ALGUMAS IDÉIAS E EXPERIMENTOS SOBRE DEMONSTRAÇÃO AUTOMÁTICA DE TEOREMAS

Emmanuel Piseces Lopes Passos

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS (D.Sc.)

Aprovada por:

Roberto Lins de Carvalho
(Presidente)

Paulo Augusto Silva Veloso
Co-orientador

Doris Ferraz de Aragon

José Lucas Mourão Rangel Netto

Jayme Luiz Szwarcfiter

RIO DE JANEIRO, RJ - BRASIL
JULHO DE 1981

PASSOS, EMMANUEL PISECES LOPES

Algumas Idéias e Experimentos sobre Demonstração Automática de Teoremas [Rio de Janeiro] 1989.

XVII, p. 82, 29,7cm (COPPE-UFRJ, D.Sc, Engenharia de Sistemas, 1981)

Tese - Univ. Fed. Rio de Janeiro, Programa de Engenharia de Sistemas

.1. Prova Automática de Teoremas I.COPPE/UFRJ II.Título(série).

DEDICATÓRIA

à Tildinha, Sonia, Mirna, Tiago
e vovó Catarina e sua família.
à Marly uma amiga do Brigadeiro
Schorth que me pediu uma dedica
tória em 1964 num trabalho quan
do me formasse e nunca mais avi.

AGRADECIMENTO

À Roberto Lins de Carvalho por tudo.

À Margarida que entendeu e deixou mais tempo para trabalharmos.

Aos amigos que ajudaram. Ao Paulo Augusto Veloso, rei do contra-exemplo, por sua valiosa colaboração.

Ao Instituto Militar de Engenharia que me proporcionou tempo e incentivo para terminar o trabalho.

Aos colegas da Informática da PUC, da Cobra, do Fundão por onde estudei e trabalhei, muito obrigado.

Aos colegas do Pio Americano, do Brigadeiro Shorcht, Profª Henríette Amado, educadora e amiga, meus agradecimentos.

Ao amigo e professor Luiz Adauto da Justa Medeiros pelo importante incentivo inicial.

E finalmente a Sonia, minha amiga e esposa, meu agradecimento por me ter ajudado e compreendido nesses anos que nos conhecemos e crescemos.

RESUMO

Este trabalho contém algumas idéias e experimentos sobre Demonstração Automática de Teoremas.

São descritas as pesquisas que o autor realizou durante os últimos cinco anos, dando continuidade ao trabalho de doutorado de Roberto Lins de Carvalho.

Neste trabalho introduziu-se uma estratégia completa para Teorias Definicionais, TRADUÇÃO POR NÍVEL, com ou sem axiomas próprios. Além de analisar-se a consistência das definições, por intermédio da construção automática de ESTRUTURAS DEFINICIONAIS.

Além disso os experimentos realizados mostraram, em resumo, que:

(i) as limitações de tempo e memória nos obrigam a sempre organizarmos um provador interativo homem/máquina.

(ii) devemos mudar a estratégia de provas de acordo com a teoria em que estamos trabalhando. Pelo fato de ser um sistema em módulos independentes, isso é possível.

Baseados em (i) e (ii.), elaboramos algumas idéias para a criação de um sistema, gerador de provador de teoremas, completamente modular e que ensina ao usuário modificá-lo, isto é, de acordo com resultados parciais de provas naquela teoria, o usuário pode mudar o provador dinamicamente.

Basicamente este sistema consta de dois programas modulares. O primeiro programa com que o usuário tem contato, *programa gerador*, proporciona a geração de um segundo programa, *programa gerado*, que será o provador de teoremas da teoria definida pelo usuário, quando da interação com o programa gerador.

As idéias novas introduzidas no trabalho sáo: gerador de estruturas definicionais. tradução por nível como uma estratégia completa para teorias definicionais e fundamentalmente o fato de podermos gerar um provador mutável com a teoria e ou com o resultado da primeira utilização do provador.

ABSTRACT

This work includes some ideas and experiments about Automatic Theorem Proving.

It describes the research that the author has made in the last five years in continuation to the Ph.D. thesis of Prof. Roberto Lins de Carvalho.

This work introduces a new and complete strategy for resolution based theorem provers for Definitional Theories.

Furthermore, the experiments showed that:

(i) memory and time limitations require one organizes a theorem prover man/machine interacting.

(ii) one must change the strategy according to the theory one is working with.

Based on (i) and (ii) we developed some ideas to create a theorem prover generator system, completely modular and that teaches the user to change it, that is, the user can change dinamicaly the prover according to the partial results in that theory.

Basically this system consists of two modular programs. The first program contacted by the user, the generator program - generates a second program - the generated program. The latter program will be the theorem prover of the theory defined by the user when he interacted with the generator program.

The new ideas included in this work are: generator of definitional theories, translation by level as a complete strategy for Definitional Theories and basically the fact that we can create a mutable prover that adjusts itself to

the theory and/or to the results of the first utilization of the prover.

RESUMÉ

Le présent travail contient quelques idées et experiments 'sur le sujet "Demonstration Automatique des théorèmes"..

On a décrit les recherches que l'auteur a realisé pendant les derniers cinq ans, pour donnes continuité au travail de doctorat de Roberto Lins de Carvalho. Dans ce travail a été introduite une stratégie complète pour les Teories Definitionelles, traduction par niveau avec ou sans des axiomes propres, en outre par l'analyse de la consistence des définitions, par l'entremise des strutures définitionelles.

De plus, les experiences realisées ont montré, en resumé que:

(i) les limitations de temps et de mémoire nous obligent toujours à organiser um prouveur interactif homme/machine.

(ii) il nous faut changer la stratégie des preuves selon la théorie avec laquelle nous travaillons. À cause d'être un système en modules indépendants, cela divient possible.

(i) et (ii) nous ont permis d'élaborer quelques idées pour créer un système, générateur de prouveurs des théorèmes, complètement modulaire et que cnscigne à l'usager le modifier c'està dire, d'accord avcc les résultats particlcs dc ladite théorie, l'usager peut changer le prouveur dynamiquement, ce système est composé essentiellement dc deux programanes modulaires. Lc premier programme nvcc lequel l'usager entre en contoct, programme générateur, provoque une interaction nvcc l'usager. Cettc

interaction homme/machine (prêmier programme) proportione la génération d'un second programme, programme engendrê, qui sera le prouveur des théorême de la théorie définie par l'usager, lorsque l'interaction avec le programme générateur est prsduite. Les nouvelles idées introduites dans le travail sont:

le générateur des strutures definitionelles;

la traduction par niveau comme une stratégie complète pour des théories définitionelles;

le fait fondamental que nous pouvons engendrer un prouveur muable avec la théorie ou avec le résultat de la prémière utilization du prouveur.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

I.1 - História

Desde o advento do computador, iniciou-se o interesse da elaboração-de'meios computacionais possíveis para prova automática de teoremas matemáticos. A pergunta que se fazia era "como pode uma máquina provar teoremas?"

Começou-se então as pesquisas construindo-se programas heurísticos baseados em conhecimentos psicológicos do comportamento humano na resolução de problemas tendo como base o fato de existir no matemático: intuição "desenvolvida","memória"de acesso rápido, repertório de "truques': percepção' global e TREINAMENTO; e uma máquina: memória, memória, memória e regras de dedução.

Então o algoritmo usado em PASSOS[1] e SHAW[2] para provar teoremas no computador era uma simulação do comportamento do matemático para provar o mesmo teorema sem o computador.

TARSKI[3] sugeriu um método de eliminação de quanti-Eicadores e o usou para provar a decibilidade das teorias de 1.ª ordem de corpos algebricamente fechados. Multas outras teorias matematicamente Interessantes foram mostradas serem decidíveis por esse método, como geometria euclidiana elementar, TAKSKI[z], grupos Abelianos, SZMIELEW[4], conjuntos densamente ordenados, MARGARIS[5], e as teorias das álgebras Booleanas, TARSKI[3]. Um algoritmo que prova teoremas para a teoria dos conjuntos ordenados densamente sem o 1º ou Último elemento e grupos Abelianos, é encontrado em GALDA[6] assim como sua implementação em LISP é encontrada em PASSOS[7].

Uma principal vantagem do trabalho de GALDA e PASSOS[6] é que dentro das teorias para qual o método foi estabelecido é que sempre decidirá se um determinado teorema é falso ou verdadeiro e de sobra determinaremos um limite sobre o número de passos necessários para provar (ou desaprovar) o teorema. Esse limite dependerá individualmente de cada teorema. Teoremas curtos terão provas curtas.

Uma desvantagem de GALDA e PASSOS[6,7] séria é a falta de universalidade; pode-se usá-lo com sucesso total somente para certas teorias decidíveis de 1? ordem. Muda a teoria muda o algoritmo. Os algoritmos dependem da estrutura da teoria envolvida.

Entretanto a ênfase nos trabalhos de demonstração automática de teoremas recentemente é bem diferente desses até então relatados.

Tudo começou com o teorema de Herbrand.

"Um conjunto de cláusulas é insatisfatível se e somente se existe um conjunto finito insatisfatível S' de cláusulas base de S" (prova em CHANG[11]).

Esse teorema sugeriu um procedimento de refutação, isto é, dado um conjunto S de cláusulas para provar, se existe um procedimento mecânico que pode gerar os conjuntos $S'_1, S'_2, \ldots S'_n$ de cláusulas base em S e testar se $S'_1 \ldots S'_n$ são insatisfatível, então o teorema de Herbrand garante que esse procedimento pode detetar um N finito tal que S'_n é insatisfatível (ou seja pára!).

GILMORE[15] implementou a idéia em 1960, escreveu um programa que gerava $S'_0, S'_1, \ldots$ onde S'_i é o conjunto de todas as cláusulas obtidas trocando-se as variáveis em S pelas constantes no nível i do conjunto de constante Hi de S. Desde que cada S'_i é um conjunto de cláusulas; pode-se usar

qualquer método disponível da lógica proposicional para verificar sua insatisfatível. GILMORE usou o método de multiplicação. Isto é, quando cada S_i' produzido, S_i' é "multiplicado" para forma disjuntiva normal-DNF. Qualquer conjunção na DNF coritendo um par complementar é removida. Se algum S_i' fica vazia então S_i' é insatisfatível, CHANG[11]. Esse procedimento é muito ineficiente requer a geração dos conjuntos S_1', S_2' de cláusulas e na maioria dos casos essa sequência cresce exponencialmente. As vezes S_1' tem, 2 elementos e S' já tem 1512 elementos e o conjunto que é insatisfatível é o S_J' que tem 10^{256} elementos.

A história moderna de demonstradores automáticos de teoremas começa com os trabalhos de ROBINSON[8,9,10]

Em 1965 introduziu o princípio de resolução que verificava diretamente em qualquer conjunto S de cláusulas se era insatisfatível ou não! A idéia essencial desse princípio é verificar se S contem uma cláusula vazia □. Caso afirmativo S é insatisfatível. Caso contrário o próximo passo é verificar se a cláusula vazia pode ser derivada de S por uma regra de derivação chamada resolução (também podemos considerar como uma regra de inferência que gera novas cláusulas). Entretanto, do ponto de vista teórico, uma regra de inferência lógica necessita apenas ser legítima ou seja permitir, apenas consequências lógicas a partir das premissas e ser efetiva. E se temos um computador aplicando a regra de inferência, a tradicional limitação da sua complexidade deixa de existir. Por outro lado, um programa destinado a aplicar uma regra de inferência deve sempre levar em consideração as limitações de tempo e quantidade de memória disponíveis e para tal deve ser "eficiente".

Baseado neste raciocínio ROBINSON[9] elaborou o chamado "PRINCÍPIO DE RESOLUÇÃO", cuja principal vantagem consiste na sua habilidade de evitar um dos maiores obstáculos combinatórios para a eficiência. Além disso, o "Princípio

de Resolução" é poderoso porque ele sozinho, como uma Única regra de inferência, forma um sistema completo de lógica de 1ª ordem. No Capítulo II, falaremos em detalhe sobre o princípio da Resolução.

I.2 - Pesquisas Iniciais

Seguindo a linha de LINS[25] foram desenvolvidas diversas pesquisas (Teses de Mestrado) no Departamento de Informática da PUC/RJ, e agora no Instituto Militar de Engenharia, buscando sempre a construção de um provador automático de teoremas, capaz de resolver problemas constantemente encontrados em provadores como por exemplo: geração de cláusulas que não serão utilizadas nas provas; programas muito grandes que mesmo assim não são bastante gerais, e finalmente estouro de memória na geração das cláusulas. Em todas essas pesquisas o autor trabalhou como orientador, co-orientador, membro de banca de exame, ou de alguma outra forma.

I.2.1 - Provador para $(BAC)_i$

Dessas pesquisas mencionadas, podemos citar as que orientamos ou co-orientamos, sempre pensando na construção desse provador automático (ou semi-automático) de teoremas. A primeira delas foi "Experimentação para um sistema e prova automática de teoremas" cujo resultado prático foi de grande utilidade no desenho do sistema geral de teoremas.

Neste trabalho foi desenvolvido um sistema cujo objetivo era implementar eficientemente (em LINS[25] já havia sido implementado de forma preliminar) toda a teoria desenvolvida por Lins de Carvalho.

Foram feitos experimentos de prova de teoremas para extensões da Álgebra Booleana de Classes, denotadas por $(BAC)_i$.

Como o objetivo desse trabalho era a eficiência em tempo de execução, ele foi programado de uma forma linear, sem nenhum tipo de estruturação.

O sistema desenvolvido consta basicamente de duas partes distintas:

(i) TRADUÇÃO

(ii) REFUTAÇÃO

Esta tradução é feita considerando-se $(BAC)_i$ munido de uma estrutura definicional e procura-se através de uma função de tradução, reduzir todos os símbolos para predicados e funções definidas nas cláusulas, para um Único símbolo predicado E.

O autor experimentou, com exemplos classificados, a utilização deste sistema PASSOS[27]. Foram feitas experiências com 400 fórmulas em $(BAC)_0$, 100 em $(BAC)_1$ e 20 em $(BAC)_2$. A tabela no Apêndice D mostra um resumo dos resultados da experiência onde o par (i, j) tem o seguinte significado:

(0,0) não foi usado o predicado de comunicação C.P.

(1,0) C.P. utilizado para igualdade

(0,1) C.P. utilizado para . , e não utilizado para igualdade

(1,1) C.P. utilizado para igualdade e para as funções.

I. 2.2 - Construção de Modelos Minimais no Universo de Herbrand

A segunda pesquisa PION[35] foi "Sistema interativo para a construção de modelos minimais no Universo de Herbrand" Este sistema tem pré-fixado no seu escopo, símbolos funcionais (SF), constantes, símbolos predicativos (SP) e variáveis da teoria.

Quando trocamos de teoria, temos que mudar praticamente todo o programa. É necessário especificar outros SF, SP constantes, variáveis. Para o programa se tornar geral, teríamos que especificar infinitos símbolos. Então teríamos um programa desproporcidnal e bem complexo. Uma palavra que numa linguagem fosse predicado em outra poderia ser variável e vice-versa.

Este sistema que foi concebido para construção de modelos minimais no universo de Herbrand e que também prova teoremas, serviu de modelo para a nosso sistema geral e consequentemente gerou nossa terceira pesquisa.

I..2.3 - Analisador Sintático para Prova Automática de Teoremas

"Analisador Sintático para prova autométicа de teorema" ROCHA[36], tenta-se melhorar as deficiências da pesquisa anterior e tenta-se incluir novas estratégias (heurísticas).

O sistema foi especificado com cuidados de uma programação modular e por isso ficou mais eficiente e econômico.

Foi implementado um analisador sintático para as cláusulas de entrada, pois antes, um pequeno erro par parte do usuário, causava resultados inesperados. A falta de um ponto após a palavra "not", por exemplo, o sistema teria sis.

tematicamente "overflow" de memória, devido a problemas no processo de "pattern-matching" do SPITBOL.

I.2.4 - Manipulador de Teorias Definicionais LANZELOTE[33]

É um sistema conversacional, implementado em SPITBOL que utiliza uma "query-language" especialmente projetada para a classificação de espaços topológicos, LANZELOTE[32]. Essa pesquisa vai ser incorporada mais tarde, 'quando da criação de um laboratório de prova automática de teoremas, veja um resumo no Apêndice D.

I.2.5 - Provador de Teoremas para "Point Set Topology"

E finalmente a Última pesquisa foi na direção de um provador automático de teoremas descrita no Apêndice B onde, o autor construiu um provador para Point Set Topology (PPST) resolvendp vários problemas do provador do Apêndice A, mas sua operação ficou um pouco artesanal, pois tentava-se provar somente nos níveis da topologia sem traduzir para $(BAC)_i$ e quando precisavamos do $(BAC)_i$, recuperava-se o resultado de PPST e davamos entrada separadamente para o provador $(BAC)_i$. Essa solução foi implementada devido aos problemas muito complexos de se fazer todo o processo em um só programa sem recuperação de memória.

Após todo esse trabalho inicial, o autor optou por uma solução que resolvesse esse problema e fosse geral. Daí surgiu a idéia do GERADOR DE PROVADOR que passa-se a explicar.

1.3 - Objetivo da Tese - Gerador de Provador de Teoremas

I.3.1 - Idéia Geral

É a proposta de um SGPT (Sistema Gerador de Prova-

dor) que basicamente consta de idéias sobre um programa que modifica outro programa, com recuperação de memória.

Esquematicamente existe um programa no nível do usuário, isto é, o usuário tem contato com esse programa. É escrito numa linguagem projetada, especificamente para esse fim, que consta de elementos do tipo: declaração da linguagem de 1ª ordem a ser usada, axiomas e definições da teoria dessa linguagem. Funciona como uma declaração em programação, onde dados seriam as constantes, símbolos funcionais, símbolos predicativos, etc.

O compilador dessa linguagem monta tabelas, e os "patterns" para o analisador sintático. Este programa modificador de programa, vai editando um outro programa, com os dados inseridos pelo usuário.

Portanto, estamos gerando um programa, para uma determinada teoria, a partir de um programa modificador.

O programa gerado vai constar de definições das tabelas, definição da linguagem, isto é, os "patterns" em SNOBOL que definem, símbolos funcionais, símbolos predicativos, etc.

Esta é a primeira parte do trabalho, ou seja, a criação de um gerador (ou editor) automático de programas.

A segunda fase é gerar, a partir das definições, um grafo definicional, como o visto no item I.5.1, isto é, monta-se uma matriz que mostra a relação entre as definições. Na apresentação do problema, discutiremos como construir essa matriz. O autor não conhece nenhuma pesquisa que USOU essa idéia e provou que funciona.

A importância da criação dessa estrutura definicional, vai aparecer quando se estudar a estratégia de prova

por niveis. Passa-se para o nível seguinte no grafo, quando não se conseguir provar o teorema naquele nível. Também o autor provou a completeza dessa estratégia.

Na terceira parte, o programa gerador (compilador) vai montar tabelas de tradução. Essa tabela de tradução começaria numa "nodo" escolhida do grafo. Então o programa faria:' Crie TRADUZ a partir do nodo X, (a partir da definição X). Daremos exemplos em "Point Set Togology".

A quarta parte, será feita também pelo usuário, são declarações de que estratégias serão montadas no programa gerado. Esses módulos de estratégias, como conjunto suporte, linear, predicados de comunicação (CP), etc., estão armazenados em algum arquivo independente. O montador (gerador de programas que provam teoremas) recupera nos arquivos esse módulos e monta (edita) no programa que está sendo gerado.

Essas estratégias farão parte do novo programa, assim como os "patterns" já gerados.

Como pesquisa paralela, tem-se alunos construindo novas estratégias, modulares, construídas para não haver conflitos de nomes, parâmetros, etc. Pretende-se ter nesses arquivos (biblioteca de estratégias), todas as estratégias existentes na bibliografia.

I.3.2 - Contribuição da Pesquisa aqui descrita

A interação homem/máquina proporciona a geração do provador semi automático de teoremas, da Teoria definida pelo usuário.

Esse sistema gerador de provador contribuirá para o pesquisador em prova automática de teoremas em vários aspectos: o primeiro é a de ser um laboratório de prova automática de teoremas em teorias definicionais, onde o matemá-

tico poderá conliccer melhor a teoria (veja ítem 1.3.3) ; o segundo de onde o pesquisador em prova automática de teoremas poderá estudar o comportamento dos provadores gerados e tentar construir um, o mais automático possível (mais rápido, menor quantidade de memória utilizada) que poderiam ser utilizados em computadores pequenos.

I.3.3 - Aspectos na Teoria do conhecimento'

Com o sistema funcionando, o pesquisador ao utilizá lo para provar teoremas, poderá descobrir o que é "inteligente" (naquela teoria]. Em outras palavras, se o sistema segue um caminho e esgota quase todas as possibilidades, o usuário define outro caminho e novamente esgota os limites, isto é, o usuário deixa para o sistema a parte "braçal" até descobrir melhor caminho, poderemos, com o uso do sistema , descobrir o que é, realmente,, difícil naquela teoria matemática. Isto do ponto de vista da teoria do conhecimento é muito importante*.

1.4 - Sumário da Tese

No Capítulo I tem-se uma história sobre prova automática de teoremas, PASSOS[28], no Capítulo II uma apresentação breve o Princípio da Resolução, com suas ~~el~~áusulas, ~~re~~gra de Resolução e ~~e~~stratégias; a motivação inicial do trabalho; uma explicação sucinta da técnica dos predicados de comunicação junto com a estratégia de P-refutação; uma descrição das pesquisas anteriores até a pesquisa atual; o objetivo e uma descrição sumária da tese e, finalmente, a idéia do autor sobre o 'resultado da tese.

No Capítulo IÍI, tem-se a apresentação do problema final e sua solução de forma esquemática. No Capítulo IV são feita; alpinas fundamentações teóricas e mostradas solu

ções das partes principais do sistema por intermédio dos algoritmos: construção dos grafos definicionais, das traduções por níveis e demonstrações sobre o funcionamento de tais algoritmos.

E finalmente no Capítulo V as conclusões e próximas pesquisas, e a bibliografia.

No Apêndice A apresenta-se uma axiomatização para a teoria em que aplicamos o sistema provador, Point Set Topology, abreviadamente P.S.T., BRAUM[29], KURATOWSKI[30], um provador automático para essa teoria com alguns exemplos e suas dificuldades.

No Apêndice B, descreve-se o provador de teoremas para PST, onde introduzimos a heurística GERADOR DE LEMAS com a estratégia de TRADUÇÃO POR NÍVEL. Foram feitos exemplos e detetadas as novas dificuldades.

No Apêndice C é feita uma descrição da linguagem bem como da implementação desse sistema para gerar provadores automáticos de teoremas.

CAPÍTULO II

RESOLUÇAO

II.1 - Princípio de Resolução

Pode-se consultar CHANG[11], para uma discussão completa de RESOLUÇÃO. Faremos aqui uma breve descrição.

Existem três problemas básicos envolvidos ao suprirmos computadores de capacidades dedutivas:

(i) acharmos representações "adequadas" para fatos e relações;

(ii) acharmos' regras de inferências "adequadas" para manipularmos esses fatos e essas relações, e

(iii) aprimorar as regras para produzir programas eficientes que possam achar provas numa quantidade razoável de tempo e memória,

Como representação adequada usamos a lógica matemática MARGARIS[5], KREISEL[12], SUPPES[13], WANG[14], como regra de inferência usamos RESOLUÇÃO ROBINSOM[9] e finalmente para aprimorar as regras usamos, "estratégias"", isto é, o refinamento de RESOLUÇAO. E nessa terceira parte que os pesquisadores ainda utilizam seus esforços e que é um dos objetivos principais da dissertação do autor.

Comecemos por escrever informalmente algo sobre a representação no computador de dados e relações através de uma entidade chamada CLÁUSULA. Faremos uma descrição de Resolução e finalmente apresentaremos algumas das mais conhecidas estratégias.

II.1.1 - Cláusulas

Resolução usa fórmulas da lógica, SUPPES[15], de 1ª. ordem que são escritas numa forma especial chamada cláusula. Cláusulas são escritas usando cinco tipos de símbolos: variáveis símbolos funcionais, constantes, símbolos relacionais, e os dois símbolos lógicos ¬ (NAO) e ∨ (OU). As variáveis, funções, e constantes são combinadas para formar termos que representam elementos de algum domínio, i.é., números reais, inteiros, etc. Exemplos: +(x,y), h(x,y), f(x,y)), z, e a onde a é uma constante, x, y, z são variáveis e +, h, f são funções. Um símbolo relacional seguido por um número apropriado de termos é chamado um átomo e é a declaração verdade-falso básica. Por exemplo, =(x,+(0,x)) que na notação infixa seria x=x+0 e P(x,y,f(x,y)) são átomos. Um literal é um átomo ou a negação lógica de um átomo; por exemplo, os dois átomos acima e P(x,y,f(x,y)) = (0,1) etc. Finalmente, uma cláusula é um conjunto de literais que podem ser interpretados como se estivessem na forma disjuntiva. Exemplo 1, se temos uma sentença lógica de 1ª ordem qualquer ∀x{P(x) ⇒ {(∀y) P(y) ⇒ P(y) P(f(x,y))} ∧ ¬(∀y) {Q(x,y) ⇒ P(y)}}} passamos para a forma normal conjuntiva (veja programa do autor em LISP, PASSOS[7]) eliminamos os quantificadores universais e teremos:

{{¬P(x) ∨ P(y) ∨ P(f(x,y))} ∧ (cláusula 1)

{¬P(x) ∨ Q(x,g(x))} ∧ (cláusula 2)

{¬P(x) ∨¬P(g(x)))}} (cláusula 3)

Finalmente as colocamos numa forma usual, que é sem as símbolos ∧ e ∨ e assim teremos um conjunto de cláusulas onde cada cláusula é um conjunto de literais e cada literal é uma forma atômica ou a negação de uma fórmula atômica.

cláusula 1 {¬P(x), ¬P(y), P(f(x,y))}

cláusula 2 {¬P(x), Q(x,g(x))}

cláusula 3 {¬P(x), P(g(x))}

Exemplo 2, agora suponhamos que R represente a relação "igual" Q represente "menor que.", f represente "+", g representa "*", e 0 e 1 os dois números reais, zero e um. E então veremos como ficam as cláusulas e quais são seus significados:

Qx,y;Qy,x;Rx,y A v B v C	se x não é menor que y e y não é menor que x, então x e y são iguais. A ∧ B → C ⇌ A v B v C ,C (cláusula 1)
Q0,x Q0,y Qf(x,y),g(x,y)	se x e y são positivos então x+y<x*y (cláusula 2)
¬Q0,x Q0,y ¬Q0,g(x,y)	se x é positivo e y não é positivo. então x*y não é ,positivo. (cláusula 3)

observa-se que sob as interpretações dos símbolos dados, a 1ª. e 3ª. cláusulas representam declarações verdadeiras enquanto a 2ª. cláusula é uma declaração falsa. É lógico que, mudando a interpretação dos símbolos, os valores verdadeiro e falso das três cláusulas podem mudar.

Essa interpretação vista em KREISEL[12], então consiste de um domínio D de objetos e uma escolha das constantes, funções, e relações para fixar os objetos de D, funções sobre D e relações em D respectivamente.

Um átomo é verdadeiro se a relação indicada pelo símbolo relacional é a verdade dos objetos de D indicados pelos termos do átomo. (veja exemplo acima). A negação de um átomo é verdadeira se o átomo é falso. Observe que nenhuma atribuição é feita a variáveis; se ocorrem variáveis, então o literal pode ser verdade para alguns valores da variável e falso para outros. Usualmente tem-se interesse em determinar se as declarações são VALIDAS, i.e., são verdades em qualquer interpretação. Entretanto, é mais prático determinar se um conjunto de cláusulas é sempre falso ou seja insatisfatível.

Uma cláusula é "tornada falsa" numa interpretação se existe uma combinação de valores do domínio para as variáveis de todos os literais que compõem a cláusula falsa.

Um conjunto de cláusulas é insatisfatível se para qualquer interpretação I, existe uma clásula no conjunto que é "tornada falsa" por I, Então normalmente se expressa a negação do teorema que se deseja provar como um conjunto de cláusulas. Ou seja, se w é uma consequência lógica de S (conj. de axiomas, por exemplo), o conjunto $S \cup \{\neg w\}$ é insatisfatível. Por outro lado se $S' = S \cup \{\neg w\}$ é insatisfatível (mostrar uma refutação para S'), então w deve ser consequência lógica de S. Por conseguinte, para se provar um teorema, escreve-se um conjunto de cláusulas que representam os axiomas da teoria e ainda as hipóteses espéciais e em seguida escreve-'se uma cláusula ou cláusulas que representem a negação do teorema a ser provado e descobre-se se esse conjunto é insatisfatível, ou não. (teorema falso).

II.1.2 - Resolução

A regra de resolução foi projetada especificamente para uso em computadores, ROBINSON[9]. É uma generalização da regra lógica "de $a \Rightarrow b$ e $b \Rightarrow c$, deduz-se que $a \Rightarrow c$" veja em HENSCHEN[16].

Consideremos primeiro duas cláusulas sem variáveis (chamada cláusula básica), pvqvr e $\neg$pvsvt. A primeira é lógicamente equivalente a $\neg(qvr) \rightarrow p$ e a segunda $p \rightarrow (svt)$ pela identidade conhecida $a \rightarrow b \equiv \neg avb$. Agora pelas regras acima podemos concluir que $\neg(qvr) \Rightarrow (svt)$ ou, em forma de cláusula qvrvsvt. Que se lembramos das duns cláusulas consideradas inicialmente essa Última tem todos os literais da 1ª e da 2ª cláusula, exccto os que tem a negação em uma cláusula e na outra não(são complementares) p, $\neg$p. Podemos deduzir então essa 3ª cláusula qvrvsvt diretamente das duns primeiras. Isso é, a cláusula deduzida é a disjiinção das 2 primeiras tirando o par complementar.

O resultado principal sobre resolução nesse exemplo é que de qualquer conjunto de cláusulas básicas que é insatisfatível pode-se deduzir, por repetição de resolução, a cláusula vazia, que é a cláusula sem nenhum literal, e representa-se por □. Para encontrar essa cláusula deve-se no último passo resolver 2 cláusulas da forma p e ¬p, isto é, um par de cláusulas opostas.

Quando existirem variáveis, temos que faze; duas modificações. Primeiro, por razões técnicas, deve-se trocar os nomes das variáveis em uma das cláusulas, tal que as duas cláusulas não tenham variáveis em comum; esse processo é chamado separação de variáveis. Segundo, os dois literais que supostamente conflitam podem conter termos em posições correspondentes que não são idênticos. 'Entretanto, pode ser possível achar uma substituição de termos por variáveis que tornará os' dois parecidos, (Lembrar que as variáveis representam valores arbitrários e, contudo, p-ode-se trocá-las por qualquer outro termo). Vários algorítmos, chamados algoritmos de unificação, existem para determinar se ou não os átomos podem ser tornados parecidos ou não. Se podem ser unificados, esses algorítmos produzem uma substituição mínima , para fazê-lo, que é o chamado, unificador mais geral. (MGU) . Então, como em resolução básica, obtemos uni resolvente formando a disjunção de todos os literais que restam, e então, aplicamos o MGU. Como no caso básico, dado um conjunto S insatisfatível de cláusulas, existe uma dedução da cláusula vazia de S por Resolução. Portanto, agora temos uma representação e uma regra.de manipulação que mesmo quando existirem variáveis, são suficientes para obter uma prova (gerar cláusulas vazia) em qualquer teoria que possamos representa-la em cláusulas.

Alguns exemplos:

Em Português	na forma de cláusula
1) a tem a propriedade P	1) Pa

2) Hipótese de indução Para qualquer x, se x tem a propriedade P então f(x) também tem.	2) $\neg Px \vee Pf(x)[\equiv Px \rightarrow P(f(x))]$
3) Teorenia a ser provado : "f(f(a)) também tem a propriedade P".	3) Negação do teorema $\neg Pf(f(a))$

Lembrar que com resolução obtem-se provas por contradição. Agora na nossa linguagem matemática formal fazemos assim: primeiro, se a tem a propriedade P e se sempre que um objeto tem a propriedade P, terá também para f desse objeto, pode-se deduzir que f(a) também tem essa propriedade P; analogamente, desde que f(a) tem essa propriedade P, f(f(a)) pelo mesmo raciocínio, terá a propriedade P como queríamos demonstrar. Mas se vamos usar a forma de cláusulas, deteremos uma refutação da seguinte maneira.

4) P(f(a))	de 1 e 2, substituindo a por x em 2.
5) □	de 5 e 3, substituindo f(a) por x em 2 contradição.

II.1.3 - Estratégias

Enquanto resolução é teoricamente suficiente para provar qualquer teorema da lógica de primeira ordem, existe um problema pratico: os computadores têm memória e quantidade de tempo limitados. Portanto mui'tos refinamentos de resolução foram desenvolvidos.com o propósito de construir programas que gerarão resoluções relevantes em grande escala, Pesquisadores continuam dispendendo grandes esforços na busca de certas restrições que possam ser impostas sobre a resolução de maneira a ainda preservar sua propriedade de ser completa. Dai' surgiram um grande número de estratégias: "set of suport" pro Wos et Al, "Semantic Resolution" por Slagle ambas em 1967; "Merging" por Andrews em 1968; "Model

Elimination" por Loveland em 1969; "Input Resolution" por Chang em 1970. E outro enfoque para eficiência de resolução utilizado por Slagle[18] em 1972 no sistema desenvolvido para Teoria dos Conjuntos, que se baseia na utilização de certos conhecimentos sobre alguns prcdicados especiais (igualdade é o mais usado).

Então um programa com alguma estratégia para refinar resolução teria-uma chance melhor de encontrar uma prpva, se ele existir, do que em programas que gerassem resolventes aleatoriamente. Nós classificaremos e descreveremos sucintamente alguns refinamentos nossos conhecidos.

