O Tempo como Modelo:

A Aplicação de Lógicas Temporais na Especificação Formal de Sistemas Distribuídos

Wamberto Weber Miranda Peixoto de Vasconcelos

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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profa. Sheila R. M. Veloso, D.Sc.
(Presidente)

Profa. Doris Ferraz de Aragon, D.Sc.

Profa. Sueli B. T. Mendes, Ph.D.

RIO DE JANEIRO, RJ — BRASIL

SETEMBRO DE 1989

DE VASCONCELOS, WAMBERTO WEBER MIRANDA PEIXOTO

O Tempo como Modelo: A Aplicação de Lógicas Temporais na Especificação Formal de Sistemas Distribuídos [Rio de Janeiro] 1989

XX, 227 p., 29,7 cm. (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1989)

Tese — Universidade Federal do Rio de Janeiro, COPPE

1. Lógica Temporal 2. Especificação Formal 3. Sistemas Distribuídos 4. Modelos de Paralelismo I. COPPE/UFRJ II. Título (série).

À Maria Celi

(In Memoriam)

Agradecimentos

À Sheila pela orientação objetiva e segura e por haver me convencido que eu conseguiria levar a termo este trabalho;

Aos membros da banca examinadora, pela valiosa presença;

A meu pai e à minha família, distantes mas constantes;

À família Saciloto ("Seu" Vergílio, "Dona" Menair e Gianini) pela ajuda durante minha estadia no Rio e a Giovani pelos co "galhos quebrados";

A Cláudia Linhares, Emília Crispim e Hércules do Prado, pela amizade;

Aos professores da COPPE–Sistemas pelos conhecimentos adquiridos;

Aos companheiros da COPPE–Sistemas, Adelina Sesconetto, Ana Dolejši, Angela Cayuna, Celina de Figueiredo, Clícia Stelling, Cristina Boeres, Einstein Lemos, Fernando Rivas, Hermes Abreu, Inês de Castro, Juan Mangione, Juarez Muylaert, Laura Calland, Lúcia Gondar, Márcia Fernandes, Marcos Villas, Mau Ling, Maurício Solar, Monica Villanueva, Morganna Dinis, Rui Chiou, Roseli Wedemann e Samuel da Mata pelo espírito de coleguismo e cooperação;

Ao pessoal do Laboratório de Computação da COPPE–Sistemas, Eduardo e Adilson pela ajuda com o equipamento e suprimentos;

À secretária Denise pela ajuda nos entraves burocráticos;

Ao povo brasileiro, pelo apoio financeiro concedido através dos órgãos CAPES e CNPq, sem o qual este trabalho não seria possível;

Ao Prof. Tarcísio Pequeno pelas "noções de lógica e aplicações" e ao Prof. Clécio Thomas

pelo incentivo e ajuda no início de tudo;

A Márcio (MDB), Marcos (Reco Prego), Edoardo, Júnior (Runho), Ricardo (Itacoatiara) e Juninho pelo *free surf* e aos "locais" de Praia Seca (Região dos Lagos — RJ) e Arpoador (Posto 7) por compartilharem suas ondas comigo.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

O Tempo como Modelo:

A Aplicação de Lógicas Temporais na Especificação Formal de Sistemas Distribuídos

Wamberto Weber Miranda Peixoto de Vasconcelos

Setembro de 1989

Orientador: Sheila R. M. Veloso

Programa: Engenharia de Sistemas e Computação

As lógica temporais têm sido usadas na especificação formal e verificação de programas paralelos. Nesta dissertação, são apresentadas de forma didática, algumas dessas lógicas, mostrando como suas estruturas subjacentes de tempo modelam sistemas distribuídos. Para cada uma das lógicas, é apontada, através de exemplos, sua adequação à tarefa de especificação formal de sistemas distribuídos.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Time as a Model:
The Application of Temporal Logics in the Formal Specification of Distributed Systems

Wamberto Weber Miranda Peixoto de Vasconcelos

September, 1989

Thesis Supervisor: Sheila R. M. Veloso

Department: Systems Engineering and Computing

Temporal logics have been used in the formal specification and verification of parallel programs. In this dissertation some of these logics are presented, showing how their underlying structures of time model distributed systems. For each one of the logics, we present, through examples, its adequacy to the task of formal specification of distributed systems.

Índice

Lista de Figuras

Lista de Tabelas

Capítulo I

Introdução

A computação distribuída tem sido alvo da atenção de pesquisadores em universidades e centros de pesquisa devido a afirmação dessa tecnologia atualmente. Sistemas distribuídos estão sendo mais utilizados e em aplicações cada vez mais importantes, estimulados pela evolução, e conseqüente barateamento, de *hardware* e *software*. O ganho em eficiência dado pelo paralelismo, o compartilhamento de recursos e o aumento de confiabilidade são fortes apelos oferecidos pelo processamento distribuído.

Sistemas distribuídos (SDs), entretanto, não são simples de serem desenvolvidos. Seu tamanho e a complexidade oriunda da necessidade de integrar vários elementos processadores, levando em conta atrasos e/ou perdas de mensagens, obriga a possuir um conhecimento profundo dessas questões e a utilizar técnicas adequadas de análise e projeto de sistemas, procurando amenizar e racionalizar o esforço empregado nessa tarefa.

Em sistemas convencionais (centralizados e seqüenciais), a especificação formal tem sido de grande utilidade no desenvolvimento de *software* confiável e correto. Sistemas distribuídos carecem, até mais urgentemente do que os sistemas convencionais, de ferramentas e métodos que antecipem ao analista ou programador detalhes importantes sobre o sistema a ser desenvolvido, permitindo mudanças sem que seja necessário chegar até a implementação.

Uma vez admitido a importância das especificações formais para os SDs, a questão que naturalmente surge é o que se deve usar para especificar. A lógica temporal, uma extensão da lógica clássica cujo valor-verdade das afirmações pode depender do tempo, tem sido largamente utilizada na especificação e análise de programas concorrentes. Con-

tudo, várias lógicas temporais vêm sendo propostas e estudadas por lógicos e cientistas da computação (RESCHER *et alia*, 1971), dando origem à questão adicional de *qual* lógica temporal é a *mais* adequada.

A contribuição deste trabalho é reunir em um só lugar uma bibliografia extensa e grandemente pulverizada, mostrando de forma didática diversas lógicas temporais e seu uso na especificação formal de SDs. As lógicas temporais são formalmente apresentadas, sendo dadas a sintaxe, a estrutura sobre a qual a lógica é interpretada e a semântica, além de um sistema formal cuja consistência e completeza são consideradas. É apontada, através de exemplos, a aplicabilidade dessas lógicas à tarefa de especificação formal de SDs, também sendo mostrados exemplos de verificações formais de algumas propriedades da especificação. Este trabalho é, desde já, de grande utilidade àqueles que estudam sistemas distribuídos e a computação paralela, munindo-os de informações que os permitirão ir mais profundamente em outras questões tais como a geração automática de programas distribuídos a partir de especificações formais, a modelagem de programas paralelos, a verificação de programas concorrentes, etc.

Os capítulos foram organizados da seguinte maneira: no capítulo II, são revisados os conceitos de computação paralela e sistemas distribuídos. No capítulo III, discorremos sobre questões concernentes à especificação formal de SDs, uma das quais é a justificativa da adequação da lógica temporal a essa tarefa. Ainda no mesmo capítulo, apresentamos formalmente a lógica modal e é dada uma visão geral da lógica temporal.

O capítulo IV traz a lógica temporal linear (LTL) em duas versões, a proposicional (LTLp) e a de primeira ordem (LTLPo), sendo exibidos sistemas formais para cada uma dessas lógicas. Essas lógicas são utilizadas na especificação formal e verificação de SDs.

No capítulo V, mostramos a lógica temporal de tempo ramificado (LTTR), sendo proposto um sistema formal para ela. A LTTR é usada na especificação formal e verificação de um exemplo de SD. Nesse capítulo é feito um estudo comparativo entre a LTTR e a LTL.

No capítulo VI, mostramos um modelo alternativo para a lógica temporal, dando origem à lógica temporal de conjuntos de intercalações (LTCI). Essa lógica une vantagens da LTL e da LTTR, considerando ainda a ordenação parcial de cada processo. Um exemplo

simples de especificação formal de SD com essa lógica é apresentado.

No capítulo VII, mostramos a união de duas lógicas modais: a lógica de conhecimento e crença e a lógica temporal linear, dando origem à lógica temporal de conhecimento e crença (LTCC). Essa lógica é usada na especificação formal e verificação de um problema na área de processamento distribuído.

No capítulo VIII, são feitas considerações gerais sobre a automatização de provas com a lógica temporal. Finalizando, no capítulo IX, são reunidas algumas conclusões adicionais sobre o trabalho.

Ao final de cada capítulo (com exceção dos capítulos VIII e IX) há uma seção de conclusões, na qual são feitas algumas conclusões sobre o assunto exposto, e uma seção de notas bibliográficas, na qual são reunidos comentários sobre a bibliografia utilizada.

Neste trabalho, buscou-se a exaustão do assunto, mas a área é extremamente fértil e novos artigos e livros são uma constante. Dadas as limitações da "velocidade de transmissão" do conhecimento, a bibliografia foi pesquisada até o início de 1989, cobrindo três décadas de referências à lógica temporal. Passados mais de 30 anos desde sua proposta (PRIOR, 1957) e 12 anos desde seu primeiro uso na computação (PNUELI, 1977), a lógica temporal continua a ser alvo de pesquisas e atenções de pesquisadores (LAMPORT, 1989) propondo novas lógicas ou novos métodos para seu uso na computação.

Capítulo II

Computação Paralela e Sistemas Distribuídos

A computação paralela surgiu junto com a computação. Só recentemente, entretanto, a partir de meados da década de setenta, ela tem sido alvo de um interesse maior. Isso se deve a avanços na tecnologia de *hardware* e *software* que a tem tornado mais acessível.

Sistemas distribuídos (SDs) são um tipo de computação paralela no qual os processadores não compartilham memória, possuindo cada um a sua própria memória local. SDs têm crescido em importância e utilização. Entretanto, devido ao elevado tamanho e complexidade do *software* dos SDs, seu desenvolvimento envolve, essencialmente, um profundo conhecimento dessa área e a utilização de técnicas adequadas de concepção e projeto de sistemas.

Esse capítulo se propõe a fazer uma breve revisão sobre os conceitos da computação paralela e, mais particularmente, de SDs. Esses conceitos serão úteis na investigação da questão da especificação formal de SDs. A estrutura do capítulo é a seguinte: a primeira seção discorre sobre a computação paralela — uma breve introdução, a motivação para o paralelismo, a definição de concorrência e seus tipos. Continuando a seção, é mostrado o problema fundamental da sincronização, compara-se os programas concorrentes com os seqüenciais, são dadas algumas definições e as diferentes maneiras de se ver um SD. Encerrando a seção, elabora-se um pouco sobre as linguagens de programação concorrentes.

A seção seguinte versa sobre os SDs, possuindo a seguinte estrutura: uma introdução sucinta, a motivação para os SDs, os tipos de SDs, a diferença entre sincronismo

e sincronização, algumas definições e as dificuldades da programação distribuída. Em seguinda, a seção de conclusões e a seção de notas bibliográficas.

II.1 Computação Paralela

A demanda por aumentos significativos nas velocidades de processamento dos computadores sobre aquelas propiciadas por sistemas uniprocessadores tem resultado no projeto de configurações de um único computador contendo várias centenas de unidades de processamento. As questões de *software* e *hardware* pertinentes às configurações de um único computador multiprocessador podem ser classificadas, de modo geral como *computação paralela.*

No processamento seqüencial convencional, um único elemento, a unidade central de processamento, é responsável pelo controle e execução das instruções que constituem os programas. No processamento paralelo, vários elementos processadores separados entre si cooperam, trabalhando em paralelo. Como diversos processadores estão operando como forças-tarefas, o ganho no desempenho será inversamente proporcional ao número de processadores utilizados, ou seja, de instruções ou grupos de instruções que são executados simultaneamente.

O conceito de processamento paralelo é tão antigo quanto os próprios computadores eletrônicos, sempre constando como opção na melhoria do desempenho de algoritmos e de sistemas de computação como um todo. A possibilidade do uso de algoritmos paralelos para a solução de equações diferenciais foi assunto de trabalhos desenvolvidos por von Neumann já na década de 40. Outros exemplos são o sistema "MODEL V" desenvolvido em meados da década de 40 nos laboratórios da *Bell Telephone* e a máquina paralela "ILLIAC VI", um computador desenvolvido na década de 60 na universidade de Illinois constituído por 64 processadores.

II.1.1 Por que a Computação Paralela?

Atualmente, há um grande interesse nas áreas de arquiteturas e processamento paralelo, com várias máquinas paralelas sendo colocadas anualmente no mercado. A computação paralela e distribuída têm merecido destaque entre projetos de pesquisa nas universidades

e centros de pesquisas. Alguns dos motivos principais que estimularam o surgimento da computação paralela e distribuída são (AMORIM *et alii*, 1988):

- *desempenho* — como vários agentes operam como executores dos programas, então o tempo de processamento diminui de forma inversamente proporcional ao número de agentes em ação;
- *modularidade* — é importante, para o fabricante e para o usuário, que a arquitetura do equipamento possa ser expandida pelo acréscimo de módulos, pois aplicações de diferentes faixas de demanda computacional poderiam ser atendidas a partir de um mesmo produto;
- *tolerância a falhas* — muitas aplicações requerem que o sistema de computação assegure operação contínua na presença de falhas de *hardware*. Sistemas multiprocessadores podem verificar se falhas ocorreram, diagnosticando-as. O componente defeituoso pode então ser retirado pelo sistema, que continua em operação, ainda que com sua configuração degradada.

II.1.2 Programas Concorrentes

Muitos algoritmos podem ser "paralelizados", isto é, refeitos de modo a explorar a execução concomitante de algumas de suas instruções ou de grupos de instruções.

Alguns problemas, entretanto, para serem resolvidos, exigem forçosamente o uso do paralelismo. Esses problemas nem sempre envolvem a transformação de dados de entrada fornecidos no início em uma saída desejada produzida no final. Ao invés disso, eles envolvem um *comportamento* desejado no decorrer do tempo, sendo exigidas várias propriedades desse comportamento. Em muitos casos o término não é verificado – os processos ficam sempre ativos realizando tarefas de acordo com as exigências. Os programas que resolvem essa classe de problemas são chamados *concorrentes* (HAREL, 1987).

Um programa concorrente especifica dois ou mais programas seqüenciais que podem ser executados concorrentemente como processos paralelos (ANDREWS *et alia*, 1983). Processo concorrente é a execução de um programa concorente. *Processo* corresponde a um programa em execução, isto é, as instruções do programa sendo executadas. A dife-

rença entre programa e processo é sutil, e respeitando o contexto, o uso de um termo pelo outro não trará prejuízos à compreensão.

II.1.2.1 Tipos de Programas Concorrentes

Um programa concorrente pode ter seus programas seqüenciais (processos) executados de duas maneiras (ANDREWS *et alia*, 1983)

- *intercaladamente* - os processos compartilham um ou mais processadores. Essa forma de execução recebe o nome de *multiprogramação*, sendo estruturada sobre um núcleo do sistema operacional que gerencia o uso dos processadores pelos processos.
- *simultaneamente* - cada processo é submetido ao seu próprio processador. A execução simultânea divide-se ainda em dois subtipos:
  1. *Multiprocessamento* - os processadores compartilham uma mesma memória.
  2. *Processamento Distribuído* - os processadores não possuem uma memória comum, são dispostos geograficamente distantes entre si e são conectados por uma rede de comunicação. Um programa distribuído é, portanto, também um tipo de programa concorrente, sua característica sendo a arquitetura subjacente da máquina.

II.1.2.2 A Velocidade dos Processos

A velocidade com a qual os processos são executados depende de qual abordagem é usada. Quando cada processo é executado em seu próprio processador, cada um dos processos é executado em uma velocidade fixa, mas talvez desconhecida. Quando processos compartilham um processador (multiprogramação), é como se cada um dos processos fosse executado em um processador de velocidade variável. Deseja-se entender um programa concorrente em termos de seus processos seqüenciais componentes e suas interações, sem preocupar em como eles são executados. Não é feita, portanto, nenhuma hipótese sobre velocidades de execução de processos executando concorrentemente, exceto que todas elas são positivas. Isso é chamado a *hipótese do progresso finito*. A correção de um programa

para o qual só o progresso finito é assumido é independente do fato de o programa ser executado em múltiplos processadores ou em um único processador multiprogramado.

II.1.3 O Problema Fundamental da Sincronização

Do ponto de vista do formalismo lógico, sistemas distribuídos não são diferentes de qualquer outro tipo de sistema concorrente. O que distingue um sistema multicomputador distribuído é o atraso da comunicação e a banda passante (*bandwidth*) da comunicação. Em sistemas distribuídos, a informação pode levar muitos milissegundos para ir de um processo para outro, ao passo que, em um multiprocessador, ela leva somente microssegundos. A banda passante para transmissão de mensagem nos sistemas distribuídos pode ser limitada por linhas telefônicas, enquanto que, nos multiprocessadores, a banda passante de transmissão de mensagens é limitada apenas por atrasos de acesso de memória.

Essas diferenças, entretanto, são quantitativas e não qualitativas. Em ambos os casos, a informação recebida de outro processo representa o estado desse processo quando a informação foi enviada, o que não é necessariamente o estado do processo quando a informação é recebida. O fato de que a situação do sistema está mudando enquanto um processo o está "vendo" é o problema fundamental de sincronização em qualquer tipo de sistema concorrente.

II.1.4 Programas Seqüenciais x Programas Concorrentes

Enquanto não há nenhuma diferença lógica entre programas concorrentes distribuídos e não-distribuídos, há uma diferença fundamental entre programas seqüenciais e concorrentes. Em programas seqüenciais, preocupa-se somente com a relação entre os estados inicial e final das partes do programa, o que é conhecido por *comportamento de entrada/saída*. Portanto, o comportamento de entrada/saída de um programa composto por dois trechos T_1 e T_2 executados seqüencialmente depende somente do comportamento de entrada/saída de T_1 e T_2 e não de como esse comportamento é implementado.

O comportamento de entrada/saída não é suficiente para falar sobre programas concorrentes. O comportamento de dois processos P_1 e P_2 executados simultaneamente não é determinado pelo comportamento de P_1 e pelo comportamento de P_2, pois os dois

processos podem "interferir" um no outro. Para tratar programas concorrentes, deve-se falar sobre seu comportamento de modo completo — o que o programa faz através de toda sua execução — não apenas o que é verdade antes e após a execução.

II.1.5 Programas Seqüenciais

Programas seqüenciais são um caso especial muito simples dos programas concorrentes: aquele caso no qual um único processo é executado. A execução de um programa seqüencial pode ser representada por uma seqüência da forma:

$$s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots \tag{II.1}$$

onde os s_i são *estados* e os α_i são *ações atômicas*. Essa seqüência será finita se, e somente se, a execução terminar.

O estado contém toda a informação necessária para determinar o futuro comportamento possível do programa: os valores de variáveis e contadores dos processos. Em um programa determinístico, o estado determina seu comportamento preciso; em um programa não-determinístico, o estado determina o conjunto de todos os possíveis comportamentos.

As ações atômicas são as ações que levam o programa de um estado para outro. Elas são atômicas porque são feitas em um único ciclo do processador, não havendo situações intermediárias entre o começo e o fim da ação. Por exemplo, uma ação α_i pode ser uma instância particular do comando

$$x := x + 1$$

Então o estado s_i seria o mesmo que o estado s_{i-1} com exceção do valor da variável x que seria acrescido de 1. Para programas seqüenciais, não há diferença em como o programa é dividido em ações atômicas, mas para programas concorrentes, isso é crucial.

Ações e estados são, de certa forma, duais entre si. Formalismos diferentes geralmente concentram-se em apenas um deles, representando a execução acima como uma seqüência $s_0 s_1 s_2 \cdots$ de estados ou como a seqüência $\alpha_0 \alpha_1 \alpha_2 \cdots$ de ações.

II.1.6 Algumas Definições

No estudo dos programas concorrentes alguns termos são importantes para a compreensão dos conceitos. A seguir, é dado uma definição para alguns deles (KATZ *et alia*, 1987).

Eventos — são execuções únicas de operações atômicas (vide def. acima). Eventos não são o mesmo que a representação sintática das operações (estas são apenas padrões de eventos). Cada execução de uma operação define um evento diferente.

Processos: são entidades abstratas representando uma tarefa que pode ser executada concorrentemente com outras tarefas. A divisão de eventos entre os processos pode ser vista como uma indexação dos eventos: eventos indexados por um indexador são ditos "ocorrendo dentro de um processo" (por exemplo, uma atribuição), eventos que são indexados por mais de um identificador de processo ocorrem conjuntamente entre vários processos (por exemplo, em um modelo com comunicação síncrona quando um "handshaking" ocorre entre dois processos).

Instantâneo Local e Global: Um instantâneo (*snapshot*) local ou global de um programa é um elemento do produto cartesiano dos valores das variáveis do processo ou do programa (respectivamente) incluindo os apontadores do código do programa. Um instantâneo global em um modelo com comunicação assíncrona inclui também, para cada canal, o conjunto de mensagens enviadas mas ainda não recebidas.

