Rio de Janeiro, 1981.

[Pol78] George Polya. *A Arte de Resolver Problemas: um novo aspecto do método matemático.* Interciência, Rio de Janeiro, 1978.

[RK93] Elaine Rich and Kevin Knight. *Inteligência Artificial.* Makron Books, São Paulo, 1993. 2ª edição.

[Swe88] J. Sweller. "Cognitive Load During Problem Solving: Effects on Learning". *Cognitive Science*, 12:257–285, 1988.

[Swe89] J. Sweller. "Cognitive Technology: Some Procedures for Facilitating Learning in Problem Solving in Mathematics and Science". *Journal of Educational Psychology*, 81:457–466, 1989.

[Vel82] Paulo A. S. Veloso. *Outlines of a Mathematical Theory of Problems.* Tech. Report 14/82, Dep. de Informática PUC-RIO, Rio de Janeiro, 1982.

[Wir89] Niklaus Wirth. *Algoritmos e Estruturas de Dados (do original em Inglês: Algorithms and Data Structures).* Prentice-Hall do Brasil, Rio de Janeiro, 1989.