Vamos classificar as estratégias em três grupos distintos: estratégias de simplificação, estratégias de refinamento e estratégias de ordenação,

As de simplificação são aquelas que determinadas cláusulas podem ser eliminados sem que o conjunto simplificado deixe de ser insatisfatível se o original o for. Assim, o emprego das estratégias de simplificação ajuda a reduzir a taxa de geração de novas cláusulas. Por exemplo: eliminação de tautologias, pois qualquer conjunto insatisfatível que contenha tautologias, continuará insatisfatível após a remoção destas; eliminação pela avaliação de predicados, se um literal em uma cláusula é avaliada como verdadeiro, então toda a cláusula pode ser eliminada sem afetar a satisfatibilidade do conjunto e, também, se um literal em uma cláusula é avaliado falso então a ocorrência desse literal na cláusula pode ser eliminada e teremos uma cláusula menor; eliminação por inclusão ("subssumption"), se C e D são duas cláusulas diferentes e não vazias, então, C inclui D se existe uma substituição 6 tal que C6 $\subseteq$ D e então podemos eliminar de um conjunto finito S de cláusulas, qualquer cláusula D que é incluída por uma outra cláusula de S. Existe teorema em **ROBINSON**[9] que mostra **tal** possibilidade e também algorítmos para fazer essa eliminação.

O segundo grupo de estratégias são as de refinamento, e são baseados no resultado da teoria de resolução que estabelece que nem todos os possíveis resolventes necessitam ser gerados para a obtenção de uma refutação, assim vamos impor determinadas condições (restrições) para que novas resolventes (cláusulas geradas a partir dos iniciais) , sejam gerados. Comecemos com o refinamento chamado Filtragem de Ancestrais (Ancestry Filter AF) LUCKHAM[19] temos que representar uma prova em resolução por um grafo e impomos a restrições. Um grafo está na forma de AF se cada nodo corresponde a: uma cláusula em S; uma cláusula descendente imediata de uma outra em S; uma cláusula descendente imediata de duas cláusulas que não estão em S, mas que é uma ancestral da outra. E veremos que a partir do nível 2' o número de resolventes gerados é menor..

Agora veremos um outro.refinamento muito utilizado chamado conjunto suporte (Set of Support,SS). Na estratégia SS, escolhemos um subconjunto K do conjunto inicial S de cláusulas tal que S-K é satisfatível. Podemos, por exemplo, escolher K, como o conjunto de cláusulas originárias, da negação 'do teorema a ser provado. Dizemos que as cláusulas em K possuem suporte. Essa estratégia deve ser usada combinada com outras estratégias que não restringem no nível 1 e em LUCKHAM[19] temos a prova do teorema que garante a completeza da resolução usando essa estratégia combinada com outra ou-não.

Para se obter o melhor proveito na ,aplicação de SS, o importante é a escolha do subconjunto K de S. Uma escolha natural,para K, seria o conjunto das cláusulas em S, relativas·a negação do teorema que se quer provar. Assim S-K seria o conjunto das cláusulas relativas aos axiomas. Como os axiomas de uma teoria devem ser um conjunto satisfatível,S-K seria satisfatível. Intuitivamente, o que SS faz, é evitar que se procure uma prova, para um conjunto que já sabemos que é satisfatível. Isso significará em prova automática de teoremas, que muitas cláusulas correspondentes a lemas triviais da teoria deixarão de ser gerados.

Resolução com merge é um refinamento em que as cláusulas "merge" (geradas com a condição merge) contém sempre um número de literais menor que uma das cláusulas que se resolveram para formá-la. Isto torna-se evidente se observarmos a definição de Merge, ANDREWS[21]. Note que se a intercessão do que resta dos conjuntos (tendo sido extraído o par complementar) não é vazia, então ao se fazer a união para se formar o resolvente este deverá ser menor, em pelo menos um literal, que é uma das cláusulas que se resolveram para formá lo. Assim sendo resolução com "merge", tende a gerar cláusulas cada vez menores, e evidentemente como o nosso objetivo é a cláusula vazia, as cláusulas menores são sempre mais promissores.

Um outro refinamento fácil de usar é o chamado P_1 - refutações e que também é completo NILSON[22]. A resolução do número de cláusulas ao se utilizar este refinamento deverá ser efetiva desde o primeiro nível, pois se está evitando gerar resolventes de duas cláusulas quando ambas não são positivas: Exemplo: seja $S = \{C_1, C_2, C_3, C_4\}$ $C_1 = \{Q(x), P(a)\}$, $C_2 = \{\neg Q(x), P(x)\}$, $C_3 = \{\neg Q(x), \neg P(x)\}$, $C_4 = \{(Q(x), \neg P(x)\}$ onde observamos ser C_1 a única cláusula positiva. Assim as cláusulas C_2, C_3 e C_4 só gerarão resolventes combinadas com C_1, isto é, $R_p^1(C_2, C_3, C_4) = \emptyset$, o que não aconte'ceria se não usássemos o refinamento. Existem outras refinamentos que vamos descrevê-los com mais detalhes no decorrer do trabalho.

Agora o terceiro e último grupo são as estratégias de ordenação que diferem dos dois grupos anteriores pois não procuram eliminar cláusulas ou evitar que as cláusulas que não satisfaçam a determinados critérios sejam geradas. Elas simplesmente se propõem a trazar uma certa organização ao processo, expressa através de uma propriedade a ser seguida para as possíveis combinações de cláusulas e portanto evitaria gerar todos os elementos de $R(S), R^2(S), \ldots R^{n-1}(S)$ ($R^o(S) = S$) e sim procurar atingir a cláusula vazia o mais diretamente possível.

A principal é a de preferência das unitárias ("Unit preference"). Nessa estratégia, procura-se resolver em primeiro lugar os pares de cláusulas unitárias (cláusulas com apenas um literal). Evidentemente se algum par de cláusulas unitárias for resolvido então teremos encontrado de pronto *a* cláusula vazia. Em seguida tenta-se resolver as cláusulas unitárias com as binárias (dois literais) e assim por diante. Sempre que for encontrada uma cláusula nova voltamos a tentar resolvê-la contra as cláusulas unitárias, depois com as binárias e assim sucessivamente. Desta maneira existirá sempre a possibilidade da pesquisa se desenvolver em direção a um ramo infinito e infrutífero, e para prevenir tal situação geralmente se estabelece um limite de nível. Quando o limite de nível não permitir mais resoluções entre cláusulas unitárias, entre unitárias e binárias, então se deve tentar resolver cláusulas binárias, binária com ternária , sempre que for encontrada uma nova cláusula unitária volta-se entretanto ao início do esquema anterior.

Finalmente temos as Heurísticas que apesar de tirarem a completeza do Princípio da Resolução de Robinson, podem melhorar a eficiência da pesquisa, Assim sendo quando as finalidades são práticas e não teóricas, a tentativa de tais heurísticas é plenamente justificada. Mesmo combinando estratêgias pode-se reduzir o número de resoluções efetuadas na procura de um refutação, entretanto esta combinação não mantem em geral a completeza do sistema. Assim só podemos aplicar tal combinação como uma tentativa. .

II.2 - Motivação Inicial

O enfoque dado por Slagle[18] possibilitou uma maior eficiência na prova automática de Teoremas, em Teoria dos conjuntos, usou-se regras similares a paramodulation ROBINSON[23], que é o conhecimento sobre alguns predicados especiais dos quais o mais comumente usado é o da igualdade.

Esse sistema, apesar de ter melhorado em relação a anteriores, tinha problemas s.e tentássemos provar teoremas simples como $\overline{A \cap B} = \bar{A} \cap \bar{B}$ (lei de Morgan). Uma grande quantidade de "adivinhação" e de interferência humana se faz necessária para conserguirmos a prova.

Posteriormente Bledsoe[24] desenvolveu sistemas mais práticos que o de Slagle. Esses sistemas se baseiam em duas regras principais: Splitting e Reduction. A regra Splitting é justificada pelo teorema "se C_1 e C_2 são cláusulas e S é um conjunto de cláusulas no cálculo de predicado de 1ª. ordem, e se C_1 e C_2 não tem variáveis comuns, então $S\{C_1 \vee C_2\}$ é insatisfatível se e somente se, ambos $S\cup\{C_1\}$ e $S \cup \{C_2\}$ são insatisfatível". Portanto esse teorema nos permite "partir" uma cláusula dada $C_1 \vee C_2$, quando C_1 e C_2 não tem variáveis em comum, em duas cláusulas C_1 e C_2. Nevine[25] enfraqueceu esta restrição, provando que o teorema aindavale desde que C_1 e C_2 não tenham mais que uma variável comum.

A regra de Reduction é justificada pelo critério da eliminabilidade da teoria das definições, que nos possibilita eliminar símbolos para funções e predicados definidos em um contexto qualquer através de um processo de redução a primitivas LINS[25].

Bledsoe e Bruell[26] desenvolveram um sistema homem / máquina que usa um subsistema baseado em Reduction e Splitting,

LINS[25] seguindo a mesma linha de Bledsoe e Nevine apresentou um novo método para transformar um dado conjunto S de cláusulas em outro conjunto de cláusulas S', mediante a introdução de novos símbolos para predicados.

Estes novos predicados, denominados "predicados de comunicação", serão introduzidos mediante esquemas válidos *c* são tais que o conjunto S será insatisfatível, se e somente se, S' o for também. Em particular, dada uma cláusula $C_1 \vee C_2$, um conjunto S de cláusulas c um novo símbolo predica-

do P, teremos que $S \cup \{C_1 \vee C_2\}$ será insatisfatível se $S \cup \{C_1 \vee P(\bar{V}), C_2 \vee \neg P(\bar{V})\}$ foi insatisfatível.

Embora exista uma grande semelhança entre a técnica acima e a Splitting de Bledsoe, a diferença básica entre elas reside no fato de que a primeira não impõe restrições quanto às cláusulas C_1 e C_2, ao passo que a última exige que as mesmas, não possuam variáveis comuns. (Nevine deixa ter só uma variável comum).

Carvalho também introduziu uma nova estratégia, chamada P.-refutação, e provou que é completa. Essa estratégia é adequada para manipular conjuntos de cláusulas P-separable, por exemplo aqueles que contém "predicados de comunicações PASSOS[27].

O autor trabalhou com o sistema provador para $(BAC)_i$ de R.L. de Carvalho e fez uma série de experiências. Esse provador usa essa nova estratégia chamada Predicados de Comunicação ver LINS[25]. Os resultados dessas experiências estão relatados em PASSOS[27].

A técnica de P.C. usada no provador para $(BAC)_i$ é explicada no ítem a seguir,

II.3 - Predicados de Comunicação (P.C.)

II.3.1 - O Procedimento

Uma maneira comum que usamos para provar que uma certa classe A de objetos é igual a outra classe B de objetos, numa dada teoria, é:

a. Provar que A está contido em B.

b. Provar que B está contida em A.

Em geral, a e b são ações independentes, no sentido que a premissa usada em a não é relacionada com aquela usada em b.

Esse procedimento de prova pode ser mecanizado num provador baseado em Resolução. Por exemplo:

Vamos assumir que igualdade de classes é definida por:

$$\forall x \forall y \; (x=y \rightleftharpoons (x \leqslant y \wedge y \leqslant x)) \qquad 3.1$$

na nossa forma de cláusulas temos

$$\begin{array}{ll} A1. & \neg x = y \vee x \leqslant y \\ A2. & \neg x = y \vee y \leqslant x \\ A3. & x = y \vee \neg x \leqslant y \vee y \leqslant x \end{array} \qquad 3.2$$

Para provar que A=B num procedimento de refutação, começamos negando A=B. Verificamos as cláusulas em (3.2) e fazemos A=B contra A3 para tentarmos a cláusula vazia.

Portanto ($\neg$A=B) e A3 por refutação obtemos a cláusula $\neg A \leqslant B \vee \neg B \leqslant A$. Essa deve ser' reduzida contra o conjunto de cláusulas, S, que definem os axiomas da teoria onde A e B são objetos, e ainda do conjunto de condições sobre esses objetos (hipóteses adicionais).

Agora usando a nova idéia fazemos:

(i) Trocar A3 por

$$\begin{array}{ll} A3'. & x=y \vee \neg x \leqslant y \vee P(x,y) \\ A4'. & \neg P(x,y) \vee \neg y \leqslant x \end{array} \qquad 3.3$$

(ii) Tentar obter $P(x,y)\delta$ de $S \cup \{A3'\}$ para alguma substituição 6.

(iii) Tentar obter a cláusula □ de $S \cup \{(\neg y \leq x)\}$

O símbolo predicado P introduzido é novo, e nós *fi*-zemos que esse—P é um predicsdo de comunicação (ou P C)

11.3.2-Motivação para o uso de P C

Dado um conjunta S de cláusulas e um conjunto P de novos símbolos predicados, trocamos S por um outro conjunto S' de cláusulas, que contém símbolos predicados em P, tal que:

1. S é insatisfatível se e somente se S' é insatis-. fatível.

2. Tem vantagem de certas características especiais de S' para o projeto de provadores de teoremas, nos quais em certo sentido imitam o homem provan do teorcmas.

Por exemplo na definição de topologia no Apêndice A, vemos que t5 (ver forma sem ser expandida) ainda usa frases, ("frase" é uma disjunção $A_1 \vee A_2 \vee \ldots \vee A_n$ onde A_i é uma con-¿unção de literais) mas para usarmos Resolução temos que usar t5 na-sua forma expandida (isto é, sem possuir frases).

A maneira natural (humana)' para provar que uma cer-ta família A de conjuntos é uma topologia para um certo con junto a, é feita no Apêndice A.

II.3.3 - A estratégia P C

Dado um conjunto de cláusulas S, introduzindo os C P'S, $P_1, P_2, \ldots, P_k$ por uma maneira válida, (Veja teorema 3.22 de LINS[25]). Obtemos um conjunto S' de cláusulas. E seja P o conjunto de C.P. 'S introduzidos. Vamos considerar S' parti-cionado em dois (2) conjuntos de cláusulas:

- um primeiro conjunto S_1 de cláusulas que não contém ocorrências negativas de literais em P. (não tem C P 'S negado).

- um segundo conjunto S_2 de cláusulas que contém pelo menos uma ocorrência negativa de um literal em P.

Por exemplo:

$S = \{p_1 \vee p_2 \vee p_3 \vee p_4,\ p_1 \vee p_2 \vee p_3 \vee p_5,\ p_1 \vee p_2 \vee p_3 \vee p_6,\ \neg p_1, \neg p_2, \neg p_3, \neg p_4 \vee \neg p_5 \vee \neg p_6\}$

$P = \{q\}$ $\quad S' = S \cup \{q\}$

$S' = \{p_1 \vee p_2 \vee p_3 \vee q, \neg q \vee p_4,\ q \vee p_5, \neg q \vee p_6, \neg p_1,\ p_2, \neg p_1\ ,\ \neg p_2\ ,\ \neg p_3, \neg p_4 \vee \neg p_5 \vee \neg p_6\}$

$S_1 = \{p_1 \vee p_2 \vee p_3 \vee q, \neg p_1, \neg p_2, \neg p_3, \neg p_4, \neg p_5, \neg p_6\}$

$S_2 = \{\neg q \vee p_4, \neg q \vee p_5, \neg q \vee p_6\}$

Vamos no início somente ativar cláusulas em S_1, resolvendo uma contra outra,

1. $p_1 \vee p_2 \vee p_3 \vee q$	em S_1	
2. $\neg p_1$	em S_1	
3. $p_2 \vee p_3 \vee q$	(1,2)	
4. $\neg p_2$	em S_1	
5. $p_3 \vee q$	(3,4)	
6. $\neg p_3$	em S_1	
7. q	(5,6)	[essa cláusula é chamada "pura" só contém e P 'S)

Quando uma cláusula pura for obtida passamos a ati-

var cláusula em S_2.

8.	$\neg q \vee p_4$	em S_2
9.	$\neg q \vee p_5$	em S_2
10.	$\neg q \vee p_6$	em S_2
11.	p_4	(7,8)
12.	p_5	(7,9)
13.	p_6	(7,10)
14.	$\neg p_4 \vee \neg p_5 \vee \neg p_6$	em S_1
15.	$\neg p_5 \vee \neg p_6$	(11,14)
16.	$\neg p_6$	(12,15)
17.	$\square$	(13,16)

II.3.4 - Conjunto P-separável

S é P-separável se e somente se cada cláusula em S contém pelo menos uma ocorrência negativa de um literal em P.

Isso nos permitirá identificar certas famílias de conjuntos de cláusulas que são adequados para serem tratados como se o conjunto P de símbolos predicados fosse um conjunto de C P 'S. Observamos que se P é o conjunto de C P 'S introduzidos e S é o conjunto transformado de cláusulas, então S é P-separável,

Observar que o S' do exemplo anterior é P-separável com P = {q}.

II.3.5 - Conjunto P-refutável

S' é P-refutável se e somente se existe uma sequência $C_1, C_2, \ldots, C_n$ de cláusulas, tal que:

(i) Para cada i, i = 1,2,...n, C_i está em S ou C_i foi obtida (resolvida) por duas (2) cláusulas anteriores , C_j, C_k da sequência.

(ii) Se uma cláusula que contém ocorrências negativas de literais em P é uma cláusula ("paterno") que gerou uma cláusula na sequência $C_1, C_2, \ldots, C_n$ então a outra cláusula ("materna") é cláusula pura em P.

(iii) $C_n = \square$

Exemplo de (ii): cláusula 8, 9, ou 10 são cláusulas que. contém ocorrências negativas de literais em P = {q}.Elas são "paternos" de 11, 12, 13, então a outra cláusula "materna" que resolveu com elas foi a 7, que é uma cláusula pura.

Em LINS[25] temos prova de que um conjunto P-separável S de cláusulas é insatisfatível se e somente se ele é P-refutável.

11.3.6- Completeza de P-refutação

Portanto pelo resultado anterior provado em LINS[25] temos a completeza de P-refutação. Agora cabe aqui uma observação. No Último exemplo que demos, nós obtivemos a cláusula vazia em 17 passos. Mas poderíamos ter 20 passos, 30 etc. Se não fosse os refinamentos usados na refutação (prova de nosso teorema),'os quais relataremos agora.

Compatividade de P-refutação com outras estratégias.

P-refutação e estratégia da preferência mais curta

(i) O comprimento de uma cláusula é o número de literais que contém. Não contar os literais em P. Nós no exemplo Último resolvemos (12,15) quando poderíamos (estava antes) ter resolvido (12,14), preferimos as cláusulas mais curtas.

(ii) Eliminações de literais em P, uma cláusula pura em P, terá preferência sobre outros possíveis resolventes. No exemplo, 11, 12 e 13 foram conseguidos porque ativamos a cláusula pura.

(iii) Um limite para o nível de cláusulas é definido, quando o nível de uma cláusula é definido por:

a) todas as cláusulas do conjunto inicial S tem nível 1.

b) Se C é o resolvente de cláusulas C_1 e C_2, então o nível de C é o maior dos níveis de C_1 e C_2 mais *1*.

Esse refinamento (preferência mais curta) para P-refutação é completo.

- P-refutação e resolução linear não é completo.

- P-refutação e conjunto suporte é completa.

A utilidade prática dessa estratégia, de usar predicados de comunicação, foi provada em PASSOS[27] no provador para $(BAC)_i$ que também vamos utilizar no nosso provador geral de teoremas e que terá como aplicações para (BAC), o Point Set Topology (PST).

CAPÍTULO III

RESUMO DO SISTEMA GERADOR DE PROVADOR

III.1 - Apresentação do Problema

III.1.1 - Definição do problema

Toda pesquisa desenvolvida pelo autor nesses últimos anos teve sempre o objetivo de construir um provador automático de teoremas. Veja Capítulo I e Apêndice A.

Com as dificuldades descritos nos capítulos citados, modificou-se a direção da pesquisa para se buscar um sistema semi-automático, onde o usuário pudesse dirigir, com a sua experiência, o processo de prova de teoremas. Veja Apêndice B.

Analisando-se os programas do Apêndice A verificou-se que eram eficientes para resolver teoremas de $(BAC)_i$ e ineficientes para "Point Set Topology" (PST). Por várias razões já ditas anteriormente. Já o sistema descrito no Apêndice B era mais eficiente para demonstrar teoremas em PST mas não era prático, pois, apesar de introduzidas duas estratégias: tradução por nível e gerador de lemas, não existia uma ligação direta com o sistema $(BAC)_i$ e portanto, quando se fazia traduções para $(BAC)_i$, era necessário chamar o sistema descrito no Apêndice A e manualmente, dava-se entrada nas cláusulas conseguidas como saída do sistema do Apêndice B. Veja esquema lógico no Apêndice B.

Além disso o sistema para $(BAC)_i$ não era modular nem, tinha esquemas de traduções inteligentes, pois considerava-se $(BAC)_i$ munido de uma estrutura definicional e através de uma função de tradução, reduzia-se todos os símbolos funcionais e predicativos encontrados na negação do teorema,

para um Único símbolo predicado "pertence" (ε). Muitas vezes, viu-se depois, não era necessário traduzir até "pertence". Daí a solução de se traduzir por nível.

Foi constatado também que o sistema parava, ou entrava em "loop" quando, sem que o usuário percebesse, entrava-se com um dado estranho ao sistema, por exemplo, a falta de um ponto em "NOT". Isso era desastroso em todos os sentidos. Foi então que se teve a idéia de construir-se um "analisador sintático" que avisasse ao usuário que seu dado de entrada estava errado e portanto deveria corrigí-lo.

A modularidade do sistema era importante, pois queria-se construir estratégias ,ao longo do tempo, e acrescentá-las ao sistema, sem modificar sua lógica.

A necessidade de criar-se um provador de teoremas que não dependesse da teoria a ser testada, isto é, nao seria necessário construir-se um programa tão grande, que a relação das constantes, predicados e símbolos funcionais fosse de tal ordem, que os símbolos relacionados para um desses três elementos não se confundisse, quando mudassemos de teoria. Daí a solução de criar-se um programa gerador de "provador de teoremas", onde somente as constantes, predicados, símbolos funcionais de uma determinada teoria fosse declarados no início, juntamente com as estratégias que já se encontravam armazenadas, daí então gerar um "provador de teoremas".

Assim, com a experiência adquirida nos sistemas descritos, o autor desenhou um sistema geral para prova de teoremas, sem esses problemas citados anteriormente, com a Ótica de um pesquisador em prova automática de teoremaç. O que se quer com esse sistema é que ele descubra novos caminhos e se auto modifique para melhor provar os teoremas. Isto só pode ser feito por meio de experiências num sistema complexo como o que estamos apresentando, isto é, sugerir

soluções mais simples pelo melhor conhecimento da teoria. O sistema foi concebido tendo-se sempre em mente, que o pesquisador quer descobrir o que é "inteligente'! quando se faz matemática com aquela teoria e, então, passar essa informação para a máquina para que se consiga resultados novos na teoria finalmente.

III.1.2 - Solução do problema - forma esquemática

A solução encontrada pelo autor foi o desenho de um grande sistema que tivesse um módulo conversacional a nível de usuário, onde ele declarasse a linguagem de 1ª ordem que usaria naquela teoria, entrasse com os axiomas e definições da teoria, selecionasse estratégias, e então este sistema gerasse um "provador de teoremas" para aquela teo-'ria, veja figuras III.1 e 111.2.

Este provador de teoremas gerado produz, 'além dos resultados referentes a demonstração do teorema, um histórico de todos os "caminhos de provas" para a teoria em questão. 'Esquematicamente teria-se :

ESQUEMA DO SISTEMA GERADOR DE PROVADORES DE TEOREMAS - SGPT

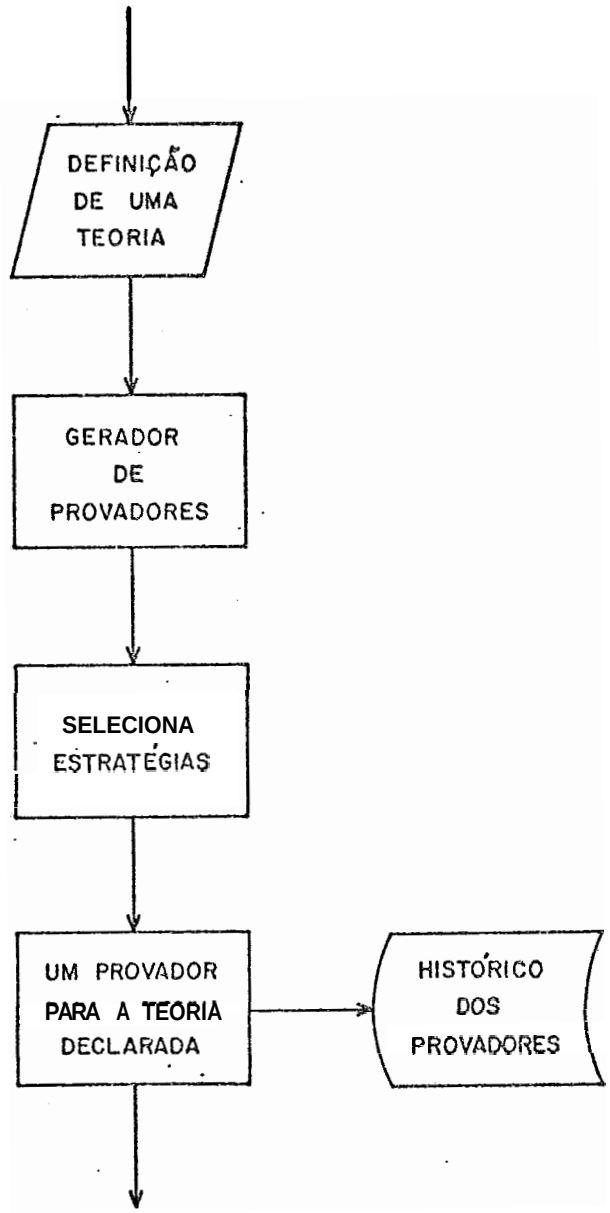


FIGURA III.1

ESQUEMA DO PROVADOR GERADO

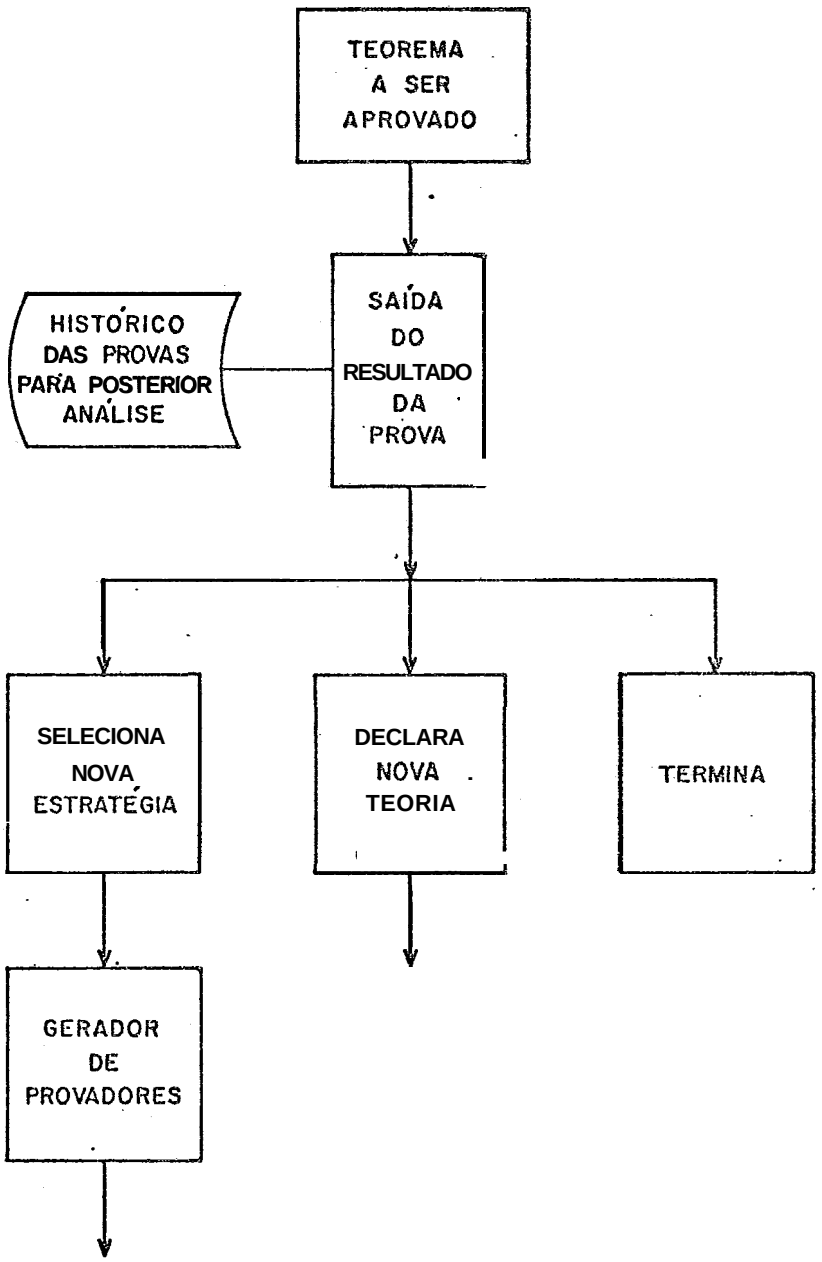


FIGURA III.2

III.2 - Descrição do Sistema (Lógica Geral do Sistema)

III.2.1 - características do Sistema Gerador de Provador de Teoremas - SGPT

O SGPT foi desenvolvido com o objetivo de pesquisar, em prova automática de teoremas, a melhor maneira de construir-se provadores inteligentes nas teorias estudadas.

É composto basicamente de dois programas. O programa que o usuário tem contacto, que é conversacional, onde ele só tem de responder às perguntas para que seja gerado automaticamente e o programa que irá provar teoremas. Estes programas são modulares com partes independentes. As estratégias ficam em locais fixos e pode-se acrescentar novas estratégias nesses locais para uma melhor escolha por parte dos usuários. Elas podem ser construídas por pessoas diferentes.

III.2.2 - Programa gerador

É composto de uma parte estática onde o usuário fornece dados para a geração do "provador" e uma parte dinâmica na qual o usuário interage com o programa para determinar ou modificar as combinações das estratégias e buscar dentre as estratégias existentes, aquelas selecionadas, por intermédio de comandos, pelo usuário.

A linguagem na qual escreve-se este programa, será projetada, completamente, numa próxima pesquisa. Aqui tem-se somente um protótipo da linguagem, com o qual testa-se o sistema que consta de:

DECLARAÇÕES - da linguagem de 1ª ordem a ser usada

AXIOMAS - da teoria dessa linguagem

DEFINIÇÕES - dessa mesma teoria

COMANDOS - para manipular as estratégias e principalmente para editar (gerar) o provador.

Este programa gerador contem, no seu módulo principal, o compilador dessa linguagem descrita acima. A função desse compilador é: (i) montar as tabelas usadas nas estratégias de Resolução, isto é, edita as tabelas que comporão o texto do programa gerado. (ii) definir os "patterns" do programa gerado, ou seja, os padrões dos símbolos funcionais, símbolos predicativos e das constantes para o analisador sintático. (iii) gerar as tabelas de traduções (começa numa folha do grafo que define a estrutura definicional da linguagem) que farão parte do programa gerado e auxiliarão na estratégia de traduzir por níveis.

III.2.2.1 - ESQUEMA DO PROGRAMA GERADOR

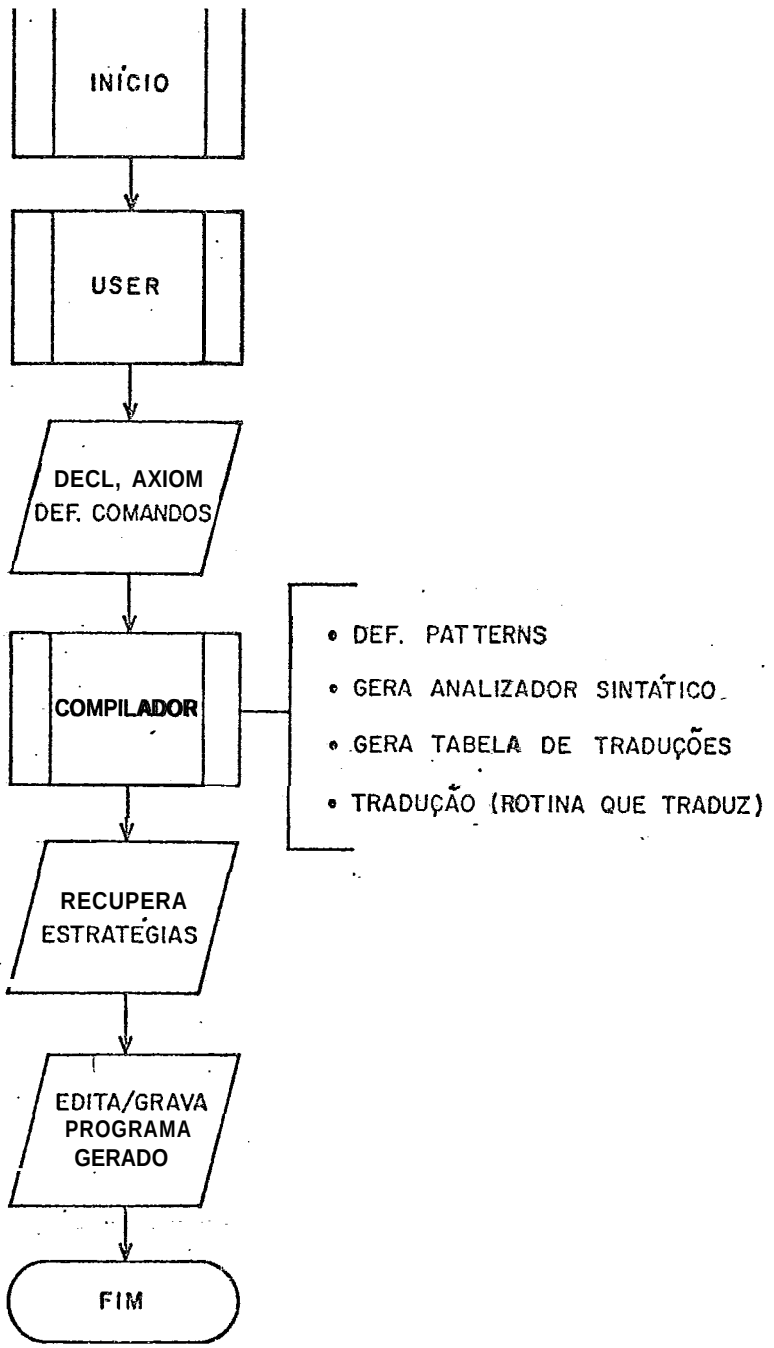


FIGURA III . 3

111.2.3 - Programa gerado

Programa construído pelo programa gerador por intermédio dos dados inseridos pelo usuário. Consta de : os "patterns" em SNOBOL que definem os símbolos predicativos, símbolos funcionais e as constantes; tabelas de traduções ; estratégias que serão usadas nas provas dos teoremas; comandos que permitam ao usuário voltar *ao* programa gerador e redefinir o programa gerado, mudando-se as estratégias ou combinações dessas; comandos que permitam mudar a maneira da busca do espaço solução; módulos de leitura e impressão, para as cláusulas que definem o teorema a ser provado; e finalmente o analisador sintático.