Evento Local: Um evento local inicia em um instantâneo local do processo no qual ele ocorre e termina com um novo instantâneo local. Em um modelo de comunicação síncrona, um evento de comunicação é mútuo a dois (ou mais) processos diferentes.

II.1.7 A Visão de Programas Concorrentes

A descrição da execução de um programa seqüencial deve ser generalizada no sentido de acomodar a execução de programas concorrentes, pois neles as ações de processos diferentes não são vistas naturalmente como seqüencialmente ordenadas. De fato, em um sistema distribuído, algumas vezes é impossível dizer, entre dois eventos, qual ocorreu primeiro, tornando necessário a adoção de uma visão que simplifique o estudo/verificação de SDs levando em conta suas características.

II.1.7.1 A Visão Espaço-Tempo

A visão espaço-tempo, discutida em profundidade por LAMPORT (1978), é talvez a abordagem mais natural. Nela, a execução de um programa concorrente é um conjunto de eventos parcialmente ordenados onde a ordenação parcial é determinada pelas seguintes restrições:

1. Os eventos em uma locação física única são totalmente ordenados no tempo;
2. Um evento α que provoca alguma mudança física no sistema deve preceder um evento β que sente aquela mudança.

Para os sistemas distribuídos, esses dois tipos de ordenações são interpretados da seguinte maneira:

1. Todas as ações executadas por um único processo são totalmente ordenadas;
2. O envio de uma mensagem deve preceder seu recebimento.

Este conjunto de ordenações temporais deve ser acíclico, de modo que seu fechamento transitivo forme uma ordenação parcial irreflexiva.

A visão espaço-tempo é muito útil no entendimento de sistemas distribuídos. Ela representa a filosofia que subjaz a maioria dos trabalhos na teoria das redes, sendo fortemente dirigida para as ações.

II.1.7.2 A Visão das Intercalações

Enquanto a visão subjacente da execução de um programa concorrente pode ser um conjunto de ações, a experiência mostra ser desejável raciocinar sobre programas (tanto seqüenciais como concorrentes) em termos de estados. Para falar sobre estados, usa-se a *visão das intercalações* (ALFORD *et alii*, 1985).

Na visão das intercalações, assume-se que, como no caso de um programa seqüencial, a execução de um programa concorrente consiste na seqüência (II.1) de estados

s_i e ações atômicas α_i. A justificativa para essa visão é que seria possível, em princípio, determinar a ordem real na qual as ações aconteceram. Se duas ações que não influenciam uma a outra são realmente concorrentes, então pode-se fazer de conta que elas ocorreram em qualquer ordem. Se duas ações concorrentes influenciam uma a outra, então elas não são realmente atômicas e devem ser subdivididas em ações menores.

Assumindo-se que as ações são totalmente ordenadas, perde-se a habilidade de distinguir se uma ação "realmente" precedeu outra ou se duas ações eram concorrentes e foram ordenadas apenas por conveniência. Essa perda de informação limita o tipo de pergunta que pode ser feita sobre o sistema. Entretanto, para aquelas perguntas que podem ser feitas, a visão das intercalações proporciona um formalismo muito conveniente para respondê-las.

II.1.8 Linguagens de Programação Concorrentes

A programação convencional é grandemente facilitada pelas linguagens de programação. Nos primeiros computadores, onde só havia a linguagem de máquina, a confecção de programas era uma tarefa extremamente cansativa, pois as instruções, seqüências de zeros e uns, eram difíceis de entender, só esclarecendo algo a programadores com muita experiência. Com o advento das linguagens de alto nível, foi verificado uma diferença qualitativa nos programas, que tornaram-se mais fáceis de fazer, analisar e depurar. Os programas ficaram mais confiáveis, dependendo menos de aptidões particulares de programadores, aproximando-se da técnica e da ciência e sendo menos artesanais.

A programação concorrente não é diferente. As linguagens concorrentes existentes tentam suavizar a dura tarefa de se fazer programas que exploram o paralelismo. Elas são muitas, e muitas outras surgem a cada ano, todas se dizendo as melhores. STOTTS JR.(1982) faz um estudo comparativo de diversas linguagens concorrentes e ANDREWS *et alia* (1983) descrevem os conceitos centrais ao projeto e construção de programas concorrentes e exploram as notações para a descrição de computações concorrentes.

As linguagens de programação convencionais, apesar de se dizerem de "propósito geral", raramente comprovam ser igualmente aplicáveis a todas as classes de problemas. No caso das linguagens concorrentes essa perda de generalidade é agravada pela arquite-

tura particular da máquina para a qual a linguagem foi concebida, pois o projetista da linguagem tenta apresentar um modelo de computação que represente o mais fielmente possível a arquitetura subjacente da máquina. Uma linguagem paralela concebida sem uma arquitetura particular ou uma classe de máquinas em mente é uma raridade.

Algumas das linguagens concorrentes historicamente mais importantes, que incorporam mecanismos para a comunicação e sincronização de processos, são (PETERSON *et alia*, 1983; CRICHLOW, 1988):

Pascal Concorrente — é uma linguagem de programação projetada para a programação estruturada de sistemas operacionais. Desenvolvida por BRINCH HANSEN (1975), a linguagem é baseada na linguagem Pascal seqüencial (WIRTH, 1971). Um dos aspectos mais importantes do Pascal concorrente é que ele, à semelhança do Pascal seqüencial, permite a modularidade na construção de programas.

As novas características do Pascal concorrente permitem a especificação de monitores e processos concorrentes. O construto *monitor* é usado para manipular a sincronização e a comunicação inter-processual.

Cada processo contém seus próprios dados e um número de sentenças que devem ser executados seqüencialmente. Nenhum processo tem acesso aos dados privativos de outro processo. Estruturas de dados compartilhados e as operações (procedimentos) que podem ser feitas nessa estrutura de dados formam um monitor.

Os processos podem comunicar-se com outros processos pela chamada de procedimentos dentro de um monitor particular. Isso resulta em mudanças sendo feitas à área de dados compartilhados. O acesso ao monitor só é autorizado a um processo de cada vez, portanto os monitores também sincronizam a atividade dos processos.

CSP — Processos seqüenciais comunicantes (*Communicating Sequential Processes*) é um arcabouço de linguagem para programação concorrente apropriado para ambientes de redes de microcomputadores com armazenamento distribuído (HOARE, 1978). Os seguintes conceitos são os mais importantes da linguagem:

- um programa CSP consiste de um número fixo de processos seqüenciais que são mutuamente disjuntos em espaços de endereçamento;

- a comunicação e a sincronização são realizadas através de construtos de entrada/saída;
- as estruturas de controle seqüencial são baseadas nos *comandos guardados* de DIJKSTRA (1975);

Em CSP, a palavra "processo" se refere a um grupo de comandos seqüenciais que, juntos, podem ser vistos como alguma sub-tarefa ou unidade de atividade. Os comandos podem ser simples ou estruturados. A execução de um comando estruturado falha se qualquer um de seus comandos falhar.

Um comando paralelo especifica um número de processos que devem ser executados em paralelo. Todos os processos começam simultaneamente e os comandos paralelos terminam com sucesso somente quando todos os processos têm terminado com sucesso.

A comunicação entre processos executados concorrentemente é verificada através de comandos de entrada e saída. Três condições devem ser satisfeitas antes que possa haver comunicação entre dois processos. Se o processo A quer enviar uma saída para o processo B, então

a) um *comando de entrada* em B deve especificar que A é a fonte da entrada;

b) um *comando de saída* em A deve especificar que B é o destino da saída; e

c) a variável-alvo especificada para o recebimento da entrada deve ser compatível com aquela especificada pelo comando de saída.

Quando essas condições são satisfeitas, os comandos de entrada e saída *correspondem* e são executados simultaneamente. Por isso, um processo não pode enviar uma mensagem para outro a menos que o processo destinatário esteja pronto para aceitá-lo. Desse modo, a atividade dos processos é sincronizada.

Ada — é uma linguagem de programação de alto-nível, cujo projeto foi financiado pelo Departamento de Defesa dos Estados Unidos (*The Ada Language Reference Manual*; United States Department of Defense; Dec. 1980). A linguagem pode ser usada para a programação convencional e também para necessidades técnicas especiais, tais como a monitoracão de aparelhos em tempo real.

Em Ada, cada uma das atividades paralelas de um programa pode ser identificada e representada por uma *tarefa* (*task*). Tarefas podem ser executadas em processadores separados ou concorrentemente em um sistema uni-processador.

As tarefas são declaradas em um unidade de programa chamadada de *pai*. Sempre que a unidade pai é executada, todas as tarefas dentro dela são inicializadas e serão executadas em paralelo a menos que haja comandos explícitos para fazê-lo de outra forma.

II.2 Sistemas Distribuídos (SDs)

Computadores podem ser fisicamente ligados via um canal de comunicações para facilitar o compartilhamento de recursos de *software* e *hardware* e para permitir a transferência, acurada e imediata, de informação através de distâncias variando de menos de um metro, em uma única sala, (LAN — *Local Area Network*), a centenas de quilômetros, através de continentes inteiros, (WAN — *Wide Area Network*). Tal organização também permite que tarefas e processos sejam distribuídos entre computadores separados onde eles possam ser executados em paralelo, resultado em um aumento no número de tarefas realizadas na unidade do tempo. Questões pertinentes ao projeto de *hardware* e *software* de uma dessas interconexões de computadores autônomos podem ser chamadas, de modo geral, de *computação distribuída* (CRICHLOW, 1988).

Processamento distribuído é a computação na qual os diversos elementos processadores não compartilham memória e, conseqüentemente, toda a comunicação entre processadores deve ser realizada através da troca de mensagens. Os processadores em um SD podem variar em tamanho e função. Eles podem incluir pequenos micro-processadores, estações de trabalho, mini-computadores e sistemas de grande porte de propósito geral. Esses processadores são referenciados por vários nomes distintos, como *locais*, *processos*, *nós* ou *nodos*, *computadores*, etc., dependendo do contexto no qual são mencionados.

II.2.1 Por que SDs?

O desenvolvimento e difusão de processadores de baixo custo e tecnologias de interconexão e transmissão de dados despertou o interesse por sistemas distribuídos. Esses fatores tornaram viável a utilização da abordagem distribuída em aplicações às quais essa solução se

adequa mais elegante e naturalmente, como, por exemplo, sistemas de automação bancária e sistemas de reservas de passagens aéreas. A esses motivos somam-se diferentes razões para se usar sistemas distribuídos, as mais importantes sendo o compartilhamento de recursos, o ganho no desempenho, a confiabilidade e a comunicação:

- *compartilhamento de carga* visando melhor exploração da capacidade de processamento;
- *compartilhamento de recursos* para utilizar recursos caros ou equipamento especial escassamente utilizado;
- *compartilhamento de informações* para acessar banco de dados distribuídos;
- *a geografia da estrutura* pode ser inerentemente distribuída. A velocidade de transmissão das linhas de comunicação ou a fraqueza de sinais analógicos pode forçar seu processamento *in loco*;
- *a estrutura lógica* pode ser mais simples, por exemplo, se cada processo paralelo é localizado em um processador separado;
- *a confiabilidade* de um sistema pode ser aumentada pela confecção de uma estrutura apropriada;
- *a flexibilidade* de um sistema é aumentada, dispondo da possibilidade de adicionar e retirar processadores.

O relacionamento entre as computações paralela e distribuída pode ser descrito da seguinte maneira: a computação pode ser distribuída em um ambiente de computador paralelo, ou que o ambiente de computação distribuída pode ser usado para explorar o paralelismo em alguma computação.

II.2.2 Tipos de SDs

Sistemas distribuídos podem ser:

- *Síncronos* - todos os processadores do sistema funcionam segundo um relógio global comum, a que todos têm acesso. Aos sistemas distribuídos síncronos é geralmente

associada uma característica: uma mensagem enviada por um nó a um de seus vizinhos no início de um ciclo do relógio global chega ao seu destino antes do início do próximo ciclo.

- *Assíncronos* - sua principal característica é a ausência total de uma base de tempo comum a todos os processadores que o compõem. Cada processador possui um relógio local, independente dos demais. Suas mensagens sofrem atrasos finitos, porém indeterminados, durante seu trânsito pelo sistema.

Sistemas distribuídos síncronos são uma abstração teórica. Não existem realmente, sendo de utilidade apenas no entendimento da área de processamento distribuído em geral. Sistemas distribuídos reais são sistemas assíncronos, daí a causa de grande parte da complexidade em sua análise – são impedidas generalizações imediatas a partir de modelos válidos para o processamento centralizado (AMORIM *et alii*, 1988).

II.2.3 Sincronismo x Sincronização

A cooperação entre os diversos processos de um programa concorrente é verificada através da comunicação e sincronização. A comunicação permite que a execução de um processo influa na execução de outro processo. A comunicação inter-processual é baseada no uso de variáveis compartilhadas (no caso do multiprocessamento) ou no envio de mensagens (processamento distribuído).

Sincronização *não* é o mesmo que sincronia. Sincronia é a propriedade apresentada pelos sistemas distribuídos cujos processadores funcionam conforme um relógio global comum (vide definição de sistema distribuído síncrono dada anteriormente).

A sincronização pode ser vista como um conjunto de restrições na ordem dos eventos de processos, sendo freqüentemente necessária quando processos se comunicam (os processos são executados com velocidades imprevisíveis e, para que eles se comuniquem, um processo deve executar alguma ação que o outro detecte, uma ação tal como a atribuição de um determinado valor a uma variável ou o envio de uma mensagem). O programador emprega um mecanismo de sincronização para atrasar a execução de um processo visando satisfazer tais restrições.

Seja o seguinte exemplo: suponha um sistema distribuído composto por dois processos, P_1 e P_2, como mostrado na figura II.1. As reticências ($\vdots$) representam instruções cuja execução não interfere no comportamento externo do processo, isto é, não são envios de mensagens nem provocam espera. As instruções estão rotuladas (l_j, l_{j+1}, m_q e m_{q+1}) para que possamos fazer referência de modo econômico e foi usada uma linguagem cuja semântica é intuitiva.

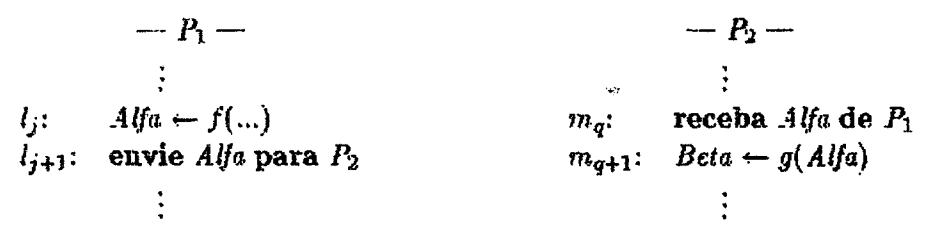

Figura II.1: Sincronização de Dois Processos

Os processos estão sincronizados: o controle da execução de P_2 só passará do ponto m_q após a variável $Alfa$ ser enviada por P_1. O programador atrasou a execução de P_2, sincronizando-o a P_1. Há, portanto, uma restrição à ordenação dos eventos (execução das ações).

II.2.4 Processos

Sistemas distribuídos são constituídos por processos. Informalmente, um *processo seqüencial* é um programa em execução. Deve ser enfatizado que um programa em si não é um processo; um programa é uma entidade *passiva*, enquanto um processo é uma entidade *ativa*. A execução de um processo deve progredir de uma maneira seqüencial (daí eles também serem chamados *processos seqüenciais*), isto é, em qualquer ponto do tempo, no máximo uma instrução é executada em nome do processo. Portanto, embora dois ou mais processos possam estar associados com o mesmo programa, eles são considerados duas seqüências de execução separadas.

II.2.5 Processos Estáticos e Dinâmicos

Um processo que não termina enquanto o sistema operacional estiver funcionando é chamado *estático*; um processo que pode terminar é chamado *dinâmico*. Se um sistema distribuído consiste de somente um número limitado de processos estáticos, então ele também é estático, isto é, ele nunca muda. Obviamente, deve-se ser criterioso quando define-se esses termos, posto que, inicialmente, o SD não possui nenhum processo, e mesmo quando este possui um número fixo de processos estáticos, ainda há aspectos dinâmicos: mensagens em trânsito pelo sistema, canais de comunicação que podem mudar, etc. Diz-se, então, que um SD é estático quando este atinge um estado, após um curto período de tempo inicial, onde nenhum processo mais é acrescido ou retirado dele. Um SD é dinâmico quando processos podem ser acrescentados ou retirados dele (PETERSON *et alia*, 1983).

II.2.6 Dificuldades da Programação Distribuída

Programar um sistema distribuído não é uma tarefa simples. Deve-se ter em mente o não-determinismo oriundo da execução simultânea dos processos por elementos processadores em diferentes velocidades. Os programas seqüenciais que compõem o programa distribuído devem ser escritos de forma a não dependerem do acaso, com comandos explícitos que os relacionem, se for necessário. Os processos devem funcionar corretamente qualquer que seja a ordem (permitida) dos eventos – a execução de um processo não deve interferir arbitrariamente na execução de outro processo. Quando for necessário, o programador deve conceber mecanismos de sincronização (vide seção II.2.3) de modo a coordenar a cooperação entre os processos.

Uma das razões da dificuldade em projetar programas distribuídos é porque a troca de mensagens, a única maneira que os processos dispõem de se comunicarem em SDs, envolvem um atraso substancial em sua entrega relativo às velocidades de execução dos processos. O recebimento de uma mensagem dá informação sobre um estado passado do processo remetente, não necessariamente seu estado atual. Por isso, o estado do sistema pode não ser conhecido por nenhum processo.

Por esses motivos, a intuição não é suficiente para a tarefa de programar/projetar SDs, sendo falível em casos de grande complexidade. Modelos, métodos e ferramentas

para a especificação formal e técnicas de verificação são mais necessárias aqui do que no desenvolvimento de sistemas convencionais.

II.3 Conclusões

A disponibilidade de microprocessadores relativamente baratos e eficientes desencadeou um reavivamento no estudo da computação paralela. O acesso *on-line* a vários processadores ao mesmo tempo dá novas dimensões ao poder computacional para resolver problemas. A tarefa pode ser dividida em várias sub-tarefas e diferentes processadores podem ser alocados para executar essas sub-tarefas. De fato, uma grande classe de problemas, para os quais soluções auxiliadas por computador são estudadas, exigem forçosamente o paralelismo (por exemplo, computações científicas de grande escala — aeronáutica, física nuclear, etc.). Sistemas paralelos dividem-se em algumas categorias, uma das quais é a dos sistemas distribuídos (SDs).

Os SDs surgem como uma opção à melhoria do desempenho computacional. Bastante utilizados no dia-a-dia em sistemas de automação bancária, reserva de passagens aéreas, monitoração de usinas nucleares, etc., eles oferecem grandes vantagens, tais como o compartilhamento de carga, recursos e informações, o aumento de confiabilidade e flexibilidade e a tolerância a falhas.

Infelizmente, essas vantagens possuem uma séria contrapartida: SDs são difíceis de projetar. Integrar a ação de vários elementos processadores funcionando de forma independente torna-se uma tarefa muito complicada, à medida que o número dos nós aumenta. Outro agravante é que a troca de mensagens, a única maneira de os processos se comunicarem, envolvem atrasos substanciais, relativo às velocidades de execução.

Dado a importância das áreas de utilização de SDs, um erro de projeto/programação pode ter conseqüências bastante danosas ou até catastróficas. SDs, até mais do que os sistemas seqüenciais convencionais, devem ser formalmente especificados e verificados para que se tenha certeza de seu correto funcionamento.

II.4 Notas Bibliográficas

A seção sobre computação paralela foi inspirada nos trabalhos de AMORIM *et alii* (1988) e CRICHLOW (1988), dois trabalhos recentes que tratam de várias importantes questões de *software* e *hardware* da computação paralela e distribuída. Usou-se, também, trechos de HAREL (1987), um trabalho bem-humorado cobrindo vários assuntos da computação, e partes do trabalho de ANDREWS *et alia* (1983), um artigo onde são descritos os conceitos centrais ao projeto e a construção de programas concorrentes. A subseção sobre visões de programas concorrentes é adaptado de ALFORD *et alii* (1985) e a subseção sobre linguagens de programação concorrentes foi escrita com o auxílio de STOTTS JR. (1982), um artigo clássico sobre o assunto, PETERSON *et alia* (1983) e CRICHLOW (1988), além das referências que constam no texto. Mais detalhes sobre a linguagem Ada podem ser encontrados no trabalho de VIDAL SILVA (1986), e sobre outras linguagens (por exemplo: PLITS, programação funcional, LISP, CAJOLE, C, etc.) são encontrados em CRICHLOW (1988). Uma extensa revisão da literatura sobre programação concorrente e distribuída acha-se em VELOSO (1985).

A seção sobre SDs foi escrita a partir de trechos de AMORIM *et alii* (1988) e CRICHLOW (1988). Usou-se, também, partes do trabalho de PETERSON *et alia* (1983), KHEI (1988) e VELOSO (1985). O artigo de LORIN (1979) inspirou parte do texto. Grande parte da bibliografia sobre SDs pesquisada pode ser encontrada em um único lugar e em português, no trabalho de KIRNER *et alia* (1988). Outra referência para os conceitos de programação concorrente e distribuída é o livro de MAGALHÃES (1986).