Haverá comandos no programa gerado, que permitirão um tipo de recuperação de memória, ou seja, será possível ao sistema retroreferenciar e guardar somente as cláusulas Úteis, eliminando da memória as que não servirão para gerar cláusulas Úteis. A lógica do programa gerado será dada a seguir.

O programa gerador e o gerado compõe o SGPT que fundamentalmente tem três níveis de ação:

(1) gera um programa

(2) modifica programa

(3) modifica espaço de memória obtido por aquele programa, gerado, isto é, dirige a busca do espaço solução.

III.2.4 - Fluxo do funcionamento da delinição de uma Teoria

A definição da Teoria é feita em duas etapas:

fig. 111.4

Etapa 1 - O usuário define as CONSTANTES, VARIÁVEIS, PREDICADOS e FUNÇÕES.

O SGPT transforma as listas fornecidas pelo usuário em "patterns" e "tables", armazenando-os no arquivo ALFABETO,

. Por exemplo:

a - Entre com as CONSTANTES da Teoria

A,B,C.$

O SGPT monta o "pattern" das constantes

SCONST = SCONT | SCONST NUMNT | NUMNT

onde o "pattern" NUMNT = SPAN(10123456789')

O significado dessa declaração de constantes é que qualquer constante da Teoria é a permitida pelo "pattern" SCONST.

b - Entre com as VARIÁVEIS da Teoria

X, Y, Z $

O SGPT monta o "pattern" das variáveis

SVAR = SVAR | SVAR NUMNT

O significado é que qualquer variável da teoria é a permitida pelo "pattern" SVAR.

c - Entre com os PREDICADOS OU FUNÇÕES

F, 2 ; I,1 $ dois nomes de funções com aridades 2 e 1

P, 2 ; = , 2 $ dois símbolos de predicados com aridades 2 e 2.

O SGPT monta os "patterns" e as "Tables" com esses símbolos funcionais.

RSIMB cria os "patterns" das funções e dos predicados.

CTABELA cria as tabelas das funções e predicados com suas respectivas aridades.

Essas duas funções acima RSIMB e CTABELA junto com os "patterns" das constantes variáveis vão ser usadas, pelo analisador sintático, para analisar as entradas dos axiomas da teoria.

O significado é que essa teoria que estamos declarando só tem F e I como símbolos funcionais e P, = como símbolo predicativos.

Etapa 2 - O usuário define os AXIOMAS, TEOREMAS e ou DEFINIÇÕES*.

Entre com os AXIOMAS, TEOREMAS OU DEFINIÇÕES.

Toda vez que um destes "comandos" é escrito, o SGPT passa a operar de uma maneira análoga ao "input" de "edit" do TSO. A diferença é que a cada linha fornecida pelo usuário, será feita uma "validação" da seguinte forma:

O SGPT invoca a execução do programa. ANALISADOR SINTÁTICO que utiliza o arquivo ALFABETO. Este programa então aceitará ou não a linha fornecida. Cada linha aceita é gravada, então, no arquivo TEORIA (veja esquema na página seguinte).

Para terminar as definições, basta fornecer uma linha com 'FIM';

* Veja Capitulo IV.

ESQUEMA DO FUNCIONAMENTO DAS DUAS PRIMEIRAS ETAPAS

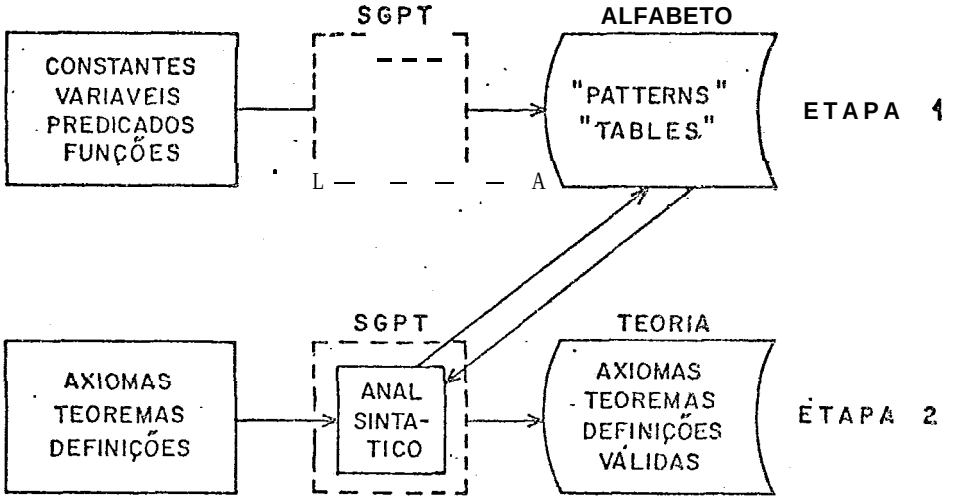


FIGURA III.4

III.2.4.1 - Fluxo do funcionamento do comando AXIOMA

Os comandos AXIOMA, TEOREMA e DEFINIÇAO têm o mesmo funcionamento. veja no Capítulo quatro a explicação teórica.

A ação do SGPT para esses comandos é dividida. em duas partes.:

(i) "Monta", internamente, o "programa" que contem o código-SNOBOL dos arquivos ALFABETO e ANALISADOR, usando a função CODE.

(ii) Coloca as iniciais AX na tela e espera) o usuário escrever uma linha.. Quando isto é feito, o SGPT entrega a linha ao programa descrito em (i). Este programa aceita ou rejeita (envia uma mensagem à tela) essa linha , repetindo (ii) até o usuário escrever uma linha começando com a palavra 'FIM'.

Veja esquema na página seguinte, figura 111.5.

ESQUEMA DO FUNCIONAMENTO DO COMANDO AXIOMA

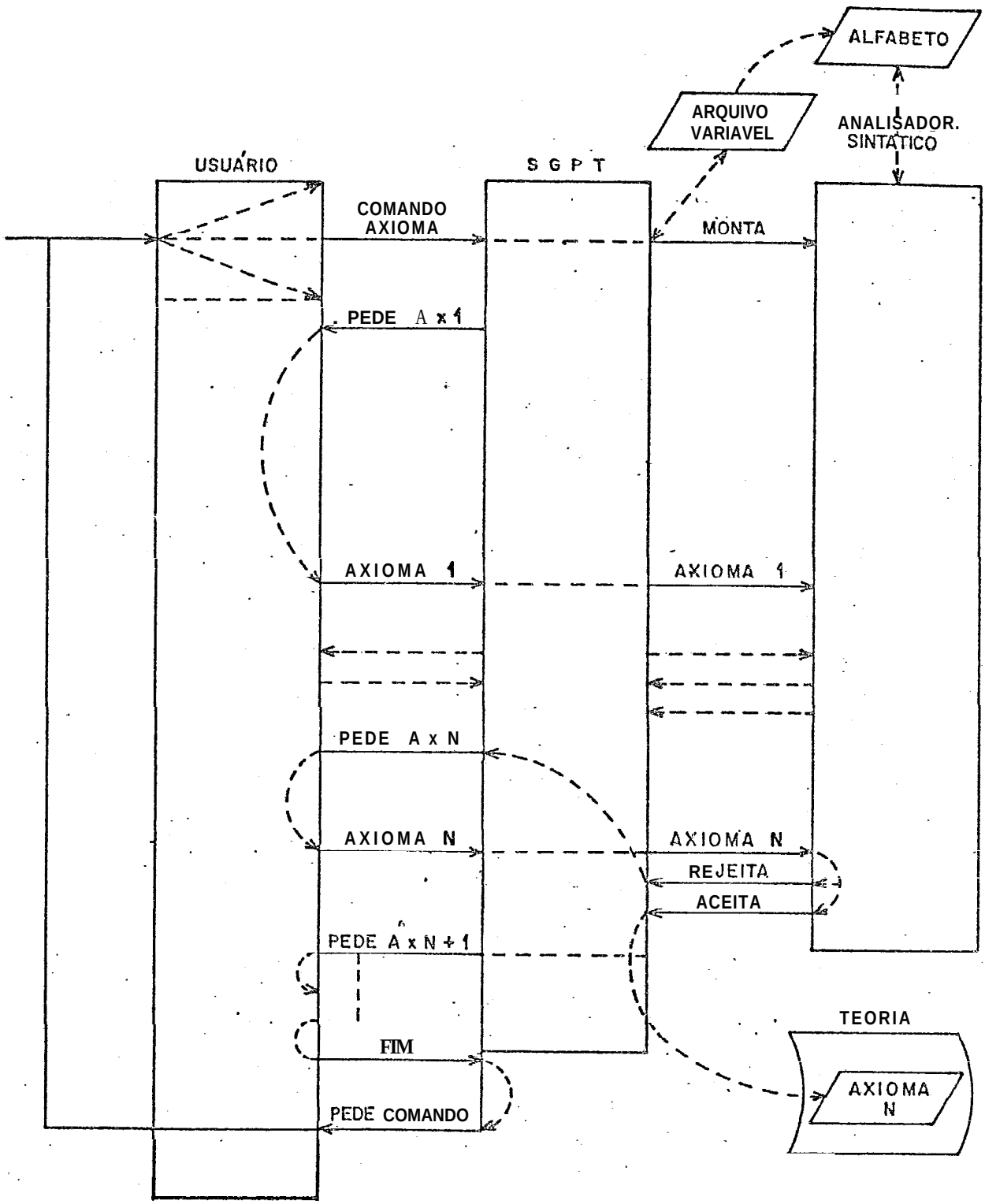


FIGURA III. 5

III.2.4.2 - Analisador Sintático

Este programa auxilia ao usuário analisar se **sua** entrada de FUNÇÕES, PREDICADOS, AXIOMAS (TEOREMAS, DEFINIÇÕES) estão sintaticamente corretas.

Por exemplo:

No nosso caso definimos a função F com aridade 2 e não definimos W como variável.

Se dermos entrada na função.

F(W) o analisador rejeitará pois a aridade da função e F é 2 e aí só temos **1** variável e também W não é uma variável de nossa teoria.

Nesse momento; depois de executadas as etapas **1** e 2 temos: Um arquivo com os "patterns" e "tables" das constantes, variáveis, predicados e funções - (ALFABETO).

Um arquivo com os AXIOMAS, TEOREMAS e DEFINIÇÕES válidas - (TEORIA),

Um arquivo com as ESTRATÉGIAS prédefinidas - (ESTRATÉGIAS).

III.2.5 - Fluxo da montagem das ESTRATÉGIAS

O SGPT conterá uma linguagem de consulta para montagem e recuperação de estratégias que serão compostas nesse primeiro ano, de sete estratégias. A saber:

1) Eliminação (subassunção de cláusulas e tautologias),

2) Preferência mais carta

3) Conjunto Supor-e .

4) .Linear

5) Hiperesolução (caso especial de resolução)

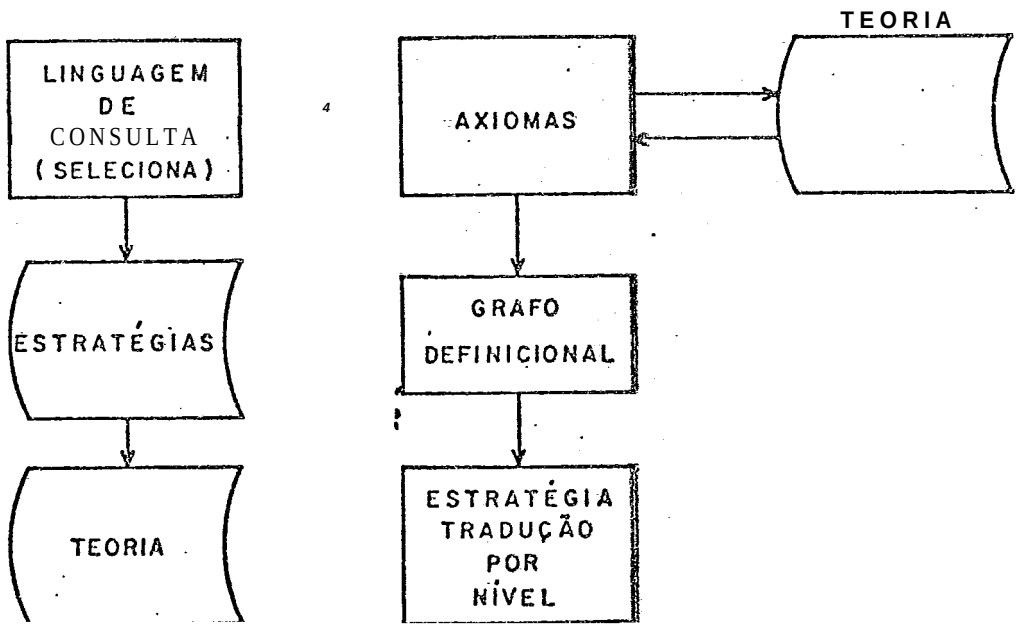
6) Lock Resolution (indexação de predicados)

7) Paramodulation

Por meio da linguagem de consulta.para estratégias existente no SGPT, o usuário monta no arquivo TEORIA as estratégias que o provador de teoremas usará nas provas de teoremas.

III.2.6 - O compilador do SGPT

O SGPT contém, no seu corpo, uma rotina que monta tabelas, "patterns" já descritos anteriormente, bem como traduz os COMANDOS da linguagem de consultas para o sistema operacional específico, (no caso para o TSO da IBM) e assim pode-se manipular os arquivos das estratégias. Finalmente a função Última do compilador é gerar o grafo definicional** da teoria para utilizarmos na estratégia especial de tradução por nível.



** Veja no Capítulo TV.

FIGURA III.6

III.2.7 - Ações do provador gerado. fig. III.7

Uma vez definida a linguagem da teoria. as estratégias que, com RESOLUÇÃO, gerarão e/ou provarão teoremas; o usuário deseja:

- provar teoremas
- gerar novos teorenas.

Essas ações serão possíveis por intermédio do comando EXECUTE.

O SGPT fará, então, a montagem das rotinas do programa gerado, que chamaremos TEORICO. Essas rotinas foram as estratégias selecionadas pela linguagem de consulta, que comporão com resolução, os componentes de TEORICO, e executarã o programa gerado.

III.2.7.1 - Algoritmo de geração e prova de Teosernas

Dentro de TEORICO tem-se ainda uma parte que o usuário pode só gerar cláusulas (teoremas), para tomar um melhor conhecimento da teoria. E uma outra parte que tentará gerar a cláusula vazia (provar teorema).

Nesta parte o usuário pode "dialogar" com o SGPT, inibindo cláusulas para não serem usadas na geração de outras, GERAR LEMAS que facilitem a prova, aumentar o limite das cláusulas que podem ser geradas etc. Isso é feito dentro do teórico por intermédio de comandos.

III.2.8 - Fluxo do funcionamento do comando EXECUTE PROGRAMA

A ação do SGPT, quando executamos o comando EXECUTE, é:

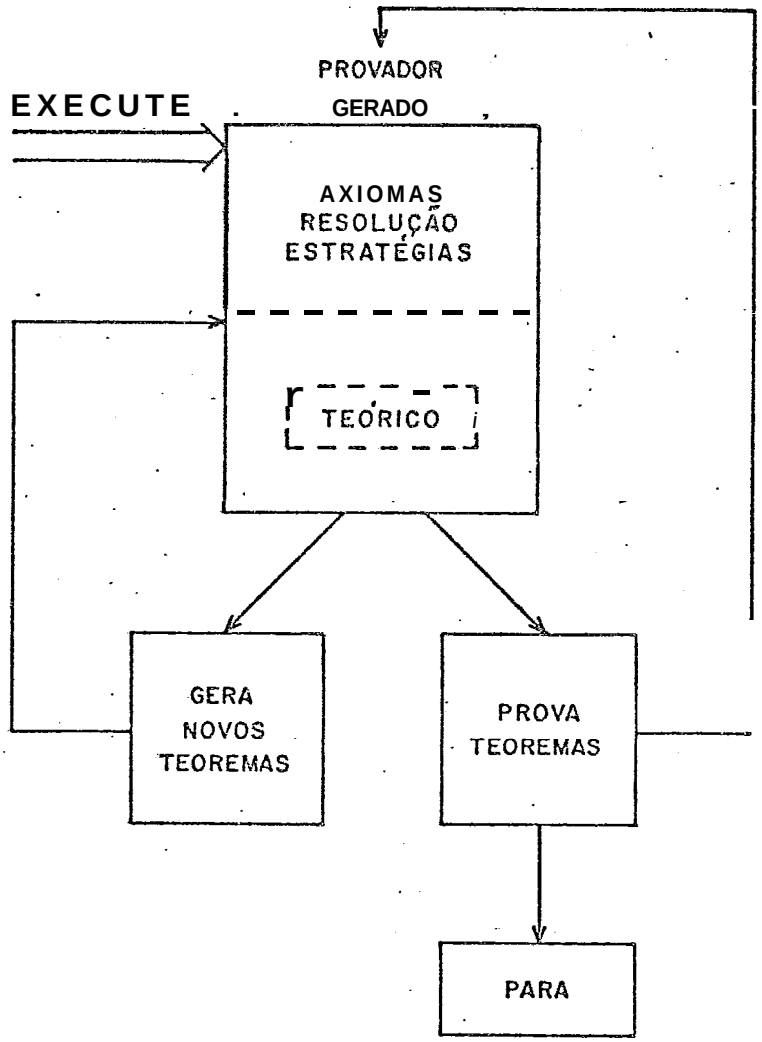

PROVADOR
GERADO
EXECUTE
AXIOMAS
RESOLUÇÃO
ESTRATÉGIAS
TEÓRICO
GERA
NOVOS
TEOREMAS
PROVA
TEOREMAS
PARA

FIGURA III.7

a) "montar" dois códigos-SNOBOL concatenados:

- o programa COMPILADOR que está em um arquivo
- as rotinas selecionadas do arquivo ESTRATÉGIA que estão no arquivo TEORIA.

b) fornecer o arquivo TEORIA como "input" do COMPILADOR.

O compilador, por sua vez, deverá gerar o arquivo do "grafo definicional", e tabelas contendo a hierarquis para a "tradução por nível", veja no Capítulo quatro.

Sintaxe: EXECUTE PROGRAMA [USANDO(N,M,...)] onde N,M,... são as estratégias selecionadas e ou combinadas . Junto com as estratégias são montadas as partes fixas, como RESOLUÇÃO, etc.

A ação deste comando é, em resumo, montar as várias partes espalhadas em arquivos em um arquivo chamado TEORICO, com as estratégias selecionadas, e então executar o provador gerado para provar teoremas.

Compilador é o código-SNOBOL do programa compilador que gera grafos, tabelas a partir de AXIOMAS e teoremas Rotinas i_1, Rotinas i_2, Rotinas são as partes do TEORICO, também em código-SNOBOL, já selecionados pela estratégia N.

III.2.8.1 - Fluxo do funcionamento do comando EXECUTE

fig. III.8

Sintaxe : EXECUTE arquivo

O objetivo deste comando é permitir ao usuário determinar a execução de um programa em código-SNOBOL contido no "arquivo" especificado.

FLUXO DO FUNCIONAMENTO DO COMANDO EXECUTE (INTEGRADO)

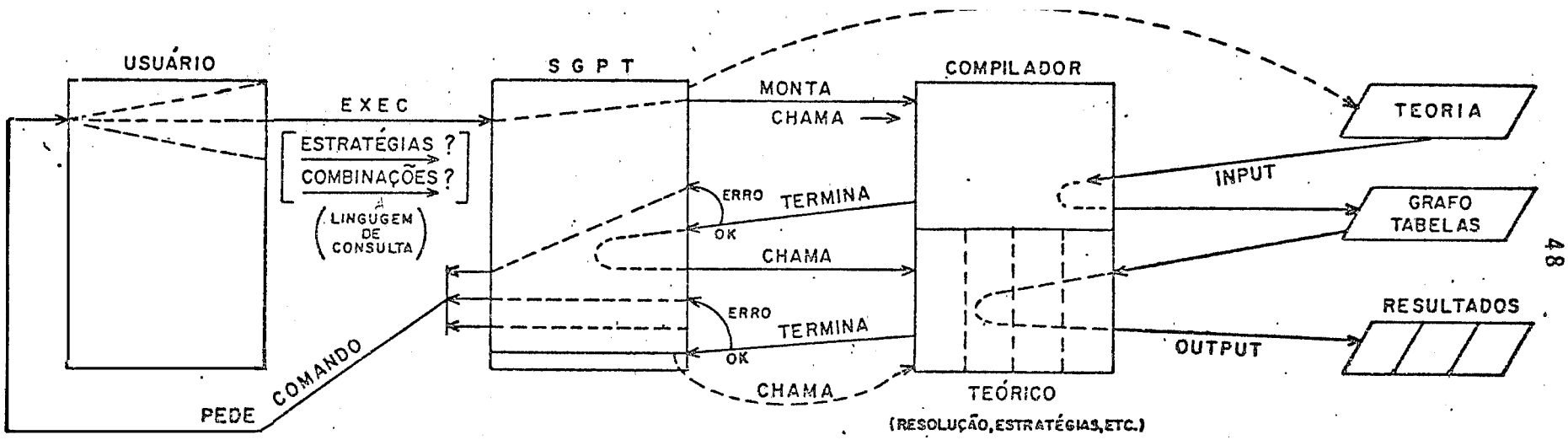


FIGURA III.8

A utilização por ora é:

-EXECUTE PASSADO. ou
EXECUTE HISTORIA .

A ação do comando é recuperar tudo o que foi feito na sessão sem precisar refazer nada, isso se a sessão, eventualmente, tiver sido cancelada.

III.3 - Linguagem que Define a Teoria.

A linguagem completa será objeto de tese de mestrado já em andamento. Descreve-se aqui um subconjunto da linguagem com a qual testa-se o sistema. O módulo USER solicita do usuário a descrição da linguagem de 1? ordem, que irá definir a teoria, na qual o usuário deseja provar teoremas e, consequentemente, o sistema gerará esse provador.

```
CONSTANTES   C1, C2,C3,...,CN;
VARIÁVEIS    V1, V2,V3,...,VN;
FUNC         F1, F2,F3,...,FN;
PRED         P1, P2,...,PN.
```

Com esses dados serão gerados os "patterns" em SNOBOL que comporão o analisador sintático. Isso implicará que. em seguida quando da entrada dos AXIOMAS da teoria, estes já sofrerão uma consistência, isto é, só poderão usar as constantes, variáveis, funções e predicados, conforme definição anterior.

Em seguida tem-se

```
AXIOMAS      A1,A2,A3,...,AN;
ESTRATEGIAS  A,B,C,...,H;
```

e os comandos disponíveis para o usuário.

TRADUZN	(traduz um nível na árvore definicional)
INSERIT	(tenta gerar um teorema a partir dos existentes e inserir como uma cláusula gerada)
SELECIONAE	estratégia A
COMB**INAE**	estratégia **A** com B
DESLIGAE	estratégia **A**
RECUPERAM	(recupera memória) o programa solicita que cláusulas não entrarão na geração dos próximos teoremas) e recupera a memória utilizada
RESTAB	(,atualiza cópia de programa gerado)
LISTA	(lista arquivo)
GERART	(usa resolução e estratégias para gerar teoremas (cláusula vazia))
INIBIR'	inibí as cláusulas que o usuário desejar. Não as usa para geração de novas cláusulas.

III.4 - Descrição das Estratégias Existentes

No ítem III.1.3 deste trabalho classificou-se as estratégias em três grupos distintos: simplificação, refinamento e ordenação. As estratégias de simplificação são aquelas que eliminam as cláusulas geradas, por serem tautologias, pela avaliação dos predicados, ou por já estar incluída (subassumption). Esta última possibilidade é explicada por meio de um algoritmo em ROBINSON[9].

O segundo grupo são os refinamentos. Neste o conjunto suporte é o mais usado. Usa-se também o refinamento P-refutação.

O terceiro e Ultimo grupo, a principal estratégia é a preferência unitária que ordena as cláusulas para

serem resolvidas.

Por fim, tem-se as heurísticas que melhoram a eficiência do Princípio da Resolução. Essas heurísticas retiram a eompleteza da regra Princípio da Resolução mas seu uso prático justifica plenamente a perda de completeza. A estratégia gerador de lemas descrita no ítem B.4 módulo 9 desta tese é uma delas.

A estratégia de Predicados de Comunicação descrita em LINS[25] combinada com P-refutação, é usada neste sistema e explicada no ítem II.3. A utilidade prática é mostrada na tabela resumo, encontrada no ítem I.2.

A análise feita nessa tabela mostra que a eficiência do sistema, que usa Predicado de Comunicação, aumenta sensivelmente.

III.4.1 - Conceito de Nível de Profundidade

Dado um conjunto S de cláusulas, diz-se que cada cláusula C pertencente a S tem nível ZERO. Se C é a cláusula obtida de resolução de C_1 e C_2, dizemos então que o nível de C, é igual ao maior nível, entre C1 e C2, somando com uma unidade. Por exemplo, seja um conjunto S de cláusulas:

1. Not.p(x,y)/p(y,x);
2. Not.p(x,y)/p(x,f(x)),
3. p(a,b).

As três cláusulas têm nível ZERO. Agora sejam as cinco cláusulas geradas:

4. p(b,a),(1,3);
5. p(a,f(a)),(2,3);
6. p(b,f(b)),(4,2);
7. p(f(a),a),(5,1);
8. p(f(a),f(f(a))),(7,2);

temos que o nível da cláusula 4 é UM, o da 5 é UM, o da 6 é DOIS, o da 7 é DOIS e finalmente o nível da cláusula 8 é TRÊS.

.Esse conceito de nível de prioridade é utilizado na geração de novas cláusulas.

O usuário informa o nível de profundidade , por exemplo um número 5 é dado como o número. máximo de cláusulas a ser gerada. O sistema começa por gerar todas as cláusulas de nível 1, todas de nível 2 e assim tem-se a geração de cláusulas "espalhada" e muitas vezes não atinge o nº máximo de cláusulas pedido pelo usuário.

O uso do "espalhamento" numa teoria é importante, pois obriga o sistema a usar vários axiomas ou teoremas de maneiras diferentes. Pode-se definir uma teoria como sendo rasa, se tem muitos teoremas em cada nível. E em caso contrário tal teoria é dita profunda.

III.4.2 - Estratégia Conjunto Suporte

Descrita no ítem II.1, do Capítulo II, e ítem A.3 do Apêndice A,

III.4.3 - Estratégia Preferência Unitária

Explicada no Capítulo II ítem II.1 e Apêndice A ítem A.3.

III.4.4 - Estratégia Tradução por Nível

Explicada no Capítulo IV,

III.4.5 - Outras estratégias

Explicada, também, no Capítulo IV.

111.5 - Descrição dos Comandos da Linguagem

III.5.1 - GERAT

Este comando quando ativado, vai tentar gerar a cláusula vazia por intermédio do Princípio da Resolução e estratégias selecionadas pelos comandos próprios descritos no ítem 111.4.

A regra de resolução (Princípio da Resolução) cujo princípio está explicado no ítem II.1.2, usa um algoritmo de unificação. Este algoritmo encontra-se descrito na bibliografia e portanto não será colocado neste trabalho.

III.5.2 - INIBIR.

O usuário pode inibir cláusulas que não deseja que sejam usadas na geração de outras. Isso é feito no programa por um mecanismo de inibição. Por exemplo se tem-se 5 cláusulas, o usuário coloca INIB=$1$5 então as cláusulas 1 e 5 não serão utilizadas para a geração de novas cláusula's.

III.5.3 - INSERIR

O usuário pode em momento solicitado pelo sistema, inserir axiomas ou teoremas, que ele "acha" Útil para apressar a geração da cláusula vazàa. Esses teoremas podem ser resultados da teoria por ele conhecidas ou lemas gerados por intermédio do mecanismo GEL descrito em B.4.

III.5.4 - Outros Comandos

Explicados no Apêndice C.

CAPÍTULO IV

TEORIA DAS DEFINIÇÕES E ALGORÍTMOS PRINCIPAIS

IV.9 - Introdução

O que nós fizemos até agora foi estudar e construir provadores de teoremas para alguma classe, de teorias axiomáticas, que chamaremos Teorias Definicionais. Essas Teorias são aquelas que contém um grande número de definições. "Point Set Topology" é um exemplo de tal teoria.

Sabemos que um determinado conjunto de sentenças, que constitui uma teoria, admite vários sistemas axiomáticos, isto é, há diferentes partições possíveis dessas sentenças em axiomas e teoremas, e de seus objetos (termos e predicados em objetos primitivps e definidos).

Dada uma teoria T que admite vários sistemas axiomáticos, é muito difícil decidir qual sistema usaremos para construir o provador de teoremas para T. É muito possível que a eficiência desse provador seja uma função dessa escolha, por isso, devemcs testar os vários sistemas axiomáticos para então decidirmos qual é o melhor. Isto o nosso gerador de provador proporciona; Mas fica existindo, ainda, a dificuldade de encontrar na literatura disponível, axiomatizações diferentes, para mesma teoria, para termos uma completa formalização,

Uma vez encontrada uma axiomatização, nós devemos verificar suas conexões com teorias conhecidas, Essas conexões são, em geral, muito difíceis de detetar se não conhecemos bem essa nova teoria. Portanto nos resta tentar conhecê-la bem, por intermédio do conhecimento de suas diferentes axiomatizações.

O sistema por nós proposto e explicado tem como ob-

jetivo, facilitar, interativamente que o usuário aprenda sobre a teoria. mudando os axiomas, as definições e gerando teoremas para se conhecê-la melhor, pela comparação dos resultados. Aí, então, decidir de qual axiomatização deverá ser construído o provador de teoremas final.

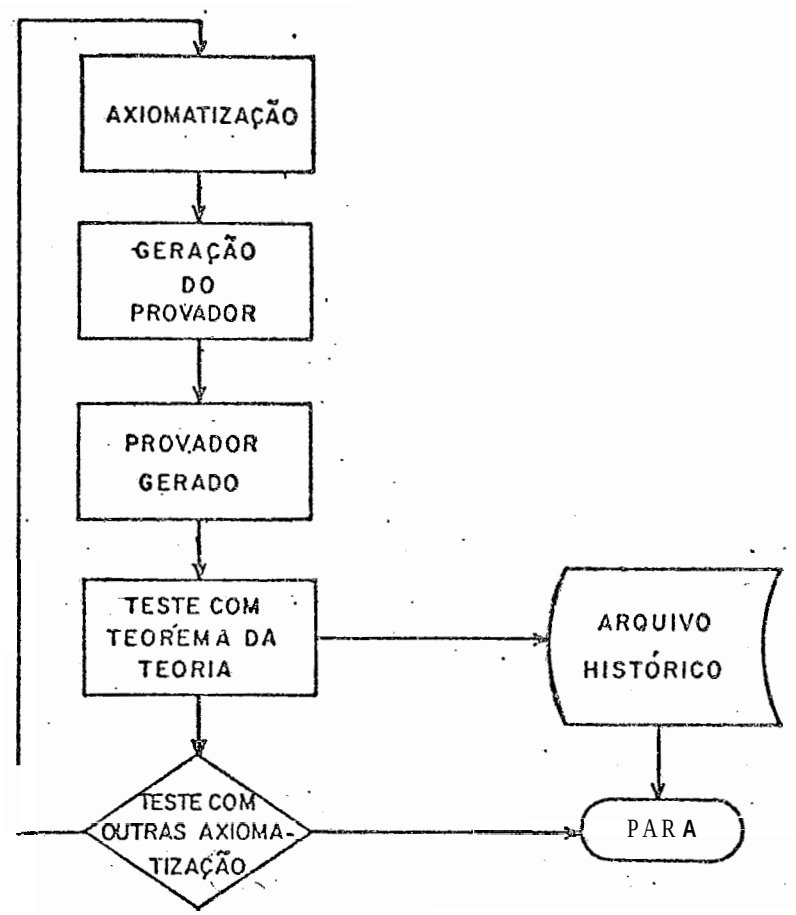


Fig. IV.1

VI.2 - Teoria das Definições

IV.2.1- Definições

Definições, segundo SUPPES[13], são afirmações que estabelecem o significadó de uma expressão. Uma teoria poderia ser composta de seus primitivos, e axiomas sobre os pri

mitivos e portanto seus teoremas seriam, relacionamentos válidos, entre esses símbolos primitivos. Mas, como sabemos, muitas noções intuitivas correspondem, na verdade, a possíveis objetos definíveis da teoria., em termos de seus objetos primitivos. Podemos pensar' numa sequência de objetos definidos, e então falar de objetos previamente definidos em relação a outros objetos definidos. Seria falso pensar nos objetos de uma teoria como constituindo uma simples cadeia de objetos definidos. Uma teoria tem, em geral, uma "Estrutura Definicional" muito complicada. Neste sistema proposto, queremos automaticamente, a partir das definições e axiomas, gerar (construir) esse "Grafo Definicional".

Uma definição é considerada pelo autor, de um ponto de vista da lógica de inferência, como um novo axioma da Teoria, mas a introdução de um novo símbolo serve somente para facilitar as deduções da estrutura da teoria, e não adicioná-lo aquela estrutura.

IV.2.2 - Critérios básicos das definições

Existem dois critérios básicos que devem ser observados pelas definições.

Eliminabilidade - Uma fórmula A que introduz um novo símbolo de uma teoria, satisfaz ao critério de eliminabilidade, se e somente se: quando A_1 é uma fórmula na qual o novo símbolo ocorre, então existe uma fórmula A_2 na qual o novo símbolo não ocorre tal que $A \Rightarrow (A_1 \Leftrightarrow A_2)$ é derivável dos axiomas e definições precedentes da teoria.

Não-criação - Uma fórmula A que introduz um novo símbolo de uma teoria satisfaz o critério de não-criação, se e somente se: não existe fórmula B na qual o novo símbolo não ocorre, tal que $(A \Rightarrow B)$ é derivável dos axiomas e definições precedentes da teoria, mas B não é derivável.