Capítulo III

Especificação Formal de SDs, Lógica Modal e Lógica Temporal

SDs estão sendo cada vez mais usados no dia-a-dia. Suas áreas de aplicação exigem a certeza da correção do projeto, pois erros podem causar grandes perdas financeiras (por exemplo, no caso dos bancos eletrônicos) ou ter graves conseqüências (por exemplo, no caso da monitoração de U.T.I.s ou usinas nucleares). O uso da especificação formal (e verificação) no desenvolvimento de SDs é uma prática que confere segurança a todo o pessoal envolvido na análise e projeto de sistemas, permitindo a confecção de SDs cujo funcionamento não causará surpresas desagradáveis.

A lógica temporal, uma lógica cujo propósito é sistematizar o raciocínio com afirmações que têm um aspecto temporalizado (isto é, sensíveis ao tempo), vem sendo utilizada como uma ferramenta no processo de especificação de programas concorrentes (PNUELI, 1977; MANNA *et alia*, 1981a e 1981b; LAMPORT, 1989). Com a lógica temporal, é possível a modelagem de SDs e o estudo de seus aspectos dinâmicos. Estreitamente relacionada com a lógica modal, ela tem sido estudada há um longo tempo. As fundações formais da lógica temporal (nos seus vários tipos) têm sido estabelecidas durante as últimas três décadas.

Este capítulo possui a seguinte divisão: na primeira seção são abordados vários itens sobre a especificação formal: o que é, suas vantagens, o uso de linguagens de programação convencionais para a especificação, o que se deve usar para especificar, os objetivos da especificação e definições de modelo, ferramenta e métodos. Continuando na primeira seção, é desenvolvido o tema da especificação formal de SDs: suas propriedades,

os modelos mais usados na literatura para especificar SDs, os problemas com esses modelos, o uso da lógica temporal para a especificação formal de SDs, o modelo de SDs da lógica temporal, os métodos requeridos pela lógica temporal e as vantagens e limitações da lógica temporal.

Na segunda seção, discorre-se sobre a lógica modal, fazendo uma introdução ao assunto e apresentando as noções modais básicas e o arcabouço modal. Após isso, são exibidos alguns sistemas formais modais e como eles se relacionam. Na terceira seção, é apresentada a lógica temporal: razões para sua proposta, um histórico, o relacionamento entre as lógicas modal e temporal, o arcabouço temporal e como se relacionam os conceitos de estado e instante. Em seguida, as seções finais de conclusões e notas bibliográficas.

III.1 A Especificação Formal de SDs

Dado a importância da especificação formal para a análise e projeto de SDs, é conveniente a investigação de questões correlatas, tais como os modelos de SDs mais usados, seus problemas, a ferramenta adequada, etc. Essa seção se propõe a abordar vários assuntos da área, tecendo alguns comentários e dando algumas definições.

III.1.1 O Que é Especificação Formal

O significado mais abrangente do termo *especificação* é qualquer informação que ajude a descrever o objeto sendo especificado. Entretanto, seu uso próprio na computação é mais restrito, envolvendo o conceito-chave "abstração". Uma especificação deve afirmar todos os requisitos que um objeto deve satisfazer e nada mais. Para ser abstrata, ela deve separar o essencial do supérfluo, restringindo-se àquele (BOCHMAN, 1985).

Pode-se ver uma especificação como um contrato entre o usuário de um sistema e seu implementador. Este contrato deve deixar bem claro as intenções do usuário para com o sistema, isto é, deve explicitar tudo o que o usuário quer do sistema. Essa informação deve ser suficiente para que o implementador execute sua tarefa de confecção do sistema. Para o usuário, a especificação deve ser o parâmetro de julgamento da implementação: se o usuário não for atendido em algum item da especificação, ele pode argumentar, com base no contrato, que a outra parte (o implementador) não cumpriu sua obrigação. Por

outro lado, a especificação é uma garantia para o trabalho do implementador: se todos os itens forem atendidos pela implementação, não há razão para queixas do usuário. Em princípio, uma vez que esse contrato foi firmado, o usuário e o implementador não têm necessidade de posterior comunicação.

Especificação é uma descrição das características que um sistema ou um programa deverá apresentar. A especificação pretende ser rigorosa e minuciosa. Especificação formal é a especificação cuja sintaxe (forma) e semântica (significado) são formalizados – é um conjunto de afirmações feitas na linguagem de algum sistema formal. Linguagens naturais são muito expressivas e muito imprecisas. Pode-se expressar qualquer propriedade de um programa em português ou em alemão, mas é difícil certificar-se que alguém mais entenderá exatamente o que se quiz dizer. Linguagens formais não são muito expressivas, mas elas são precisas. Não é possível dizer tudo em uma linguagem formal, mas o que se diz tem seu significado completamente livre de ambigüidades (LAMPORT, 1983b).

Para a especificação ser formal, a questão de saber se uma implementação satisfaz a especificação deve ser redutível à questão de saber se uma afirmação é demonstrável em algum sistema dedutivo. Para demonstrar que ele cumpriu os termos do contrato, o implementador vale-se da lógica (LAMPORT, 1989).

Um sistema (um conjunto de programas, um conjunto de instruções ou uma máquina) pode ter suas propriedades ou comportamento especificados por afirmações feitas na linguagem $\mathcal{L}$ de um sistema formal. Uma linguagem formal $\mathcal{L}$ é aquela cuja semântica é precisa, isto é, toda afirmação feita na linguagem tem um significado preciso. Todas as linguagens de programação são formais neste sentido, o significado sendo determinado pelo compilador. Nenhuma linguagem natural goza de tal propriedade.

Uma especificação Ψ em um sistema formal Σ consiste em um conjunto de afirmações em $\mathcal{L}$, isto é, $\Psi \subset \mathcal{L}$. Alguns dos símbolos destas afirmações podem ser usados para representar coisas no sistema real sobre o qual se deseja falar, tipicamente objetos que exibem algum comportamento. Outros termos na linguagem proporcionam "mecanismos observacionais" que são necessários para se falar sobre os objetos e seu comportamento. Por exemplo, seja $\mathcal{L}_e$ uma linguagem formal possuindo três símbolos: o, $\bullet$ e $\triangleright$. Seja ainda um sistema a ser especificado nessa linguagem composto por um *buffer* de uma única

posição e dois processos, um que produz informação a ser colocada no *buffer* e outro que consome essa informação e seja Ψ_e uma especificação formal feita em $\mathcal{L}_e$ consistindo de uma única afirmação (seqüência de símbolos):

$$\circ \triangleright \bullet \qquad (\dagger)$$

O símbolo $\circ$ é usado para representar o fato de que a posição única do *buffer* está vazia. O símbolo $\triangleright$ pode ser entendido como uma representação para o fato de que foi feito uma solicitação para colocar algo no *buffer* e o símbolo $\bullet$ representa o fato de que a posição do *buffer* está ocupada. A afirmação ($\dagger$) pode ser entendida como uma representação dos fatos acima, ilustrando o comportamento do sistema.

III.1.2 Vantagens da Especificação Formal

O desenvolvimento de um sistema ou programa encontra problemas de várias naturezas. Ações conflitantes, limitações físicas de equipamentos e o tratamento de erros são exemplos de problemas recorrentes em muitas das fases da confecção de sistemas. Muitas vezes, esses problemas são detectados somente na fase de implementação, quando a solução é várias vezes mais cara do que se fossem detectados em fases anteriores. A prática da especificação formal disciplina o projetista pois o obriga a considerar esses problemas em fases anteriores à implementação. Desde que a especificação pretende ser rigorosa e minuciosa, é inevitável que os problemas sejam encontrados e que decisões sejam tomadas. Essas decisões podem, às vezes, ser não decidir, adiando o detalhamento para etapas mais próximas da implementação.

Vantagens adicionais de uma especificação formal, especialmente em um arcabouço matemático, são (MULLERY, 1985):

- permite a realização de experimentos matemáticos (na forma de provas e hipóteses) no objeto especificado;
- os resultados destes experimentos podem ser usados no processo de projeto mais cedo (e de forma muito mais barata) do que seria possível de outra forma;
- uma especificação formal oferece uma maior possibilidade de, efetivamente, ser mostrada como sendo satisfeita por implementações propostas.

III.1.3 Linguagens de Programação como Especificação

O conceito de linguagem formal foi exemplificado pelas linguagens de programação. Deve se explicado, entretanto, por que as linguagens de programação convencionais não são aceitas como linguagens para especificação.

Uma especificação funcional deveria afirmar O QUÊ um sistema deve fazer, e não COMO ele deve funcionar. Esta é a primeira razão para não se empregar linguagens de programação convencionais na tarefa de se escrever especificações funcionais, pois programas são exatamente receitas de COMO executar uma tarefa (especificação implícita) e o uso de linguagens de programação forçaria a consideração de detalhes que deveriam ser adiados da análise funcional para a fase de projeto.

O segundo problema com as linguagens de programação convencionais diz respeito a sua coleção de tipos de dados. O que é requerido de uma especificação é a capacidade de se escolher tipos de dados que se adeqüem ao problema do modo mais natural possível. O uso de uma linguagem de programação para especificações novamente forçaria a introdução de questões de projeto na análise funcional.

III.1.4 O Que Usar?

Se não é adequado usar linguagens naturais nem linguagens de programação para a especificação formal, o que, finalmente, devemos usar? A fonte original da notação formal é a matemática e é para a matemática que devemos nos voltar para obter uma base íntegra para linguagens de especificação.

Uma boa especificação funcional deve ser, primeiramente, menor do que qualquer programa que puder ser escrito para atender à tarefa. Segundo, ela deve informar o usuário aquilo que ele deseja saber — o comportamento externo é seu interesse, não o projeto. A concisão é uma das virtudes desta abordagem.

O uso da notação formal foi primeiramente motivado pelo desejo de se obter precisão. Esse objetivo pode ser atingido adotando a notação de áreas estabelecidas como a matemática. As lógicas (proposicional, de primeira ordem, modal, etc.), por exemplo, têm seu significado precisamente definido em termos de conceitos matemáticos.

III.1.5 Objetivos de Uma Especificação

A primeira afirmação precisa do que um sistema fará é a especificação formal. Muitas especificações tornam-se desnecessariamente complexas por incluírem detalhes implementacionais. É crucial que uma especificação se limite aO QUÊ o sistema faz. Portanto, somente aqueles aspectos de um sistema visíveis por um usuário externo devem ser descritos, evitando detalhes de COMO a implementação deve funcionar, sendo o COMO objetivo da fase de implementação. Mesmo em situações nas quais a mesma pessoa ou grupo especifica formalmente e implementa, deve-se assegurar que a distinção entre as duas atividades seja preservada.

A especificação formal deve servir a dois objetivos: ela é a base do contrato firmado entre o usuário e a equipe de desenvolvimento e, como tal, ela deve afirmar precisamente o que aquele espera deste. Para que tal objetivo seja atingido, a especificação deve ser não-ambígua, não-contraditória e completa. Os primeiros dois requisitos são óbvios e o objetivo da completude se refere somente às suas características externas. Deve-se, entretanto, adicionar à lista de requisitos algumas considerações sobre a usabilidade: uma extensão razoável, estruturada para facilitar a localização de informação e escrita em uma linguagem que o usuário possa aprender.

A especificação formal deve também providenciar o ponto de referência para o projeto — o responsável pelo desenvolvimento deve ser capaz de relacionar o projeto escolhido com a especificação de uma maneira que mostre que aquele está correto.

Finalmente, pode-se usar a especificação como um modelo para o sistema antes que este seja construído. Este é o objetivo mais importante e também o mais difícil de ser atingido. Como uma planta de arquiteto, o modelo torna possível experimentar e mudar a especificação de modo bem mais barato do que quando o sistema já está implantado.

Este modelo é o instrumento para se pensar — a notação é o instrumento para se escrever. Deve-se ter uma notação na qual registrar o modelo e vale a pena gastar algum tempo na notação. Tem sido defendido que a matemática é uma boa fonte de notação: a razão para se adotar notações como a lógica e a teoria dos conjuntos é que elas proporcionam uma abreviação/codificação conveniente para descrever coisas. Ao olhar para a notação somente como uma notação, entretanto, corre-se o risco de subestimar seu

valor real: uma notação apropriada pode ser tão mais fácil de se manipular que é possível, com sua ajuda, resolver problemas previamente inconcebíveis.

III.1.6 Modelos para a Especificação

Por *modelo* referimo-nos a uma abstração de um objeto (um programa, um SD ou uma máquina) para um outro objeto de menor complexidade, preservando certas propriedades do objeto original. Um modelo representa uma visão simplificada do objeto real, permitindo uma análise rigorosa deste. Modelos possuem como objetivo definir precisamente propriedades específicas ou características de um objeto sob consideração de ser construído ou analisado, proporcionando a fundamentação para verificar essas propriedades (COHEN *et alii*, 1986).

Modelos diferentes são usados para especificar propriedades diferentes; alternativamente, para expressar uma propriedade específica de um sistema, deve-se escolher dentre os modelos o que melhor se adeque para representar e estudar aquela propriedade (ALFORD *et alii*, 1985). No caso de SDs, alguns modelos seriam:

- Um grafo não-orientado conexo $G = (N, E)$ onde $N = \{p_1, \ldots, p_n\}$ é o conjunto de processos (nós) e $(p_i, p_j) \in E$ se, e somente se, p_i e p_j compartilham um recurso $R = \{r_1, \ldots, r_w\}$ (AMORIM *et alii*, 1988);
- O modelo subjacente à visão espaço-tempo (seção II.1.7.1), no qual um SD é visto como uma coleção de processos e cada processo consiste em um conjunto de eventos com uma ordenação total *a priori* (uma seqüência de eventos). Nesse modelo, o SD é representado por um "diagrama espaço-tempo" (LAMPORT, 1978).
- O modelo subjacente à visão das intercalações (seção II.1.7.2), no qual um SD é abstraído para uma seqüência de instantâneos (estados) globais, representando a situação do sistema, e ações atômicas que causaram a transição de um estado global para outro (LAMPORT, 1980; CHANDY *et alia*, 1985);

III.1.7 Ferramentas para a Especificação

Por *ferramentas* referimo-nos aos instrumentos ou aparatos empregados sobre um modelo no sentido de se obter a especificação formal. A união da ferramenta com o modelo permite o estudo do objeto real abstraído de detalhes irrelevantes e concentrado naqueles aspectos importantes.

Ferramentas diferentes podem ser usadas em um mesmo modelo, o critério de escolha sendo quais propriedades estudar. Um exemplo de ferramenta para o modelo subjacente da visão espaço-tempo (vide capítulo II) é a linguagem de especificação baseada em eventos (EBS - *Event Based Specification Language*) de CHEN *et alia* (1983), uma extensão ao cálculo de predicados de primeira ordem. Essa ferramenta oferece a sintaxe e a semântica de um sistema formal para especificar formalmente e verificar SDs.

Ferramentas para a especificação podem ser divididas naquelas que dão suporte ao processo de especificação e aquelas que dão apoio à geração ou execução de código que implemente a especificação (por exemplo, MANNA *et alia* (1984)). A maioria das discussões detalhadas sobre ferramentas tem concentrado-se nas ferramentas do último tipo. Pode-se ver em outros trabalhos sobre técnicas formais (por exemplo, LAMPORT (1983) e OWICKI *et alia* (1982)), que um forte apoio ferramental é desejável e possível.

III.1.8 Métodos para a Especificação

Por *métodos* referimo-nos ao procedimento ou modo de uso da ferramenta (agindo sobre o modelo) visando a obtenção da especificação ou a condução da verificação. O relacionamento entre modelo, ferramenta e método pode ser representado pela figura III.1.

Quando a ferramenta é um sistema formal, diz-se que o método que a utiliza, bem como a especificação resultante, são também formais.

Um *método* para a especificação consiste em um procedimento determinando as atividades a serem realizadas visando a obtenção de uma especificação. Métodos para a especificação podem dirigir-se a duas questões: *aquisição* de informação e *expressão* da informação que foi adquirida.

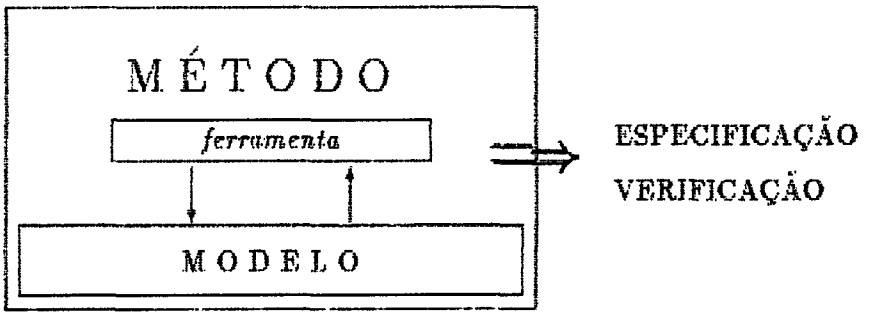

Figura III.1: Representação do relacionamento entre método, ferramenta e modelo

Aquisição significa mais do que simplesmente escrever uma lista de afirmações sobre o sistema e seu ambiente. Significa estabelecer e policiar o escopo da informação, obter a informação na "melhor" ordem, usando técnicas conhecidas para a manipulação de aspectos específicos do sistema, provocando decisões quando são necessárias e lidando com omissões forçadas.

Expressão significa mais do que simplesmente estabelecer regras sintáticas e uma terminologia consistente. Significa usar uma terminologia consistente que permita a expressão de toda a especificação. A expressão exige que a terminologia adotada (sintaxe) possua uma semântica formalmente definida. Em muitos casos, a expressão pode também exigir que subconjuntos da expressão formal completa possa ser mapeada em uma ou mais terminologias menos completas para leitores com menor treinamento no uso de notações formais (MULLERY, 1985).

III.1.9 Especificação Formal de SDs

Sistema distribuídos precisam, até mais urgentemente do que os sistemas convencionais, ser especificados formalmente. Suas qualidades inerentes de concorrência, assincronia e não-determinismo requerem estudos mais completos sobre o futuro comportamento de um sistema a ser implementado. A necessidade de um modelo formal no qual seja possível raciocinar de forma precisa, antecipando resultados e prevendo situações indesejáveis é até mais preeminente dado a natureza dinâmica dos sistemas distribuídos.

Entretanto, sistemas concorrentes são mais difíceis de especificar e analisar do que os sistemas seqüenciais convencionais porque eles exigem a conceitualização não somente dos processos seqüenciais, mas também das complexas interações entre eles (LAUER, 1983). A independência entre os processos do SD e a ausência de um relógio global para o sistema como um todo pioram ainda mais o quadro.

III.1.9.1 Propriedades de SDs

Várias (e tristes) experiências indicam que um programa concorrente pode resistir aos mais severos escrutínios sem revelar seus erros. A única maneira de poder ter certeza que um programa concorrente faz o que se pensa que ele faz é provar rigorosamente que ele o faz. Quando se estuda SDs, geralmente se está interessado na análise de dois tipos de propriedades (OWICKI *et alia*, 1982; LAMPORT, 1983):

- *Propriedades de segurança* — aquelas afirmando que algo ruim nunca acontece, isto é, o SD nunca entra em um estado não-aceitável;
- *Propriedades de vida* — aquelas afirmando que algo bom eventualmente acontece, isto é, o SD eventualmente entra em um estado desejável.

Algumas propriedades de segurança são:

- *Correção Parcial*: se o SD começa com a pré-condição verdadeira, então ele nunca pode terminar com a pós-condição falsa;
- *Ausência de Bloqueio Perpétuo*: o SD nunca entra em um estado no qual nenhum progresso posterior é possível;
- *Exclusão Mútua*: dois processos diferentes nunca estão em suas seções críticas simultaneamente.

A única propriedade de vida que tem recebido um tratamento formal cuidadoso é a da terminação de programa. Entretanto, programas concorrentes são capazes de muitos outros "pecados por omissão" do que simplesmente falha na terminação. Sem dúvida alguma, para muitos programas concorrentes — sistemas operacionais são um exemplo

maior -- a terminação, conhecida por "crashing", é indesejável e quer-se provar que isso não acontece. Para tais programas outros tipos de propriedades de vida são importantes, por exemplo:

- cada pedido para um serviço eventualmente será respondido;
- uma mensagem eventualmente atingirá seu destino;
- um processo eventualmente entrará em sua seção crítica;

Propriedades de vida envolvem o conceito temporal "eventualmente". Uma maneira de formalizar este conceito é usando fórmulas do cálculo de predicados contendo uma variável "tempo" (CHEN *et alia*, 1983). Entretanto, a introdução explícita do tempo desta maneira leva a fórmulas complicadas que tendem a obscurecer as idéias subjacentes. Como veremos, a lógica temporal evita isso, pois incorpora implicitamente o conceito do tempo.