As regras para definições condicionais estão na pá-

gina 165 de SUPPES[13] assim como as regras para símbolos predicados e funcionais,

IV.3 - Estruturas Definicionais - ED's - Exemplos

Nessa seção vamos mostrar o que são estruturas definicionais e como criamos o Grafo Definicional.

IV. 3.1 - Estrutura Definicional para $(BAC)_o$

Considere um conjunto de fórmulas, que definam $(BAC)_o$, e seus respectivos conjuntos de cláusulas.

Veremos que esse conjunto de cláusulas é suficiente, do ponto de vista da eliminabilidade dos símbolos predicados C_i, C e $=$; das ocorrências de $\vee$, $\cap$, $-$, $\complement$ (complemento), associados com os símbolos predicados, e ocorrências destes símbolos no lado direito de ' '.

Observaremos que $x,y,z,\ldots$ são pontos e $X,Y,Z,\ldots\ldots$ são conjuntos.

FÓRMULAS

A1 - $\forall x \neg(x \in \emptyset)$
A2 - $\forall x\ (x \in V)$
A3 - $\forall X \forall Y (X \subseteq Y \Leftrightarrow \forall x\ (x \in X \Rightarrow x \in Y))$
A4 - $\forall X \forall Y (X = Y \Leftrightarrow (X \subseteq Y \wedge Y \subseteq Y))$
A5 - $\forall X \forall Y (XCY\ (X \subseteq Y \wedge \neg Y \subseteq X))$
A6 - $\forall X \forall Y \forall x (x \in (X \cup Y) \Leftrightarrow (x \in X \vee x \in Y))$
A7 - $\forall X \forall Y \forall x (x \in (X \cap Y) \Leftrightarrow (x \in X \wedge x \in Y))$
A8 - $\forall X \forall Y \forall x (x \in (X - Y) \Leftrightarrow (x \in X \wedge \neg x \in Y))$
A9 - $\forall X \forall x (x \in \complement(X) \Leftrightarrow \neg x \in X)$

A representação de A1 à A9 em forma de cláusulas é:

CL(0) $= \neg x \in \emptyset$

CL(1) $= x \in V /$

CL(2) $= \neg X \subseteq Y \ v \ \neg x \in X \ v \ x \in Y$

CL(3) $= X \subseteq Y \ v \ f_1(X,Y) \in X$

CL(4) $= X \subseteq Y \ v \ \neg f_1(X,Y) \in Y$

CL(5) $= \neg X = Y \ v \ X \subseteq Y$

CL(6) $= \neg X = Y \ v \ Y \subseteq X$

CL(7) $= X = Y \ v \ \neg X \subseteq Y \ v \ \neg Y \subseteq X$

CL(8) $= \neg X \subset Y \ v \ X \subseteq Y$

CL(9) $= \neg X \subset Y \ v \ \neg X \subseteq Y$

CL(10) $= X \subset Y \ v \ \neg X \subseteq Y \ v \ Y \subseteq X$

CL(11) $= \neg x \in (X \cup Y) \ v \ x \in X \ v \ x \in Y$

CL(12) $= x \in (X \cup Y) \ v \ \neg x \in X$

CL(13) $= x \in (X \cup Y) \ v \ \neg x \in Y$

CL(14) $= \neg x \in (X \cap Y) \ v \ x \in X$

CL(15) $= \neg x \in (X \cap Y) \ v \ x \in Y$

CL(16) $= x \in (X \cap Y) \ v \ \neg x \in X \ v \ \neg x \in Y$

CL(17) $= \neg x \in (X - Y) \ v \ x \in X$

CL(18) $= \neg x \in (X - Y) \ v \ \neg x \in y$

CL(19) $= x \in (X - Y) \ v \ \neg x \in X \ v \ x \in Y$

CL(20) $= \neg x \in C(x) \ v \ \neg x \in X$

CL(21) $= x \in C(x) \ v \ x \in X$

IV.J.1.2-Grafo Definicional para $(BAC)_o$ - Algorítmo

Parte 1

INICIALIZAÇÃO

Leitura das definições e criação da Tabela DEF e NOME

CL(I) Tabela das Cláusulas

Tabela dos nomes associados aos símbolos		Tabela dos símbolos associados aos nºs das cláusulas que os definem	
NOME (Ø)	ε	DEF(ε)	{0,1}
NOME (1)	⊆	DEF(⊆)	{2,3,,4}
NOME(2)	=	DEF(=)	{5,6,7 }
NOME(3)	⊂	DEF(C)	{8, 9,101
NOME(4)	∪	BEF(U)	{11,12,13}
NOME(5)	n	DEF(∩)	{14,15,16}
NOME(6)	—	DEF(-)	{17,18,19}
NOME(7)	C	DEE(C)	(20,211

Parte 2

criação da Tabela PAI	Para todo nome definido em DEF, selecionar as cláusulas que definem esse nome - dentro de cada definição marcar cada predicado encontrado, como pai desse nome na Tabela PAI

Tabela PAI	
PAI (E)	E
PAI (C)	E
PAI (=)	C
PAI (⊂)	⊂

PAI (∪)	ε
PAI (∩)	ε
PAI (-)	E
PAI (C)	E

Parte 3

Criação da Tabelo NÍVEL	Para cada nome de PAI zerar nível desse nome se nome não tem pai, - se tem, pegar cada pai e .fazer NÍVEL (nome) igual ao máximo entre o valor já obtido *e* o nível de cada pai mais um.

TABELA NÍVEL	
NÍVEL(ε)	0
NÍVEL(⊂)	1
NÍVEL(=)	2
NÍVEL(⊂)	2

NÍVEL(∪)	1
NÍVEL(n)	1
NÍVEL(-)	1
NÍVEL(C)	1

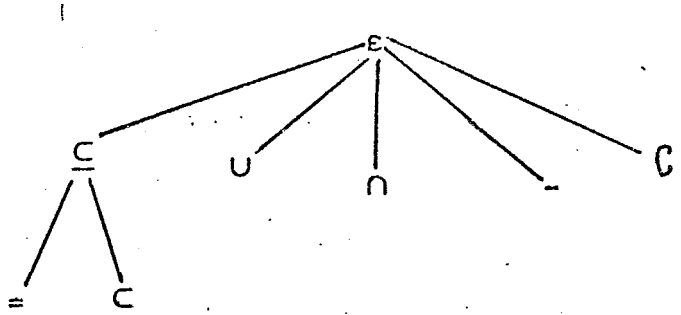
Parte 4

Criação do VETOR LEVEL	para cada nome definido pegar o NÍVEL desse nome colocar em LEVEL desse nível o nome (inverter NÍVEL)

LEVEL (0)	'ε'
LEVEL (1)	'⊆ : ∪ : ∩ : - : ∁ :'
LEVEL (2)	'= . ⊂ .'

Com o VETOR LEVEL podemos desenhar o grafo definicional de $(BAC)_o$

Parte 5 - opcional



IV.3.2 - Estrutura Definicional para Point Set Topology - PST

Considere um conjunto de cláusulas que definam PST,

Use o mesmo algorítmo para gerar o grafo Definicional,

IV.3.2.1 - Cláusulas que a definem

Sejam as definições do ítem A.2 do Apêndice A em forma de cláusulas na Tabela CL(I), com (BAC) feita analogamente ao exemplo anterior,

IV.S.2.2 - Grafos Definicional para PST

Parte 1

INICIALIZAÇÃO ---------- CL(I) - Tabela das cláusulas de PST

Tabela dos nomes associados aos símbolos	
NOME(0) = (BAC)	(Algebra Booleana de classes extcndida)
NOME(1)	NBH
NOME(2)	CLOSED
NOME(3)	LIM
NOME(4)	INT
NOME(5)	DER
NOME(6)	BD
NOME(7)	CLOS
NOME(8)	DENSE
NOME(9)	PERF
NOME(10)	SEP
NOME(11)	CONCT

Tabela de símbolos associados nºs das cláusulas que os definem	
⋮	
DEF('NBH')	{12,13,14}
DEF('LIM')	{15,16,17}
⋮	

Parte 2

criação da Tabela PAI

Tabela PAI	
PAI(BRC)	grafo já criado
PAI(NBH)	BAC
PAI(CLOSED	BAC
PAI(LIM)	NBH
PAI(INT)	NBH
PAI(DER)	LIM
PAI(CLOS)	LIM
PAI(BD)	LIM
PAI(DENSE)	DER
PAI(PERF)	DER
PAI(SEP)	CLOS
PAI(CONCT)	SEP

Parte 3

criação da Tabela NÍVEL

Tabela NÍVEL	
NÍVEL(BAC)	0
NÍVEL(NBH)	1
NÍVEL(CLOSED)	1
NÍVEL(LIM)	2
NÍVEL(INT)	2
NÍVEL(DER)	3
NÍVEL(CLOS)	3
NÍVEL(BD)	3
NÍVEL(DENSE)	4
NÍVEL(PERF)	4
NÍVEL(SEP)	4
NÍVEL(CONCT)	5

Parte 4

Criação do VETOR LEVEL

LEVEL(0)	BAC
LEVEL(1)	NBH: CLOSED
LEVEL(2)	LIM: INT:
LEVEL(3)	DER: CLOS: BD:
LEVEL(4)	DENSE: PERF: SEP:
LEVEL(5)	CONCT

Parte 5 - opcional

grafo - ítem B4 - Apêndice B

IV.4 - ALGORÍTMO GERAL PARA GERAR GRAFOS DEFINICIONAIS

Daremos agora um algorítmo geral para criação automática de grafos definicionais. Esse algorítmo que provaremos funcionar, detetará também, se as definições são mal formuladas (símbolos definidos a partir de símbolos não definidos), aceitará definições não ordenadas (símbolos definidos fora de ordem).

Esse "grafo" será usado pela estratégia de TRADUÇÃO POR NÍVEL.

IV.4.1 - Notação

Sejam: (a) $C_{min(i)}$, ... , $C_{max(i)}$ as cláusulas que definem os símbolos t_i $1 \leq i \leq m$; $C_j = L_1^j \vee L_2^j \vee ..$

(b) A_j o conjunto de símbolos que ocorrem nos literais L_2^j , ... , L_m^j de C_j

(c) $A^i = \bigcup_{j=min(i)}^{max(i)} A_j$

(d) s_1 , s_2 , ... , s_n símbolos primitivos

(e) t_1 , t_2 , ... , t_m símbolos definidos

(f) R a relação a ser construída (GRAFO)

(g) $x = \{x_1 , x_2 , ... , x_q\}$ conjunto

$y = \{y_1 , y_2 , ... , y_q\}$ lista

$z = \{z_1 , z_2 , ... , z_q\}$ conjutno

tal que x ← x.a represente $\{x_1 , x_2 , ... , x_k , a\}$ - a colocado ao fim do conjunto.

(h) Inicialmente comprimento de y é m.

Embaixo, y_k representa o K-ésimo elemento da lista, A^k representa inicialmente os números **primitivos** e depois os "já" definidos,

$\mathcal{R} = \emptyset$, $x = \{s_1 , s_2 , \ldots , s_n\}$, inicialmente

$y = \langle t_1 , t_2 , \ldots , t_m \rangle$ e $z = \{t_1, t_2, \ldots t_m\}$, também.

IV.4.2 - Algorítmo

O algorítmo que produz o GRAFO DEFINICIONAL da TEORIA dada pela axiomatização p.

1 - $K \leftarrow 0$

2 - $K \leftarrow K+1$, LIM $\leftarrow$ nº e$\ell(y)$-K , $CT \leftarrow 0$

3 - Se $A^K \subseteq X$ então faça

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u, y_k) \mid u \in A^k\}$

$x \leftarrow x \cup \{y_k\}$

$z \leftarrow z - \{y_k\}$

se $z = \emptyset$ termine (correto)

$z \neq \emptyset$, se $K = m$, termine, **ERRO** (definição mal formulada) .

se $K \neq m$ vá para (2)

Se $A^k \not\subseteq X$ então faça

'(Coloque y_k na última posição deslocando as demais).

$$(y_1 , \ldots , y_k , y_{k+1} , \ldots , y_m)$$
$$\downarrow$$
$$(y_1 , \ldots , y_{k+1} , \ldots , y_m , y_k)$$

$CT \leftarrow CT + 1$

Se CT>LIM então termine ERRO (def. mal formulada) caso contrário vá para (3).

IV.4.2.1 - Prova do Funcionamentp

Demonstração:

Por indução finita, sobre o número de elementos definidos - para = 1

- A Condição $A^1 \subseteq X$ é satisfeita, pois os primitivos estão inicialmente em x e como as definiçóes estão ordenadas, só ocorrem elementos primitivos para t_1. Então $R = \{(u,y_1) \mid u \in A^1\}$ e $z = \emptyset$, para
  $z = \{t_1\}$ $z = z - \{t_1\} = \emptyset$
  o algorítmo termina (correto) em (3).

- Suponha que o algorítmo está funcionando para k = **i** (ou seja **i** elementos definidos para k = **i**)

  Como as definições estão ordenadas, pois o algorítmo as ordena, $x = \{s_1,...s_n,t_1,...t_i\}$ (estão no grafo ou são primitivos]

  Logo $A^{i+1} \subseteq X$

  $R = R \cup \{<u, y_{i+1}> \mid u \in A^{i+1}\}$

  $Z \leftarrow Z - \{y_{i+1}\}$

  portanto vale para K = **i** + 1, isto é, pára. Pois
  Z = ∅ Termina correto em (3),
  se Z ≠ ∅ Termina ERRO (definição mal formulada em(3).

IV.5 - ESTRATÉGIA DE TRADUÇÃO POR NÍVEL

IV.5.1 - Idéia da Estratégia

Nos trabalhos oriundos de LINS, as cláusulas eram traduzidas até ε(último nível), dentro do programa, e só

então eram "resolvidas". Esse'programa, apesar de complexo e grande, só servia para provar teoremas daquela teoria , cuja tradução estivesse embutida no texto. Com a experiência feita no provador do Apêndice B, verificamos que nem sempre era necessário traduzir até o Último nível. Pois isso fazia com que, 'ao começarmos a prova, a memória já contivesse uma grande quantidade de cláusulas. Essas cláusulas tinh'am outro incoveniente, eram muito compridas, pois a tradução para o nível máximo (E) acarretava um aumento considerável da cláusula. Veja ítem B.4-do Apêndice B.

IV.5.2 -Exemplo de Aplicação

Provar o teorema de PST

$$\forall X \forall Y \forall Z \forall W((T(x,y) \wedge Z \subseteq W) \rightarrow DER(X,Y,Z) \subseteq DER(X,Y,W)$$

Prova usando a estratégia de tradução por nível.

1) $T(X,Y)$

2) $Z \subseteq W$

3) $\neg DER(X,Y,Z) \subseteq DER(X,Y,W)$

4) $\neg x \varepsilon Z \vee x \varepsilon W$ de (2)

Foi colocado no provador e com um tempo dado e um limite para o número de cláusulas geradas, não conseguimos gerar a $\square$.

Foi pedida, então, pelo usuário a tradução para o nível seguinte da Cláusula **3.**

5) $LIM(X,Y,Z,x)$ (1,3) Tradução de DER

Novamente não se conseguiria nada, portanto foi solicitado's tradução para o nível seguinte.

6) $\neg NBH(X,Y,V,x) \vee f_1(g(V),\emptyset) \varepsilon V$

7) $\neg NBH(X,Y,V,x) \subset \neg f_1(g(V),\emptyset) = x$ — Definição do DER

8) $\neg NBH(X,Y,V,x)$ v ((V ϵ Z

9) $\neg$ x' $\epsilon h_4(W,x),x)$ v x' = x v x' ϵW

10) $NBH(h_4,(x,x),x)$

onde $x = f_1(DER(X,Y,Z), DER(X,Y,W)$ e $g(V) = (V-\{x\}\cap Z)$

11) $\neg NBH(h_4(W,x),x)$ por (6,7,8,9)

12) □ (10,11)

IV.5.3 - Algorítmo de TRADUÇÃO POR NÍVEL (fig. IV.2)

Neste momento, temos as tabelas NÍVEL e o vetor LEVEL que são as facilidades requeridas pelo algorítmo do TRADUÇÃO POR NÍVEL:

a) O sistema (PROVADOR) interage com o usuário solicitando o número da.~cláusulas, CL(I), que deseja traduzir.

b) Das cláusulas selecionadas, o TRAD,NIV procura (considerando o grafo deficional da teoria já criado) o maior nível.

c) Traduz todas as cláusulas com este maior nível para o nível seguinte menor,

IV.5.3.1 - Interação com o Provador

a) desenvolve o controle para o provador. Este vai gerar cláusulas até um,LIM,limite dado pelo usuário; se não gerar a cláusula vazia o usuário seleciona novamente um conjunto de cláusulas e solicita a tradução para o nível seguinte.

b) essa geração de cláusulas é feita por RESOLVE utilizando as estratégias selecionadas no arquivo das estratégias, gerando mais teoremas ou a cláusula □.

FLUXO DENTRO DO PROVADOR UTILIZANDO ESTRATÉGIA DE TRADUÇÃO POR NÍVEL

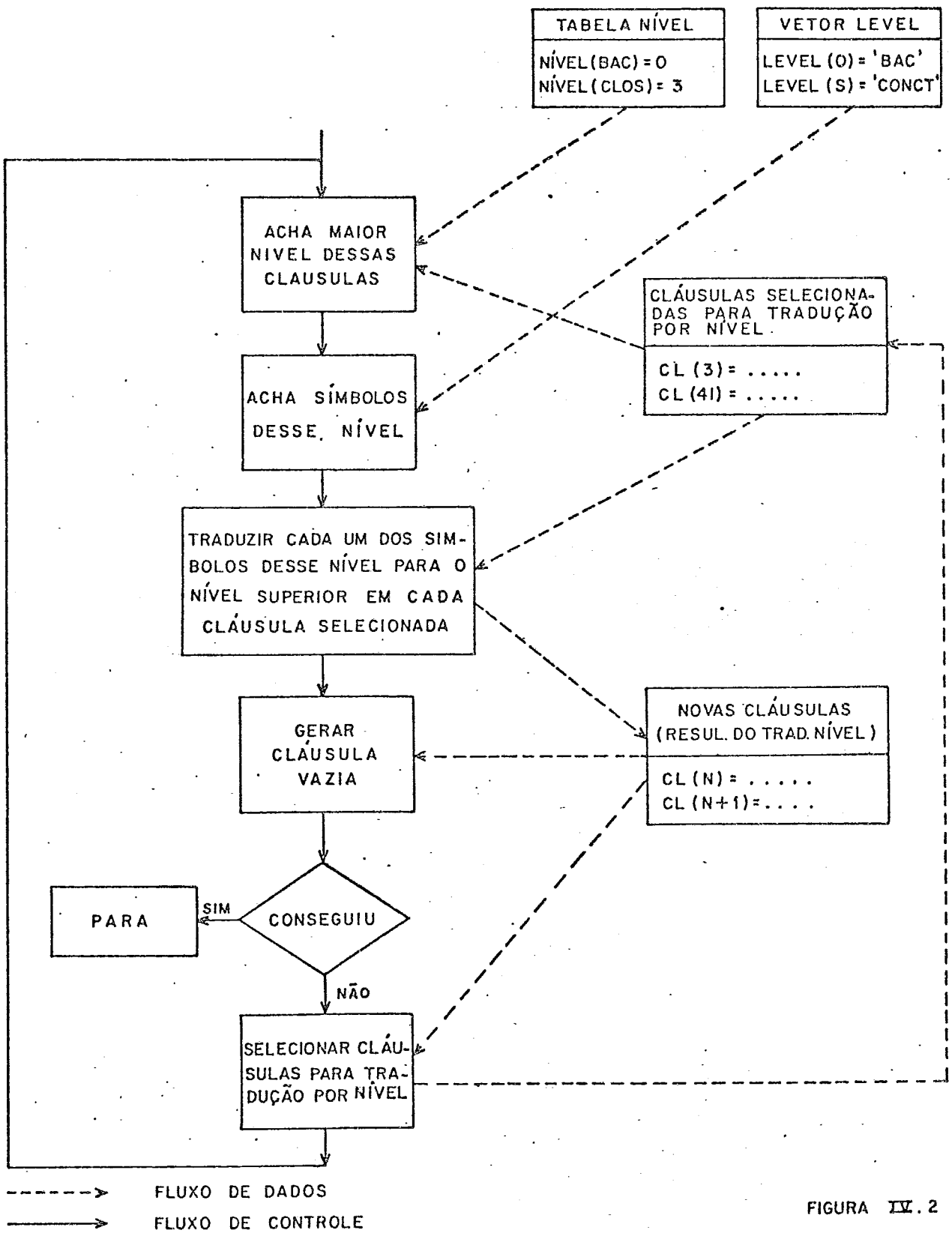


FIGURA IV.2

c) RESOLVE é um algorítmo de RESOLUÇÃO que usa um unificador que foi provado funcionar em CHANG[11].

IV.5.4 - Completeza da Estratégia de TRADUÇÃO POR NÍVEL

IV.5.4.1 - Observação Inicial sobre a Completeza da Estratégia

Observe que uma prova por níveis (usando tradução por nível) pode ser considerada como uma sucessão de "locking resolution". Portanto, a estratégia de Tradução por nível mantêm a completeza pelo Teorema de Boyer.

IV,5.4.2 - Definição de "Lock Resolution"

É um refinamento de Resolução, introduzido por Boyer, que usa um conceito similar ao de cláusulas ordenadas.

Dado um conjunto S de cláusulas, indexamos, arbitrariamente números inteiros para cada ocorrência de literais em S. Diferentes ocorrências do mesmo literal em S pode ser indexada diferentemente, e dois literais diferentes podem receber o mesmo índice.

A Resolução nesta estratégia é usada somente nos literais de menor índice em cada cláusula.

Por exemplo:

Considere as cláusulas:

$C_1 = P_1(x) \vee Q_2(x) \vee R_3(x)$ e

$C_2 = \neg P_4(a) \vee Q_5(a)$. Como $P_1(x)$ e $\neg P_4(a)$ tem o menor índice respectivamente nas cláusulas C_1 e C_2, escolhe-se $L_1 = P(x)$ e $L_2 = \neg P(a)$ para unificar e resolver.

IV.5.4.3 - Aplicação de "Lock Resolution"

Considere o seguinte conjunto de cláusulas já indexados:

$CL(0) = P_1 \vee Q_2 \quad \rightarrow P_1$

$CL(1) = P_3 \vee \neg Q_4 \quad \rightarrow P_3$

$CL(2) = \neg P_6 \vee Q_5 \quad \rightarrow Q_5$

$CL(3) = \neg P_8 \vee \neg Q_7 \quad \rightarrow \neg Q_7$

Das cláusulas de (0) a (3) existe somente um "lock" resolvente:

$CL(4) = \neg P_6$ de (2) e (3) $\rightarrow \neg P_6$

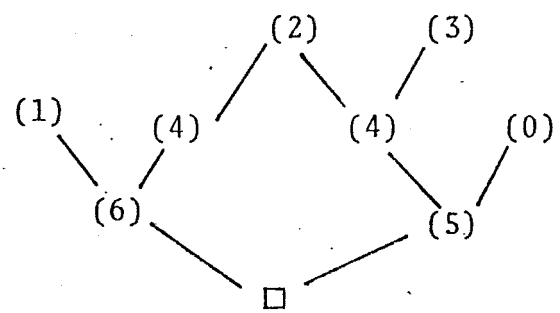
Das cláusulas de (O) a (4) existem dois "lock" resolventes:

$CL(5) = Q_2$ de (0) e (4) $\rightarrow Q_2$

$CL(6) = \neg Q_4$ de (1) e (4) $\rightarrow \neg Q_4$

Finalmente, resolvendo cláusulas (5) e *(6)* obtemos:

$CL(7) = \square$



Portanto obtemos uma "lock deduction" de □ (cláusula vazia), figura acima. Se não usassemos "Lock deduction" geraríamos a cláusula vazia depois de gerarmos 37 resolventes.

IV.5.4.4-Notação Usada na Prova de Completeza

$\mathcal{R}_o$ é resolução com Locking que atribuímos a todos literais os mesmos índices, portanto é equiva-

lente a $\mathcal{R}$ (sesolução sem"locking")

.S^A conjunto de axiomas da teoria

.S^T cláusulas obtidas da negação do teorema

.$S_o^j = \mathcal{R}_o^j (S^A \cup S^T)$ para $j>0$, $j\varepsilon$

.$S_o^o = S_A \cup S = \mathcal{R}_o^o(S^A \cup S^T)$ para $j = 0$.

.S^D conjunto de definições

.$\mathcal{R}_S^D$ estrutura definicional da Teoria com S^D definições.

. ℓ nível máximo de tentativas

.N nível máximo de $\mathcal{R}_S^D$ com respeito a S

.$S_i^j = \mathcal{R}^j(S_i^A \cup S_i^T \cup S_i^D)$ e $\ell>1$ resolução com "LOCK"

onde i significa que as cláusulas de $S(S^D, S^T, \ldots)$ foram indexadas de acordo com o esquema de indexação resultante de se considerar o nível N-i de $\mathcal{R}_S^D$.

IV.5 4 5 - Algoritmo

Abaixo descreveremos o algoritmo de refutação com resolução com tradução por nível, baseado na estrutura definicional da teoria.

As entradas para o algoritmo são:

S^A, S^T, S^D, ℓ e N. Representaremos o conjunto de cláusulas geradas pelo algoritmo por $\mathcal{R}_n(S^A, S^D, S^T, \ell)$.

		Observações
1.	$j \leftarrow 0$; $K \leftarrow 0$; $S \leftarrow S^A \cup S^T$; $j_{máx} \leftarrow \ell$; N	(*1)
2.	Se $\square \in \mathcal{R}^j_K(S)$ então termine."T é um Teorema"	(*2)
	Se $\square \notin \mathcal{R}^j_K(S)$ e $\mathcal{R}^j_K(S) = \mathcal{R}^{j-1}_K(S)$ e $j > 1$	
	então se K=N+1, Termine "T não é Teorema".	(*3)
	de outra forma vá para (3)	(*4)
	Se $\square \in \mathcal{R}^j_K(S)$ e $\mathcal{R}^j_K(S) \supset \mathcal{R}^j_K(S)$ se $j < j_{máx}$	(*5)
	senão faça $j \leftarrow j+1$ e vá para (2)	(*6)
	senão	
	se K = N+1 Termine "T não é um Teorema"	(*7)
3.	Faça $K \leftarrow K+1$ $S = S' \cup S^D_K$, $j \leftarrow 1$ e vá para (2)	(*8)

IV.5.4.6.-Observações sobre os passos do algorítmo $\mathcal{R}_n(S^A, S^D, S^T, \ell, N)$.

(*1) Tendo em vista que mesmo numa teoria definicional alguns teoremas (sentenças a serem testadas) conterão apenas símbolos primitivos, inicialmente o conjunto S ao qual vai se aplicar resolução, não conterá nenhuma definição*

Definições só serão introduzidas no passo, vide observação, *8 . K = 0 significa não usar definições. K cresce a medida que o nível relativo a estrutura definicional decresce.

("2) Muito embora o algorítmo comece com j = 0, o que significa que testaremos se $\square \varepsilon$ S, que parece um contrasenso, em algumas teorias se acrescenta como estratégia a geração de inconsistência, logo após o processo de definições. Por exemplo: se estamos na "Teoria dos conjuntos" e temos uma literal $(A \cup B) = \emptyset$ e uma cláusula é $x \in A \cup B$, durante o processo de tradução (introdução de definições) poderíamos introduzir a cláusula $\square$ em substituição a $x \varepsilon (A^+ B)$. Isso significa gerar a cláusula vazia antes de traduzir.

(*3) Como K já é igual a N+1, nesta altura não existirá no sistema, para $j \geq 1$, nenhum símbolo definido. E o fato da resolução que a esta altura será irrestrita (equivalente a não usar "Locking" pois todos os índices serão iguais a 2, vide "8) e portanto o não crescimento de $\mathcal{R}_k^{j-1}(S)$ por resolução significa que o sistema é satisfatível. Isto acontecerá sempre que a teoria for decidível e, eventualmente no caso que a teoria seja indecidível. Nos dois casos para um j suficientemente grande.

(*4) Aqui muito embora resolução aplicada ao conjunto de cláusulas existente não seja efetiva (nenhuma cláusula nova foi gerada) concluímos que existem no sistema símbolos definidos, ou o sistema é satisfatível. Mas a essa altu

ra não poderemos. saber, a não ser que se modifique o algoritmo para verificar se ainda existem símbolos definidos em $\mathcal{R}_k^j(s)$. Pensemos que de um ponto de vista de especificação tal ítem adicional é desnecessário (Teoria decidível é rara),

(*5) Nesse caso o nº de tentativas de resolução é menor do que o permitido pelo usuário, portanto devemos reaplicar resolução sem introduzir novas definições.

Note que se usarmos este algoritmo interativamente, o usuário, pela análise das cláusulas poderá decidir modificar $j_{máx}$, proibindo resolução, indo para **(3)** onde serão introduzidas novas definições.

(*6) um sistema interativo o usuário pode modificar ℓ, atribuindo-lhe um valor maior,

(*7) Para $j_{máx} = \ell$ dado, não foi possível verificar, se T era um teorema. O usuário poderá mudar ℓ.

(*8) A introdução de novas definições é feita através de uma atribuição de índices aos literais $\mathcal{R}_k^j(S)$ obtendo-se o conjunto S'. E S_k^D por atribuição de índices às cláusulas de S^D segundo o seguinte esquema de atribuição de índice.

- S' é obtido de $\mathcal{R}_k^j(S)$ atribuindo-se índice 2 a todos os seus literais que não contém símbolos do nível N-(k-1) e índice 1 aos demais literais (que contém símbolos de nível N-(k-1)),

- S_k^D é obtido de S^D atribuindo-se índice 1 ao primeiro literal das cláusulas que definem símbolos de nível, exatamente, N-(k-1) e índice 2 aos demais literais dessas cláusulas e índice 3 aos literais das cláusulas de S^D que não definem símbolos de nível N-(k-1).

IV.5.4.7 - Observação sobre o funcionamento do Algoritmo

Observe que o algoritmo sempre Termina, ou pela saída de (*3) ou pela safda (*2) que é o ponto de término que nos interessa, pois devemos mostrar que sempre saímos por (*2). E como vimos na observação (*6), qualquer outra saída é fictícia, pois do ponto de vista do teorema, equivale a hipótese "para algum $\ell>0$".

Observe também que o Valor mfnimo de R para que, realmente, seja possível se obter $\square$, é pelo menos igual ao número mínimo de cláusulas usadas para a definição dos símbolos em S^D.

Na prova do Teorema IV.5.4.8, nos limitamos a demonstrar que o algoritmo sempre termina pela saída (2).

IV. 5.4.8 - Teorema da Completeza da Regra

TEOREMA

.Se $S^A \cup S^D \models T$ então $\square \in \mathcal{R}_n(S^A, S^D, S^T, \ell, N)$ para algum ℓ, $\ell \in N$.

Demonstração

Se $S^A \cup S^D \models T$ então
(Como S^T são as cláusulas obtidas por negação de T) $S^A \cup S^D \cup S^T$ é inconsistente.

Se em T não ocorrem símbolos (N=0) definidos(S^D) então $S^A \models T$ e portanto $S^A \cup S^T$ é inconsistente, por conseguinte $\square \in \mathcal{R}^\infty(S^A \cup S^T)$ e assim para algum j, $\square \in \mathcal{R}^j(S^A \cup S^T)$. Como $\mathcal{R}_o = \mathcal{R}$ (vista anteriormente) o algoritmo pára com K = 0, no passo 2.

Como "LOCK RESOLUTION" é refutacionaimente completa (Teorema Boyer) em CHANG[11], após K=N+1 (Todas as definições foram usadas e portanto os símbolos definidos eliminados). Portanto pela Teoria da definição se $S^A \cup S^D \models T$, $S^A \models T'$ e pelo algoritmo onde T' é a fórmula obtida-de T pela eliminação dos símbolos definidos .através das definições em S^D. $S^A \cup S^{T'} \subseteq \mathcal{R}^j_{N+1}(S)$. Para j suficientemente grande, $S^A \cup S^{T'}$ é inconsistente. E assim para algum $j' \geq j$, $\square \in \mathcal{R}^{j'}_{N+1}(S)$ c.q.d.

REFERÊNCIAS BIBLIOGRÁFICAS

1 . PASSOS, E.P.L - Introduction to Mechanical Theorem Proving --Monogrnphs in Computer Science and Computer Applications 2, 1971, INF-PUC/RJ.

2 . SHAW, NEWELL and SIMON - Empirical Explorations with the Logic Theory Machine: A case study in Heuristics - Computer and Thought, 1958.

3 . TARSKI, A. - Decision Method for Elementary Algebra and Geometry - Berkeley, 1951.

4 . SZMIELEW, W. - Elementary properties of Abelian groups.

5 . MARGARIS, A. - First Order Mathematical Logic - Waltham, Mass, 1967.

6 . GALDA, K. and PASSOS, E.P.L. - Application of Quantifier Elimination to Mechanical Theorem Proving - Monographs in Computer Science and Computes Applications 6, 1972, INF-PUC/RJ.

7 . PASSOS, E.P.L. and SILVEIRA, G.G. - Quantifier Elimination to Mechanical Theorem Proving - Implementation-works from 3th Intern. Cong. of Cybernetics and System August 1975, Bucarest - Romênia, Publicado pela NORTH HOLLAND em 1976.

8 . ROBINSON, J.A. - THEOREM PROVING ON THE COMPUTER - J. ACM 10, 1963.

9 . ROBINSON, J.A. - A machine-oriented logic based on the resolution principle - J.ACM 12, 1965.

10 . ROBINSON, J.A. - 'A review of automatic theorem proving - Proceedings of Symposia in applied mathematics volume 19, 196.7.

11 . CHANG, C.L. and LEE, R.C.T - Symbolic Logic and Mechanical Theorem Proving - New Yark, Academic Press,1973.

12 . KREISEL, G. and KRIVINE, J.L. - Elemcnts of Mathématical Logic: Modcl Theory - Amsterdam, 1967.

13 . SUPPES, P. - Introduction to Logic - Prentice-Hall , N.Y., 1957.

14 . WANG, H. - Logic: of Many Sorted Theories - JSL 17, 1952.

15 . GILMORE, P.C. - A proof method for quantification theory: its justification and realization, IBM, J. Rcs. Develop.