Probidade

Um dos conceitos mais importantes em processamento concorrente é a *probidade*. Probidade (do inglês *fairness*) significa que todo processo tem uma chance de fazer progresso, sem importar com o que os outros processos façam. A probidade é garantida por um sistema verdadeiramente concorrente no qual cada processo é executado em seu próprio processador sem comunicação com outros processadores: um processo não pode parar a execução física de um outro processo executando em um processador diferente. Entretanto, SDs nos quais o progresso de um processo depende do recebimento de uma mensagem podem ou não apresentar probidade, caso a entrega da mensagem seja eventualmente feita ou para sempre adiada. Sistemas multiprogramados, nos quais um único processador é compartilhado por vários processos, podem ou não proporcionar a probidade, dependendo do algoritmo de escalonamento usado para alocação de processos.

III.1.9.2 Modelos de SDs

Nesta seção, é dado uma visão geral de alguns modelos selecionados comumente usados como a fundamentação para a especificação de SDs (ALFORD *et alii*, 1985).

Funções Matemáticas

A especificação de SDs via funções matemáticas consiste na especificação de um domínio de entrada (por exemplo, um conjunto de variáveis de entrada), um domínio de saída (por exemplo, um conjunto de variáveis de saída) e uma regra para transformar as entradas nas saídas. Para a transformação ser uma função, ela deve sempre produzir as mesmas saídas para o mesmo conjunto de dados de entrada.

Há vários aspectos relevantes das funções matemáticas que afetam sua aplicabilidade como um modelo para especificar SDs. Primeiro, uma função matemática não é um algoritmo — uma função *pode ser* especificada providenciando-se um algoritmo, mas é uma questão de projeto conceber um algoritmo que executa uma transformação dentro de uma exatidão especificada.

Um segundo aspecto relevante das funções matemáticas é que elas podem ser "decompostas", isto é, especificadas por uma combinação de lógica e funções de nível inferior. Isso tem o efeito de especificar uma função em termos de uma estrutura de funções, e portanto, especificando uma abordagem algorítmica. A estrutura pode ser descrita em termos de tabelas de decisão, pré-condições e pós-condições ou fluxogramas. Desde que deseja-se especificar uma transformação e não um algoritmo, no sentido de separar requisitos do projeto, o uso de uma função matemática parece ser desejável.

Entretanto, essa abordagem é limitada porque uma função matemática inerentemente não possui memória — dado uma entrada, ela produz uma saída mas não guarda nenhuma informação. Isso significa que uma coleção de funções matemáticas não pode ser usada para especificar o conteúdo requerido da informação de SDs. As tentativas de usar a recursão para superar essa ausência de memória fizeram a complexidade da descrição da função crescer exponencialmente.

Em vista dessa limitação, parece que uma função matemática é um ingrediente necessário, mas não um modelo suficiente para a especificação de um SD. Esse modelo, sozinho, só pode ser usado para especificar funções que não exigem memória; o modelo deve ser aumentado para se referir aos problemas para os quais os SDs são usados mais comumente.

Máquina de Estados Finitos (MEF)

O conceito das Máquinas de Estados Finitos (MEFs) parece ser feito sob medida para a especificação do processamento de dados. Essencialmente, uma MEF é composta por um conjunto de entradas X, um conjunto de saídas Y, um conjunto de estados S, um estado inicial s_0 e um par de funções f e g que são usadas para especificar as saídas e transições de estado que ocorem como um resultado de uma entrada. A função de transformação f especifica as saídas que resultam de uma entrada quando a MEF está em cada um de seus possíveis estados e a função de transição de estados g especifica o próximo estado que resulta de uma entrada para cada estado possível. Em outras palavras:

- $X = \{x_i\}$ é o conjunto de entradas;
- $Y = \{y_i\}$ é o conjunto de saídas;
- $S = \{s_j\}$ é o conjunto de estados;
- s_0 é o estado inicial;
- $f : X \times S \rightarrow Y$;
- $g : X \times S \rightarrow S$

A figura III.2 ilustra este modelo.

Este modelo parece ideal para a especificação de sistemas de processamento de dados como um todo porque pode-se associar as entradas e saídas dos processadores com aquelas de uma MEF, e associar os valores das posições de memória com o estado e o código com as transformações. A fundamentação para a análise é a de que as transformações para todas as entradas devem ser especificadas para todos os estados de saída, todas as saídas

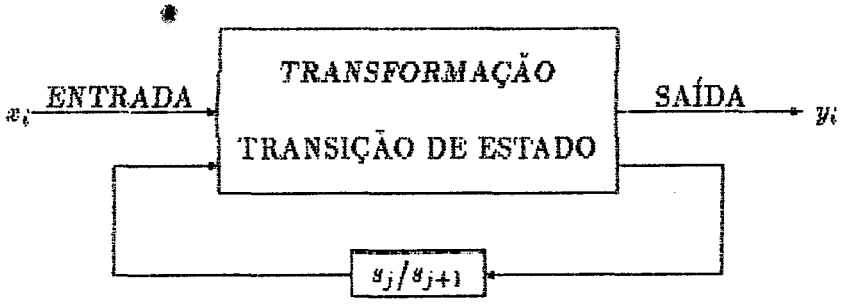

Figura III.2: Modelo de Máquina de Estados Finitos (MEF)

devem ser produzidas para alguma combinação de entrada e estado e que o estado inicial deve ser especificado.

O modelo da máquina de estados finitos é um dos modelos fundamentais da ciência da computação. Ele tem sido usado para a especificação e validação de protocolos de comunicação. Ele está na base de quase todas as técnicas de provas, tendo sido incorporado ao método-padrão IBM para desenvolvimento de software.

Entretanto, há duas limitações fundamentais para o uso deste modelo na especificação de SDs. Primeiro, o modelo da MEF "serializa", de modo inerente, toda a concorrência subjacente que gostaríamos de explorar com múltiplos processadores. O modelo assume explicitamente que todo o processamento de uma entrada é completado antes que a próxima entrada chegue. Como resultado disso, um dos erros mais comuns no uso do modelo de MEF para a especificação de processos concorrentes é que estados de colisão (isto é, estados nos quais duas chegadas ocorrem ao mesmo tempo) são negligenciados; isso resulta em provas de correção para um modelo inapropriado. Tentativas vêm sido feitas para estender o modelo da MEF para a descrição de concorrência usando múltiplas MEFs paralelas.

Uma segunda limitação fundamental do modelo da MEF é que ele atualmente não possui nenhuma maneira inerente de tratar da complexidade: o modelo é "achatado", isto é, se for necessário 2000 estados para descrever um processo, a MEF não será inteligível sem uma estruturação adicional. A descrição da função que mapeia uma entrada e um

estado em uma saída e o estado atualizado pode tornar-se extremamente complexo de se especificar, mesmo para pequenos problemas. Algum mecanismo é necessário para decompor a transformação e as funções de transição de estado em pedaços menores e mais inteligíveis. Deve ser notado que a MEF estende o conceito de função matemática com o conceito de estado no sentido de descrever como as seqüências de entradas são mapeadas em seqüências de saídas e, portanto, os problemas de complexidade encontrados na representação de funções matemáticas ainda permanecem.

A discussão acima sugere que o modelo da MEF não é suficiente para a especificação de SDs, mas o modelo deve ser um ingrediente necessário numa técnica descritiva para esse fim. O modelo da MEF proporciona a fundamentação para propriedades ligando entradas, saídas e o processamento e proporciona o elo para técnicas de prova de programas. Entretanto, para se obter uma fundamentação prática para tais especificações, os problemas de complexidade devem ser resolvidos. Uma maneira de se conseguir isso é usando o conceito de uma *hierarquia de funções*.

Hierarquia de Funções

A abordagem mais comum para a especificação de requisitos de processamento de dados é o uso de uma hierarquia de funções. Começamos identificando uma função do sistema que possui um domínio de entrada consistindo de todos os possíveis conjuntos de seqüências de dados de entrada e possui o contra-domínio de todas as seqüências possíveis de dados de saída. Essa função do sistema é, então, decomposta em uma coleção de funções interagentes as quais podem ter relacionamentos de entrada/saída; essas funções podem, por sua vez, ser decompostas em uma coleção de funções interagentes de nível mais baixo.

Os benefícios desse modelo advêm de sua habilidade em organizar uma grande massa de dados e checar a consistência de entradas e saídas entre uma função de alto nível e sua decomposição em um número de funções interconectadas de nível inferior. O processo de decomposição pode ser repetido tantas vezes quanto for necessário para se obter uma hierarquia de funções com muitos níveis. Isso trata explicitamente com os problemas de representar concorrência e complexidade, mas às custas de ambigüidade — só o fluxo de dados é representado, não o fluxo de controle.

A limitação fundamental desse modelo é a de que ele não diferencia entre uma função f que recebe x *e* z, x *ou* z ou x *seguido de* z. Portanto, o seqüenciamento e informação de condição não são expressos pelo modelo e não podem ser verificados usando o modelo. De modo semelhante, quando uma função f é decomposta nas funções f_1, f_2 e f_3, o modelo não especifica se elas são simultâneas ou executadas em uma seqüência específica. Isso limita a utilidade deste modelo para a especificação de condições e seqüências de funções.

A discussão acima sugere que qualquer modelo usado para descrever SDs complexos devem possuir a hierarquia de funções como um ingrediente necessário, mas esse modelo é insuficiente para representar todas as propriedades requeridas de tais sistemas. Em outras palavras, o modelo "achatado" da MEF oferece precisão para a descrição do processamento de seqüências de entradas, mas é difícil de usar para descrever grandes sistemas; enquanto a hierarquia de funções pode descrever a concorrência e propicia visibilidade no conteúdo de grandes sistemas, mas carece de precisão. Um método preciso de descrever concorrência se faz necessário e isso é proposto pelas *Redes de Petri*.

Redes de Petri

Uma rede de Petri (ou rede Petri) propicia uma notação para representar formalmente o seqüenciamento e a concorrência e para identificar e resolver ambigüidades potenciais. Uma rede de Petri consiste em um grafo direcionado que alterna dois tipos de nós, os *lugares* e as *transições*, conforme a figura III.3. Os *lugares*, representados por círculos ou retângulos, definem locais no grafo onde os símbolos (*tokens*) podem residir. Eles são ligados por arcos às *transições*, representadas por barras horizontais. As transições governam o movimento de símbolos de lugares precedentes para lugares subseqüentes se a condição da transição for satisfeita. As pré-condições da transição podem especificar que ela pode mover símbolos se uma combinação especificada de símbolos existir nos lugares imediatamente precedentes; a pós-condição da transição pode especificar a movimentação de símbolos para alguma combinação dos lugares subseqüentes. Para redes de Petri simples, as pré-condições são limitadas à especificação de todas ou uma (isto é, a transição ocorre se todos os lugares que levam à transição possuem um símbolo ou se somente um possui um símbolo); a pós-condição é limitada à especificação de se um símbolo é gerado

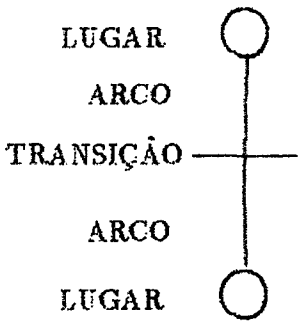

Figura III.3: Representação da Rede de Petri

para todos os lugares que advêm da transição ou somente um. Quando a pré-condição é satisfeita, a transição "dispara" e símbolos são removidos dos lugares precedentes e gerados para os lugares subseqüentes. Versões mais complexas de redes de Petri permitem que combinações arbitrárias de arcos de entrada e de saída sejam especificados como pré- e pós-condições. A rede de Petri pode ser usada para especificar precisamente conceitos de seqüência, seleção, concorrência e sincronização.

Muitas propriedades interessantes de sistemas concorrentes podem ser definidas e verificadas usando a rede de Petri. Por exemplo, uma rede de Petri pode ser analisada para determinar se o término correto sempre ocorrerá (por exemplo, para uma única entrada de símbolo, somente um sairá, e nenhum símbolo será deixado no grafo quando ele existir), ou se a possibilidade de bloqueio perpétuo (*deadlock*) existe (por exemplo, a rede pode entrar em um estado a partir do qual nenhuma transição é possível, ou onde a rede está em "laço" (*loop*).

A despeito de suas vantagens, há duas limitações para o uso de uma rede de Petri para descrever SDs. Primeiro, como o modelo da MEF, redes de Petri são "achatadas", isto é, a rede aparece em um nível, essencialmente. Entretanto, foi desenvolvido o conceito de sub-rede Petri consistindo em uma hierarquia de sub-redes para manipular redes grandes e complexas. Quando feito propriamente, isso não afeta a semântica da rede de Petri, mas habilita a resolver um grande problema através de vários problemas menores.

Uma limitação mais significante ao uso das redes de Petri é que elas especificam

somente o fluxo de controle, esquecendo o fluxo de dados. Mesmo que as condições para as transições possam ser especificadas em termos de valores de dados, a semântica para mudar ou salvar valores de variáveis não é uma parte inerente da rede de Petri, portanto devemos procurar em outro lugar a modelagem desses conceitos. Tentativas têm sido feitas para corrigir esse problema, mas os detalhes não foram completamente elaborados. Devemos notar que redes de Petri são uma das ferramentas para a especificação e validação de protocolos de comunicação.

III.1.9.3 Os Problemas com os Modelos

Os modelos que consideram explicitamente os estados de um SD na especificação formal (por exemplo, MEFs, Redes de Petri e lógica de primeira ordem) possuem duas sérias desvantagens. À medida que o número de possíveis estados aumenta, analisar todas as interações torna-se impraticável (SCHWARTZ *et alia*, 1981; CHEN *et alia*, 1983). Além disso, a análise rigorosa do possível comportamento, quando é praticável, garante a *segurança*, mas não garante a *vida* do sistema (vide seção III.1.9.1, acima). Propriedades de vida, isto é, requisitos de que certos eventos eventualmente ocorrem, são difíceis de afirmar ou provar usando uma abordagem assim.

Um problema mais geral no uso de um modelo abstrato é a dificuldade de separar requisitos de implementações. Enquanto é desejável dizer os requisitos que o sistema deve atender, deixando o COMO para o implementador, os modelos operacionais especificam os requisitos de um sistema dando uma implementação abstrata. Não há indicação de quais aspectos do modelo devem ser rigorosamente seguidos e quais aspectos meramente ilustram a funcionalidade. No caso de um modelo abstrato para um programa seqüencial, isso não causa nenhum problema particular. Qualquer implementação com comportamento de entrada/saída idêntico (isomórfico) é aceitável. Para um sistema concorrente torna-se mais difícil identificar aqueles aspectos dos dados e controle que podem ser modificados sem afetar o comportamento do sistema. Uma especificação para um SD deve ser até mais explícita quanto ao que é necessário para uma operação correta do sistema.

Para programas seqüenciais, as técnicas de especificação algébricas ou axiomáticas (MANNA, 1978; MANNA *et alia*, 1978; COHEN *et alii*, 1986) proporcionam a abstração

necessária para se afirmar propriedades de um programa sem dar uma implementação. Infelizmente, técnicas algébricas/axiomáticas convencionais, definindo propriedades de um estado particular do programa, não são adequadas para programas concorrentes: propriedades dependentes do tempo em SDs tais como concorrência ou exclusão mútua são difíceis, senão impossíveis, de especificar nessas abordagens. Além disso, para raciocinar-se sobre interações entre os processos de um SD, devem ser especificadas propriedades de cada processo através de toda sua execução ao invés de simplesmente seu comportamento de entrada/saída.

III.1.9.4 A Especificação Formal de SDs com a Lógica Temporal

Foi visto, em seções anteriores, que deve-se escolher o modelo para representar o SD, bem como a ferramenta a ser usada sobre o modelo, em função de quais aspectos e/ou propriedades a serem estudados. Mostrou-se também que, na análise de SDs, geralmente se está interessado em dois tipos de propriedades: as de segurança e as de vida. O problema da especificação formal de SDs consiste, então em *encontrar* uma ferramenta e seu modelo subjacente que representem convenientemente as propriedades citadas, permitindo o estudo e a verificação.

Pode-se argumentar que o termo *conveniente* é muito subjetivo para orientar uma decisão, entretanto, alguns itens podem tornar essa escolha mais objetiva:

- possuir uma notação sucinta — a linguagem para a especificação deve ser mais concisa do que qualquer implementação a ser feita;
- permitir ao usuário definir o nível de abstração desejado — à medida que se conhece mais sobre o problema, deseja-se detalhar mais a especificação: a ferramenta deve permitir ao usuário escolher qual o nível de abstração que o satisfaz e mudar desse nível para outros;
- oferecer meios de verificação — apesar de a verificação ser uma atividade distinta da especificação, é interessante que a ferramenta tenha, à disposição do usuário, condições de verificar se a especificação está correta, consistente ou se corresponde à descrição informal. Essa facilidade de verificação pode ser, por exemplo, um provador automático de teoremas ou um método manual de provas.

A lógica temporal (PRIOR, 1957; RESCHER *et alia*, 1971) é uma ferramenta que satisfaz os três itens acima, além de ser adequada para o estudo das propriedades mencionadas, surgindo como um candidato natural para a especificação formal de SDs.

III.1.9.5 O Modelo de SD Subjacente à Lógica Temporal

A lógica temporal é um sistema formal que pode ser usado para a especificação e estudo de SDs. Adotando a visão das intercalações (vide capítulo II), seu modelo subjacente é uma seqüência discreta de *estados globais* ligados por ações atômicas. O estado global representa a situação do sistema em instantes distintos — uma seqüência de instantâneos (fotografias) do SD, descrevendo sua execução.

Cada estado consiste de um possível instantâneo obtido no meio de uma execução, contendo toda a informação necessária para prosseguir com a execução daquele ponto. Portanto, o estado deve descrever o valor de cada variável dos processos, o conteúdo dos canais de comunicação, o valor do contador de cada processo, etc. Uma ação é uma ação atômica específica de um processo — por exemplo, a execução de algum comando de atribuição ou o envio de uma mensagem. Os comportamentos no modelo representam todas as possíveis execuções do programa.

É importante observar que o paralelismo é modelado como uma intercalação de ações atômicas. Cientistas da computação geralmente sentem que alguma coisa é perdida através da seqüencialização de um programa paralelo da maneira feita, e que deveria-se usar uma ordenação parcial entre as ações (visão espaço-tempo — vide capítulo II). Entretanto, enquanto falarmos apenas de propriedades de segurança e vida, não há perda de generalidade em considerar seqüências de ações totalmente ordenadas. O modelo inclui todas as seqüências possíveis, e uma ordenação parcial é completamente equivalente ao conjunto de todas as ordenações totais consistentes com ela. Por exemplo, seja um SD com dois processos i e j. O processo i pode ser caracterizado pela seqüência de símbolos $\alpha e_j \alpha'$, na qual α e α' representam a execução de comandos (ou seqüência de comandos) "internos" ao processo i, isto é, não são envios nem recebimentos de mensagens, e e_j representa a execução de um comando de envio de mensagem para o processo j. O processo j, de maneira análoga, pode ser caracterizado pela seqüência $\beta r_j \beta'$ na qual β e β' são semelhantes a α e α'

(não são envios nem recepções de mensagens) e r_j representa o recebimento da mensagem enviada por i. A ordenação parcial descrita por essas seqüências é consistente com todas as ordenações totais que apresentem e_i precedendo r_j, pois o envio de uma mensagem deve sempre preceder seu recebimento. Algumas ordenações totais consistentes seriam: $\alpha\boxed{e_i}\alpha'\beta\boxed{r_j}\beta'$, $\alpha\beta\boxed{e_i}\boxed{r_j}\alpha'\beta'$, $\beta\alpha\boxed{e_i}\boxed{r_j}\beta'\alpha'$ e $\beta\alpha\boxed{e_i}\alpha'\boxed{r_j}\beta'$, todas preservando a precedência de r_j por e_i.

A hipótese real implícita no modelo é a existência de ações atômicas, e isso é feito em virtualmente todos os modelos de paralelismo. A lógica temporal exige apenas que, em algum nível suficientemente baixo — talvez ao nível de micro-código — a execução do programa seja acuradamente descrita por uma coleção de ações atômicas (LAMPORT, 1983).

Para ilustrar como os SDs são modelados pela lógica temporal, seja o sistema composto por dois processos P e Q, conforme a figura III.4. Por questões de simplicidade, será assumido que os canais de comunicação já estão estabelecidos, sendo 100% confiáveis

```
        — P —                    — Q —
        comece                   comece
p0 :      até falso faça   q0 :    até falso faça
          comece                   comece
p1 :        x ← x + 1      q1 :      receba y de P
p2 :        envie x para Q   q2 :    z ← y + 2
          termine                  termine
        termine                  termine
```

Figura III.4: SD composto por dois processos cíclicos P e Q

e que não há *bufferização* de mensagens (as mensagens são entregues imediatamente, isto é, no próximo estado a partir do envio). Os comandos rotulados ($p_i, q_i, i = 0, \ldots, 2$) são somente aqueles executáveis. Os elementos **comece** e **termine** são apenas delimitadores de bloco e por isso não precisam de rótulos. Quando o controle estiver em p_i (ou q_i) é porque o comando correspondente ao rótulo ainda será executado.

O modelo da execução desse SD, pela visão das intercalações, pode ser representado pela figura III.5. Os retângulos são os estados globais do sistema: cada coluna representa os valores dos processos P (coluna esquerda) e Q (coluna direita). O retângulo

superior da coluna traz o valor do contador do processo e o inferior o valor da(s) variável (variáveis).