16 . HENSCHEN, L.J. - Tutoria1 on Resolution - IEEE Transactions on Computers, August 1976.

17 . WILSON, G.A. and NINKER J. - Resolution, Refinements , and Search Strategies: A comparative study - IEEE Transactions on Computers, August 1976.

18 . SLAGLE, J.R. - Automatic Theorem Proving with Built - in - Theories Including. Equality, Partial Ordering , and Sets - J.ACM, 1972.

19 . LUCKHAM, D. - IRIA Symposium on Automatic Deduction - Versailles, França, 1968.

20 . WOS, LAIVRENCE, ROBINSON e CARSON - J.ACM 12, 1965.

21 . ANDREWS, P.B. - J.ACM 15, 1968.

22 . NILSON, N.J. - Problem Solving Methods in Artificial Intelligence - McGraw-Hill Book Company, 1971.

23 . ROBINSON, G. - Paramodulation and Theorem Proving in First-Order Theories with Equality - Machine Intelligence 4, 1969.

24 . BLEDSOE, W.W. - Splitting and Reduction Heuristics in Automatic Theorem Proving - Artificial Intelligence 2, 1971.

25 . CARVALHO, R. Lins de - Some Results in Automatic Theorem-Proving with Applications in Elcmcntary Set Thcory and

Topology - Terhnical Report 71, 1974, Departament of Computer Scicnce, University of Toronto, Canada.

26 . BLEDSOE, W.W. e BRUELL, P. - A Man-Machine Theorem - Proving System - Artificial Intelligence 5, 1974.

27 . PASSOS, E.P.L., de CARVALHO, R.L. e PEIXOTO, S.R. - Communication Predicates: Complete Strategy for Resolution-Based Thcorcm-Provers An Evaluation of An Implementation - Current Topics in Cybernetics and Systems WOGSC, 1978, Springer-Verlag.

28 . PASSOS, E.P.L. - Sistema Auxiliar do Matemático (S.A.M) - Anais do 6º Painel de Discussão sobre Tópicos de Computação, 1979, Valparaiso - Universidade Católica - Chile.

29 , BAUM, J.D. - Elements of Point Set Topology - 1964 ., Prentice-Hall, Inc.

30 . KURATOWSKI, K. - Introduction to Set Theory and Topology, 1972, Pergamon Press.

31 . PEIXOTO, S.R.G. - Experimentação para um sistema de prova automática de teoremas - Tese de Mestrado, Dept. INF-PUC/RJ, 1975.

32 . LANZELOTE, R.S.G., PASSOS, E.P.L., de Carvalho, R.L. - MID a Conversational System Oriented for Topology , EMCR-80, Austria.

33 . LANZELOTE, R.S.G. - Manipulador de Teorias Definicionais - Tese de Mestrado, INF-PUC/RJ, 1977.

34 . PION, M.M.B., de Carvalho, R.L. e PASSOS, E.P.L. - Interactive System to Construct Minimal Models on the Herbrand Universe, Publ. *8*, IME, 1980.

35 . ROCHA, E.S. - Análisador Sintático para prova de Teoremas - Tese de Mestrado, INF-PUC/RJ, 1981:

APÊNDICE A

UM "PROVADOR" PARA TOPOLOGIA

A.1 - Introdução.

Neste capítulo apresentaremos uma axiomatização para Point Set Topological - PST BAUM[29], KURATOWSKI[30], um irovador automático de teoremas .baseado nessa axiomatização e finalmente os problemas encontrados com esse provador.

A.2 - Axiomatização para PST

Nosso objetivo aqui é estudar o que comumente chamamos Point Set Topology (topologia por conjuntos de pontos). PST sabemos, é básico para estudos mais avançados da: Análise .na qual as noções de álgebra e topologia, são relacionadas nos estudos de .sistemas automáticos; a própria PST avançada onde extendemos as idéias; e Topologia Algébrica que estuda algumas técnicas da álgebra para aprofundar o estudo de espaços topológicos.

Pressupomos saber todas as noções da Álgebra dos conjuntos (Teoria dos Conjuntos) e que sabemos também estarmos estudando geometria, dos conjuntos de pontos. Portanto os elementos dos nossos conjuntos são pontos; Introduziremos axiomas nesses conjuntos de postos via definição.

Mas antes apresentaremos uma lista de predicados e símbolos funcionais, veja capitulo II de LINS[25].

TOPOLOGIA T(X,Y)

O CONJUNTO Y É UMA TOPOLOGIA PARA O CONJUNTO X.
SIGNIFICA QUE:

1- Y É UMA FAMILIA DE SUBCONJUNTOS DE X.
QUE CONTÉM X

2- A UNIÃO DE QUAISQUER DUAS SUBFAMILIAS DE Y ESTA EM Y.

3- A INTERSEÇAO DE 2 CONJUNTOS DE Y ESTA EM Y.
OS ELEMENTOS DE Y SAO OS CONJUNTOS ABERTOS DA TOPOLOGIA T(X,Y) ,

VIZINHANÇA NBH(X,Y,Z,x)

O CONJUNTO Z É UMA VIZINHANÇA PARA O PONTO W (ISSO RELATIVAMENTE A TOPOLOGIA (X,Y) SIGNIFICA:

Z É UM CONJUNTO ABERTO QUE CONTÉM x.

LIMITE LIM(X,Y,Z,x)

x É PONTO LIMITE PARA O CONJUNTO Z SIGNIFICA:
QUALQUER VIZINHANÇA DE x CONTÉM PONTOS DE Z DISTINTOS DE x.

- CONJUNTO DERIVADO DE Z - É O CONJUNTO DE TODOS OS PONTOS LIMITES DE Z.
  DER(X,Y,Z)
- O INTERIOR DE UM CONJUNTO Z - É O CONJUNTO DOS PONTOS QUE POSSUEM UMA VIZINHANÇA CONTIDA EM Z.
  INT(X,Y,Z)
- Z É UM CONJUNTO FECHADO - SIGNIFICA QUE O COMPLEMENTO DE Z EM RELAÇÃO A X E UM CONJUNTO ABERTO NA TOP(X ,Y) .
  CLOSED(X,Y, Z)
- O FECHO DE Z - É A UNIÃO DE Z E SEUS PONTOS LIMITES.
  CLOS(X ,Y, Z)
- A BORDA DE UM CONJUNTO Z - É O CONJUNTO DE PONTOS LIMITES DE Z E (X-Z)
  BD(X,Y,Z)
- O CO'NJUNTO Z É DENSO - SIGNIFICA QUE TODOS OS PONTOS DE Z SÃO PONTOS LIMITES.
  DENSE(X,Y,Z)

- Y É UM CONJUNTO PERFEITO . PERF(X,Y,Z) - SIGNIFICA QUE TODOS OS PONTOS DE Z SÃO PONTOS LIMITES, E TODOS OS PONTOS LIMITES DE Z SAO ESSES.

- OS CONJ. Y Z SAO SEPARADOS SEP(x,Y, Z,V) - SIGNIFICA QUE NÃO HÁ PONTO OU PONTO LIMITE EM COMUM A V B Z.

- O CONJ. Y É LIGADO CONCT(X,Y,Z) - SIGNIFICA QUE Z NÃO É A UNIÃO DE 2 . COJUNTOS SEPARADOS.

Esses predicados e símbolos funcionais acima são definidas para serem utilizados em nossos sistemas provadores por RESOLUÇÃO, pelos seguintes conjuntos de cláusulas .aqui explicados:

Se nós queremos dizer que Y é uma topologia para X escrevemos T(X,Y) . Isso implica que:

t_1 "Y é uma família de subconjuntos de X"

t_2 "que contém X e tal que"

t_3 a união de qualquer subfamília de Y está em Y"

t_4 "se 2 conjuntos Z e W estão em Y, sua interseção também está".

$T(A1,A2) \Leftrightarrow A2 < P(A1)$ e $A1 \in A2$ e $FAM1 < A2 \mid U(FAM1) \in A2$ e $(\neg VAR1 \in A2) \mid (\neg VAR2 \in A2) \mid (VAR1 \text{ INTER } VAR2) \in A2$

Fazendo por partes temos:

$T(A_1,A_2) \Rightarrow t_1$ ou $T(A_1,A_2) = A_2 \subseteq P(A_1)$

ou: T1. $\neg T(A_1,A_2 \vee A_2 \subseteq P(A_1)$

$T(A_1,A_2) \Rightarrow t_2$ ou $T(A_1,A_2) \Rightarrow A_1 \in A_2$ ou

T2. $\neg T(A_1,A_2) \vee A_1 \in A_2$

A.4

$T(A_1A_2) \Rightarrow t_3$ ou $T(A_1,A_2) \Rightarrow FAM1 \subseteq A_2$ v...

... $\sqcup(FAM1) \in A_2$ ou

T3. $\neg T(A_1,A_2) \vee \neg FAM1 \subseteq A_2 \vee \sqcup(FAM1) \in A_2$

$T(A_1,A_2) \Rightarrow t_4$ ou $T(A_1,A_2) \Rightarrow (\neg VAR1 \in A2) \vee (\neg VAR2 \in$

$\in A_2) \vee ... \vee ((VAR1 \cap VAR2) \in A_2))$ ou

T4. $\neg T(A_1,A_2) \vee (\neg VAR1 \in A2) \vee (\neg VAR2 \in A_2) \vee (VAR1 \cap VAR2) \in A_2))$

Portanto se A_2 é uma topologia para A_1 temos quatro cláusulas T1, T2, T3 e T4 que definem para resolução esse fato.

Reciprocamente se Y é uma família de subconjuntos de X que contém X e a união de qualquer subfamília de Y está em Y e se dois conjuntos Z e W estão em Y a sua interseção também está, temos que Y é uma topologia para X, ou seja, temos T(X,Y). E em forma de cláusulas temos:

$$\forall X\ \forall Y\ (Y \subseteq P(X) \wedge \cup(Y) = X \wedge \forall Z(Z \subseteq Y \rightarrow \cup(Z) \in Y) \wedge$$

$$\wedge \forall V_1\ \forall V_2\ ((V_1 \in Y \wedge V_2 \in Y) \rightarrow (V_1 \cap V_2) \in Y)) \Rightarrow T(X,Y).$$

Agora traduzindo para cláusulas. na D.N.F (fórmula normal disjuntiva), teremos: (observando as funções de SKOLEM).

A_1 A_2 A_3

t5: $T(X,Y) \vee \neg Y \subseteq P(X) \vee \neg \cup(Y) = X \vee (G(X,Y) \subseteq X \wedge \neg \cup(G(X,Y)) \in Y) \vee$

A_4

$(g(X,Y) \in Y\ \ h(X,Y) \in Y\ \ \neg(g(X,Y) \cap h(X,Y) \in Y)$

As frases são disjunções de A_i*.

Agora lembramos que um caminho natural para provar que certas famílias A de conjuntos é uma topologia para um certo conjunto a é:

(i) provar que $A \subset P(a)$;

(ii) provar que $\cup(A) = a$;

(iii) provar que para todas as famílias Y, se $Y \subseteq A$ então $(Y) \varepsilon A$

[iv) provar que se $Y \in A$ e $Z \in A$ então $(Y \cap Z) \in A$.

Agora pela introdução dc três prcdicados de comunicação sugeridos por LINS[25], P_1, P2 e P3 nós podemos "imitar" **(i)** → (iv). Para isso. troca-se t5 por sua forma expandida.

t'5 $\neg T(X,Y) \vee \neg y \subseteq P(X) \vee P_1(X,Y)$ — onde $P_1(X,Y)$ é o t5 menos $y \subseteq P(X)$

t'6 $\neg P_1(X,Y) \vee \underline{\neg \cup(Y) = X}$ ou $\neg X \varepsilon Y \vee P_2(X,Y)$ — onde $P_2(X,Y)$ é o t5 menos $\exists Y \subseteq P(X)$ menos $\neg \cup(Y) = X$

t'7 $\neg P_2(X,Y) \vee (G(X,Y) \subseteq X \wedge \neg \cup(G(X,Y) \in Y) \vee P_3(X,Y)$ novamente — onde $P_3(X,Y)$ é o t5 menos $\subset P(X)$ menos $\neg \cup(Y) = X$ menos $G(X,Y) \subset X \wedge \neg \sqcup(G(X,Y)) \varepsilon Y$

t'8 $\neg P_3(X,Y) \vee (g(X,Y) \varepsilon y \wedge h(X,Y) \varepsilon X \wedge \neg(g(X,Y) \cap h(X,Y) \varepsilon Y)$

Aqui novamente em t'7 e t'8 usamos frases. Precisamos colocá-los naforma expandida. Usamos uma estratégia semelhante aquela discutida acima ((i) (iv), retirando-se as frases.

De t'7 vem:

t"7 $\neg P_2(X,Y) \vee (G(X,Y) \subseteq X) \vee P_3(X,Y)$ onfe P_3 é o resto.

t"7 $\neg P_2(X,Y) \vee \neg \cup(G(X,Y)) \varepsilon y \vee P_3(X,Y)$

De t'8 vem:

t"8 $\neg P_3(X,Y)$ v $g(X,Y)$ ε Y v resto (não tem mais disjunção ; logo não tem resto, não tem P_4)

t"'8 $\neg P_3(X,Y)$ v $h(X,Y)$ ε Y

t'v8 $\neg P_3(X,Y)$ v $\neg((g(X,Y) \cap h(X,Y)) \varepsilon Y)$

Portanto se temos T5, T6, T7, T8, T9, T10, T11 nós podemos dizer (temos definido) que A2 é uma topologia para A1.

T5. $T(X,Y)$ v $\neg Y \subseteq P(X)$ v $P_1(X,Y)$

T6. $\neg P_1(X,Y)$ v $\dfrac{\neg \cup(Y)=X}{\neg X \varepsilon Y}$ v $P_2(X,Y)$

T7. $P_2(X,Y)$ v $(G(X,Y) \subseteq X)$ v $P_3(X,Y)$

T8. $P_2(X,Y)$ v $\cup(G(X,Y)$ ε Y v $P_3(X,Y)$

T9. $P_3(X,Y)$ v $g(X,Y)$ Y

T10. $P_3(X,Y)$ v $h(X,Y)$ ε Y

T11. $P_3(X,Y)$ v $((g(X,Y)$ $h(X,Y))$ ε Y)

Agora verificaremos como definimos, em cláusulas , os predicados e os símbolos funcionais para TOPOLOGIA.

Em termos de implementação no computador temos:

T(A1,A2) $\rightleftharpoons$ A2 < P(A1) ε A1 E A2 ε FAM1 < A2 | (FAM1) E A2

(VAR1 e A2) | (VAR2 E A2) | (VAR1 INTER VAR2) E A2

onde ε representa AND,

E representa pertence e

< representa contido.

$\neg$T(A1,A2) $\rightleftharpoons$ A2 < P(A1) | S(A1,A2)

Analogamente temos as cláusulas que definem os outros predicados e símbolos funcionais são:

NBH.

C1. $\neg T(X,Y) \vee \neg NBH(X,Y,Z,x) \vee x \in z$

C2. $\neg T(X,Y) \vee \neg NBH(X,Y,X,x) \vee z \in Y$

C3. $\neg T(X,Y) \vee NBH(X,Y,Z,x) \vee \neg x \in Z \vee \neg z \in Y$

em termos de implementação no computador

$NBH(A1,A2,A3,A4) \leftrightharpoons \neg T(A1,A2) \mid (A4 \text{ E } A3)$ e

$\neg T(A1,A2) \mid (A3 \text{ E } A2)$

$\neg NBH(A1,A2,A3,A4) \leftrightharpoons T(A1,A2) \mid \neg(A4 \text{ E } A3) \mid \neg(A3 \text{ E } A2)$

LIM

C4. $\neg T(X,Y) \vee \neg lim\ (X,Y,Z,x) \vee \neg nbh\ (X,Y,V,x) \vee \neg(V-\{x\}) \cap Z = \emptyset$

[redacted] $\neg T(X,Y) \vee lim\ (X,Y,Z,x) \vee nbh\ (X,Y,h_4(X,Y,Z,x),x)$

C6. $\neg T(X,Y) \vee lim\ (X,Y,Z,x) \vee (h_4(X,Y,Z,x) - X) \cap Z = \emptyset$

em termos de implementação

$LIM(A1,A2,A3,A4) \leftrightharpoons \neg T(A1,A2) \mid \neg NBH(A1,A2,VAR1,A4) \mid$

$\neg(INTER(VAR1 - (A4)),A3) = VAZIO)$

$\neg LIM(A1,A2,A3,A4) \leftrightharpoons \neg T(A1,A2) \mid NBH(A1,A2,FUN5,A4)$ e

$\neg T(A1,A2) \mid$

$(INTER(VAR1 - (A4)),A3) = VAZIO)$

DER

C.7. $\neg T(X,Y) \vee \neg x \in der\ (X,Y,Z) \vee lim\ (X,Y,X,x)$

C.8. $\neg T(X,Y) \vee x \in der\ (X,Y,Z) \vee \neg lim\ (X,Y,Z,x)$

em termos de implementação

$A4 \text{ E } DER(A1,A2,A3) \leftrightharpoons \neg T(A1,A2) \mid LIM(A1,A2,A3,A4)$

$\neg A4 \text{ E } DER(A1,A2,A3) \leftrightharpoons \neg T(A1,A2) \mid \neg LIM(A1,A2,A3,A4)$

INT

C.9. $\neg T(X,Y) \vee \neg x \varepsilon \underline{int}\ (X,Y,Z) \vee \underline{nbh}\ (X,Y,h_3(X,Y,Z,x),x)$

C.10. $\neg T(X,Y) \vee \neg x \varepsilon \underline{int}\ (X,Y,Z) \vee h_5(X,Y,Z,x) \subseteq Z$

C.11. $\neg T(X,Y) \vee x \varepsilon \underline{int}\ (X,Y,Z) \vee \neg \underline{nbh}\ (X,Y,V,x) \vee V \subseteq Z$

em termos de implementação

A4 E INT(A1,A2,A3) $\rightleftharpoons$ $\neg T(A1,A2) \mid NBH(A1,A2,FUN6,A4)$ e

$\neg T(A1,A2) \mid (FUN6 < A3)$

$\neg$A4 E INT(A1,A2,A3) $\rightleftharpoons$ $\neg T(A1,A2 \mid \neg NBH(A1,A2,VAR1,A4) \mid$

$\neg VAR1 < A3$

CLOSED

C.12. $\neg T(X,Y) \vee \neg \underline{closed}\ (X,Y,Z) \vee (X-Z) \varepsilon Y$

C.13. $\neg T(X,Y) \vee \underline{closed}\ (X,Y,Z) \vee \neg (X-Z) \varepsilon Y$

em termos de implementação

CLOSED(A1,A2,A3) $\rightleftharpoons$ $\neg T(A1,A2) \mid (A1-A3)$ E A2

$\neg$CLOSED(A1,A2,A3) $\rightleftharpoons$ $\neg T(A1,A2) \mid \neg (A1-A3)$ E A2

CLOS

C.14. $\neg T(X,Y) \vee \neg x \varepsilon \underline{clos}\ (X,Y,Z) \vee x \varepsilon Z \vee \underline{lim}\ (X,Y,Z,x)$

C.15. $\neg T(X,Y) \vee x \varepsilon \underline{clos}\ (X,Y,Z) \vee \neg x \varepsilon Y$

C.16. $\neg T(X,Y) \vee x \varepsilon \underline{clos}\ (X,Y,Z) \vee \neg \underline{lim}\ (X,Y,Z,x)$

A4 E CLOS(A1,A2,A3) $\rightleftharpoons$ $\neg T(A1,A2)$ (A4 E A3) LIM(A1,A2,A3,A4)

$\neg$A4 E CLOS(A1,A2,A3) $\rightleftharpoons$ $\neg T(A1,A2)$ $\neg$A4 E A3 e

$\neg T(A1,A2)$ $\neg$LIM(A1,A2,A3,A4)

BD

C.17. $\neg T(X,Y) \vee \neg x\ \underline{bd}\ (X,Y,Z) \vee \underline{lim}\ (X,Y,Z,x)$

C.18. $\neg T(X,Y) \vee \neg x\ \underline{bd}\ (X,Y,Z) \vee \underline{lim}\ (X,Y,(X-Z),x)$

C.19. $\neg T(X,Y) \vee x\ \underline{bd}\ (X,Y,Z) \vee \neg \underline{lim}\ (X,Y,Z,x) \vee \neg \underline{lim}\ (X,Y,(X-Z),x)$

em termos de implementação

A4 E BD(A1,A2,A3)⇌ ¬T(A1,A2) | LIM(A1,A2,A3,A4) e

¬T(A1,A2) | LIM(A1,A2,A1-A3,A4)

¬A4 E BD(A1,A2,A3)⇌ ¬T(A1,A2) |¬LIM(A1,A2,A3,A4) |

¬LIM(A1,A2,A1-A3,A4)

DENSE

C.20. ¬T(X,Y) v ¬dense (X,Y,Z) v Z⊂der (X,Y,Z)

C.21. ¬T(X,Y) v dense (X,Y,Z) v ¬Z⊂der (X,Y,Z)

em termos de implementação

DENSE(A1,A2,A3)⇌ ¬T(A1,A2) | A3 < DER(A1,A2,A3)

¬DENSE(A1,A2,A3)⇌ ¬T(A1,A2)| ¬A3 <DER(A1,A2,A3)

PERF

C.22. ¬T(X,Y) v ¬perf (X,Y,Z) v Z=der (X,Y,Z)

C.23. ¬T(X,Y) v perf (X,Y,Z) v Z=der (X,Y,Z)

em termos de implementação

PERF(A1,A2,A3)⇌ ¬T(A1,A2) | A3 = DER(A1,A2,A3)

¬PERF(A1,A2,A3)⇌ ¬T(A1,A2)| ¬A3 = DER(A1,A2,A3)

SEP

C.24. ¬T(X,Y) **v** sep (X,Y,Z,V) v ¬Z=∅

C.25. ¬T(X,Y) v sep (X,Y,Z,V) v ¬V=∅

C.26. ¬T(X,Y) **v** sep (X,Y,**Z**,V) v Z∩clos **(X,Y,V)** = ∅

C.27. ¬T(X,Y) v sep (X,Y,Z,V) v V clos (X,Y,Z) = ∅

C.28. ¬T(C,Y) **v** sep (X,Y,Z,V) v ¬x∈Z **v** S_1(X,Y,Z,V)

C.29. ¬S_1(X,Y,Z,V) v ¬y∈V v S_2(X,Y,Z,V)

C.30. ¬S_2(X,Y,Z,V) v ¬(Z∩clos (X,Y,V)) = ∅ v S_3(X,Y,Z,V)

C.31. ¬S_3(X,Y,**Z**,V) **v** ¬(V∩clos (X,Y,Z))=∅

onde S_1, S_2 e $_3$**S** são pcedicados de comunicação.

em termos de implementação

SEP(A1,A2,A3,A4)⇌ ¬T(A1,A2) | ¬A3 = W e

A.10

$\neg T(A1,A2) \mid \neg A4 = W$ e

$\neg T(A1,A2) \mid .. A3$ INTER CLOS$(A1,A2,A3) = W$ e

$\neg T(A1,A2) \mid R4$ INTER CLOS$(A1,A2,A3) = W$

$SEP(A1,A2,A3,A4) \rightleftharpoons \neg T(A1,A2) \mid \neg VAR1$ E $A3 \mid S1(A1,A2,A3,A4)$

CONCT .

C.32. $\neg T(X,Y)$ v $\neg$conct (X,Y,Z) v $\neg$sep (X,Y,Z,W) v $\neg Y=(Z \cup W)$.

C.33. $\neg T(X,Y)$ v conct(X,Y,Z) v sep$(X,Y,h_6(X,Y,Z),h_7(X,Y,Z))$

C.34. $\neg T(X,Y)$ v conct (X,Y,Z) v $Y=(h_6(X,Y,Z)\ h_7(X,Y,Z)$

em termos de implementação

$CONCT(A1,A2,A3) \rightleftharpoons \neg T(A1,A2) \mid SEP(A1,A2,VAR1,VAR2) \mid$

$\mid A3 = (VAR1 \cup VAR2)$

$CONCT(A1,A2,A3) \rightleftharpoons \neg T(A1,A2) \mid SEP(A1,A2,FUN7,FUN8)$ e

$\neg T(A1,A2) \mid A3 = (FUN7 \cup FUN8)$

A.2 - Um Provador Automático de Teorema para $(BAC)_i$

Em 1974, desenvolveu-se ums sistema que transformava uni conjunto S de cláusulas em um conjunto S' de cláusulas, através da introdução de novos símbolos predicativos, chamados "communication predicates" LINS[25], PASSOS[27]

Em particular, dado uma cláusula C_1 v C_2, um conjunto S de cláusulas e um novo símbolo predicativo P, nós dizemos que $S \cup \{C_1 \text{ v } C_2\}$ é insatisfatível $\rightleftharpoons S \cup \{C_1 \text{ v } P(\bar{v})\ C_2 \text{ v } P(\bar{v})\}$ é insatisfatível, onde $\bar{v}$ contém todas as variáveis comuns a C_1 e C_2 (similar a splitting mas permite variáveis comuns)

E uma nova estratégia, P-refutação é provada ser completa.

P-refutação é adequada para manipular conjuntos "separable", por exemplo aqueles que contém "predicados de comunicação". P-refutação pode ser considerado uma generalização de técnica de "splitting" de Bledsoe[24].

Predicados de comunicação e P-refutação tem se mostrado ferramenta muito Útil para teorias que são definicionais, isto é, teorias que contém em grande número de definições.

O sistema desenvolvido em LINS[27] possui basicamente duas(2) partes distintas:

a) Tradução - Essa parte reduz todos os símbolos funcionais e predicativos para um Único símbolos predicativo, E.

b) Refutação - Essa parte é um procedimento P-refutação junto com alguma estratégia compatível.

O provador é "extendível" no sentido que somente a parte de (TR1) tradutores' é modificada quando novos símbolos funcionais são introduzidos.

Lins mostra que pelo uso de um sistema tradutor pode-se construir um procedimento de decisão para $(BAC)_0$, que é o conjunto de todos universal closures of quantifier-free fórmulas no alfabeto com símbolos predicativos $\subseteq$ = e $\subset$ e símbolos funcionais $\cap$ (inter) $\cup$ (união) - (diferençâ) $\subset$(complemento).

Nenhum procedimento de decisão foi encontrado para qualquer extensão mais forte.

Alguns resultados de uma experiência preliminar de um provador para $(BAC)_1$ (uma extensão do $(BAC)_0$ que inclui conjunto potenciais, união e interseção de famílias de conjunto) também foram conseguidos.

A.4, - Um Provador Automático de Teoremas para PST

A extensão do provador para Point Set Topological será feita acrescentando a TRI e TR2 ao provador de $(BAC)_i$.

Por exemplo:

Se encontramos $<$, $=$, $\varepsilon\cup$, $\varepsilon\cap$, $\varepsilon+(A,B)$ ε (A o provador já sabe traduzir, AGORA se encontramos T(X,Y) vamos acrescentar a traduçás em TR1.

T1, T2, T3 e T4 já descritos anteriormente

e se encontrarmos T(X,Y), vamos acrescentar em TR2

T5, T6, T7, T8, T9, T10, T11 também já descritos anteriormente e T12 até T45.

O provador aqui descrito é uma modificação, para provar teoremas em Point Set Topology, do Trabalho de Peixoto31 sugerido por Lins de Carvalho. Era originariamente um provador automático de Teoremas somente para Álgebra Booleana das Classes, $(BAC)_i$, $i=\overline{0,2}$.

A.4.1 - FLUXOGRAMA GERAL DO PROVADOR

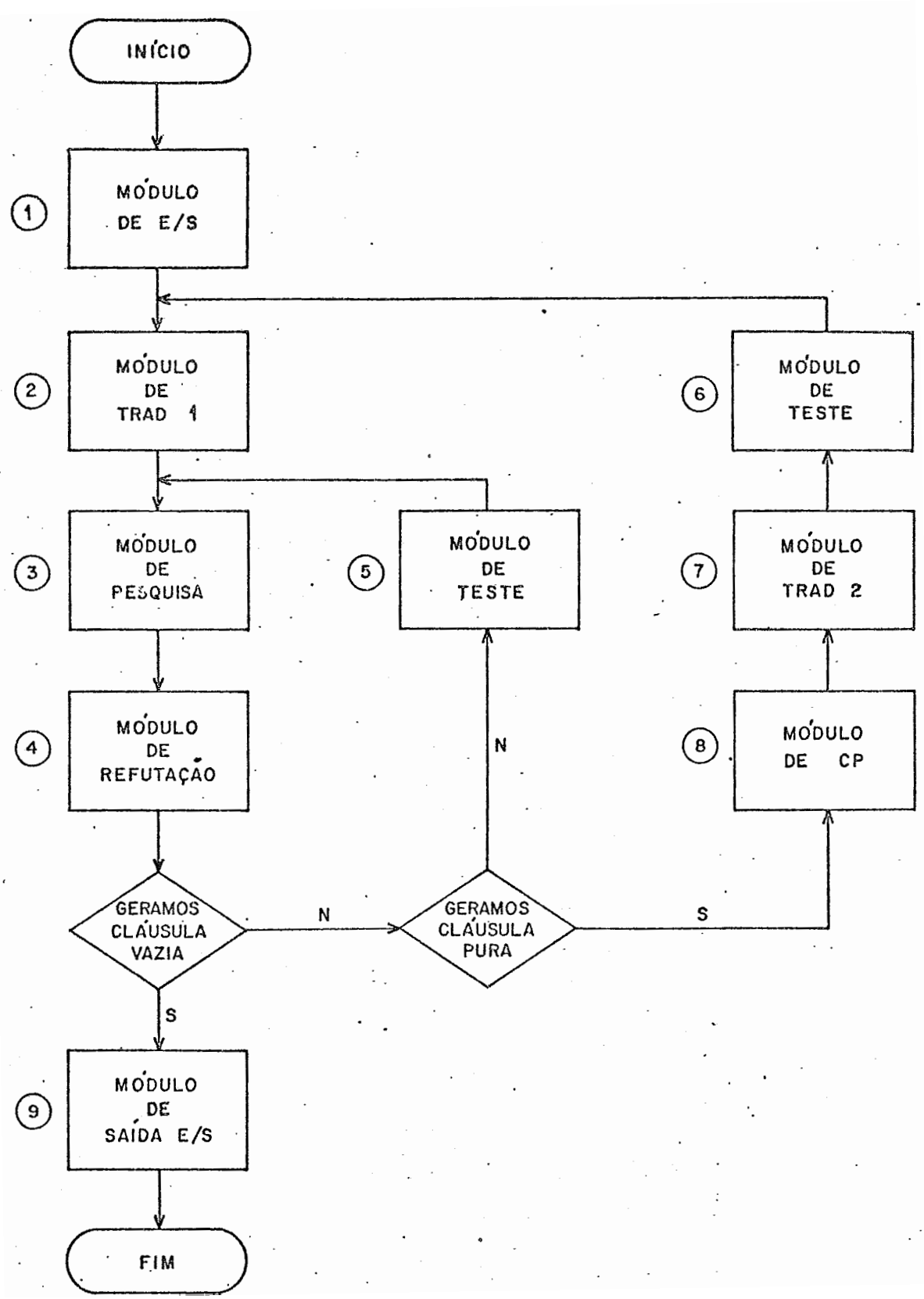


FIGURA A.1

Comentários sobre os módulos dos componentes do provador.

Modificou-se o programa para $(BAC)_i$ original para provar teoremas em topologia e as modificações foram:

1) Entrada das cláusulas que definem a negação do teorema a ser provado, montagem da lista de cláusulas e impressão de mensagens iniciais, (MODIFICADO).

2) Esse módulo fará a tradução do conjunto de cláusulas, sendo que estas serão lidas separadamente uma a uma. (modificada).

   Basicamente a sua tarefa é transformar uma lista de cláusulas dada em outra equivalente a esta, constituída por apenas cláusulas básicas, que são obtidas a partir de um processo de redução a primitivas, tendo como base, o conjunto de axiomas e definições da teoria.

   Por exemplo, suponhamos a cláusula $Y_1 \in A = Y_2 \in B * A = B * A \subset B$, suponhamos também que o literal a ser traduzido, seja o terceiro, isto é, A=B. Como sabemos a tradução para A=B será (A⊆B e B⊆A) ou seja 2 cláusulas consequentemente obteremos: $Y_1 \in A * Y_2 \in B * A \subset B * A \subset B$ e $Y_1 \in A * Y_2 \in B * BCA * A \subset B$, e continuamos com a tradução do 3º literal e assim por diante.

   A modificação aqui foi a inclusão da tradução para os predicados e símbolos funcionais para Point Set Topology.

3) Aqui faz-se uma pesquisa na lista de cláusulas para resolução. É um algorítmo para a estratégia da cláusula mais curta usando "subssumption" estratégia compatível com P-refutação. (não foi modificado].

4) Este módulo será o provador propriamente dito, envolvendo a estratégia de P-refutação, princípio de resolução para lógica de 1ª ordem e algorítmo de unificação. Terá como entrada um conjunto de cláusulas básicas e rejeitará qualquer conjunto de cláusulas não básicas que tenha sido gerado. Sempre que uma cláusula pura for obtida, este módulo ativar5 o módulo de CP (predicados de comunicação) que

cuidará desta; (Não foi modificado).

5 e 8) Aqui faz-se testes para cada cláusula resolvente ou pura retraduzida, com todas as demais cláusulas já no sistema. Esses testes verificarão se a nova cláusula é "boa", isto é, deve ser acrescida ao sistema. (modificado).

6) Cuidará também das cláusulas puras obtidas por tradução ou resolução. Estas serão testadas com as demais puras anteriormente obtidas, para verificação de sua ocorrência ou não anterior. Se esta pura não tiver sido obtida ainda, será retraduzida e integrará a lista de cláusulas. (Não foi modificado).

7) Este módulo tratará da retradução das "puras", acima mencionadas. (Modificado).

9) Impressão das cláusulas do processo de prova e do resultado. (Modificada).

A.4.2 - Modificações sugeridaç'e resultados .

As modificações sugeridas por Lins de Carvalho de somente extendes os procedimentos de tradução TRAD1 e TRAD2 não foram suficientes "para provas: todos os exemplos do capítulo 6 de LINS[25] exceto dois". Já no terceiro exemplo 6.1.3, de LINS[25].

"A interseção de uma família de topologias para X é uma topologia para X"

esse provador gerou uma grande quantidade de "lixo" sem que conseguissemos chegar ao final da prova por problema de memória.

Os exemplos 6.1.1 e 6.1.2 chegaram a um bom termo, como veremos a seguir:

Exemplo 6.1.1 de LINS[25]

$\forall X(T(X,P(X)))$ que significa: "O conjunto potência de um conjunto X é uma topologia para X".

No Provador modificado por nós temos:

conjunto de cláusulas dadas (que define a negação do teorema a ser provado).

NOT.$3T(A, δ 2%(A));

conjunto de cláusulas traduzi'das

(1)
⋮
(9)

tradução após eliminação da cláusula pura

⋮
(2)
⋮
(8)

( ) 'cláusula vazia obtida por tradução

Resposta

O conjunto de cláusulas dadas é inconsistente. Provou-se o teorema.

O detalhe da prova de 6.1.1

1. $\neg T(A,P(A))$ negação da fórmula dada
2. $\neg P(A) \subseteq P(A) \vee R_1(A,P,(A))$ (1,T5)
3. $R_1(A,P(A))$ (2,$(BAC)_1$)

   $R_1(A,(P(A))$ é cláusula pura, e então é eliminada.

2'. $\neg A \in P(A) \vee R_2(A,P(A))$ (3,T6)

3'. $R_2(A,P(A))$ (2,$(BAC)_1$)

   $R_2(A(P(A))$ é cláusula pura, e então é eliminada.

2". $h_1(A,P(A)) \subseteq P(A) \vee R_3(A,P(A))$ (3',T7)

3". $\neg\sqcup(h_1(A,P(A)) \in P(A) \vee R_3(A,P(A))$ (3',T8)

Geração de 4

(a) $h_1 \subseteq P(A) \rightarrow \neg\forall x \in h \vee x \in P(A);\quad x \in h_1 \vee x \subseteq A;$

$\neg x \in h_1 \vee \neg y \in x \vee y \in A$

(b) $\neg\cup h_1 \in P(A)\ \neg\cup h_1 \subseteq A;\ a \in \cup h_1 \vee \neg a\ A;$

Em (a) fazendo $\neg y \in b \vee y \in A$ $\quad y=a \Rightarrow \neg a \in b$ $\quad a \in b$ $\}$ □

$b \in h_1 \vee a \in b$

4. $R_3(A,P(A))$ ($\{2'', 3''\}$ e $(BAC)_1$)

$R_3(A,P(A))$ é pura, e então é eliminada e de 2'' e 3''

2'''. $h_2(A,P(A)) \in P(A)$ (4,T9)

3'''. $h_3(A,P(A)) \in P(A)$ (4,T10)

4'. $\neg h_2(A,P(A)) \cap h_3(A,P(A)) \in P(A)$ (4,T11)

Geração de 5.

(a1) $h_2 \in P(A) \rightarrow h_2 \subseteq A \rightarrow \neg x \in h_2 \vee x \in A$

(a2) $h_3 \in P(A) \rightarrow h_3 \subseteq A \rightarrow \neg x\ h_3 \vee x \in A$

(a3) $\neg(h_2 \cap h_3) \in P(A) \rightarrow f_1(h_2 \cap h_3, A) \in h_2$

(a4) $\neg(h_2 \cap h_3) \in P(A) \rightarrow f_1(h_2 \cap h_3, A) \in h_3$

(a5) $\neg(h_2 \cap h_3) \in P(A) \rightarrow \neg f_1(h_2 \cap h_3, A) \in A$

(a6) $f_1(h_2 \cap h_3, A) \in A$ (a3,a1)

(a7) □ (a5,a6)

Logo

5. □ por $(\{2''', 3''', 4'\}, (BAC)_1)$

Exemplo 6.1.2 de LINS[25].

$\forall(T(X,\{\emptyset,X\}))$ de que significa

"O conjunto X mais o conjunto vazio $\emptyset$ é uma topologia para X."

DETALHE DA PROVA

1. $\neg T(X,\{\emptyset,X\})$ negação da fórmula dada

2. $\neg\{0,X\} \subseteq P(X) \vee R_1(X,\{\emptyset,X\})$ (1,T5)

3. $R_1(X,\{\emptyset,X\})$ $(2,(BAC)_1)$

$R_1(X,\{\emptyset,X\})$ é cláusula pura e subassumiremos em 2.

2'. $\neg X \in \{\emptyset,X\} \vee R_2(X,\{0,X\})$ (3,T6)

3'. $R_2(X,\{0,X\})$

$R_2(X,\{0,X\})$ é cláusula pura e subassume 2'

2". $h_1(X,\{\emptyset,Z\}) \subseteq \{\emptyset,X\} \vee R_3(X,\{\emptyset,X\})$ (3',T7)

3". $\neg\sqcup(h_1(X,\{\emptyset,X\})) \in \{\emptyset,X \vee R_3(X,\emptyset,X\})$ (3',T8)

Geração de 4.

(a) $h_1 \subseteq \emptyset,X \rightarrow \neg x \in h_1 \vee x \in \{\emptyset,X\} \rightarrow \neg x\ h_1 \vee x \subseteq X\ x \subseteq \emptyset \rightarrow$

$\neg x \in h_1 \vee \neg y \in x \vee y \in X \vee z \in x \vee A \in \emptyset$

(b) $\neg\sqcup h_1 \in \{\emptyset,X\} \rightarrow \sqcup h_1 \subseteq X \rightarrow a \in \sqcup h_1 \vee \neg a \in X;$

$\sqcup h_1 \subseteq \emptyset \rightarrow b \in \sqcup h_1 \vee \neg b \in \emptyset;$

Em (a) fazendo $\neg y \in c \vee y \in X$ $\quad y=a \rightarrow$ $\neg a \in c$

Em (b) fazendo $c \in h_1 \vee a \in c$ $\quad$ $a \in c$ $\}$ □

4. $R_3(X,\{\emptyset,X\})$ $(\{2",3"\}$ e $(BAC)_1)$

$R_3(X,\{\emptyset,x\})$ é cláusula pura e então é eliminado e subassumida 2" e 3".

temos:

2"'. $h_2(X,\{\emptyset,X\}) \in \{\emptyset,X\}$ (4,T9)

3"'. $h_3(X,\{\emptyset,X\}) \in \{\emptyset,X\}$ (4,T10)

4'. $\neg(h_2(X,\{\emptyset,X\}) \cap h_3(X,\{\emptyset,X\})) \in \{\emptyset,X$ (4,T11)

Geração de 5.

(a1) $h_2 \leq \{\emptyset,X\} \rightarrow h_2 \subseteq X \rightarrow \neg x \leq h_2 \vee x \in X$

(a2) $\cap_3 \leq \{\emptyset,X\} \rightarrow h_3 \subseteq X \rightarrow \neg x \in \cap_3 \vee x \in X$

(a3) $\neg(h_2 \quad h_3) \in \{\emptyset,X\} \rightarrow f_1(h_2 \cap h_3,X) \in h_2 \subseteq X$

(a4) $\neg(h_2 \cap h_3) \in \{\emptyset,X\} \rightarrow f_1(h_2 \cap h_3,X) \in h_3 \subseteq X$

(a5) $(h_2 \cap h_3) \in \{\emptyset,X\} \rightarrow f_1(h_2 \cap h_3,X) \in X$

(a6) $f_1(h_2 \cap h_3,X) \in X$ (a3,a1)

(a7) □ (a5,a6)

LOGO

5. □ por $(\{2''', 3''^{i}, 4'\}, (BAC)_1)$

Exemplo 6.1.3 de LINS[25] - O provador não deu resposta apesar de ter gcrndo uma grnndc quantidade de cláusulas. Ele poderia ter feito, pois "na mão" temos a prova:

$$\forall x\ \forall x((\neg w=\emptyset \wedge \forall y\ (y\ x \rightarrow T(x,y))) \rightarrow T(x, \sqcap(w)))$$

negar $\rightarrow\neg((\neg x = \emptyset \wedge \forall y(y \in w \rightarrow T(x,y))) \vee T(x,\amalg(w)))$

$$((\neg w = \emptyset \wedge \forall y(y \in w \rightarrow T(x,y))) \wedge \neg T(x,\amalg(w)))$$

$$((\neg w = \emptyset \wedge \forall y(\neg y \in w \rightarrow T(x,y))) \wedge \neg T(x,\amalg(w)))$$

$\underline{1}$ $\underline{3}$ $\underline{3}$

ou seja "a interseção de uma família de topologias para X é uma topologia para X.

1. $l\ x = \emptyset$
2. $\neg x \in \Omega \vee T(A,x)$ 3 cláusulas que negam à fórmula dada
3. $\neg T(A,\amalg(\Omega))$

Reescrevendo e traduzindo.

1'. $f_1(\Omega,\emptyset) \in \Omega$ * [$\emptyset_2(1)$] :....[$(BAC)_i$]
2'. $\neg x \in \Omega \vee T(A,x)$ [igual a 2]
3'. $\neg\amalg(\Omega) \subseteq P(A) \vee R_1(A,x)$ [3, T5]
4'. $T(A,f_1(\Omega,\emptyset))$ [1', 2']

como chegar a 4' por [1',3'] e [2"]

1. $f_1 \in \Omega$
2. $\neg\pi(\Omega) \subseteq P(A) \vee R_1(A,\pi(\Omega))$
3. $f_1 \subseteq P(A)$

novamente:

1. $f_1 \in \Omega$
2". $f_2 \in \pi(\Omega)$ **
3". $\neg f_2 \in P(A)$ **
4". $\neg f_2 \subseteq A$

traduzidno 3:5'. $\exists x \in f_1 \vee x \in P(A) \rightarrow \exists x \in f_1 \vee x \subseteq A \rightarrow$

$\rightarrow \exists x \in f_1 \vee y \in x \vee y \in A$

ou seja $\exists x \in f_1 \vee \exists y \in x \vee y \in A$.

1a. $f_1 \in \Omega$

2a. $\exists z \in ? \vee f_2 \in z$ de 2"

2a.1. $\exists f_3 \in f_2$

2a.2. $\exists f_3 \in A$

3a. $\exists x \in f_1 \vee \exists y \in x \vee y \in A$

4a. $\exists x \in f_1 \vee \exists f_3 \in x$ [2a.2,3a]

5a. $\exists f_2 \in f_1$ $(x=f_2)$ [2a.1,4a]

6a. $\exists f_1 \in \Omega$ [2a.,5]

7. $\square$ [6a,1a]

OBS:

52 tipo 3

A tipo 2

f_1 tipo 2

f_2 tipo 1

x tipo 1

y tipo 0

z tipo 2

f_3 tipo 0

Ω família de topologia para A.

* f_1 é uma topologia para A, $f_1 \subseteq P(A)$

** $\exists(\exists x \in \pi(\Omega) \vee \in X \in P(A)$ fazendo

$f_2 = x \Rightarrow f_2 \ \pi(\Omega) \wedge \exists f_2 \in P(A)$

A.5 - Dificuldades com esse Provador Modificado

Nos testes que fizemos o provador não correspondeu à expectativa, Pois logo no 2º exemplo 6.1.2 gerou uma quantidade muito grande de cláusulas (67) até chegar a cláusula vazia, O exemplo 6.1.3 não conseguimos gerar a cláusula vazia apesar de conseguirmos fazendo "na mão". Talvez teríamos de modificar o unificador e outros componentes desse provador, mas a experiência do autor sugeriu que fizessemos outro provador para PST usando o provador primitivo de $(BAC)_1$ quando fosse necessário (pois nesse caso está comprovado que funciona bem), Esse novo provador será discutido no apêndice B a seguir.

APÊNDICE B

DESCRIÇÃO DO PROVADOR-PST E A ESTRATÉGIA GERADOR DE LEMAS

B.1 - Introdução

Descreveremos um sistema homem-máquina que prova teoremas em Point Set Topology (PST) sem as dificuldades encontradas no Provador do Apêndice A..

As cláusulas que negam o teorema de PST, a ser provado, são passadas pelo módulo de tradução num primeiro nível. Em seguida, o resultado dessa tradução, é passado para um Módulo Principal que comandará a prova para PST.

Esse módulo Principal vai interagir diretamente com o Módulo de simplificação, indiretamente com o provador para $(BAC)_i$ e com o Gerador de Lemas (GEL) por intermédio do próprio usuário.

Essa interação indireta descrita acima, é feita através de uma rotina chamada SON que, quando solicitada pelo usuário faz ligações com o GEL e com o provador $(BAC)_i$.

B.2 - POR QUE SEMI-AUTOMÁTICO? (HOMEM-MÁQUINA)

Os pesquisadores da área de prova automática de teoremas, inclusive o autor desse trabalho, acreditam que precisariam de muitos anos para que as máquinas provem sozinhas teoremas difíceis em matemática BLEDSOE[26]. Portanto quem espera ver suas máquinas como Sistemas Auxiliares práticos dos matemáticos puros, têm redirigido sua atenção para provadores semi-automáticos homem/máquina, BLEDSOE[26], PASSOS[28]..

B.3 — Estrutura Geral do Provador para PST

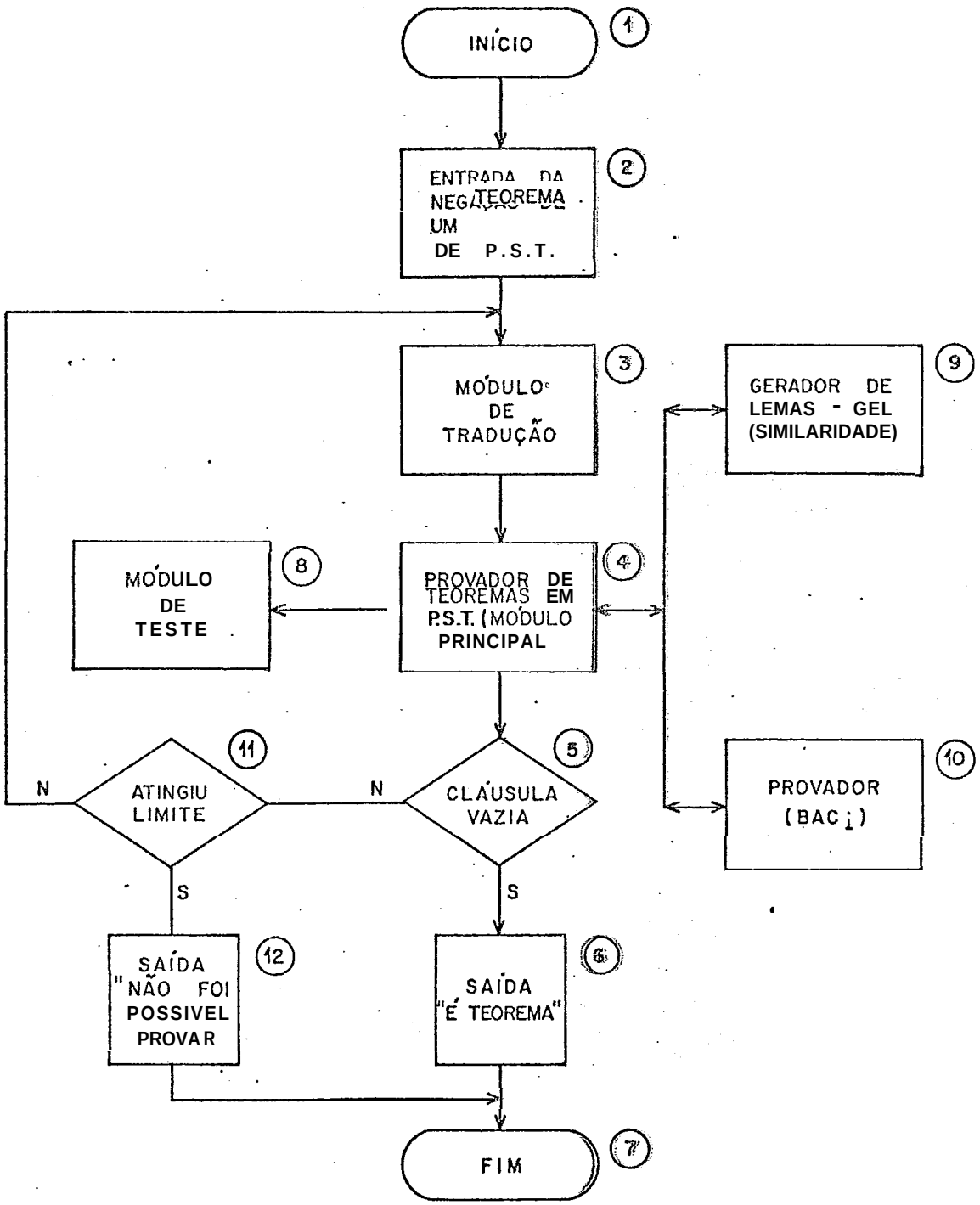


FIGURA B.1

B.4 - DESCRIÇÃO GERAL DOS MÓDULOS QUE COMPÕEM O PPST

1 - Módulo INICIO - contém uma rotina que imprimirá as instruções resumidas de como usar o PPST e opcionalmcnte uma explicação resumida dc como funciona o provndor. O PPST já contém todos os axiomas da PST assim como alguns lemas LANZELOTE[32] prontos para auxiliarem na prova.

2 - Rotina que recebe do usuário as cláusulas que definem a negação do teorema a ser provado.

3 - Módulo de Tradução - Este módulo fará a tradução do conjunto de cláusulas básicas. As rotinas que compõem o Módulo de Tradução são 12(TTOP, TNBH, TCLOSED, TLIM, TINT, TDER, TCLOS, TRD, TDENSE, TPERF, TSEP, TCONCT) e independentes uma da outra, podendo ser chamadas cada uma separadamente.

A 1ª. tradução será para um nível imediatamente superior, isto é, se CONCT(X,Y,Z) é o que estamos traduzindo (conj. Y é ligado ou Z não é a união de 2 conjuntos separados) traduziremos por C.32, C.33 e C.34 (veja Apêndice A) que usa somente o fato de ser

$$\neg SEP(X,Y,Z,W) \vee \neg Y = (Z \cup W) \quad e$$
$$SEP(X,Y,h_6(X,Y,Z) \cup \neg h_7(X,Y,Z))$$

3 - Módulo de Tradução - continuação - figura B.2

e tentaremos provar nesse nível $\neg SEP(...)$
$SEP(...)$

e conseguirmos OK! caso contrário traduziremos SEP para o nível imediatamente superior que é CLOS (C14, C15 e C16) conforme gráfico a seguir, e tentaremos com o nome LIM (C.14, C.15, C.16) provar; se conseguirmos OK! Caso contrário vamos traduzir LIM por NBH (C1, C2 e C3) e novamente tentamos a prova; assim sucessivamente com os outros conceitos, de acordo com a árvore das traduções da página seguinte, até chegar-se a $(BAC)_i$ que tentar-se-á também chegar a cláusula vazia através do provador do Apêndice A.

B.4

GRAFO DE TRADUÇÕES POR NÍVEL

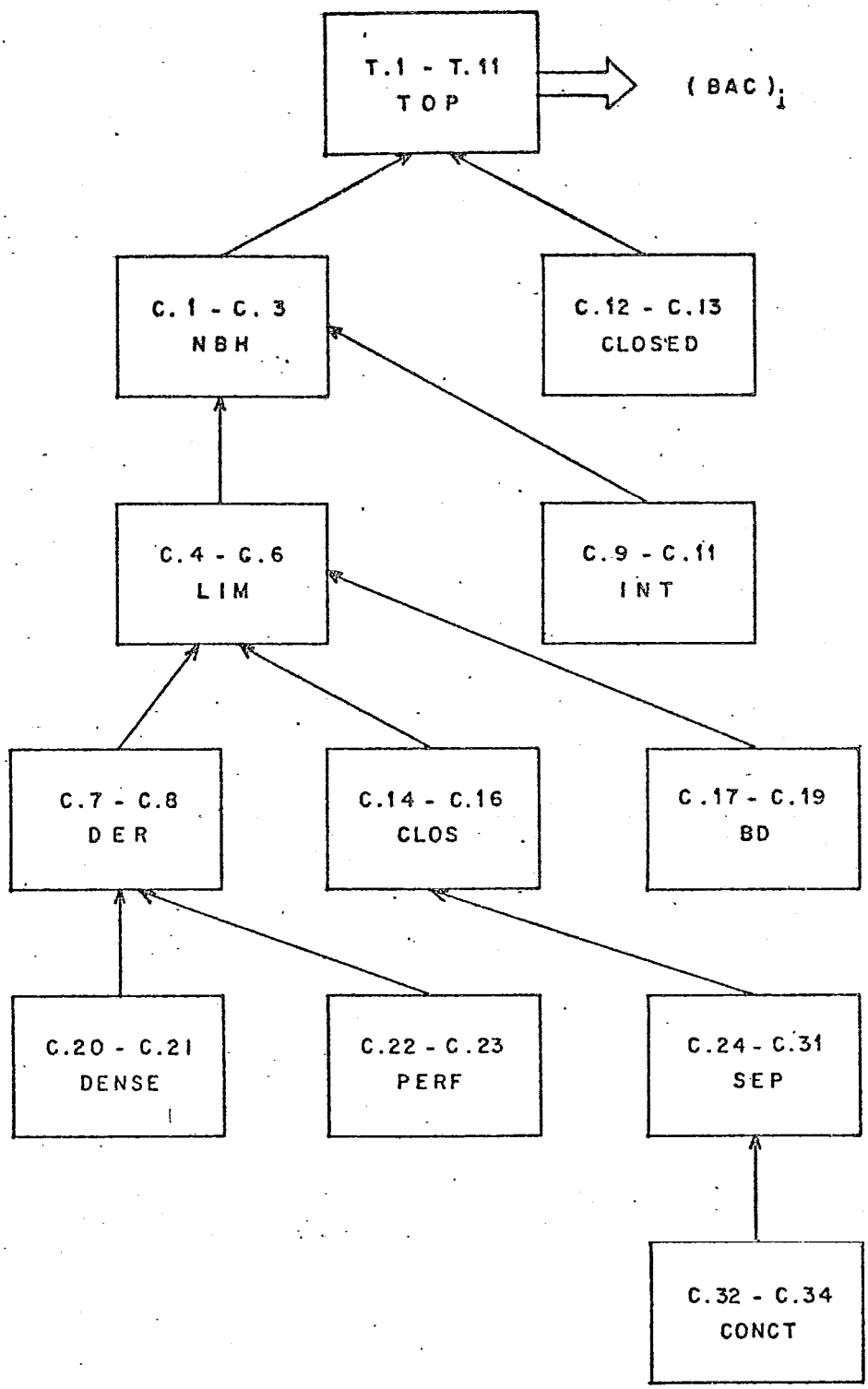


FIGURA B.2

3 - Módulo de Tradução - continuação. Exemplo de aplicação.

OBS: A estratégia usada pelo PPST em resumo é: (a) traduzir os símbolos funcionais e predicativos presentes no sistema (negação da fórmula dada) para o símbolo funcional c predicativo dc um "caminho" (profundidade na árvore de tradução) menor, os quais têm algumas ocorrências presentes no sistema no momento do início da redução. (b) não executar o passo (a) antes que todos os resolventes possíveis forem obtidos.

Por exemplo, provar que: "Se Y é uma topologia para X e. $Z \subseteq W$ então o conjunto de todos os pontos limites de Z está contido no conjunto de todos os pontos limites de W".

$\forall X\ \forall Y\ \forall Z\ \forall W\ ((T(X,Y) \wedge Z \subset W) \rightarrow DER(X,Y,Z) \subset DER(X,Y,W)$

$(p \rightarrow q \Leftrightarrow \neg p \vee q;\ \neg(\neg p \vee q);\ p \wedge \neg q)$

1) $T(X,Y)$
2) $Z \subseteq W$
3) $\neg DER(X,Y,Z) \subseteq DER(X,Y,W)$
4) $\neg X \in Z \vee X \in W$ de (2) e porque $\{X \subseteq Y\} = \{\neg Z \in X \vee Z \in Y$, Z nova variável.)

Traduzindo 3 com a definição de DER e LIM, tem-se as seis cláusulas abaixo:

5) $LIM(X,Y,Z,x)$ (1,3, traduzindo DER 1 nível)
6) $\neg NBH(X,Y,V,x) \vee f_1(g(V),\emptyset) \in V$ (traduzindo LIM)
7) $\neg NBH(X,Y,V,x) \vee \neg f_1(g(V),\emptyset) = x$
8) $\neg NBH(X,Y,V,x) \vee f_1(g(V),\emptyset) \in Z$ (por 7,3)
9) $\neg x' \in h_4(W,x),x) \vee x' = x \vee \neg x' \in W$
10) $NBM(h_4(W,x),x)$

onde $x = f_1(DER(X,Y,Z),DER(X,Y,W)$ e $g(V) = (V-\{x\} \cap Z)$

11) $\neg NBH(h_4(W,x),x)$ por (6,7,8,9)
12) $\square$ (10,11)

4 - Módulo PRINCIPAL .

(Veja fluxo na página seguinte)

É composto de submódulos especiais:

FLUXO DO MÓDULO PRINCIPAL

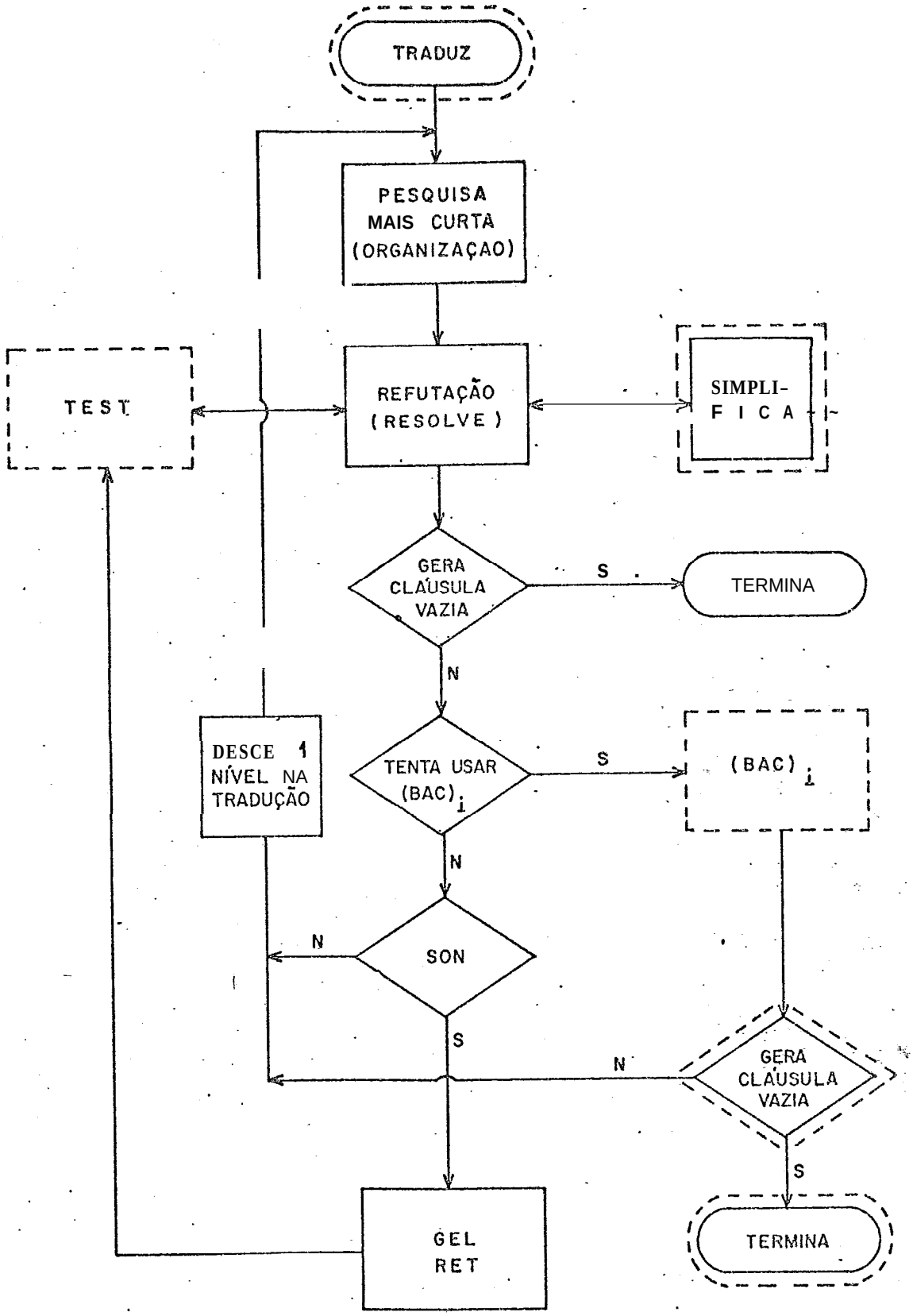


FIGURA B.3

i) de Pesquisa - que é a organizadora da lista de cláusulas que entrarão na Resolução. Contém um algorítmo para estratégia da cláusula mais curta. (compatível com P-refutação).

ii) de Refutação - que tentará gerar a cláusula vazia. Envolve a estratégia de P-refutação, princípio da resolução. Terá como entrada as cláusulas básicas enviadas pelo módulo anterior. Se conseguir gerar a cláusula vazia termina. Sempre interagindo com o módulo de simplificação. Caso contrário tenta usar o provador $(BAC)_i$ e gerar cláusula vazia. Se não conseguir usar o provador $(BAC)_i$ isto é $(BAC)_i$ não gerou a cláusula vazia, volta para traduzir um nível a mais ou opcionalmente interrompe a prova automática e passa o controle a SON que dirigirá a prova, pedindo algum lema a GEL ou pedindo alguma relação entre espaços topológicos a GEL.

iii) SON - rotina que permite ação do usuário para auxiliar a prova através de lemas vindos de GEL e de definições ou axiomas.

5 - TESTE DA CLÁUSULA VAZIA

6 - IMPRESSÃO DA RESPOSTA

7 - FIM - pela impressão da resposta ou pela impressão de "estouro de memória" ou de "estouro de tempo".

8 - MÓDULO DE TEST - rotina que verificará se incorporaremos a cláusula gerada no sistema ou não e se elimina tautologia (subassumption de 1 só tipo - só se é pura).

9 - MÓDULO GEL - gerador de lemas - contém uma rotina que por SIMILARIDADE descobre LEMAS que introduzidos após a tradução (de 1 nível) facilita a refutação. Por exemplo:

Para provarmos que
"Se um conjunto ligado está contido na união de conjun-.tos separados então ele deve estar contido em cada um dos dois conjuntos".

Em PST

$\forall X \forall Y \forall Z$ ((SEP(X,Y,Z,W) $\cup$ CONCT(X,Y,V) $\cup$ V $\sqsubset$ (Z $\cup$ W)) (V $\cup$ Z $\cup$ V $\subseteq$ W)).

1) $SEP(A_1,A_2,A_3,A_4)$
2) $CONCT(A_3,A_4,A_5)$
3) $A_5 \subseteq (A_3 \quad A_4)$
4) $A_5 \subseteq A_3$
5) $A_5 \subseteq A_4$

} negação da fórmula dada

Não se conseguiu refutar o conjunto de cláusulas acima. Mas usando nossa metodologia (de traduzir um nível de cada vez e gerando de 3) ($C \subseteq A \cup B$) o lema $C = (A \cap C)(B \cap C)$ através da rotina GEL. Tem-se:

1) $SEP[A_1,A_2,A_3,A_4]$ (igual a 1)
2) $SEP[A_1,A_2,A_3,A_4] \vee \neg A_5 = A_3 \cup A_4$ (traduz 1 nível CONCT)
3) $A_5 = (A_3 \cap A_5) \cup (A_4 \cap A_5)$ (lema gerado por GEL (por similaridade com (3) anterior)
4) $\neg A_5 \subseteq A_3$ (igual a (4))
5) $\neg A_5 \subseteq A_4$ (igual a (5))
6) $SEP(A_1,A_2,A_3 \cap A_5, A_4 \cap A_5)$ (2,3)

que agora consegue-se provar,

10 - PROVADOR. $(BAC)_i$

É o existente em LINS[25], PASSOS[27], PEIXOTO[31] implementado em SNOBOL e prova teoremas para extensões da Álgebra Booleana de Classes $(BAC)_i$, veja Fluxograma Geral do provador ítem A, 2, do Apêndice A.

B.5 - O PPST

Apesar do melhor comportamento deste provador, ficou patente a inflexibilidade do mesmo quando teve-se que provar teoremas mais complexos. Somente no caso em que consegue-se gerar a cláusula vazia sem a tradução para $(BAC)_i$, pois não há ainda mecanismos de ligação para a entrada automática de cláusulas geradas pela tradução da topologia, usando as novas estratégias, e o provador para $(BAC)_i$.

Toda essa idéia foi incorporada no desenho do novo sistema gerador de provadores que apresentamos no Capítulo III.

APÊNDICE C

ESPECIFICAÇÃO DE UMA LINGUAGEM PARA GERAR DEMONSTRADORES AUTOMÁTICOS DE TEOREMAS - PROTOTIPO DA IMPLEMENTAÇÃO DO SGPT

C.1 - INTRODUÇÃO

O nosso sistema, conforme descrito no Capítulo III tem basicamente dois grandes programas. Um a nível do usuário, que chamamos programa gerador onde., por intermédio de uma linguagem de consulta, é possível comunicar-se com o programa. E outro'programa, provador automático de teoremas, para a teoria declarada no primeiro programa, que é o programa gerado.

Está sendo implementado no IBM/370 modelo 165 da PUC/RJ, por razões históricas. Deverá ser copiado para o B/6800 do IME.

A linguagem de consulta foi desenvolvida, em protótipo, para atender mais amplamente às necessidades do usuário. Esta linguagem denominada LIGP - Linguagem Interativa para Geração de Provadores será descrita a seguir.