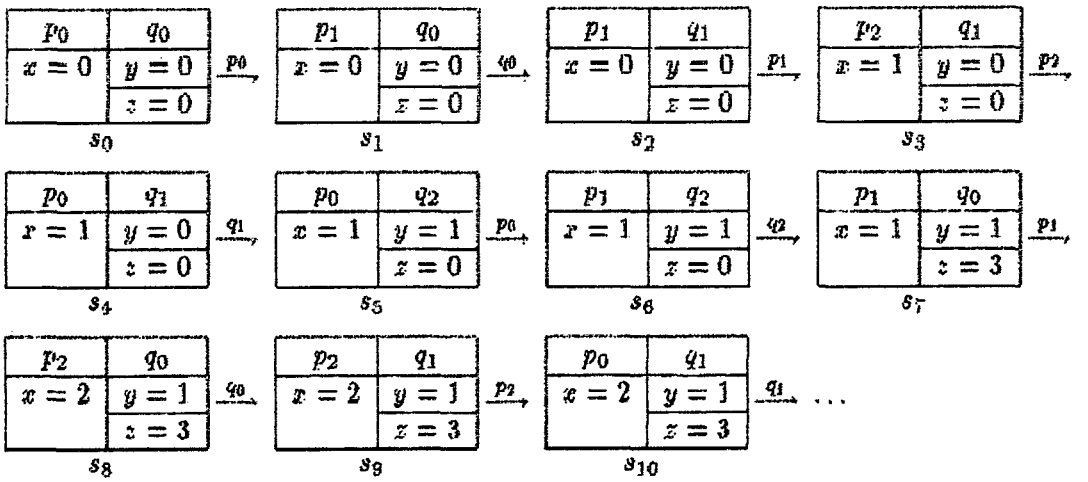

Figura III.5: Representação de execução de um SD

Desse exemplo, apreende-se que até SDs simples podem tornar-se bastante complexos se representados por uma seqüência de estados. Alguns autores (SCHWARTZ *et alia*, 1981; CHEN *et alia*, 1983) sugerem ser este o problema de muitas ferramentas e modelos para a especificação de SDs: a consideração explícita de todos os estados da execução do sistema (vide III.1.9.3). Entretanto, apesar de o modelo subjacente à lógica temporal ser uma seqüência de estados, não há a menção explícita a eles: o formalismo (sintaxe e semântica) assume essa representação, oferecendo meios (os operadores temporais) convenientes de expressar as propriedades temporais (isto é, aquelas propriedades com aspectos sensíveis ao tempo) do SD sem considerar explicitamente qualquer estado. Isso é devido a lógica temporal ter sido desenvolvida para descrever a *ordem* na qual as coisas devem acontecer (e seu relacionamento), ao invés do momento preciso no qual as coisas acontecem.

III.1.9.6 Os Métodos da Lógica Temporal

Pode-se classificar os métodos de especificação da seguinte maneira (LAMPORT, 1983):

- *Métodos Construtivos* — aqueles que descrevem um modelo abstrato dizendo como o SD deve comportar-se;
- *Métodos Axiomáticos* — aqueles que afirmam quais propriedades o SD deve possuir.

Não há maneira de formalizar essa distinção, desde que é possível descrever um modelo abstrato afirmando suas propriedades. Contudo, é útil classificar os métodos dessa maneira.

Os métodos construtivos especificam um programa essencialmente escrevendo outro programa presumivelmente mais simples. Entretanto, programas concorrentes que parecem simples podem exibir um comportamento inesperado. Como estar seguro do entendimento de um programa se sua especificação só diz que ele comporta-se como outro programa que, talvez, não é entendido?

Os métodos axiomáticos, os requeridos pela lógica temporal, tentam superar essa dificuldade afirmando diretamente quais as propriedades que o SD deve possuir. Os métodos só serão bem-sucedidos se as propriedades especificadas forem fáceis de entender. O desafio desses métodos é expressar as propriedades requeridas de forma precisa e inteligível.

III.1.9.7 Vantagens e Limitações da Lógica Temporal

Uma das principais vantagens da lógica temporal é que ela permite a especificação hierárquica e o raciocínio de uma maneira simples e natural. A experiência mostra que a melhor maneira de descrever um sistema complexo é através de uma hierarquia de níveis de abstração, começando de uma especificação de alto nível e terminando com uma implementação em alguma linguagem de programação. A lógica temporal proporciona um único sistema lógico para a descrição do SD em qualquer nível de abstração. Portanto, métodos de projeto hierárquicos têm um suporte direto, sem que nenhum mecanismo extra seja necessário para ligar os diferentes níveis de descrição (LAMPORT, 1983b).

Quando se descreve um SD, não se pode restringir a atenção ao que é verdade antes e depois de sua execução — deve-se considerar o que acontece *durante* sua execução. No estudo de propriedades simples de segurança, só é necessário considerar as

propriedades verdadeiras em todos os instantes durante a execução do SD. Desde que "em todos os instantes" é um conceito temporal simples, pode-se raciocinar sobre essas propriedades de segurança sem qualquer raciocínio temporal explícito. Entretanto, no estudo de propriedades de vida, o raciocínio temporal necessário é mais complexo, não podendo ser deixado implícito. A lógica temporal oferece um meio formal para o estudo sistemático dos aspectos que variam em função do tempo (LAMPORT, 1983b).

Na lógica temporal, imerge-se em um único SD, considerado como universo, e concentra-se em possíveis desenvolvimentos dentro daquele universo: traduz-se características do SD em regras gerais de comportamento que serão posteriormente analisadas. Quando se está interessado em provar um programa complexo, é vantajoso trabalhar em um contexto fixo. A lógica temporal tenta, nesse sentido, equipar o projetista com o aparato mais forte possível para formalizar seu pensamento intuitivo e facilitar uma prova rigorosa (PNUELI, 1977).

Todavia, como qualquer sistema formal, a lógica temporal tem suas limitações. Uma limitação é óbvia: a lógica temporal *não* irá tornar a especificação e verificação de SDs mais fácil. Projetar um SD é uma tarefa difícil: nenhum formalismo pode tornar isso fácil. O que a lógica temporal faz é proporcionar um meio para especificar precisamente o que o SD deve fazer e analisar rigorosamente o que ele vai fazer. Essa ferramenta pode ser utilizada com níveis diferentes de rigor, variando de uma especificação completamente formal com uma prova de correção da implementação verificada por uma máquina até uma descrição informal auxiliada por fórmulas da lógica temporal. O raciocínio rigoroso em qualquer sistema formal é difícil e consome tempo, mas é a única maneira de eliminar os sutis erros dependentes do tempo que são endêmicos nos programas concorrentes.

A lógica temporal possui limitações menos óbvias. Sua expressividade é limitada, mas é isso que a torna ainda mais adequada para especificar e raciocinar sobre programas paralelos. Entretanto, essa limitação se torna desvantajosa em outras aplicações. Por exemplo, ela não pode expressar a afirmação que dois programas são equivalentes. Isto é porque dois programas são equivalentes se eles têm o mesmo conjunto de comportamentos possíveis, e a lógica temporal não pode expressar a noção de possibilidade.

Em geral, a lógica temporal não é boa para comparar programas ou descrever

conjuntos de programas. Quando especificando e raciocinando sobre um único programa, a única comparação relevante entre programas é que uma versão de nível mais baixo implemente corretamente uma de nível mais alto. Só há um único programa que interessa: a implementação funcionando no computador. É este programa que se especifica e sobre o qual se raciocina com a lógica temporal.

III.2 Lógica Modal

A lógica modal pode ser descrita brevemente como a lógica da necessidade e possibilidade, do "deve ser" e "pode ser" (HUGHES *et alia*, 1968). A lógica temporal é uma lógica modal à qual são impostas restrições adicionais como será visto mais adiante. Para o bom entendimento da lógica temporal é necessário conhecer formalmente a lógica modal.

III.2.1 Introdução

A lógica modal[1] moderna foi desenvolvida a partir de certas limitações do conceito de *implicação* como notado por Hugh MacColl no final do século passado. Ela foi axiomatizada por C.I. Lewis em 1912 utilizando o método de *Principia Mathematica* (1910) de Whitehead e Russell. Segundo esses lógicos, o cálculo de predicados é suficiente para permitir a expressão dos conceitos matemáticos de dedução e inferência, mas é limitado pela sua forma de expressão. Certas noções, como as da *necessidade lógica* e *possibilidade lógica*, elaboradas adiante, dificilmente poderiam ser exprimidas através do cálculo de predicados.

No desenvolvimento da lógica como uma ferramenta de formalização, pode-se observar uma crescente habilidade em expressar mudança e variabilidade. O *Cálculo Proposicional* foi desenvolvido para expressar verdades absolutas ou constantes, afirmando fatos básicos sobre o universo de discurso. O arcabouço proposicional lida, principalmente, com a questão de como a verdade de uma sentença composta depende da verdade de seus constituintes. No *Cálculo de Predicados* lidamos com verdades relativas ou variáveis através da distinção da afirmação (o predicado) de seus argumentos. Entende-se que a afirmação pode ser verdadeira ou falsa de acordo com os indivíduos particulares aos quais ela é aplicada. Portanto, podemos considerar predicados como proposições parametrizadas. O

[1] HUGHES *et alia* (1968) apresentam uma excelente introdução à lógica modal, compreendendo igualmente uma extensa bibliografia do assunto

Cálculo Modal adiciona outra dimensão de variabilidade a esta descrição por predicados. Se contemplarmos uma transição maior na qual não somente indivíduos mas também o significado de funções e predicados variem, então o cálculo modal propicia uma notação especial para tal transição. Por exemplo, qualquer cadeia de raciocínio que é válida no planeta Terra pode tornar-se inválida em Marte porque alguns dos conceitos básicos naturalmente usados na Terra podem assumir significados completamente diferentes (ou até tornarem-se sem sentido nenhum) em Marte. Conceitualmente, isso sugere uma partição do universo de discurso em mundos de estrutura semelhante mas conteúdos diferentes. A variabilidade dentro de um mundo é manipulada variando-se os argumentos de predicados, enquanto as mudanças entre mundos são expressadas pelo formalismo modal especial.

Outra maneira de ver a generalização oferecida pela lógica modal é afirmar que o cálculo de predicados é apropriado para descrever *situações estáticas*. Ele dá sentenças sobre objetos básicos e suas interrelações. A dimensão adicional proporcionada pela lógica modal é aquela da *mudança dinâmica* de uma situação em outra. Uma das características de mudanças devido a transições de tempo é o fato que os mesmos objetos básicos e entidades existem em cada uma das situações estáticas, mas seus atributos e interrelações podem mudar. A lógica modal representa, fiel e convenientemente, uma *situação dinâmica* consistindo de um conjunto de situações estáticas e regras de mudanças entre elas (MANNA *et alia*, 1979 e 1981a).

III.2.2 As Noções Modais Básicas

Entre as afirmações verdadeiras podemos distinguir aquelas que meramente acontecem de ser verdadeiras e aquelas que são forçosamente verdadeiras, isto é, não podem ser falsas. Semelhantemente, entre as afirmações falsas podemos distinguir aquelas que meramente acontecem de ser falsas e aquelas que forçosamente são falsas, isto é, não podem ser verdadeiras. Uma afirmação forçosamente verdadeira é chamada uma afirmação *necessariamente verdadeira* ou uma *verdade necessária* ou simplesmente uma *afirmação necessária*. Uma afirmação forçosamente falsa é chamada uma afirmação *impossível* e aquela afirmação que não é necessária nem impossível é chamada uma afirmação *contingente*. Uma afirmação *possível* é uma afirmação que não é impossível.

"Necessidade", nesse contexto, significa *necessidade lógica*. O sentido no qual o

termo "necessário" é usado pode ser dado da seguinte forma: quando é dito que uma certa afirmação é necessária, ela não pode deixar de ser verdade não importando como as coisas fossem, ou não importando com o quê o mundo viesse a ser. Por exemplo, mesmo se a afirmação de que nenhum corpo viaja mais rápido que a luz é amparada por tamanha evidência científica que somos levados a dizer que num sentido importante é *impossível* um corpo viajar mais rápido que a luz, essa afirmação não seria necessária no sentido pretendido, pois a razão que a apóia consiste de fatos sobre o universo físico como ele é, e o universo físico pode, presumivelmente, ter sido outro que não esse. Exemplos de verdades necessárias no sentido pretendido seriam: "Todos os solteiros não são casados", "Não há quadrados redondos" e "Hoje é terça-feira ou não é terça-feira" (HUGHES *et alia*, 1968).

Da mesma forma, por "impossibilidade" referimo-nos a impossibilidade *lógica*; por "contingência", contingência *lógica*; e por "possibilidade", possibilidade *lógica*. O sentido de cada uma dessas expressões pode ser apreendido pelo que foi dito no caso da necessidade. Essas quatro noções, necessidade, impossibilidade, contingência e possibilidade são chamadas noções *modais*. Elas são estreitamente relacionadas umas com as outras — na realidade, pode-se explicar qualquer três delas em termos da quarta. De importância particular é a relação entre necessidade e possibilidade: dizer que uma proposição p é necessariamente verdadeira é equivalente a dizer que não é possível que p seja falsa, e dizer que p é possível (ou possivelmente verdadeira) é equivalente a dizer que não é uma verdade necessária que p é falso.

III.2.3 O Arcabouço Modal

Visando explicar o significado de uma fórmula (seja ela clássica, de primeira ordem ou modal) fala-se de *interpretações*, as quais, no caso da lógica proposicional (Lp — vide apêndice A) significa uma atribuição dos valores-verdade a cada um dos símbolos proposicionais e no caso da lógica de predicados de primeira ordem (LPPo — vide apêndice A) significa a denotação de todos os símbolos de constantes, funções e predicados em um universo fixo. O arcabouço modal geral é uma *estrutura Kripke* — uma coleção de interpretações como as acima mencionadas.

Mais especificamente, uma estrutra Kripke considera um universo consistindo de

vários *mundos* (ou *possíveis mundos*), cada um dos quais é uma interpretação clássica. As interpretações em mundos distintos podem ser bem diferentes: uma fórmula pode ser verdade em um mundo mas não em outro. Se os mundos são interpretações da Lp, então temos uma *lógica modal proposicional*; se os mundos são interpretações da LPPo, então temos uma *lógica modal de primeira ordem*. Em ambos os casos, os mundos são conectados por elos diretos: quaisquer dois mundos podem ser conectados em uma direção, em ambas as direções ou em nenhuma. Seja $\mathbf{E} = \{s_0, s_1, s_2, \ldots\}$ o conjunto de mundos que constitui o universo de discurso da lógica modal. A relação $R : \mathbf{E} \times \mathbf{E} \mapsto \{\mathbf{v}, \mathbf{f}\}$ tal que $R(s_i, s_j) = \mathbf{v}$ se, e somente se, o mundo s_j é R-acessível a partir do mundo s_i, é chamada *relação de acessibilidade*, *relação de atingibilidade* ou de *relação de alternatividade*. Diferentes restrições nessa relação resulta em diferentes sistemas modais.

Se na lógica clássica a questão é se uma fórmula é ou não satisfeita por uma interpretação, na lógica modal, a questão é se uma fórmula é ou não satisfeita por uma estrutura Kripke *em um mundo particular*, desde que, como foi dito, a verdade varia entre os mundos. O fato que a fórmula φ é verdade em uma estrutra Kripke $\mathbf{K}$ em um mundo s_i é denotado por $\mathbf{K}_i(\varphi) = \mathbf{v}$ (SHOHAM, 1988).

A idéia notacional principal é evitar qualquer menção explícita a tanto o parâmetro mundo quanto a relação de acessibilidade. Ao invés disso, são propostos dois operadores simbolizando as noções de necessidade e possibilidade apresentadas anteriormente, e que descrevem propriedades de mundos que são acessíveis de um dado mundo em um universo.

Os dois *operadores modais* são $\Box$, chamado o operador de necessidade e simboliza a noção de necessidade lógica, e $\Diamond$, chamado o operador de possibilidade e simboliza a noção de possibilidade lógica[2]. Seus significados são dados pelas seguintes regras de interpretação:

$$\mathbf{K}_i(\Box\varphi) = \mathbf{v} \Leftrightarrow \forall s_j[R(s_i, s_j) \supset \mathbf{K}_j(\varphi) = \mathbf{v}] \qquad \text{(III.1)}$$

isto é, $\Box\varphi$ é verdade em um mundo s_i se, e somente se, a fórmula φ é verdade em todos os mundos s_j R-acessíveis a partir de s_i e

$$\mathbf{K}_i(\Diamond\varphi) = \mathbf{v} \Leftrightarrow \exists s_j[R(s_i, s_j) \wedge \mathbf{K}_j(\varphi) = \mathbf{v}] \qquad \text{(III.2)}$$

[2] Alguns autores utilizam L e M para representar a necessidade e a possibilidade lógicas, respectivamente. No apêndice 4 do trabalho de HUGHES *et alia* (1968) podem ser encontradas diversas notações de outros autores.

isto é, $\Diamond\varphi$ é verdade no mundo s_i se, e somente se, φ for verdade em pelo menos um mundo s_j R-acessível de s_i.

Da maneira como foram definidos os operadores modais acima, a fórmula geral

$$\Box\sim\varphi \equiv \sim\Diamond\varphi \tag{III.3}$$

é verificada, ilustrando a relação entre os operadores modais: todos os mundos R-acessíveis satisfazem $\sim\varphi$ se, e somente se, não existir nenhum mundo R-acessível satisfazendo φ. Essa fórmula é verdadeira em qualquer mundo para qualquer universo com uma relação R arbitrária (HUGHES *et alia*, 1968).

Também pelas definições acima, tem-se que,

$$\Box\varphi \equiv \sim\Diamond\sim\varphi \tag{III.4}$$

é válido, afirmando que "φ é necessariamente verdade" é equivalente a "não é possível que φ seja falso" e

$$\Diamond\varphi \equiv \sim\Box\sim\varphi \tag{III.5}$$

também é válido, afirmando que "φ é possível" é equivalente a "não é necessário que φ seja falso".

III.2.4 Os Sistemas Modais

Vários sistemas foram propostos visando formalizar a lógica modal em um conjunto mínimo de símbolos, regras e axiomas. A seguir mostramos alguns deles.

III.2.4.1 O Sistema T

O sistema modal mais fraco, construído a partir do cálculo proposicional é o sistema T proposto por Robert Feys em 1937.

A base do sistema T é:

1. Os símbolos primitivos

   - $p, q, r, \ldots$ – variáveis proposicionais;

- $\sim$, $\Box$ – operadores monádicos;
- $\vee$ – operador diádico;
- "(" e ")" – parênteses;

2. As regras de formação (RF)

RF1 – Uma variável é uma fórmula bem-formada;

RF2 – Se A é uma fórmula bem-formada, $\sim A$ e $\Box A$ também são;

RF3 – Se A e B são fórmulas bem-formadas, então $(A \vee B)$ também é;

3. Definições

O símbolo $\stackrel{\text{def}}{=}$ deve ser entendido como "igual por definição".

- [Def $\wedge$] $(A \wedge B) \stackrel{\text{def}}{=} \sim(\sim A \vee \sim B)$;
- [Def $\supset$] $(A \supset B) \stackrel{\text{def}}{=} (\sim A \vee B)$;
- [Def $\equiv$] $(A \equiv B) \stackrel{\text{def}}{=} ((A \supset B) \wedge (B \supset A))$;
- [Def $\Diamond$] $\Diamond A \stackrel{\text{def}}{=} \sim \Box \sim A$;
- [Def $\Rightarrow$] $(A \Rightarrow B) \stackrel{\text{def}}{=} \Box(A \supset B)$;
- [Def $=$] $(A = B) \stackrel{\text{def}}{=} ((A \Rightarrow B) \wedge (B \Rightarrow A))$

4. Esquemas de Axiomas (onde A, B e C representam fórmulas bem-formadas quaisquer)

A1. $(A \vee A) \supset A$

A2. $B \supset (A \vee B)$

A3. $(A \vee B) \supset (B \vee A)$

A4. $(B \supset C) \supset ((A \vee B) \supset (A \vee C))$

A5. $\Box A \supset A$ (Axioma da Necessidade)

A6. $\Box(A \supset B) \supset (\Box A \supset \Box B)$

5. Regras de Inferência (onde *teorema* é qualquer instância de um esquema de axioma, isto é, uma substituição de A, B e C nos esquemas de axiomas por fórmulas bem-formadas quaisquer, ou o resultado da aplicação de uma regra de inferência a uma dessas instâncias (MENDELSON, 1964) — vide apêndice A)

R1. *Regra Modus Ponens*: de A e $A \supset B$ obtem-se B;

R2. *Regra da Necessitação*: se A é um teorema, então, $\Box A$ é um teorema.

III.2.4.2 O Sistema S4

O sistema **S4** é construído a partir do sistema **T**, acrescentando o esquema de axioma

A7. $\Box A \supset \Box\Box A$

ao conjunto de esquemas de axiomas.

III.2.4.3 O Sistema S5

O sistema **S5** é construído a partir do sistema **T**, acrescentando o esquema de axioma

A8. $\Diamond A \supset \Box\Diamond A$

ao conjunto de esquemas de axiomas.