C.2 - LIGP - LINGUAGEM INTERATIVA PARA GERAÇÃO DE PROVADORES

O programa que implementa o LIGP é escrito em SPITBOL, e dua chamada é feita atualmente via PHOENIX (Sistema Interativo existente no IBM-370 da PUC/RJ), por intermédio da execução do procedimento

PBTX. TORPIN. PROC: LIGP

que torna disponível todos os arquivos necessários antes de executar o programa

PBTX. TORPIN. LOAD:, LIGP

C. 2.1 - Comandos de Conversação

a - EXPLICAR
b - AJUDAR
c - SUGERIR
d - TERMINAR
e - ARQUIVOS
f - HISTÓRIA

C. 2.2 - Comandos de Definição de Parâmetros
Declarações do usuário

a - CONSTANTES
b - VARIÁVEIS
c - FUNÇÕES
d - PREDICADOS
e - Definições (ou Axiomas)

C. 2.3 - Comandos de Manipulação de Arquivos

a - Listar
b - Incluir
c - Retirar
d - Gerar
e - Limpar
f - Executar (executa o programa gérado)

O LIGP permite a abreviação do nome do comando até a primeira letra. No caso de existir coincidência, o sistema recusa o comando e o solicita por extenso.

C.2.4 - Descrição dos Comandos dc Conversação

Três eventos básicos podem ocorrer durante uma ses-

são do SGPT, além daqueles de uso normal do sistema:

1 - O usuário deseja terminar a sessão

2 - O usuário precisa de uma ajuda local a respeito da sintaxe de um comando

3 - Houve "queda" do sistema operacional e o usuário deseja recuperar o que já havia sido feito antes da "queda".

Para atender ao evento 1,- bastará o usuário entrar como comando TERMINAR.

Para atender ao evento 2 o SGPT conservará sempre ativa (em uma sessão) uma'biblioteca de. documentação· de cada c'omando.

Para atender ao evento 3, o SGPT copiará sempre , qualquer linha escrita pelo usuário, em arquivo permanente. Através do comando RECUPERAR, o usuário indica ao SGPT, para esta execução, tudo o que já havia sido feito (inclusive todas as inicializações serão feitas e o arquivo HISTÓRIA será usado como ENTRADA).

C.2.4.1 - EXPLICAR

Sintaxe : EXPLICAR tópico

Semântica: Lista uma ou mais telas com o texto explicativo de um determinado TOPICO. Este texto será armazenado no arquivo de documentação da LIGP.

Exemplo : EXPLICAR comando

Resposta :

comando : qualquer comando. da LIGP obedece as normas de abreviação.
Será listado um resumo da descrição do comando.

C.2.4.2 - AJUDAR

.Sintaxe : AJUDAR

Semântica: Lista todos os comandos cxistentes na **LIGP**

C. 2.4.3 - SUGERIR

Sintaxe : SUGERIR texto

Semântica: Cópia texto escrito pelo usuário num arquivo de sugestões

A finalidade é permitir uma maior participação do usuário na melhoria do sistema. em geral (SISTEMA).

O usuário do SISTEMA poderá, com esse comando, pro por novos comandos ou criticar algum? pqrte do SISTEMA.

Exemplo: SUGERIR texto

A forma de introduzir um texto é a seguinte: após digitar a palavra SUGERIR e teclar "ENTER", o sistema estará funcionando no modo INSERÇAO, que lerá cada linha subsequente da tela para o arquivo de sugcstõcs. Para voltar ao modo normal o usuário deverá digitar * FIM. .

C.2.4.4 - TERMINAR

Sintaxe : TERMINAR

Semântica: Termina a sessão, na qual estava geran do um provador, voltando ao sistema PHOENIX.

C.2.4.5 - ARQUIVOS .

Sintaxe : ARQUIVOS Alocados?

semântica: Lista uma tela mostrando as associa-

ções dos nomes lógicos (dduames) usados pela LIGP e os nomes dos arquivos físicos alocados.

A finalidade é fornecer informações sobre os arquivos que podem ser usados em comandos de manipulação de arquivos, de forma a ajudar o usuário quando este recebe uma mensagem de erro "arquivo não alocado".

C.2.4.6 - HISTÓRIA

Sintaxe : HISTÓRIA

Semântica: Listar todos os comandos que já foram executados até a momento na presente sessão de LIGP.

A finalidade é proporcionar ao usuário ajuda. no caso de "parada do computador". Com isso será mais fácil retornar a execução da LIGP, recuperando atualizações.

A cada início de sessão do SISTEMA é gravada uma linha no arquivo HISTÓRIA, contendo:

* INÍCIO DATA = data HORA = hora

A cada fim (normal) de sessão do SISTEMA é gravada a linha

* FIM DATA = data HORA = hora

C.2.5 - Descrição dos Comandos de Definição de Parâmetros

São comandos que devem ser usados para declarar os símbolos da linguagem de 1ª ordem que definirá a Teoria.

C. 2.5.1 - CONSTANTES

Sintaxe : CONSTANTES lista de constantes

Semântica: A lista de constantes espccificnda será transformada em um "pattern" e este, gravado no arquivo "constantes" (C).

Na primeira utilização a linha gerada no arquivo será:

PATC = 'const1' / 'const2'

Na segunda, ou mais vezes, a linha gerada será:

+ 'constn' / 'constn + 1'

O arquivo será 'esvaziado' toda vez que mudarmos de teoria.

Os separadores entre o nome CONSTANTE e a lista de constantes serão branco (S) ou (:) (dois pontos) seguidos ou não por brancos.

A lista de constantes é um string não nulo que contém constantes separadas por uma (,) ou (/) seguido (ou não) de branco (S). Cada constante pode começar por uma letra ou um dígito, seguido por um número qualquer de letras ou dígitos.

C.2.5.2 - VARIÁVEIS

Sintaxe : VARIÁVEIS lista de variáveis

semântica: A lista de variáveis especificada será transformada em um "pattern" e este, gravado no arquivo "variáveis" (V).

A linha gerada é:

PATV = 'var1' | 'var2' ['ou
+ 'varn' | 'varnt1'| de forma idêntica ao arquivo C.

Diferentemente das constantes, as variáveis só podem começar por letra, e em seguida letras ou dígitos.

C.2.5.3 - FUNÇÕES

Sintaxe : FUNÇÕES lista de-funções

Semântica: Os nomes de funções declarados são usados para gerar o "pattern" PATF.

A seguir uma tabela de aridade é adicionada ao arquivo de funções (F).

A lista de funções é um string não nulo que contém os pares nome-aridade separados por (,) ou branco (S).

Os pares devem ser separados por (:) ou branco (S).

Os nomes devem começar por letra e em seguida letras ou dígitos e a lista de argumentos (constantes e variáveis) deve ser colocada entre parênteses. Aridade só pode conter dígitos.

C.2.5.4 - PREDICADOS

Sintaxe : PREDICADOS lista de predicados

semântica: Os nomes dos predicados declarados são usados para gerar o "pattern" PATP, da mesma forma que os anteriores, e este será gravado no arquivo dos "predicados" (P).

C. 2.5.5 - DEFINIÇÕES

Sintaxe : DEFINIÇAO descrição

Semântica: Os nomes das definições declaradas são usados para gerar o "pattern" PATD, da mesma forma que os anteriores, e este será gravado no arquivo das "definições" (D).

C.2.6- Descrição dos comandos de Manipulação de Arquivos

Incluir A em B antes de C
depois de C
no início do arquivo
no .fim do arquivo

Retirar A e B

Destruir' A → O arquivo.é reinicializado (contendo ϕ registros)

Imprimir A → O arquivo A é inserido na JCL que listará na impressora após o. término da sessão SGPT

Listar A → Uma cópia do arquivo A é listada no vídeo do usuário.

C.3 - Comandos de EXECUÇAO DO PROVADOR

Uma vez fornecidos os símbolos (da linguagem de 1.ª ordem), axiomas, teoremas e definições da teoria, o usuário deve usar o comando EXECUÇAO DO' PROVADOR para obter:

(i) A demonstração formal de seus teoremas ou
(ii) A geração dc novos teoremas.

A ação do SGPT será montar os módulos (necessários) do programa provndor da seguinte forma:

- incluir as Tabelas da Teoria (arquivo gerado pelo Analisador)
- incluir o programa Compilador (que gera grafos a partir dèssa teoria)
- incluir as rotinas do PROVADOR que serão usadas para executar uma das tarefas (i) e (ii) . que o usuário selecionou, através de uma estratégia.

Uma vez montado o programa gerado (PROVADOR) o SGPT invocará sua execução.

Como o PROVADOR é também conversacional, deverá haver maneiras do usuário interromper sua execução e retornar ao LIGP para definir novas tarefas.

C.4 - ALGORÍTMOS DETALHADOS PARA A IMPLEMENTAÇAO

C.4.1 - Criação da Estrutura Definicional

O algorítmo que cria a estrutura definicional de uma teoria genérica é conceitualmente simples:

. Consiste em achar, para cada símbolo s da teoria T, todos os símbolos em função dos quais s está definido a guardar este relacionamento em uma tabela para posterior consulta.

Algorítmo PAI:= ϕ
Para cada símbolo s de T
Para cada cláusula C que define s
Para cada símbolo s' $\neq$ s que ocorre em C
PAI:= PAI {(s',s)} .

Sendo S o conjunto de símbolos de T
PAI= {(x,y)/x,y $\in$ S e x$\neq$y e x ocorre na definição de y}

Podemos visualizar esta seleção PAI como um grafo que nos mostra a "hierarquia" dos símbolos da teoria. O par (x,y) seria representado por:

Assim, os símbolos fonte (de onde só saem flexas) seriam os primitivos da teoria.

Para facilitar a tradução por nível torna-se necessário saber a "profundidade" definicional de cada símbolo da teoria.

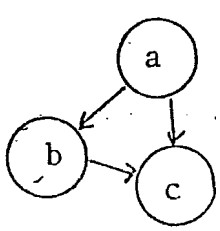
Novamente, o algorítmo é simples:

Para cad-a símbolo s de T
se s não "tem pai" então PROF (s):= ϕ
senão PROF (s):= PROF (PAI(s)) + 1

Onde "ter pai" significa "ser 2º termos de algum par ordenado de PAI"

As dificuldades surgem do fato que, como regra geral, o pai de um símbolo não é único.

Examinando-se o grafo abaixo chegamos a dois valores diferentes para a profundidade de C:



Tomando-se o par (a,c) a profundidade de C é 1. Por outro lado, usando o par (b,c) obtemos PROF *(c)*.= 2.

Essa ambiguidade é eliminada definindo-se a profundidade de um símbolo como sendo a maior profundidade obtida analisando-se todos os caminhos possíveis até um símbolo fonte.

No caso acima, PROF (c) = 2.

Outro problema que se apresenta (e que esse algorítmo simplificado não resolva) é o caso das definições desordenadas.

No grafo anterior, corresponderia a dar a definição de "b" antes de definir "a". No cálculo da profundidade de "b" teríamos:

PROF (b) := PROF (a) + 1
mas PROF (a) ainda não foi calculada.

Estes detalhes complicam sobremaneira a realização prática do algorítmo, por isso, apresentamos uma versão simplificada do mesmo apenas para ilustrar o texto.

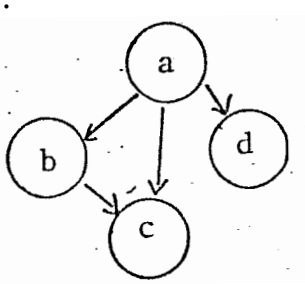
Uma outra estrutura de dados auxiliar é criada invertendo-se a tabela PROF. Cria-se, assim, o vetor NÍVEL onde teremos, para cada profundidade, uma lista dos símbolos que estão no mesmo nível.

Algorítmo

Para cada símbolo s de T
NÍVEL (PROF (s)) := NÍVEL (PROF (s))//","// s

Exemplo:

Seja a seguinte estrutura definicional representada pelo grafo abaixo:



PAI = {(a,b), (a,c), (a,d), (b,c)}
PROF (a) = ϕ . NÍVEL ( ) = "a"
PROF (b) = 1 NÍVEL (i) = "b;d"
PROF (c) = 2 NÍVEL (z) = "c"
PROF (d) = 1

C.4.2 - Tradução por Nfvel

A Tradução por Nível é uma das muitas estratégias de prova de teoremas. Pode ser assim sintetizada:

Dado um conjunto C de cláusulas de uma teoria T queremos "eliminar" todos os símbolos de maior profundidade que ocorrem em C. Essa "eliminação" é feita traduzindo-se cada ocorrência do símbolo pela sua definição.

1º Passo - Cálculo do nível mais profundo

MAX:= ϕ
Para cada cláusula c de C
Para cada símbolo s de c
MAX:= 'máximo (MAX,PROF (s))

2º Passo - Obtenção dos símbolos que serão traduzidos

NÍVEL (MAX) fornece uma lista de todos os símbolos do nível mais profundo.

3º Passo - Tradução propriamente di'ta

Para cada cláusula c de C
Para cada símbolo s de NÍVEL (MAX)
se s ocorre em c então substituir a ocorrência de s por sua definição

Essa substituição consiste em procurar entre as cláusulas que definem s aquelas que contêm a negação de s e usar o algorítmo de resolução de cláusulas.

Por exemplo: a cláusula c é $\neg p \vee \neg q \vee r$
e na definição de p temos:
C. $\neg p \vee a \vee b$ e
C_z. $p \vee d \vee f$

A cláusula resultante da resolução de C_1 e C_2 será:

C_3. $d \vee f \vee \neg q \vee r$

Já existe um programa em SNOBOL que faz a resolução de duas cláusulas genéricas por isso não nos ocuparemos do mesmo.

C.4.3 - Exemplo

Um subconjunto da Álgebra Booleana de Classes é definido pelas cláusulas abaixo:

$\neg x \in \phi$

C_2 $x \in v$

C_3 $\neg \chi \subseteq y \vee \neg x \in \chi \vee x \in y$

C_4 $\chi \subseteq Y \vee f_1(\chi,y) \in \chi$ Definição de C

C_5 $\chi \subseteq y \vee f_1(\chi,y) \in y$

C_6 $\neg \chi = y \vee \chi \subseteq y$

C_7 $\neg \chi = y \vee y \subseteq \chi$ Definição de =

C_8 $\chi = y \vee \neg \chi \subseteq y \vee \neg y \subseteq \chi$

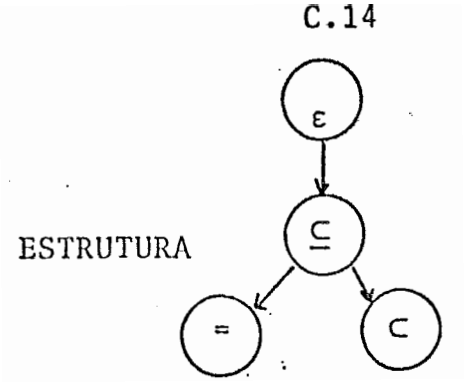
C_9 $\neg \chi \subset y \vee \chi \subseteq y$

C_{10} $\chi \subset y \vee \neg y \subseteq \chi$ Definição de C

C_{11} $\chi \subset y \vee \neg \chi \subseteq y \vee y \subseteq \chi$

PAI = $\{(\in,\subseteq), (\subseteq,=), (\subseteq,\subset)\}$

C.14



ESTRUTURA **DEFINICIONAL**

PROF (E) = ϕ .
PROF ($\subset$) = 1
PROF (=) = 2
PROF ($\subset$) = 2

NIVEL (ϕ) = "ε"
NIVEL (1) = "c"
NIVEL (2) = "=;c"

Suponhamos. que desejamos provar que:

$X \subset y \rightarrow X = y$ é teorema nesta teoria

Reescrevendo nosso c^i teorema ? em forma de cláusula:

$\neg X \subset y \vee \neg X = y$

Adicionamos agora, ao conjunto de 11 cláusulas que definem a teoria, a negação da tese que yretendcmos provar:

C_{12} . $X \subset Y$.
C_{13} . $X = Y$.

Seja $C = \{C_{12}, C_{13}\}$

.Façamos a tradução por nível:

1º Passo

MAX:= 2

2º Passo

NIVEL (MAX)= "=;c"

3º Passo ('detalhado)

C = C_{12}

s = "="

s não ocorre em C_{12}

s = "C"

s ocorre em C_{12}

Substituição

cláusulas que definem C: C_9, C_{10}, C_{11}

C_9 . $\neg X \subset Y \vee X \subseteq Y$

C_{10} . $\neg X \subset Y \vee \neg Y \subseteq X$

C_{11} . $X \subset Y \vee \neg X \subseteq Y \vee Y \subseteq X$

Usaremos as cláusulas C_9 e C_{10} onde C aparece negado.

Resolução de C_{12} e C_9 produz C_{14} . $X \subseteq Y$

Resolução de C_{12} e C_{10} produz C_{15} . $\neg Y \subseteq X$

C = C_{13}

s = "="

s ocorre em C_{13}

Substituição

cláusulas que definem = : C_6, C_7 e C_8

C_6 . $\neg X = Y \vee X \subseteq Y$

C_7 . $\neg X = Y \vee Y \subseteq X$

C_8 . $X = Y \vee \neg X \subseteq Y \vee \neg Y \subseteq X$

Usaremos as cláusulas C_6 e C_7 onde = aparece negado.

Kesolução de C_{13} e C_6 produz C_{16} . $X \subset Y$

Resolução de C_{13} e C_7 produz C_{17} . $Y \subset X$

s = "C"

s não ocorre em C_{13}

Ficamos, então, com:

C_{14} . $X \subseteq Y$

C_{15} . $\neg Y \subseteq X$

C_{16} . $X \subseteq Y$

C_{17} . $Y \subseteq X$

Torna-se evidente que a resolução de C_{15} e C_{17} produzirá a cláusula vazia sem que seja necessária a tradução até o nível "E".

Logo: $X \subset Y + \neg X = Y$ é o teorema nesta teoria.