III.2.4.4 O Sistema B

O sistema **B** (de Brower) é construído a partir do sistema **T**, acrescentando o esquema de axioma

A9. $A \supset \Box\Diamond A$

ao conjunto de esquemas de axiomas.

III.2.4.5 O Sistema S4.3

O sistema **S4.3** é construído a partir do sistema **S4**, acrescentando o esquema de axioma

A10. $\Box(\Box A \supset \Box B) \vee \Box(\Box B \supset \Box A)$

ao conjunto de esquemas de axiomas.

III.2.4.6 O Sistema S4.3.1(D)

O sistema **S4.3.1(D)** é construído a partir do sistema **S4.3**, acrescentando o esquema de axioma

A11. $((A \Rightarrow \Box A) \Rightarrow A) \supset (\Diamond\Box A \supset A))$

ao conjunto do esquemas de axiomas.

III.2.5 Relação entre os Sistemas Modais

O diagrama da figura III.6 ilustra a relação verificada entre os sistemas descritos anteriormente. A notação $S \longrightarrow S'$ significa que S contém mas não é contido em S'.

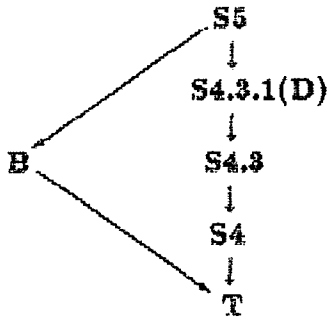

Figura III.6: Diagrama representando a relação entre sistemas modais

III.2.6 Outros Sistemas Modais

A lista de sistemas modais mostrada não é exaustiva: há outros sistemas modais propostos, por exemplo, os sistemas **S0.4**, **S3** e as extensões de **S4** não contidas em **S5**. Muitos desses sistemas são formalmente apresentados no trabalho de HUGHES *et alia* (1968).

III.3 Lógica Temporal

A teoria da lógica temporal é uma preocupação integral da inquirição filosófica. Questões concernentes à natureza do tempo e conceitos temporais preocupam os filósofos desde a inauguração do assunto.

É o objetivo primário da lógica temporal esclarecer o conteúdo, elaborar as conseqüências e elucidar os interrelacionamentos entre os membros (e candidatos a membros) da família de princípios apodícticos, isto é, demonstráveis, no que diz respeito às relações de tempo ou aos axiomas de tempo em geral. A lógica temporal mune o filósofo e o cientista natural com ferramentas para conseguir um melhor entendimento da natureza do tempo em si (RESCHER *et alia*, 1971).

III.3.1 Porque a Lógica Temporal

A lógica convencional não toma nenhum cuidado especial com proposições relacionadas com o tempo. Devido a esse fato, ela manipula tais proposições de modo desgracioso ou até inadequado. Não há sequer maneira direta de se lidar com verbos conjugados. "Sócrates está sentado" torna-se, no melhor possível, "Todos os momentos-idênticos-ao-presente são momentos-nos-quais-Sócrates-está-sentado".

O propósito da lógica temporal é sistematizar o raciocínio com proposições que têm um aspecto temporalizado. Lida-se com afirmações envolvendo o tempo, nas quais alguma referência essencial ao relacionamento antes-depois ou ao relacionamento passado-presente-futuro está em questão, e as idéias de sucessão, mudança e constância estão presentes. A lógica temporal procura proporcionar o aparato lingüístico e inferencial para se obter um discurso exato e um raciocínio rigoroso nesta esfera.

A teoria lógica de tais proposições relacionadas com o tempo é de interesse substancial porque tais considerações explicitamente temporais surgem em uma grande variedade de contextos filosoficamente relevantes. À parte de sua significância óbvia para a análise do discurso conjugado, essas considerações são de importância capital para vários interesses de filósofos da ciência — a estrutura do tempo, a análise de relações temporais e a caracterização de processos naturais, entre outros.

III.3.2 Um Breve Histórico

O estudo da lógica temporal modernamente teve seu ímpeto maior dado pela publicação, no final dos anos 50, do trabalho de PRIOR (1957), *Time and Modality*. Virtualmente todos os trabalhos na área até em torno de 1966 é coberto pelo tratado *Past, Present and Future* do mesmo autor (PRIOR, 1967).

A antiga história dos quantificadores temporais como "algumas vezes" e "sempre" — e da teoria de modalidades temporalizadas relacionadas a elas através da mediação de princípios tais como "o que é algumas vezes verdade é sempre possível" — permanece encoberta na obscuridade. Sabe-se que os rudimentos de tal teoria foram desenvolvidos ativamente pelos antigos gregos: os megarenses e os estóicos e Aristótelos e os antigos Peripatéticos.

A lógica aristotélica e estóica de relações temporais foi tomada pelos lógicos medievais árabes que a desenvolveram subseqüentemente. Avicenna desenvolveu o tratamento temporal de implicação na maneira de Diodorus numa teoria geral de proposições categóricas. Além disso, Avicenna também desenvolveu consideravelmente a teoria megarense-estóica de modalidades temporais. A teoria de modalidades temporais desenvolvida pelos árabes na idade média era de natureza sutil e complexa.

Os europeus medievais (latinos e saxões) parecem ter desenvolvido seu trabalho a partir dos árabes. Os latinos desenvolveram a uma extensão modesta; os saxões (como William de Ockham, Alberto da Saxônia e John Buridam) trataram as considerações cronológicas mais profundamente.

Um reavivamento maior do interesse na lógica temporal tem florescido desde o final da década de 40. O estímulo para esse reavivamento pode ser justificado por três fontes: o estudo de materiais históricos (especialmente os estudos da lógica estóica e lógica medieval), a análise lógica de tempos gramaticais e, principalmente, o esforço do filósofo polonês Jerzy Łos em preparar um sistema de lógica temporal para a análise de questões na filosofia da ciência.

III.3.3 Lógica Temporal e Lógica Modal

A lógica temporal, como será apresentada adiante, é um ramo da lógica modal. Como foi visto, a lógica modal lida com dois operadores proposicionais, $\Box$ e $\Diamond$ (além dos operadores usuais como $\sim$, $\vee$, $\supset$, etc.) interpretados como "necessariamente" e "possivelmente". Isto é baseado na idéia que a verdade de uma afirmação é uma noção relativa dependendo de mundos possíveis. PRIOR (1957) foi o primeiro a sugerir uma interpretação "temporal" para os operadores $\Box$ e $\Diamond$: "sempre" e "alguma vez". Seguindo esse trabalho, vários sistemas diferentes de lógica temporal foram estudados. Uma visão geral desses sistemas pode ser encontrado em RESCHER *et alia* (1971).

III.3.4 O Arcabouço Temporal

O arcabouço da lógica temporal é um arcabouço modal ao qual são impostas restrições adicionais aos modelos de interpretações (RESCHER *et alia*, 1971). A interpretação dada pela lógica temporal à relação de acessibilidade básica R é a da passagem do tempo: um mundo s' é R-acessível de um mundo s se, através do decorrer do tempo, s pode transformar-se em s'. Nesse caso, é mais natural considerar o universo de discurso como consistindo de um único mundo no qual são estudados os efeitos da passagem do tempo. Os vários mundos s_i da lógica modal são agora os diferentes *estados* de um único mundo visto em instantes distintos. $R(s, s')$ é verificado se, e somente se, o estado (do mundo em) s' estiver no futuro do estado (do mundo em) s.

A relação R na lógica temporal, refletindo as noções intuitivas (geralmente aceitas) acerca do tempo, deve ser:

1. *reflexiva*, de modo que todo estado s é R-acessível de si mesmo, isto é,

$$\forall s[R(s, s)] \tag{III.6}$$

afirmando que todo estado $s \in \mathbf{E}$ satisfaz R consigo mesmo.

2. *transitiva*, expresso por

$$\forall s_i \forall s_j \forall s_k \{[R(s_i, s_j) \wedge R(s_j, s_k)] \supset R(s_i, s_k)\} \tag{III.7}$$

afirmando, para três estados quaisquer s_i, s_j e s_k, se s_j é R-acessível de s_i e s_k é R-acessível de s_j, então s_k é R-acessível de s_i.

Os operadores modais adquirem um novo significado, incorporando o conceito de tempo. O operador de necessidade $\Box$ agora é interpretado como "sempre" ou "doravante". A regra de interpretação III.1 afirma que $\Box\varphi$ é verdade em um estado s_i se, e somente se, φ for verdade em todos os estados s_j R-acessíveis a partir de s_i (inclusive em s_i). Já o operador de possibilidade $\Diamond$, na lógica temporal, é interpretado como "eventualmente". A regra de interpretação III.2 afirma que $\Diamond\varphi$ é verdade no estado s_i se, e somente se, φ for verdade em algum estado s_j R-acessível de s_i (podendo ser no próprio s_i).

As equivalências modais mostradas na seção anterior (III.3, III.4 e III.5) continuam válidas, mas são interpretadas diferentemente: por exemplo, a fórmula III.3 é agora interpretada da seguinte maneira: $\sim\varphi$ é *sempre* verificado se, e somente se, não é o caso que φ é *eventualmente* verificado.

III.3.5 Estado e Instante

Como foi visto, a lógica temporal interpreta o conjunto $\mathbf{E} = \{s_0, s_1, \ldots\}$ de mundos possíveis da lógica modal como um mesmo mundo visto em instantes diferentes. Seja $\mathbf{T} = \{t_0, t_1, \ldots\}$ um conjunto de instantes representando uma escala temporal qualquer (segundos, minutos, dias ou séculos), então cada $t_i \in \mathbf{T}$ determina um, e somente um, estado $s_i \in \mathbf{E}$, numa função injetora de $\mathbf{T}$ em $\mathbf{E}$. A escala $\mathbf{T}$ de tempo é interpretada diferentemente (isto é, séculos, dias ou milissegundos) conforme a variabilidade do aspecto do mundo a ser estudado. Doravante, evitando considerações sobre escalas de tempo, será falado somente em estados s_i, significando a situação do mundo em um instante t_i.

III.4 Conclusões

Pelo que foi exposto, pode-se concluir que a lógica temporal é uma *ferramenta* adequada para a especificação formal de SDs. Por *adequada* queremos dizer que ela satisfaz aos três critérios de avaliação de ferramentas descritos anteriormente:

- sua notação é sucinta;

- permite a seu usuário definir o nível de abstração desejado e
- oferece meios de verificar a especificação formal.

O modelo formal subjacente à lógica temporal abstrai o SD para uma seqüência de estados, incorporando a visão das intercalações. Entretanto, a lógica temporal não considera as ações atômicas que causam as transições dos estados (isto é, seu formalismo não permite representar as ações atômicas) e a idéia notacional principal é evitar qualquer menção explícita aos estados — os operadores temporais representam as noções temporais.

O formalismo da lógica temporal permite a expressão e o estudo das propriedades de vida e segurança de modo natural, através dos operadores temporais $\Box$ e $\Diamond$. Uma propriedade de segurança φ é prefixada pelo operador $\Box$, formalizando os conceitos temporais "sempre" ($\Box\varphi$) e "nunca" ($\Box \sim\varphi$), e uma propriedade de vida ϕ é prefixada pelo operador $\Diamond$, formalizando o conceito temporal "eventualmente" ($\Diamond\phi$). A lógica temporal possui, ainda, as seguintes vantagens:

- permite a especificação hierárquica, proporcionando um único sistema lógico para a descrição do SD em qualquer nível de abstração,
- oferece um meio formal para o estudo sistemático dos aspectos dos SDs que são sensíveis ao tempo e
- equipa o projetista com um aparato forte o suficiente para formalizar seu pensamento intuitivo e facilitar uma prova rigorosa.

Apesar de todas essas vantagens, a lógica temporal *não* irá tornar mais fácil a especificação e verificação de SDs. O que ela faz é proporcionar um *meio* para especificar precisamente o que o SD deve fazer e analisar rigorosamente o que ele vai fazer.

Outras conclusões dignas de nota são:

- *As Lógicas Modal e Temporal* – a lógica modal representa, fiel e convenientemente, uma situação dinâmica consistindo de um conjunto de situações estáticas e regras de mudanças entre elas. A lógica temporal é uma lógica modal com algumas restrições adicionais visando refletir a idéia da passagem do tempo. Ela captura os

conceitos temporais intuitivos (sempre e eventualmente) utilizados informalmente na descrição da execução de um SD. Após serem dadas as definições das lógicas modal e temporal, é possível concluir que, do ponto de vista puramente técnico, a lógica modal é desnecessária. Qualquer lógica modal pode ser substituída por uma lógica de primeira ordem, na qual os "possíveis mundos", sejam quais forem seus significados, são simplesmente feitos objetos na linguagem, a relação de acessibilidade se tornaria um símbolo relacional (como já foi usado), e os operadores modais ($\Box$ e $\Diamond$) seriam quantificadores. Tomando como exemplo a lógica temporal, pode-se substituir a seguinte fórmula

$$\sim chove \supset \Box \sim chove$$

pela fórmula de primeira ordem

$$\sim chove(s_1) \supset \forall s_2(R(s_1, s_2) \supset \sim chove(s_2)).$$

Entretanto, é conveniente usar a lógica modal porque esta é concisa e elegante, como demonstra o exemplo dado. Além disso, a lógica modal nos dá a oportunidade de atribuir um *status* especial a um certo tipo de indivíduos, aqueles que correspondem aos possíveis mundos, de forma que eles são entendidos implicitamente sem necessidade de menção explícita (SHOHAM, 1988).

- *Erros de Sincronização* – SDs são inerentemente complexos e são freqüentemente assolados por erros de sincronização dependentes do tempo impossíveis de achar por métodos de teste convencionais. Há razões para suspeitar que 90% das "quedas" em alguns sistemas são devido a tais erros de sincronização. O raciocínio formal e rigoroso é a melhor maneira de eliminá-los (LAMPORT, 1980).

  Escrever uma especificação formal pode ser de grande ajuda mesmo sem verificação, porque força-se a entender precisamente o que o sistema deve fazer. O raciocínio informal pode capturar muitos erros.

- *Ferramentas Automatizadas* – A questão de ferramentas automatizadas (isto é, implementadas por um computador) freqüentemente causa controvérsias. Gerentes de projeto dizem que só considerarão o uso de métodos formais quando ferramentas de suporte adequadas forem disponíveis. Embora aparentemente razoável, esse posicionamento é mal colocado. Métodos formais são, primeiro e mais importante, instrumentos que melhoram a capacidade analítica do projetista. As ferramentas só

existem para dar apoio ao uso do método, e métodos formais podem geralmente ser usados a um bom termo simplesmente com papel e lápis.

- *Sobre Modelos* – É extremamente importante lembrar que modelos não são a realidade em si. Sua utilidade depende tanto da facilidade computacional que eles proporcionam quanto da validade de suas teorias subjacentes. Isso implica que os modelos geralmente têm uma vida útil limitada, devendo ser abandonados em favor de outros melhores, quando a complexidade da aplicação excede seu poder computacional ou quando novos aspectos do objeto em estudo, não representados adequadamente pelo modelo, devem ser levados em consideração.

  O projetista deve estar em posição de endereçar o meta-problema de conceber novos modelos ou pelo menos avaliar aqueles projetados por outros, considerando os requisitos de abstração (remoção de detalhes desnecessários), representação (poder expressivo e analítico e naturalidade de expressão) e na manipulação de representações.

- *Sobre Métodos* – Os métodos axiomáticos, os requeridos pela lógica temporal, são os mais adequados para a especificação formal de SDs, pois afirmam diretamente quais propriedades o sistema deve possuir. Os métodos construtivos possuem séria desvantagem na especificação formal de programas concorrentes.

- *A Expressividade da Lógica Temporal* – A expressividade da lógica temporal é limitada, sendo adequada para especificar e raciocinar sobre programas paralelos. Entretanto, a lógica temporal não é útil em outras aplicações, como o estudo comparativo de dois programas ou a especificação de propriedades que apresentem o conceito de probabilidade.

III.5 Notas Bibliográficas

A seção sobre especificação formal de SDs é fruto de vários artigos e livros. Além das referências que constam no texto, o livro de COHEN *et alii* (1986) é digno de ser mencionado como referência maior. As definições de ferramenta e modelo são originais — na realidade, foram dados nomes a conceitos que são fundamentais para o trabalho. Na bibliografia pesquisada, não havia uma definição precisa dos termos nem consenso quanto ao seu uso — procurou-se com isso estabelecer critérios para o estudo da lógica temporal.

Os trechos que relacionam a lógica temporal e a especificação formal foram escritos principalmente a partir do artigo de LAMPORT (1983b), uma importante bibliografia desse assunto.

A lógica modal foi apresentada a partir dos trabalhos de GUIDACCI DA SILVEIRA (1982), um apanhado de grande parte da literatura clássica sobre o assunto, de HUGHES *et alia* (1968), onde é feito uma excelente introdução a essa lógica, e de MANNA *et alia* (1979 e 1981a). A seção sobre lógica temporal, por sua vez, foi escrita a partir do trabalho de RESCHER *et alia* (1971), um trabalho que abrange as questões principais do assunto, sendo a referência bibliográfica mais constante em trabalhos sobre a lógica temporal, além do trabalho de GUIDACCI DA SILVEIRA (1982), novamente. Usou-se, também, vários artigos sobre a lógica temporal, destacando-se MANNA *et alia* (1979 e 1981a), onde é feito uma interessante abordagem ao relacionamento entre as lógicas de predicado, de primeira ordem e modal, além de uma apresentação didática da lógica temporal.

PRIOR (1957) propõe uma interpretação temporal aos operadores modais. Nesse trabalho, o autor, entre outras coisas, demonstra que a lógica temporal por ele proposta, a lógica de tempos verbais (*tense logic*), é uma lógica modal, propondo sistemas formais análogos aos sistemas S4 e S5 de Lewis. Muitas das questões levantadas nesse trabalho são resolvidas no livro do mesmo autor publicado 10 anos mais tarde (PRIOR, 1967). Nesse último trabalho, são mostrados os precursores da lógica de tempos verbais, numa visão geral do que havia sido feito até então na lógica temporal, estendendo também o assunto em áreas ainda não investigadas. Versões não-monotônicas da lógica modal são encontradas em SMETS *et alii* (1988) e SHOHAM (1988).

Capítulo IV

Lógica Temporal Linear (LTL)

A Lógica Temporal Linear (LTL) (RESCHER *et alia*, 1971; PNUELI, 1977) oferece como modelo semântico uma seqüência linear discreta de estados. Essas seqüências podem ser utilizadas para descrever processos dinâmicos que passam por mudanças em instantes discretos.

O uso da lógica temporal na computação foi originalmente feito com a versão linear (LTL) (KRÖGER, 1987; PNUELI, 1977). Esta lógica temporal é a mais antiga e, por isso, a que mais teve tempo para ser revista e ampliada. Na bibliografia existente disponível, questões endereçadas a essa lógica temporal são mais abundantes do que a quaisquer outras.

A LTL apresenta-se sob duas formas: a lógica temporal linear proposicional (LTLp — PNUELI, 1977) e sua extensão, a lógica temporal linear de primeira ordem (LTLPo — MANNA, 1981; MANNA *et alia*, 1981a e 1981b). Quando falarmos simplesmente em LTL, deve ser entendido as duas formas.

O capítulo possui a seguinte estrutura: na primeira seção fazemos uma introdução à versão linear da LT apresentada de forma geral no capítulo anterior, sendo dadas as características distintivas. Apresentamos, então, nas próximas duas seções, a LTLp e LTLPo. Para cada uma dessas LTLs damos a sintaxe, a semântica, exemplos de obtenção de valor-verdade, algumas definições e teoremas e exibimos um sistema formal. Para cada um dos sistemas formais provamos sua consistência, consideramos a completeza, damos algumas regras derivadas e alguns teoremas. Concluindo cada uma das seções, mostramos um exemplo de especificação formal de SDs utilizando as LTLs.

Na seção seguinte, discorremos sobre propriedades inexprimíveis na LTL e, na outra seção, exibimos mais alguns operadores temporais. Encerrando o capítulo, temos as seções de conclusões e de notas bibliográficas.

IV.1 Introdução

A LTL considera o tempo como tendo uma estrutura linear. Podemos representar a história de um mundo (o universo de discurso — vide capítulo III) consistindo de um conjunto $E = \{s_0, s_1, s_2, \ldots\}$ de estados, onde o estado s_i descreve a situação do mundo no instante t_i, da maneira mostrada pela figura IV.1.a). Podemos também ver o conjunto E com seus elementos distribuídos entra os pontos de uma reta orientada conforme a figura IV.1.b) como uma forma alternativa de representação.