C.4.4 - FLUXOGRAMAS DOS ALGORITMOS

C.4.4.1 - PROVADOR

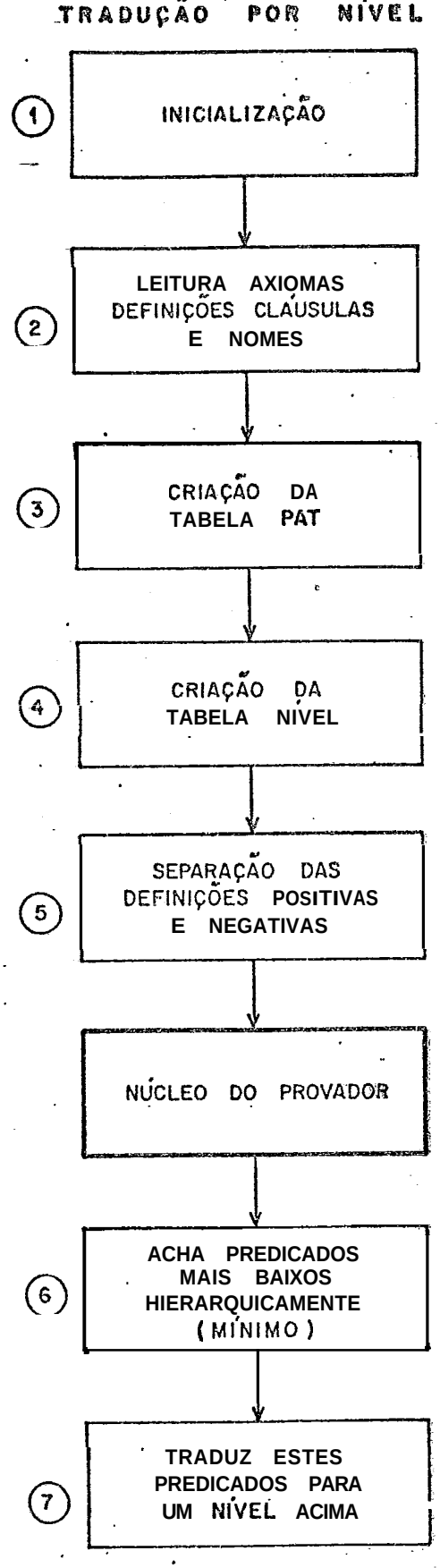


FIGURA C.1

C.4.4.2 - LEITURA DOS AXIOMAS E DEFINIÇÕES

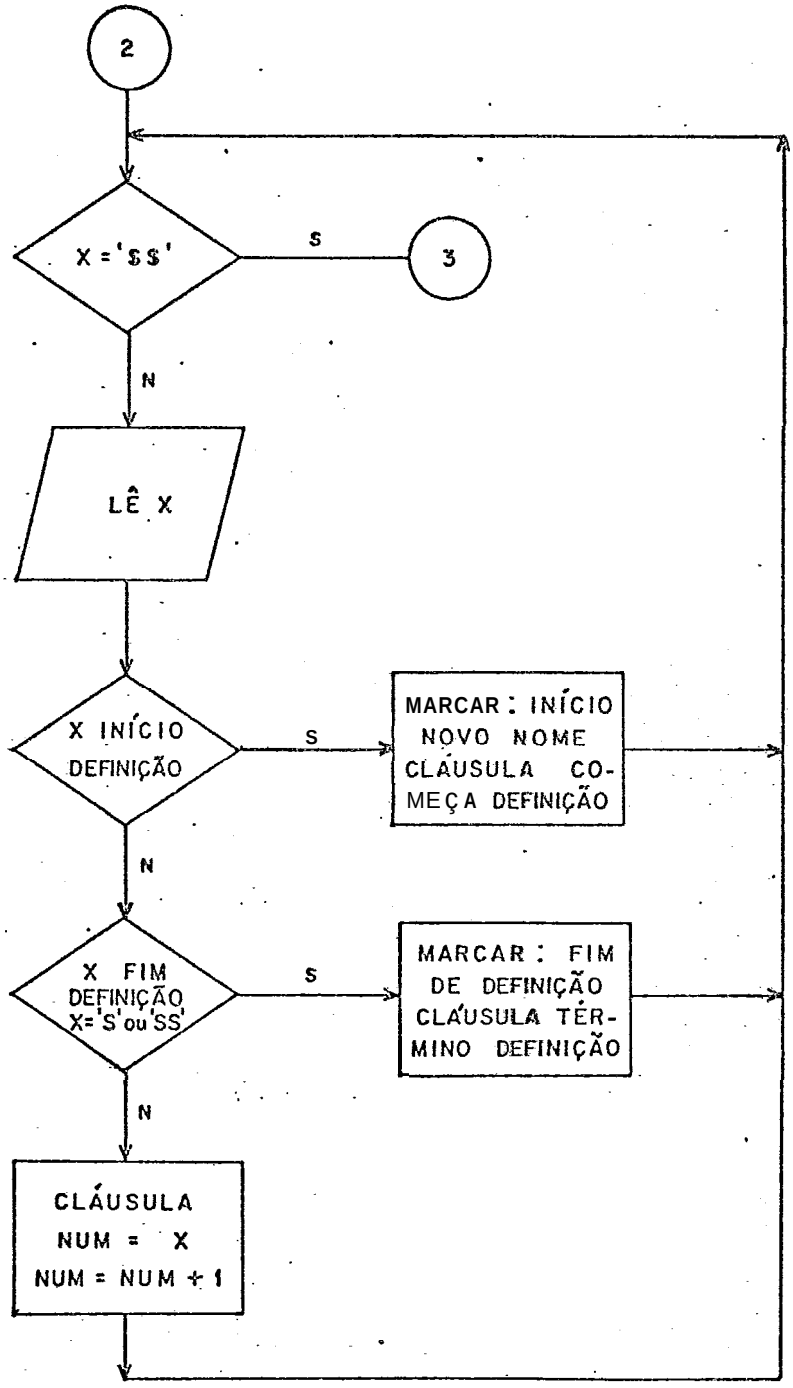


FIGURA C.2

C.4.4.3 - CRIAÇÃO DA TABELA PAT

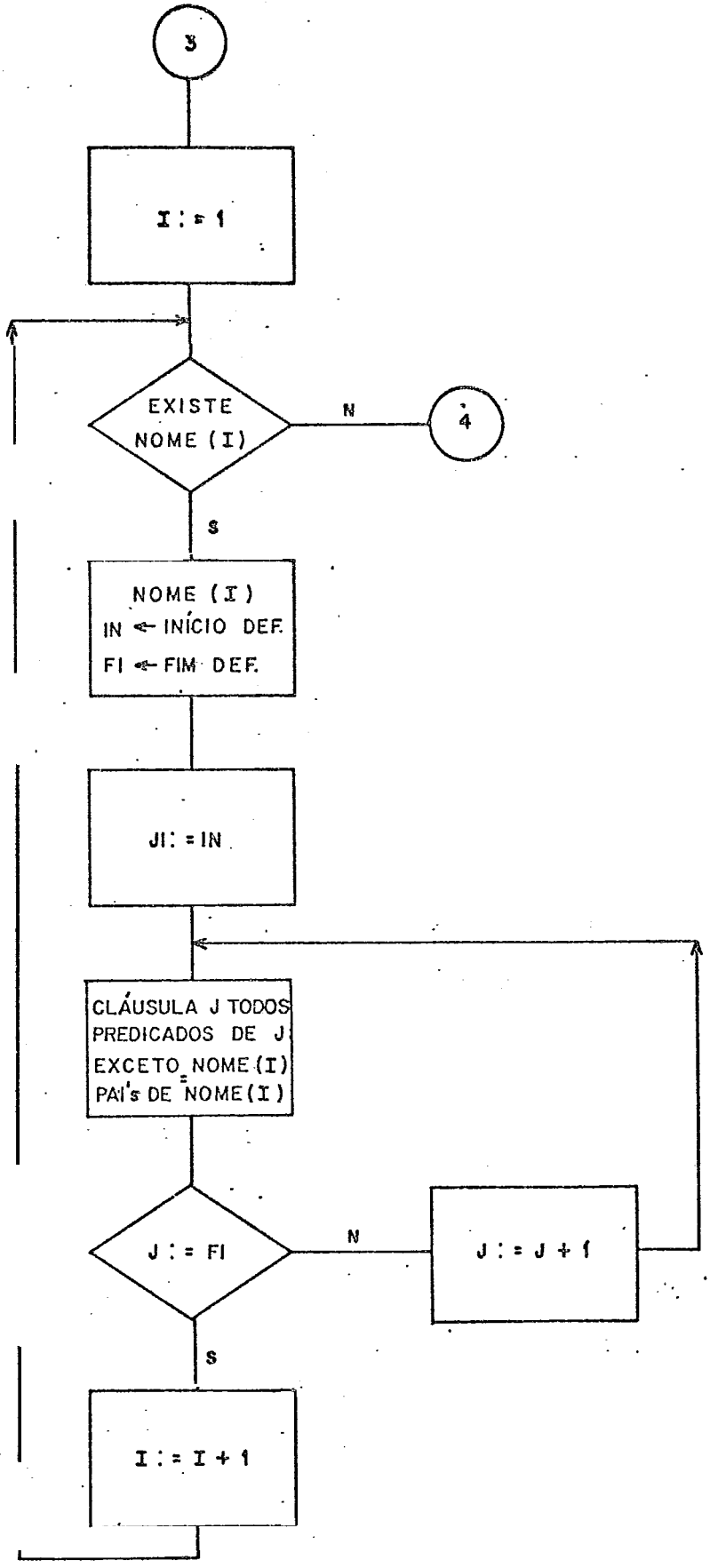


FIGURA C.3

C.4.4.4. - CRIAÇÃO DA TABELA DE NÍVEL

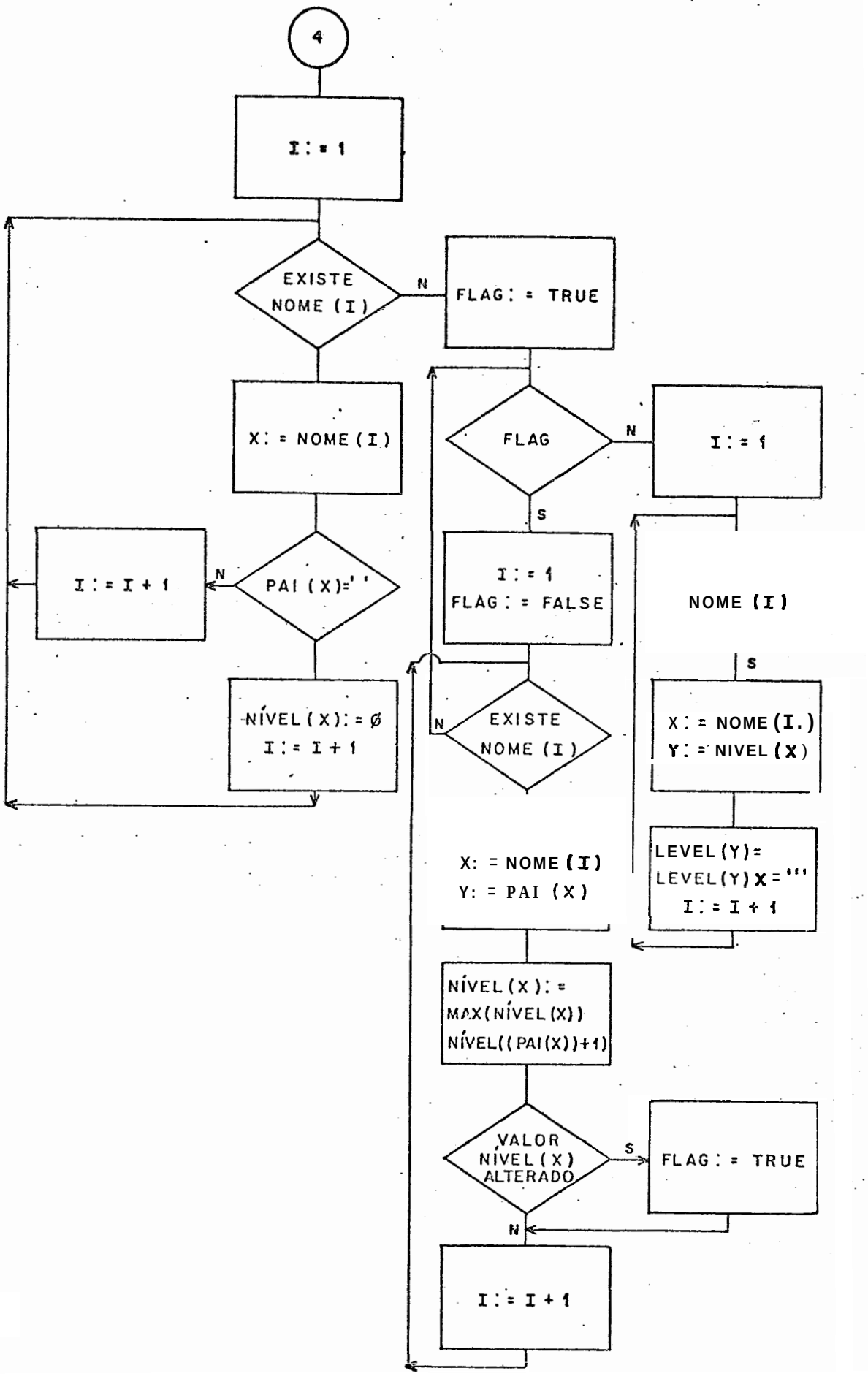


FIGURA C.4

C.4.4.5 - SEPARAÇÃO DAS DEFINIÇÕES POSITIVAS E NEGATIVAS

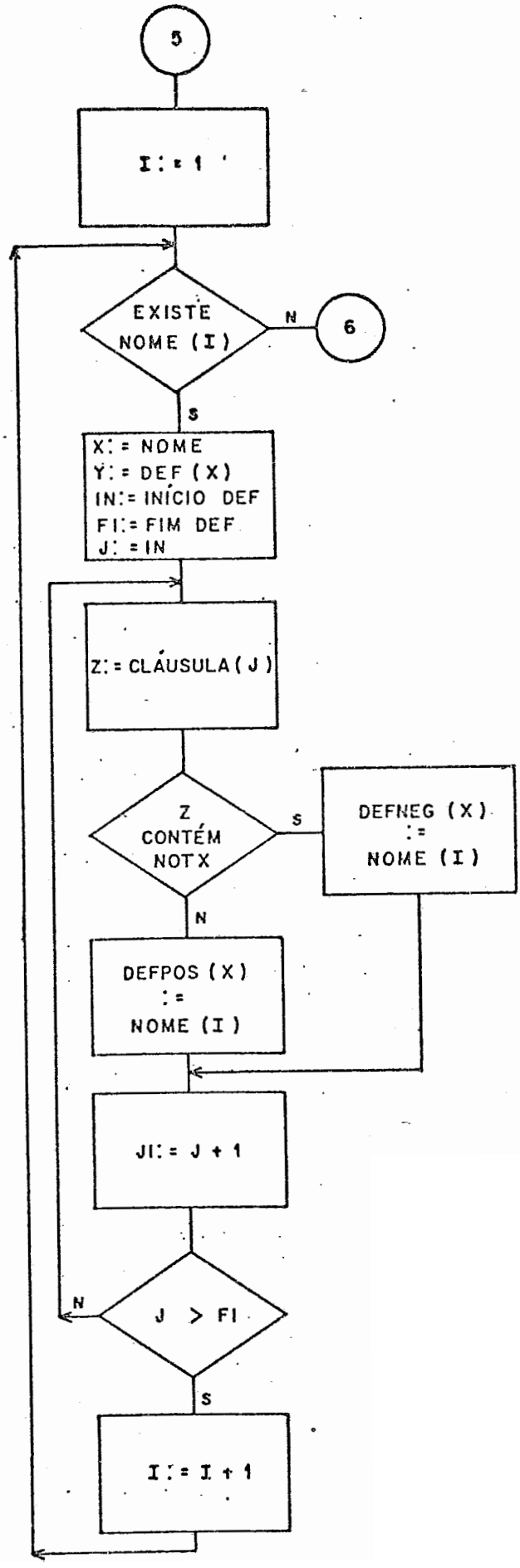


FIGURA C.5

C.4.4.6 - MÍNIMO

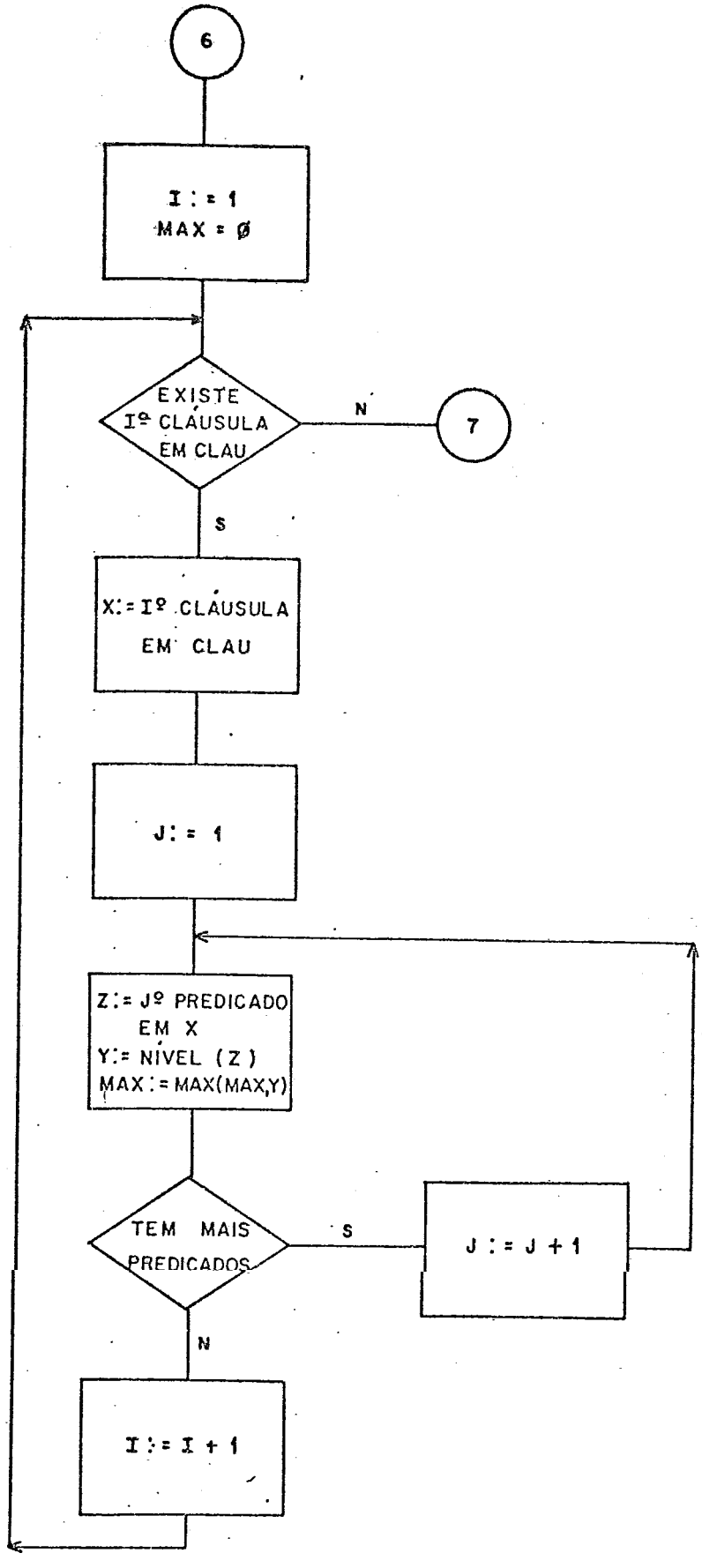


FIGURA C.6

C.4.4.7 - TRADUZ OS PREDICADOS PARA UM NÍVEL ACIMA

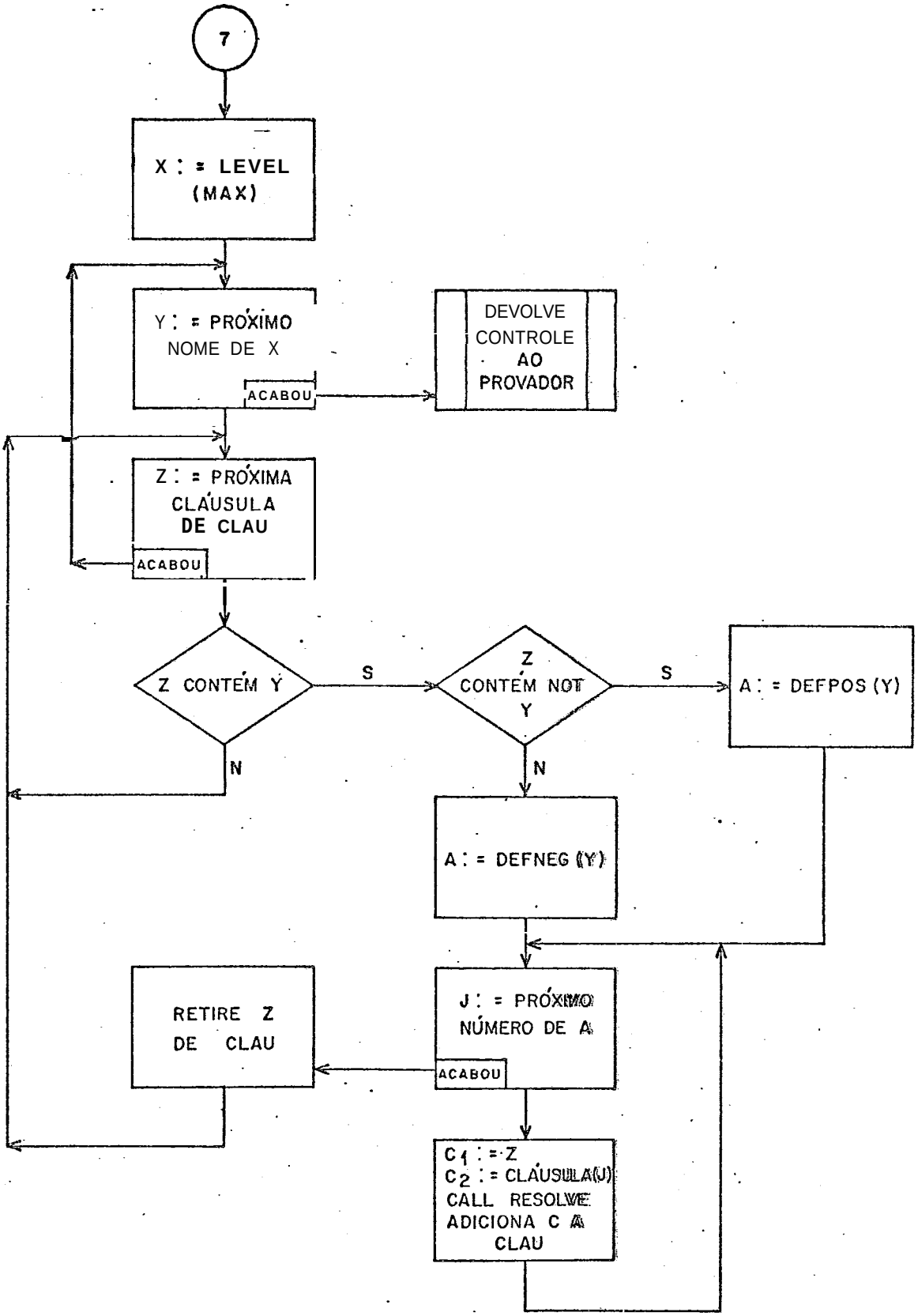


FIGURA C.7

APÊNDICE D

RELAÇÕES ENTRE ESPAÇOS TOPOLÓGICOS
(Sistema Manipulador de Teorias Definicionais)

É um sistema conversacional, implementado em SPITBOL, que utiliza uma "query-language" especialmente projetada para a classificação de espaços topológicos LANZELOTE[32].

Foi criado um "data-base" com as definições de propriedade de espaços topológicos. Foi também criada uma estrutura para interrelacionar essas propriedades..

Na topologia pode-se caracterizar seis grupos principais de propriedades de espaços. O número de propriedades consideradas fundamentais ultrapassa sessenta e pode-se estabelecer mais de uma centena de relações entre essas propriedades. Como exemplo, consideremos as propriedades de separação tem-se os axiomas de separação Ti, $0 \leq i \leq 6$ que obedecem as seguintes relações

T2 → T1 → T0
T5 → T4
(T4 e T2) → T3

Além disso, novas propriedades são definidas a partir dos axiomas de separação, tais como:

Regular ↔ T0 e T3
Normal ↔ T1 e T4
Completamente normal ↔ T1 e T5

Até aqui estamos vendo propriedades de um mesmo grupo. Elas também podem se relacionar com propriedades de outros grupos. A rede geral englobando todas as relações entre propriedades será apresentada no final desse ítem, bem como um resumo de um sistema que fornece resulta-

dos parciais e totais a consultas apresentadas pelo usuário. O sistema também pode deduzir propriedades como

Normal → TO
Completamente Normal → Normal

Para efeito de apresentação das propriedades de espaços topológicos, elas foram grupadas em 6 conjuntos segundo a classificação de Steen e Seebach, veja LANZELOTE[33].

Os grafos que se seguem, de 1 a 5, resumem todas as relações existentes entre propriedades do mesmo grupo e de grupos diferentes.

No grafo 1 só foram representadas as relações cujo antecedente é constituído de apenas um termo. Os demais grafos representam relações com antecedentes compostos por mais de um termo.

O RET é composto de um sistema de "question answering" que tem como objetivo fornecer respostas a perguntas de usuários, utilizando um tipo de recuperação de informação de forma a obter as respostas a partir de deduções lógicas das informações armazenadas no data-base.

A Única relação aceita pelo sistema é a relação de implicação: compõe-se de um antecedente, que pode ter um ou mais termos operados por AND's, e de um consequente.

A seguir um quadro resumo dos comandos da "query-language".

	COMANDO	OPERANDOS	PROPÓSITO
Comandos de Alteração do data-base	INSERE	(nome) (definição)	inserção de um novo "nome" no data-base
	REL	A1ε...εAn→C1,...,Cn A←→C1εC1εC2ε...εCn	inserção de uma nova relação no data-base
	MODIF	(nome) (nova definição)	modificação da definição de um "nome" anteriormente implementado
	DELETA	nome	deleção de um "nome" e de todas as informações relacionadas a .ele
	NREL	número A1ε...εAn→C1,...,Cn A ←→ C1εC2ε...ε Cn	deleção de uma relação anteriormente implementada
Comandos de Consulta	DEF	ncme	fornece a definição de "nome"
	LISTA	-	lista todos os nomes implementados
	LISTAREL	-	lista todas as relações implementadas
	PCONSEQ	nome	fornece os consequentes em primeiro nível de "nome"
	COXSEQ	nome	fornece todos os consequentes de "nome"
	PANTEC	nome	fornece os antecedentes em primeiro nível de "nome"
	ANTEC	nome	fornece todos os antecedentes de "nome"
	VERIF	A1εA2ε...εAn→C	fornece todos os caminhos que tornam a relação válida
	PESQ	nome1, nome2	fornece as relações que interligam total ou parcialmente os nomes
	ENCERRA	-	indica fim do fornecimento de comandos.

COMO UTILIZAR O RET

Podemos usar qualquer um dos 10 comandos de consulta do RET.

Alguns exemplos: 1) o comando PCONSEQ nome

Este comando fornece ao usuário uma lista de relações envolvendo um nome e todos os seus consequentes em primeiro nível. Listará inclusive as relações em que "nome" aparece como um dos termos do antecedente.

Aplicação: PCONSEQ REGULAR
causará a seguinte resposta,

(Grafo 1) REGULAR → T3, SEMIREGULAR, T2.1/2
(Grafo 3) KEGULAR'E CONTAVELMENTE COMPACTO → SEGUNDA CATEGORIA
(Grafo 3) REGULAR E SEGUNDO CONTAVEL → METRIZAVEL
(Grafo 3) REGULAR E TEM SIGMA-BASE LOCALMENTE FINITA → METRIZAVEL

: 2). o comando CONSEQ nome

A forma é a mesma do PCONSEQ, só que o comando CONSEQ produz uma lista de relações que envolvem todas as implicações a partir de um determinado "nome" em todos os níveis.

Aplicação: CONSEQ → PARACOMPACTO
causará a seguinte resposta

PARACOMPACTO → METACOMPACTO, CONTAVELMENTE PARACOMPACTO
METACOMPACTO → CONTAVELMENTE METACOMPACTO
CONTAVELMENTE PARACOMPACTO → CONTAVELMENTE METACOMPACTO

As consequentes de uma propriedade poderiam gerar relações com antccedcntes compostos.

: 3) o comando PANTEC e ANTEC

O primeiro fornece os antecedentes em primeiro nível de um "nome" e o segundo fornece todos os antecedentes em todos os níveis de um "nome".

: 4) o comando VERIF nome

É usado quando o usuário deseja saber se uma determinada relação é válida ou não.

O seu formato é VERIF A1 e A2 E ... AN → C. Ao processar o VERIF o sistema estabelece o caminho, ou caminhos que torna(m) a relação de implicação válida. Por exemplo, o comando

VERIF METRICO E PSEUDOCOIIPACTO → COMPACTO

CAMINHO 1:
METRICO → PERFEITAMENTE NORMAL → PERFEITAMENTE T4 → T4
METRICO → PERFEITAMENTE NORMAL → T1
T1 E T4 → NORMAL
NORMAL E PSEUDOCOFIPACTO → CONTAVELMENTE COMPACTO
METRICO E TOTALMENTE NORMAL → PARACOMPACTO → METACOMPACTO
METACOMPACTO E CONTAVELMENTE COMPACTO → COMPACTO.

CAMINHO 2:
METRICO → PERFEITAMENTE NORMAL → PERFEITAMENTE T4 → T4
METRICO → PERFEITAMENTE NORMAL → T1
T1 E T4 → NORMAL
NORMAL E PSEUDOCOMPACTO → CONTAVELMENTE COMPACTO
METRICO E CONTAVELJIENTE COIIPACTO → COMPACTO

CAMINHO 3:
METRICO → PERFEITAMENTE NORMAL → PERFEITAMENTE T4 →
METRICO → PERFEITAMENTE NORMAL → T1
T1 E T4 → NORMAL
NORMAL E PSEUDOCOMPACTO → CONTAVELMENTE COMPACTO → FRACAMENTE
METRICO E FRACAMENTE CONTAVELMENTE COMPACTO → COMPACTO

No caso do antecedente da relação a ser verificada ser constituido de apenas uma propriedade, duas possibilidades podem ocorrer:

- se a relação se verifica, são fornecidos todos os caminhos que levam do antecedente ao consequente. Por exemplo,

VERIF COMPLETAMENTE REGULAR → T2

produzirá

CAMINHO 1:
COMPLETAMENTE REGULAR → URYSOHN → T2 1/2 → T2

CAMINHO 2.:
COMPLETAMENTE REGULAR → T3 1/2 → T3
COMPLETAMENTE REGULAR → TO
TO E T3 → REGULAR SEMIREGULAR T2

se a relação não se verifica, é fornecida uma mensagem e os caminhos que interligam parcialmente o antecedente ao consequente, se existir algum. Por exemplo,

VERIF COMPACTO NORMAL

causará a seguinte resposta:

NÃO HA RELAÇAO DIRETA ENTRE AS PROPRIEDADES, MAS SAO FORNECIDAS A SEGUIR ALGUMAS RELAÇOES QUE AS INTERLIGAM:
COMPACTO → CONTAVELMENTE COMPACTO → FRACAMENTE CONTAVELMENTE COMPACTO
METRICO E FRACAMENTE CONTAVELMENTE COMPACTG → COMPACTO → SIGMA-LOCALMENTE COMPACTO → SIGMA-COMPACTO → LINDELOF
LINDELOF ε T3 → PARACOMPACTO
T2 ε PARACOMPACTO T4
T1 E T4 → T3 1/2 → T3
T2 E T3 → T2 1/2 → T2 → T1
T1 E T4 → NORMAL

: 4) o comando PESQ nome 1, nome 2

Causa a pesquisa de relações, que interligam os dois nomes.

O seu formato é PESQ nome1, nome2 e o seu processamento é exatamente análogo ao dos dois comandos seguintes:

VERIF nome1 → nome2
VERIF nome2 → nome1

Se a pesquisa ocasionada pelo primeiro comando tiver sucesso, o segundo não será processado.

Observamos que o comando PESQ não obriga ao usuário o prévio conhecimento do sentido da relação de implicação, como é o caso do comando VERIF.

Aplicação: PESQ REGULAR*,T2

causará a seguinte resposta'

CAMINHO 1:
REGULAR SEMIREGULAR T2

CAMINHO 2:
REGULAR T2 1/2 T2

D2- GRAFOS ENTRE AS PROPRIEDADES DO MESMO GRUPO E DE GRUPOS DIFERENTES

GRAFO 1

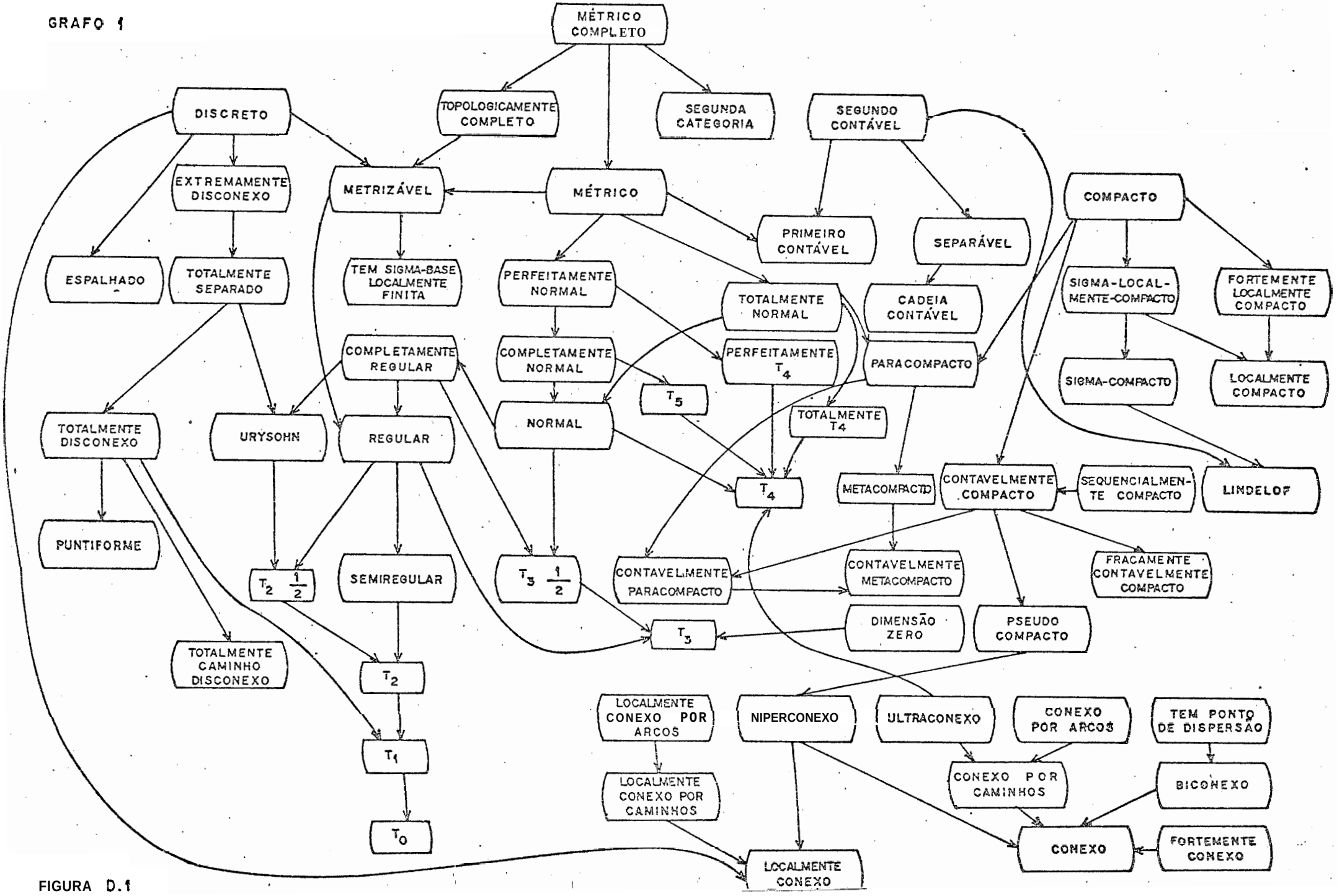


FIGURA D.1

GRAFO 2

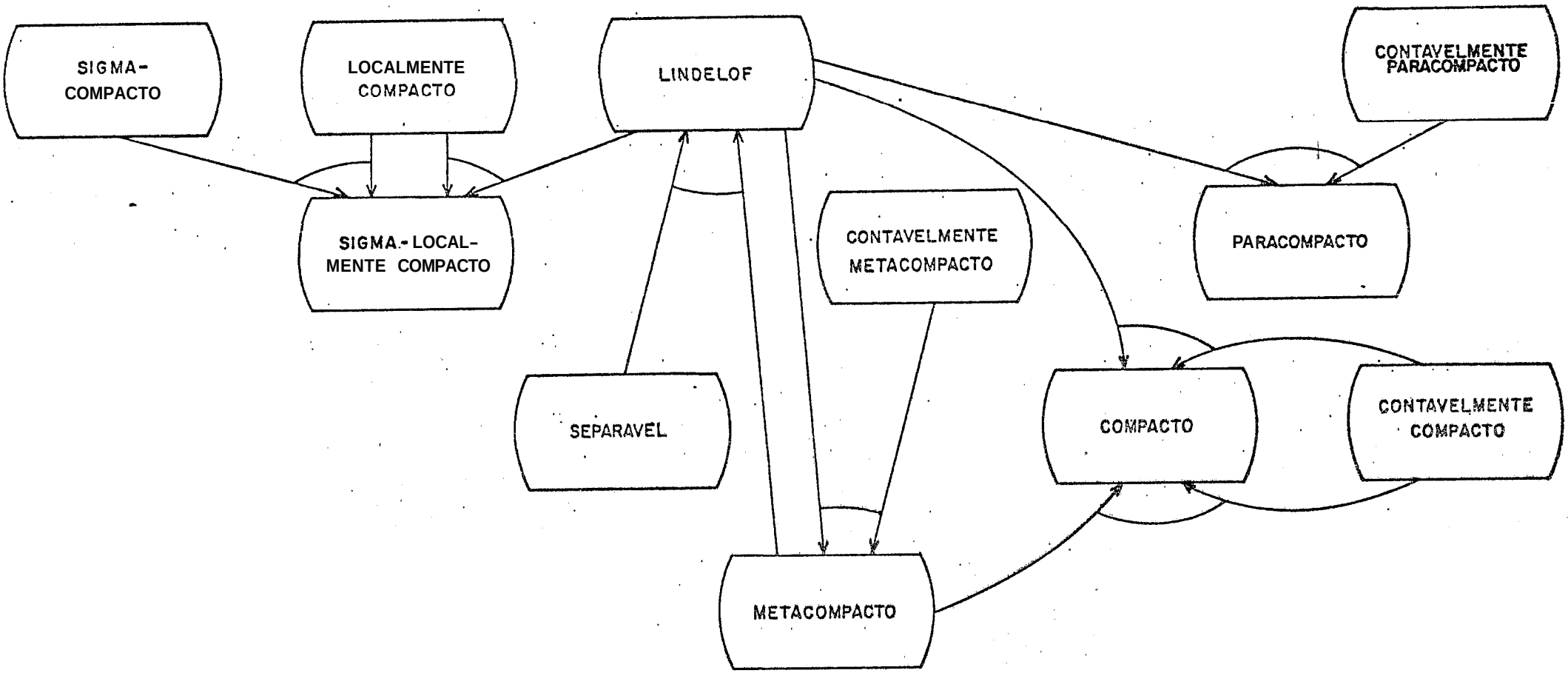


FIGURA D.2

GRAFO 3

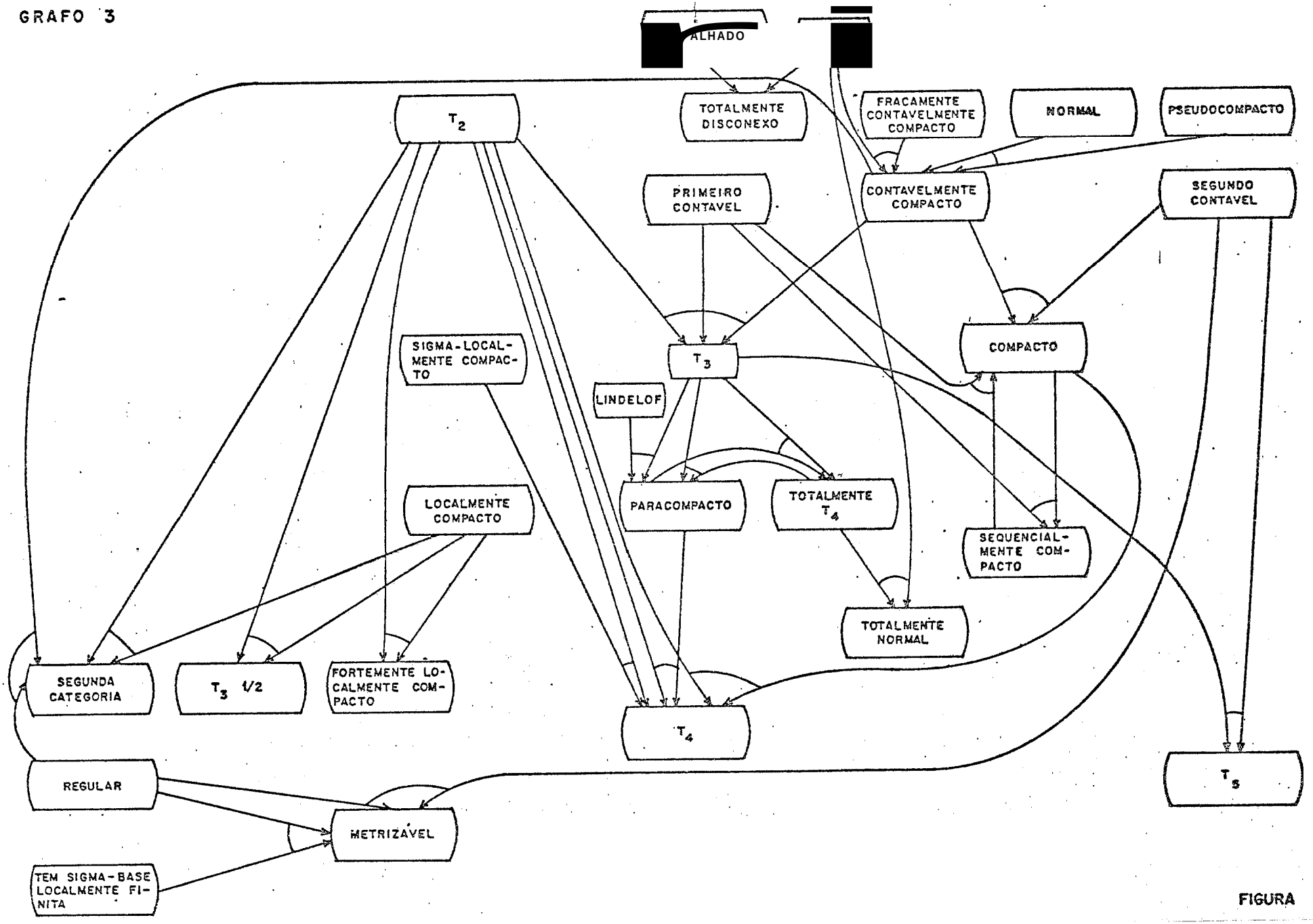


FIGURA D.3

GRAFO 4

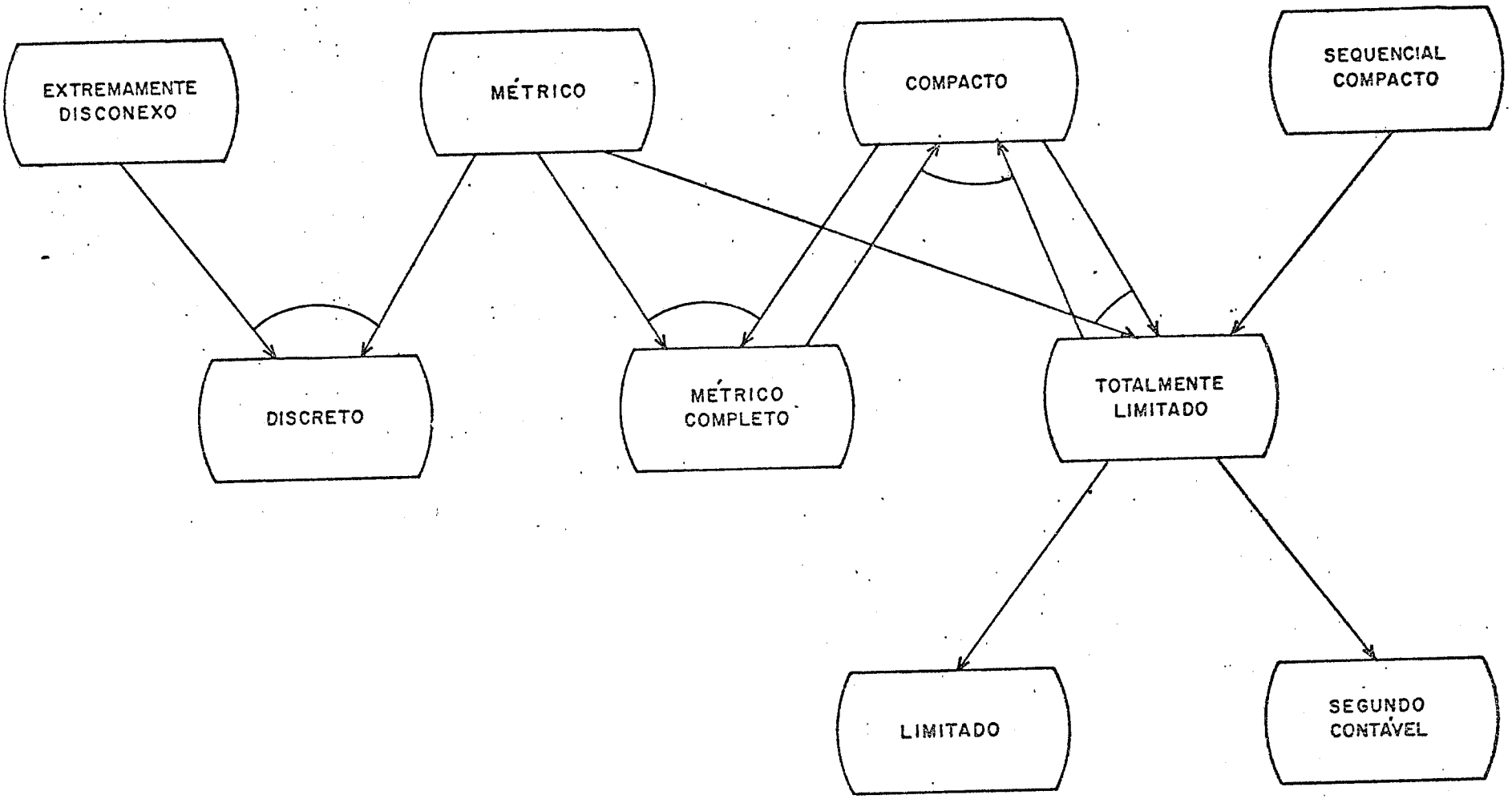


FIGURA D.4

GRAFO 5

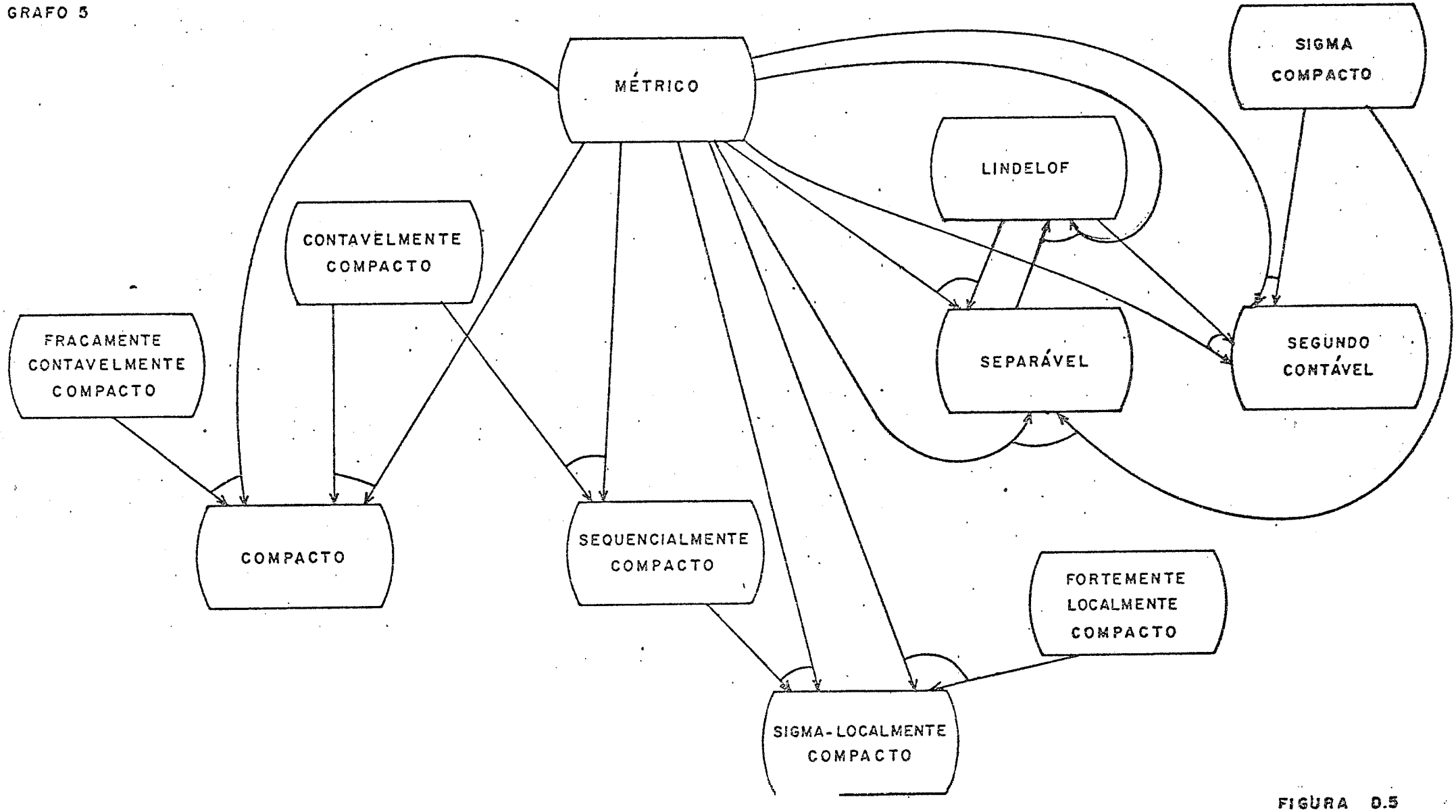


FIGURA D.5