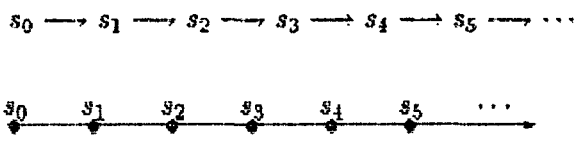

Figura IV.1: Representações lineares do tempo

Na LTL, o tempo é visto de forma *discreta*, isto é, sempre é possível encontrar dois estados s_i e s_j, representando instantes distintos, entre os quais não há nenhum outro estado s_k. Em termos da relação básica de acessibilidade R (vide capítulo III), podemos expressar essa característica pela fórmula de primeira ordem

$$\exists s_i \exists s_j \forall s_k \{R(s_i, s_j) \wedge [\sim R(s_i, s_k) \vee \sim R(s_k, s_j)]\} \qquad \text{(IV.1)}$$

O tempo, linear e discreto, é ainda considerado *infinito*, característica expressa por

$$\forall s_i \exists s_j [s_i \neq s_j \wedge R(s_i, s_j)] \qquad \text{(IV.2)}$$

afirmando, para todo estado s_i, a existência de um estado s_j diferente de s_i e R-acessível a partir de s_i. Além de linear, discreto e infinito, o tempo possui *um menor elemento*, chamado de s_0, a partir do qual o tempo se desenvolve. Essa característica pode ser

expressa pela fórmula

$$\exists s_i \forall s_j [R(s_i, s_j)] \tag{IV.3}$$

afirmando a existência de um estado s_i tal que todos os estados s_j são R-acessíveis a partir dele.

Portanto, os elementos $s_0, s_1, s_2, \ldots$ do conjunto E formam uma seqüência $\sigma = s_0 s_1 s_2 \cdots s_n \cdots$ de estados $s_i, i \in \mathbb{N}_0$, que constitui o modelo de interpretação da LTL. Nos estados $s_i \in \mathbf{E}$, definimos uma *relação de acessibilidade imediata* $\rho : \mathbf{E} \times \mathbf{E}$, tal que $\rho(s, s')$ é verificado se s for *seguido imediatamente* por s'. Em termos das representações gráficas dadas na figura IV.1, $\rho(s, s')$ é verificado se entre s e s' houver uma, e somente uma, seta ($\longrightarrow$), no caso da figura IV.1.a), e no caso da representação da figura IV.1.b), se entre o ponto s e o ponto s' só houver um segmento de reta sem pontos marcados. Formalmente, temos a fórmula de primeira ordem

$$\forall s_i \forall s_j \{\rho(s_i, s_j) \equiv \forall s_k [R(s_i, s_j) \wedge (\sim R(s_i, s_k) \vee \sim R(s_k, s_j))]\} \tag{IV.4}$$

afirmando que s_j é ρ-acessível de s_i se, e somente se, s_j é R-acessível de s_i e s_j não é R-acessado por nenhum estado R-acessível de s_i.

A característica de linearidade do tempo é dada pela relação ρ, se limitarmos ρ a ser uma função, $\rho(s, s')$ só sendo satisfeita para o *único* estado s' imediatamente seguinte a s, formalmente expresso pela fórmula

$$\forall s_i \exists! s_j \{s_i \neq s_j \wedge \rho(s_i, s_j)\} \tag{IV.5}$$

afirmando, para todo s_i, a existência de um, e somente um, estado s_j ρ-acessível de s_i.

IV.2 A Lógica Temporal Linear Proposicional (LTLp)

A LTLp é uma extensão da lógica proposicional. À sintaxe da lógica proposicional são acrescidos os operadores temporais $\Box$, $\Diamond$, $\circ$ e $\mathcal{U}$. Esses operadores dão às fórmulas proposicionais uma nova dimensão da sua semântica, estendendo-a de forma a expressar a temporalidade de afirmações.

IV.2.1 Sintaxe

As fórmulas da LTLp são construídas a partir dos seguintes símbolos (alfabeto):

1. Sinais de pontuação: "(" e ")";
2. Símbolos de valor-verdade: **V** (verdade) e **F** (falso);
3. Conectivos: $\sim$ (*não*), $\supset$ (*implicação lógica*), $\wedge$ (*e*), $\vee$ (*ou*) e $\equiv$ (*equivalência*);
4. Operadores temporais: $\Box$ (*sempre ou doravante*), $\Diamond$ (*alguma vez ou eventualmente*), $\circ$ (*próximo*) e $\mathcal{U}$ (*até*);
5. Constantes proposicionais: $p_1, p_2, \ldots$.

Por questões de simplicidade, também denotaremos as constantes proposicionais por $p, q, r, \ldots$. Usando esses símbolos, definimos recursivamente as *fórmulas bem-formadas* (ou simplesmente *fórmulas*) tomando por base as *fórmulas atômicas*:

1. Fórmulas Atômicas — são os construtos mais simples da LTLp, a partir dos quais as fórmulas (bem-formadas) são formadas. As fórmulas atômicas, referenciadas genericamente por $v, v_1, v_2, \ldots$, são:
   a) os símbolos de valor-verdade **V** e **F** e
   b) as constantes proposicionais $p_i, i = 1, 2, \ldots$.

   Chamaremos de $\mathcal{V}$ o conjunto enumerável das fórmulas atômicas.
2. Fórmulas Bem-formadas — referenciadas de modo genérico pelas meta-variáveis $A, A_1, A_2, \ldots, B, B_1, B_2, \ldots, F, \ldots$, são obtidas a partir das fórmulas atômicas, às quais são aplicados os sinais de pontuação, os conectivos e os operadores temporais:
   a) toda fórmula atômica é uma fórmula bem-formada;
   b) se A é uma fórmula bem-formada então (A) também é uma fórmula bem-formada;
   c) se A e B são fórmulas bem-formadas, então $\sim A$, $A \supset B$, $A \wedge B$, $A \vee B$ e $A \equiv B$ também são fórmulas bem-formadas;
   d) se A e B são fórmulas bem-formadas, então $\Box A$, $\Diamond A$, $\circ A$ e $A \mathcal{U} B$ também são fórmulas bem-formadas;

Observacões:

i) Poderia ter sido dada uma versão da LTLp mais condensada, considerando as abreviações da lógica clássica (vide apêndice A). Também poderíamos ter definido o operador $\Box$ em termos de $\Diamond$ ou vice-e-versa (vide capítulo III). Por razões de documentação e conveniência técnica, foi mostrada uma LTLp com todos os operadores, o que é, na realidade, redundante.

ii) Estendemos a lista de prioridades da Lp (apêndice A):

$\sim$, $\circ$, $\Box$ e $\Diamond$ têm maior prioridade que todos os operadores binários;

$\mathcal{U}$ tem maior prioridade que $\wedge$, $\vee$, $\supset$ e $\equiv$;

$\wedge$ e $\vee$ tem maior prioridade que $\supset$ e $\equiv$;

$\supset$ tem maior prioridade que $\equiv$;

iii) A observação da Lp (apêndice A) acerca dos sinais de pontuação continua valendo na LTLp.

IV.2.2 Estados e Estrutura Temporal (K)

A semântica da lógica proposicional clássica é obtida a partir da avaliação, isto é, da atribuição de valores-verdade **v** (verdade) e **f** (falso), a cada constante proposicional. Com esses valores e as regras de manipulação de cada conectivo ($\sim$, $\wedge$, $\vee$, $\supset$ e $\equiv$) consegue-se finalmente o valor-verdade da fórmula como um todo. Para a LTLp, é necessário estender esse conceito conforme a idéia informal de que suas fórmulas são avaliadas sobre uma escala de tempo.

Uma *estrutura temporal* (ou *Kripke*) **K** para a linguagem $\mathcal{L}$ da LTLp ($\mathcal{L}_{LTLp}$) consiste de uma seqüência infinita $s_0 s_1 s_2 \cdots$ de mapeamentos

$$s_i : \mathcal{V} \mapsto \{\mathbf{v}, \mathbf{f}\}$$

onde $\mathcal{V}$ é o conjunto das fórmulas atômicas e, por definição, $s_i(\mathbf{V}) = \mathbf{v}$ e $s_i(\mathbf{F}) = \mathbf{f}$ para $i \in \mathbb{N}_0$. Os s_i são chamados *estados* e s_0 é o estado inicial. Cada estado é uma avaliação de fórmulas no sentido clássico: intuitivamente, s_i atribui um valor **v** (verdade) ou **f** (falso) a cada uma das fórmulas atômicas $v \in \mathcal{V}$. Com esses valores é possível, através da manipulação de conectivos e operadores temporais, deduzir o valor final de fórmulas.

IV.2.3 Semântica

Para toda estrutura temporal $\mathbf{K}$, todo $i \in \mathbb{N}_0$ e toda fórmula F, definimos indutivamente o valor-verdade $\mathbf{K}_i(F) \in \{\mathbf{v}, \mathbf{f}\}$, informalmente significando "o valor-verdade de F no estado s_i":

1. $\mathbf{K}_i(v) = s_i(v)$ para $v \in \mathcal{V}$. Informalmente quer dizer "o valor-verdade de v no estado s_i é dado pela atribuição que s_i fizer a v";
2. $\mathbf{K}_i(\sim A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{f}$;
3. $\mathbf{K}_i(A \wedge B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{v}$ e $\mathbf{K}_i(B) = \mathbf{v}$;
4. $\mathbf{K}_i(A \vee B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{v}$ ou $\mathbf{K}_i(B) = \mathbf{v}$;
5. $\mathbf{K}_i(A \supset B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{f}$ ou $\mathbf{K}_i(B) = \mathbf{v}$;
6. $\mathbf{K}_i(A \equiv B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A \supset B) = \mathbf{v}$ e $\mathbf{K}_i(B \supset A) = \mathbf{v}$;
7. $\mathbf{K}_i(\Box A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_j(A) = \mathbf{v}$ para todo $j \geq i$;
8. $\mathbf{K}_i(\Diamond A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_j(A) = \mathbf{v}$ para algum $j \geq i$;
9. $\mathbf{K}_i(\circ A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_{i+1}(A) = \mathbf{v}$;
10. $\mathbf{K}_i(A \mathcal{U} B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo k, $i \leq k < j$

Observações:

i) Os operadores $\sim$, $\wedge$, $\vee$, $\supset$ e $\equiv$ são os conectivos clássicos interpretados em cada estado, sem qualquer aspecto temporal;

ii) Na definição dos operadores $\Box$ e $\Diamond$ foi incluido o presente (i) como parte do futuro (j), afirmando-se $j \geq i$ e não apenas $j > i$;

iii) O operador $\mathcal{U}$ possui um aspecto "existencial" no sentido de que há, necessariamente, um ponto no tempo (j) onde é verificado B.

Exemplos típicos de expressões temporais e suas interpretações intuitivas são:

$\Diamond\Box A$: "Em algum momento (agora ou no futuro) A será verificado permanentemente";

$\Box\Diamond A$: "Para todo estado há um estado posterior onde A é verificado", isto é, "A se verifica infinitamente freqüentemente de agora em diante".

$\Box(A \supset B)$: "Doravante, se é verdade A então B é verdade";

$\Diamond(A \wedge \circ\sim A)$: "Há um instante futuro no qual A é verdade e no próximo instante A é falso".

IV.2.4 Um Exemplo da LTLp

Seja **K** a estrutura temporal de acordo com a matriz da figura IV.2. As duas linhas representam os valores das fórmulas atômicas v_1 e v_2, respectivamente, para $s_0, s_1, s_2, \ldots$. O símbolo "*" indica valores arbitrários.

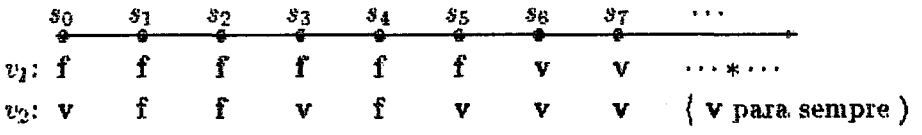

Figura IV.2: Matriz representando uma estrutura temporal **K**

Da estrutura **K**, obtemos:

- Seja A a fórmula $\sim v_1$, então $\mathbf{K}_0(A) = \mathbf{K}_1(A) = \mathbf{K}_2(A) = \mathbf{K}_3(A) = \mathbf{K}_4(A) =$ **v** isto é, v_1 possui valor **f** nos estados s_0, s_1, s_2, s_3 e s_4;
- Seja B a fórmula $\circ\sim v_1$, então $\mathbf{K}_0(B) = \mathbf{K}_1(B) = \mathbf{K}_2(B) = \mathbf{K}_3(B) = \mathbf{K}_4(B) =$ **v** isto é, a fórmula $\circ\sim v_1$ é verdade (valor **v**) nos estados s_0, s_1, s_2, s_3 e s_4, pois ela afirma que $\sim v_1$ possui valor **v** (ou seja v_1 tem valor **f**) no próximo estado, o que ocorre nesses estados;
- Seja C a fórmula $\Box v_2$, então $\mathbf{K}_5(C) = \mathbf{K}_6(C) = \mathbf{K}_7(C) = \ldots =$ **v** isto é, a fórmula $\Box v_2$ é verdade em todos os estados a partir dos quais v_2 tenha sempre valor **v**, o que ocorre do estado s_5 em diante;

- Seja D a fórmula BUC, isto é, $(\circ\sim v_1 U \Box v_2)$, então $\mathbf{K}_0(D) = \mathbf{K}_1(D) = \mathbf{K}_2(D) = \ldots = \mathbf{v}$ isto é, é sempre verdade que B possui valor $\mathbf{v}$ até que C se torne verdade. De fato, $\circ\sim v_1$ é verdade até o estado s_4 e do estado s_5 em diante v_2 é verdade (a fórmula $\Box v_2$ é verdade). Deve ser notado que, após o estado que torna $\Box v_2$ verdade, o valor-verdade de $\circ\sim v_1$ é irrelevante para a avaliação da fórmula D.

IV.2.5 Definições e Teoremas

Definição IV.2.5.1 Uma fórmula A é dita *válida na estrutura temporal* $\mathbf{K}$, denotando-se por $\models_{\mathbf{K}} A$, se $\mathbf{K}_i(A) = \mathbf{v}$ para todo $i \in \mathbb{N}_0$.

As definições de fórmulas *validas* e de fórmulas *que seguem de um conjunto de fórmulas* são feitas à semelhança das definições A.1.3.2 e A.1.3.3 (apêndice A).

O teorema seguinte afirma que se uma fórmula A segue de um conjunto $\mathcal{F}$ de fórmulas válidas (B), então A também é válida:

Teorema IV.2.5.1 *Se* $\mathcal{F} \models A$ *e* $\models B$ *para todo* $B \in \mathcal{F}$, *então* $\models A$.

Prova: Seja $\mathbf{K}$ uma estrutura temporal. Então $\models_{\mathbf{K}} B$ para todo $B \in \mathcal{F}$ e, portanto $\models_{\mathbf{K}} A$. Desde que isso é verificado para todo $\mathbf{K}$, temos que $\models A$. ∎

O teorema IV.2.5.2 é semelhante ao fato clássico

$$A_1, \ldots, A_n \models B \text{ se, e somente se, } \models A_1 \wedge \ldots \wedge A_n \supset B$$

que não é mais verificado na $\mathcal{L}_{LTL_p}$. Um contra-exemplo é $A \models \Box A$, o qual se verifica, ao passo que $A \supset \Box A$ não é válido.

Teorema IV.2.5.2 $A_1, \ldots, A_n \models B$ *se, e somente se,* $\models \Box A_1 \wedge \ldots \wedge \Box A_n \supset B$.

Prova:

($\Rightarrow$) Sejam $A_1, \ldots, A_n \models B$ (logo, $\models_{\mathbf{K}} B$ para todo $\mathbf{K}$ tal que $\models_{\mathbf{K}} A_1, \ldots, \models_{\mathbf{K}} A_n$), $\mathbf{K} = \{s_0, s_1, s_2, \ldots\}$ e $i \in \mathbb{N}_0$. Vamos assumir que $\mathbf{K}_i(\Box A_1 \wedge \ldots \wedge \Box A_n \supset B) = \mathbf{f}$. Isso significa que $\mathbf{K}_i(\Box A_1) = \ldots = \mathbf{K}_i(\Box A_n) = \mathbf{v}$ e $\mathbf{K}_i(B) = \mathbf{f}$ e, portanto, $\mathbf{K}_j(A_1) = \ldots = \mathbf{K}_j(A_n) = \mathbf{v}$ para todo $j \geq i$ e $\mathbf{K}_i(B) = \mathbf{f}$. Seja agora $\mathbf{K}' = \{s'_0, s'_1, s'_2, \ldots\}$ uma nova estrutura temporal com $s'_j = s_{i+j}$ para todo $j \in \mathbb{N}_0$. Então temos $\mathbf{K}'_j(A_1) = \ldots = \mathbf{K}'_j(A_n) = \mathbf{v}$ para

todo $j \in \mathbb{N}_0$ e $\mathbf{K}'_0(B) = \mathbf{f}$ (pois s_i é agora s'_0), daí $\models_{\mathbf{K}'} A_1, \ldots, \models_{\mathbf{K}'} A_n$ mas não $\models_{\mathbf{K}'} B$. Isso é uma contradição, pois $\mathbf{K}'$ é um contra-exemplo da primeira premissa, portanto, $\mathbf{K}_i(\Box A_1 \wedge \ldots \wedge \Box A_n \supset B) = \mathbf{v}$ e como $\mathbf{K}$ e i são arbitrários, temos que $\models \Box A_1 \wedge \ldots \wedge \Box A_n \supset B$.

($\Leftarrow$) Sejam $\models \Box A_1 \wedge \ldots \wedge \Box A_n \supset B$, $\mathbf{K}$ uma estrutura temporal tal que $\models_{\mathbf{K}} A_1, \ldots, \models_{\mathbf{K}} A_n$ e $i \in \mathbb{N}_0$. Então $\mathbf{K}_j(A_1) = \ldots = \mathbf{K}_j(A_n) = \mathbf{v}$ para todo $j \in \mathbb{N}_0$, e, portanto, também $\mathbf{K}_j(A_1) = \ldots = \mathbf{K}_j(A_n) = \mathbf{v}$ para todo $j \geq i$. Isso implica que $\mathbf{K}_i(\Box A_1 \wedge \ldots \wedge \Box A_n) = \mathbf{v}$ e por causa de $\mathbf{K}_i(\Box A_1 \wedge \ldots \wedge \Box A_n \supset B) = \mathbf{v}$ temos que $\mathbf{K}_i(B) = \mathbf{v}$. Isso significa que $\models_{\mathbf{K}} B$ e temos o resultado desejado que $A_1, \ldots, A_n \models B$. ∎

O teorema IV.2.5.3 afirma que a regra de inferência *modus ponens* da lógica clássica preserva a validade na LTLp:

Teorema IV.2.5.3 *Se* $\mathcal{F} \models A$ *e* $\mathcal{F} \models A \supset B$ *então* $\mathcal{F} \models B$.

Prova: Sejam $\mathbf{K}$ uma estrutura temporal tal que $\models_{\mathbf{K}} C$ para todo $C \in \mathcal{F}$ e $i \in \mathbb{N}_0$. Então $\mathbf{K}_i(A) = \mathbf{K}_i(A \supset B) = \mathbf{v}$ e portanto $\mathbf{K}_i(B) = \mathbf{v}$. Isso significa que $\mathcal{F} \models B$. ∎

Teorema IV.2.5.4 *Se* $\mathcal{F} \models A$ *então* $\mathcal{F} \models \circ A$ *e* $\mathcal{F} \models \Box A$.

Prova: Sejam $\mathbf{K}$ uma estrutura temporal tal que $\models_{\mathbf{K}} C$ para todo $C \in \mathcal{F}$ e $i \in \mathbb{N}_0$. Então $\mathbf{K}_j(A) = \mathbf{v}$ para todo $j \in \mathbb{N}_0$, em particular, $\mathbf{K}_{i+1}(A) = \mathbf{v}$ e $\mathbf{K}_j(A) = \mathbf{v}$ para todo $j \geq i$. Isso significa que $\mathcal{F} \models \circ A$ e $\mathcal{F} \models \Box A$. ∎

Uma dúvida que naturalmente surge é se as fórmulas válidas, isto é, as "leis lógicas" da lógica proposicional clássica permanecem válidas na LTLp, podendo ser usadas substituindo-se as fórmulas atômicas que as compõem por fórmulas temporais. O teorema IV.2.5.5 abaixo responde afirmativamente. Para chegarmos ao teorema, são necessárias duas definições:

Definição IV.2.5.2 Uma *substituição uniforme* em uma fórmula A é feita quando substituímos uniformemente as ocorrências de uma determinada subfórmula B de A, isto é, todas as ocorrências de uma subfórmula B de A são substituídas por uma mesma fórmula C.

Definição IV.2.5.3 Uma fórmula da $\mathcal{L}_{LTLp}$ é chamada de *tautologia da LTLp* se ela for o

resultado de uma substituição uniforme das constantes proposicionais de uma tautologia A da lógica proposicional (Lp — apêndice A) por fórmulas da $\mathcal{L}_{LTL_p}$.

Teorema IV.2.5.5 *Toda tautologia da LTLp é válida.*

Prova: Sejam A^* o resultado da substituição uniforme das fórmulas atômicas $v_1, \ldots, v_n$ de uma fórmula A por fórmulas $A_1, \ldots, A_n$ da $\mathcal{L}_{LTL_p}$, $\mathbf{K}$ uma estrutura temporal e $i \in \mathbb{N}_0$. Definimos uma avaliação (atribuição de valores-verdade v ou f) clássica $\mathbf{B}$ da linguagem da lógica clássica proposicional por $\mathbf{B}(v_j) = \mathbf{K}_i(A_j)$ para $j = 1, \ldots, n$ (e com valores arbitrários para outros $v_k \in \mathcal{V}$). Afirmamos que:

$$\mathbf{B}(A) = \mathbf{K}_i(A^*) \qquad (*)$$

o que prova o teorema, pois $\mathbf{B}(A) = \mathbf{v}$ se A é uma tautologia. A prova de $(*)$ é feita por indução em (na sintaxe de) A:

1. A é da forma v_j: então $A^* \equiv A_j$, portanto $\mathbf{B}(A) = \mathbf{B}(v_j) = \mathbf{K}_i(A_j) = \mathbf{K}_i(A^*)$

2. A é da forma $\sim B$, então, com o significado correspondente do "operador" * em B, temos A^* é $\sim B^*$, de onde temos, usando a hipótese de indução $(\mathbf{B}(B) = \mathbf{K}_i(B^*))$

$$\mathbf{B}(A) = \mathbf{v} \Leftrightarrow \mathbf{B}(B) = \mathbf{f} = \mathbf{K}_i(B^*) \Leftrightarrow \mathbf{K}_i(A^*) = \mathbf{v}$$

3. A é da forma $B_1 \wedge B_2$, então, com o * em B_1 e B_2, temos que A^* é $B_1^* \wedge B_2^*$ e com a hipótese de indução $(\mathbf{B}(B_1) = \mathbf{K}_i(B_1^*)$ e $\mathbf{B}(B_2) = \mathbf{K}_i(B_2^*))$, obtemos:

$$\begin{aligned} \mathbf{B}(A) = \mathbf{v} &\Leftrightarrow \mathbf{B}(B_1) = \mathbf{v} \text{ e } \mathbf{B}(B_2) = \mathbf{v} \\ &\Leftrightarrow \mathbf{K}_i(B_1^*) = \mathbf{v} \text{ e } \mathbf{K}_i(B_2^*) = \mathbf{v} \\ &\Leftrightarrow \mathbf{K}_i(B_1^* \wedge B_2^*) = \mathbf{v} \\ &\Leftrightarrow \mathbf{K}_i(A^*) = \mathbf{v} \end{aligned}$$

■

Queremos também interpretar esse resultado de um modo diferente: suponhamos que uma fórmula B seja conseqüência lógica de algumas fórmulas $A_1, \ldots, A_n$ da lógica clássica proposicional. Novamente, se substituirmos (uniformemente) fórmulas da $\mathcal{L}_{LTL_p}$ nas fórmulas $A_1, \ldots, A_n$ e B, não destruiremos o relacionamento lógico. Por exemplo, teríamos:

$$\circ A \supset \Box B, \Box B \supset \Diamond C \models \circ A \supset \Diamond C$$

desde que, na lógica clássica, nós temos:

$$A' \supset B', B' \supset C' \models A' \supset C'$$

IV.2.6 O Sistema Formal Σ_{LTLp}

Chamaremos de Σ_{LTLp} o sistema abaixo que formaliza a LTLp. A notação $\vdash A$ significa que A é um teorema (axioma ou teorema — vide A.1.4).

Axiomas

Os esquemas de axiomas que constituem a base do Σ_{LTLp} são:

A1. $\vdash \sim \Diamond A \equiv \Box \sim A$

A2. $\vdash \Box(A \supset B) \supset (\Box A \supset \Box B)$

A3. $\vdash \Box A \supset A$

A4. $\vdash \sim \circ A \equiv \circ \sim A$

A5. $\vdash \circ(A \supset B) \supset (\circ A \supset \circ B)$

A6. $\vdash \Box A \supset \circ A$

A7. $\vdash \Box A \supset \circ \Box A$

A8. $\vdash \Box(A \supset \circ A) \supset (A \supset \Box A)$

A9. $\vdash A \mathcal{U} B \equiv B \vee (A \wedge \circ(A \mathcal{U} B))$

A10. $\vdash A \mathcal{U} B \supset \Diamond B$

Os axiomas A1–A3 vêm do sistema modal T (vide capítulo III) onde estão na [Def $\Diamond$], A6 e A5, respectivamente. O axioma A4 estabelece o operador $\circ$ como seu próprio dual. Conseqüentemente, ele implica que o próximo instante existe e é único, restringindo nossos modelos a seqüências lineares. O axioma A5 é o análogo de A2 para o operador $\circ$. A6 afirma que o próximo instante é parte do futuro. A7 é uma versão mais fraca do axioma A7 do sistema S4 (vide capítulo III). O axioma A8 é o axioma de indução computacional — ele afirma que, se uma propriedade é "herdada" sobre transições de um passo, ela é invariante sobre qualquer seqüência cujo primeiro estado satisfaz A. A9 caracteriza o operador $\mathcal{U}$ no que diz respeito ao presente e ao próximo instante. O axioma A10 simplesmente afirma que "A até B" implica que B eventualmente será verificado.

Regras de Inferência

As regras de inferência do Σ_{LTLp} são:

R1. Se A é uma tautologia da LTLp, então $\vdash A$ (*Tautologia Proposicional — TP*)

R2. Se $\vdash A \supset B$ e $\vdash A$ então $\vdash B$ (*Modus Ponens — MP*)

R3. Se $\vdash A$ então $\vdash \Box A$ (*Inserção do* $\Box$ *— I*$\Box$)

IV.2.7 Consistência do Σ_{LTLp}

Seja A uma fórmula e $\mathcal{F}$ um conjunto de fórmulas. Afirmamos que:

Se $\mathcal{F} \vdash A$ *então* $\mathcal{F} \models A$

ou seja, admitindo que $\mathcal{F}$ é Σ_{LTLp},

O sistema Σ_{LTLp} *é consistente.*

O termo *consistência* está sendo usado (talvez indevidamente) como tradução livre do termo *sound*, geralmente traduzido como *correto* ou *robusto*. Deve ser lembrado que a notação $\mathcal{F} \vdash A$ denota "A é uma conseqüência de $\mathcal{F}$" (vide subseção A.1.4, cujas definições também se aplicam ao Σ_{LTLp}) e $\mathcal{F} \models A$ denota "A segue de um conjunto $\mathcal{F}$". Como $\mathcal{F}$ é qualquer, podemos escrever que, se $\vdash A$, então $\models A$.

Prova: A prova segue por indução, assumindo-se a dedução de A a partir de $\mathcal{F}$ ($\mathcal{F} \vdash A$):

1. A é um esquema de axioma do Σ_{LTLp}. Provamos a validade de todos os esquemas de axiomas **A1—A10** no apêndice B: em todos os casos, $\mathcal{F} \models A$.
2. $A \in \mathcal{F}$. Nesse caso, temos trivialmente que $\mathcal{F} \models A$.
3. (Consistência de *TP*) Seja A a conclusão de *TP* com a premissa satisfeita de que A é uma tautologia da LTLp. Então, pelo teorema IV.2.5.5, $\models A$, isto é, $\mathcal{F} \models A$ para $\mathcal{F}$ qualquer.

4. (Consistência de *MP*) Seja A a conclusão de *MP* com premissas B e $B \supset A$. Então, temos que $\mathcal{F} \vdash B$ e $\mathcal{F} \vdash B \supset A$, obtendo $\mathcal{F} \models B$ e $\mathcal{F} \models B \supset A$ pela hipótese de indução, e portanto, $\mathcal{F} \models A$, pelo teorema IV.2.5.3.

5. (Consistência de $I\Box$) Seja A a conclusão de $I\Box$ com a premissa B. Então, temos que $\mathcal{F} \vdash B$ e A é $\Box B$, e pela hipótese de indução, $\mathcal{F} \models B$. Pelo teorema IV.2.5.4 obtemos $\mathcal{F} \models \Box B$ (e também $\mathcal{F} \models \circ B$, que não nos interessa aqui) ou seja, $\mathcal{F} \models A$.

■

IV.2.8 Completeza do Σ_{LTL_p}

O sistema Σ_{LTL_p} é completo. Ele é redutível ao sistema DUX(II) que é provado como completo (GABBAY *et alii*, 1980). O Σ_{LTL_p} também é redutível ao sistema Σ_{TA} que é completo (KRÖGER, 1987).

IV.2.9 Regras Derivadas

Exibimos algumas regras derivadas que serão úteis nas demonstrações adiante. As demonstrações dessas regras podem ser encontradas em MANNA (1981).

1. *RP — Raciocínio Proposicional*

$$\frac{\vdash (A_1 \wedge \ldots \wedge A_n) \supset B \quad \vdash A_1, \ldots, \vdash A_n}{\vdash B}$$

Sempre que *RP* for aplicado sem indicarmos o antecedente ($\vdash (A_1 \wedge \ldots \wedge A_n) \supset B$) é porque ele é uma instância de uma tautologia proposicional.

2. *I*○ — *Inserção do* ○

$$\frac{\vdash A}{\vdash \circ A}$$

3. *I*◇ — *Inserção do* ◇

$$\frac{\vdash A}{\vdash \Diamond A}$$

4. *Regras* □□

a) $\dfrac{\vdash A \supset B}{\vdash \Box A \supset \Box B}$ b) $\dfrac{\vdash A \equiv B}{\vdash \Box A \equiv \Box B}$

5. *Regras* $\Diamond\Diamond$

a) $\dfrac{\vdash A \supset B}{\vdash \Diamond A \supset \Diamond B}$ b) $\dfrac{\vdash A \equiv B}{\vdash \Diamond A \equiv \Diamond B}$

6. *Regras* $\circ\circ$

a) $\dfrac{\vdash A \supset B}{\vdash \circ A \supset \circ B}$ b) $\dfrac{\vdash A \equiv B}{\vdash \circ A \equiv \circ B}$

7. *IC — Regra de Indução Computacional*

$$\frac{\vdash A \supset \circ A}{\vdash A \supset \Box A}$$

8. *IT — Regra de Indução para Trás*

$$\frac{\vdash \circ A \supset A}{\vdash \Diamond A \supset A}$$

9. *PP — Regra Próximo do Presente*

$$\frac{\begin{array}{l}\vdash (\circ A \equiv \circ B) \supset (A \equiv B) \\ \vdash A \supset \Diamond(A \wedge B) \\ \vdash B \supset \Diamond(A \wedge B)\end{array}}{\vdash A \equiv B}$$

10. *RE — Regra da Equivalência*

Seja A' o resultado da substituição de uma ocorrência de uma subfórmula v_1 em A por v_2. Então

$$\frac{\vdash v_1 \equiv v_2}{\vdash A \equiv A'}$$

11. *Regras da Conseqüência — Q*$\Box$, *Q*$\Diamond$ *e Q*$\circ$

$$\frac{\begin{array}{l}\vdash A \supset B \\ \vdash B \supset \Box C \\ \vdash C \supset D\end{array}}{\vdash A \supset \Box D} \qquad \frac{\begin{array}{l}\vdash A \supset B \\ \vdash B \supset \Diamond C \\ \vdash C \supset D\end{array}}{\vdash A \supset \Diamond D} \qquad \frac{\begin{array}{l}\vdash A \supset B \\ \vdash B \supset \circ C \\ \vdash C \supset D\end{array}}{\vdash A \supset \circ D}$$

12. *Regra da Concatenação —* $\Diamond C$ *e* $\Box C$

$$\frac{\begin{array}{l}\vdash A \supset \Diamond B \\ \vdash B \supset \Diamond C\end{array}}{\vdash A \supset \Diamond C} \qquad \frac{\begin{array}{l}\vdash A \supset \Box B \\ \vdash B \supset \Box C\end{array}}{\vdash A \supset \Box C}$$

IV.2.10 Teoremas

Nessa subseção, são apresentados alguns teoremas usados em provas nas seções seguintes. Suas demonstrações podem ser encontradas em MANNA (1981):

T1. $\vdash \Box\Box A \equiv \Box A$

T2. $\vdash \Diamond\Diamond A \equiv \Diamond A$

T3. $\vdash \sim\Box A \equiv \Diamond\sim A$

T4. $\vdash \Box(A \supset B) \supset (\Diamond A \supset \Diamond B)$

T5. $\vdash \Box(A \wedge B) \equiv (\Box A \wedge \Box B)$

T6. $\vdash \Diamond(A \vee B) \equiv \Diamond A \vee \Diamond B$

T7. $\vdash \Diamond(A \wedge B) \supset (\Diamond A \wedge \Diamond B)$

T8. $\vdash \Box A \vee \Box B \supset \Box(A \vee B)$

T9. $\vdash (\Box A \wedge \Diamond B) \supset \Diamond(A \wedge B)$

T10. $\vdash \circ A \supset \Diamond A$

T11. $\vdash \circ(A \wedge B) \equiv \circ A \wedge \circ B$

T12. $\vdash \circ(A \vee B) \equiv \circ A \vee \circ B$

T13. $\vdash \circ(A \supset B) \equiv \circ A \supset \circ B$

T14. $\vdash \circ(A \equiv B) \equiv (\circ A \equiv \circ B)$

T15. $\vdash \Box\circ A \equiv \circ\Box A$

T16. $\vdash \Diamond\circ A \equiv \circ\Diamond A$

T17. $\vdash \Box A \equiv A \wedge \circ\Box A$

T18. $\vdash \Diamond A \equiv A \vee \circ\Diamond A$

T19. $\vdash (A \wedge \Diamond\sim A) \supset \Diamond(A \wedge \circ\sim A)$

T20. $\vdash \circ A \mathcal{U} \circ B \equiv \circ(A \mathcal{U} B)$

T21. $\vdash (A \wedge B) \mathcal{U} C \equiv (A \mathcal{U} C) \wedge (B \mathcal{U} C)$

IV.3 A Especificação Formal de SDs com a LTLp

Dado uma descrição informal de um SD, é possível preparar uma especificação formal das propriedades que possíveis implementações devam possuir. A tarefa de verificar se

uma implementação está correta, isto é, se uma implementação satisfaz aos requisitos da descrição informal, reduz-se à tarefa de provar que a implementação atende às propriedades formalmente especificadas. Pode-se, ainda, usar a especificação formal como um modelo do sistema real, realizando verificações de aspectos relevantes, como as propriedades de segurança ou probidade do SD.

IV.3.1 Um Exemplo

Seja um SD composto por 3 processos P_1, P_2 e S e um recurso a ser compartilhado pelos processos $P_i, i = 1, 2$. Este recurso deve ser usado exclusivamente por *um* processo, isto é, um processo a cada vez, podendo ser, por exemplo, uma impressora, um programa ou informações. A figura IV.3 pode representar este SD. Na figura, os processos estão

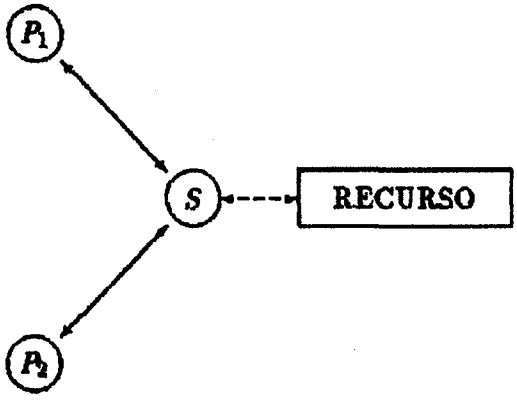

Figura IV.3: Exemplo de SD com processo sincronizador S

representados por círculos e o recurso por um retângulo. As retas orientadas representam canais de comunicação estabelecidos entre os processos, cuja existência será assumida na especificação formal. Também será assumido que não há perdas de mensagens, isto é, os canais são 100% confiáveis — toda mensagem enviada eventualmente chega ao seu destino.

O SD funciona de seguinte maneira: o processo P_i manifesta seu desejo de utilizar o recurso enviando um pedido para o processo S. O processo S recebe o pedido e, se o recurso estiver disponível, autoriza o processo P_i a utilizar o recurso. Esta autorização é uma mensagem enviada por S para P_i. Quando P_i recebe essa autorização, ele começa a

usar o recurso. Nesse exemplo, o uso do recurso pelos processos se verifica em um tempo finito: eventualmente o processo deixará de usar o recurso. Quando isso ocorrer, o processo P_i envia para S uma mensagem avisando que está liberando o recurso.

Por questões de simplicidade, no exemplo, não há nenhum tipo de *bufferização* de mensagens — estas são entregues à medida que são enviadas. Por isso, quando um processo atinge um comando de envio (ou recebimento) de mensagem, ele espera até que o outro processo correspondente atinja a operação de recebimento (ou envio) que case. Essa característica corresponde aos comandos de transmissão de mensagens da linguagem CSP (HOARE, 1978 — vide seção II.1.8).

IV.3.2 Uma Especificação Formal

Seguindo a idéia de MANNA *et alia* (1984), a especificação formal do SD será feita apenas em função de mensagens enviadas/recebidas. Serão usadas as seguintes constantes com o significado pretendido:

p_i: o processo P_i enviou um *pedido* para S — P_i manifesta seu desejo de utilizar o recurso;

a_i: o processo S enviou uma *autorização* para P_i — S autoriza P_i a usar, de modo exclusivo, o recurso;

l_i: o processo P_i enviou uma mensagem informando que o recurso está *liberado*;

Como a transmissão das mensagens é sincronizada, o envio e o recebimento de uma mensagem pode ser visto como uma única ação. Portanto, p_i representa tanto o envio como a recepção do pedido, o mesmo podendo ser dito para a_i e l_i. Algumas propriedades que o SD da figura IV.3 deve possuir são:

- *Propriedade 1*: Como há uma comunicação síncrona sempre é verdade que, somente um dos fatos p_i, a_i ou l_i pode se verificar a cada vez. Isto é, sempre é verdade que, se p_i for verificado, então nem a_i nem l_i são verificados, o que pode ser formalizado por

$$\forall_1 a : \Box[p_i \supset \sim(a_i \vee l_i)]$$

Semelhantemente, sempre é verdade que, quando a_i for verificado, nem p_i nem l_i são verificados:

$$\psi_1 b : \Box[a_i \supset \sim(p_i \vee l_i)]$$

Também, sempre é verdade que, quando l_i for verificado, nem p_i nem a_i são verificados:

$$\psi_1 c : \Box[l_i \supset \sim(p_i \vee a_i)]$$

- *Propriedade 2*: Sempre é verdade que, todo pedido feito deve eventualmente ser autorizado. Podemos formalizar isso por

$$\psi_2 : \Box(p_i \supset \Diamond a_i)$$

- *Propriedade 3*: Assumiu-se que o tempo de uso do recurso pelos processos P_i é finito. Logo, sempre é verdade que, após uma autorização ser dada, eventualmente o processo que utiliza o recurso liberará o recurso:

$$\psi_3 : \Box(a_i \supset \Diamond l_i)$$

- *Propriedade 4*: Sempre é o caso que, se for dada uma autorização a P_i, então não pode ser dada uma autorização a P_j até que P_i libere o recurso:

$$\psi_4 : \Box(a_i \supset (\sim a_j \mathcal{U} l_i)) \qquad i, j \in \{1,2\}, i \neq j$$

IV.3.3 Verificando a Especificação Formal

Após ter sido feita a especificação formal, pode-se usá-la para analisar o comportamento das implementações que a satisfizerem. Seja Ψ a conjunção das propriedades acima. Então com base em Ψ, podemos afirmar que o SD apresenta as propriedades da exclusão mútua e da probidade.

Exclusão Mútua: Toda implementação que satisfizer Ψ não autorizará os dois processos P_i e P_j a utilizarem o recurso simultaneamente, isto é, dado Ψ, então $\Box \sim (a_i \wedge a_j)$ é verificado.

Prova:

1. $\vdash \Psi \supset \Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i)$	*TP* ($\psi_1 b$ e ψ_4)
2. $\vdash \Psi \supset \Box[(a_i \supset \sim l_i) \wedge (a_i \supset \sim a_j \mathcal{U} l_i)]$	*RP* (**T5**)
3. $\vdash \Psi \supset \Box(a_i \supset \sim l_i \wedge \sim a_j \mathcal{U} l_i)$	*RP*
4. $\vdash (\sim a_j \mathcal{U} l_i) \equiv (l_i \vee (\sim a_j \wedge \circ(\sim a_j \mathcal{U} l_i)))$	**A9**
5. $\vdash (a_i \supset \sim l_i \wedge \sim a_j \mathcal{U} l_i) \supset$ $[a_i \supset (\sim l_i \wedge (l_i \vee (\sim a_j \wedge \circ(\sim a_j \mathcal{U} l_i))))]$	*RP*
6. $\vdash (a_i \supset \sim l_i \wedge \sim a_j \mathcal{U} l_i) \supset$ $[a_i \supset \sim l_i \wedge \sim a_j \wedge \circ(\sim a_j \mathcal{U} l_i)]$	*RP*
7. $\vdash (a_i \supset \sim l_i \wedge \sim a_j \mathcal{U} l_i) \supset (a_i \supset \sim a_j)$	*RP*
8. $\vdash (a_i \supset \sim l_i \wedge \sim a_j \mathcal{U} l_i) \supset (\sim a_i \vee \sim a_j)$	*RP*
9. $\vdash \Box(a_i \supset \sim l_i \wedge \sim a_j \mathcal{U} l_i) \supset \Box(\sim a_i \vee \sim a_j)$	*I*$\Box$, 8 e **A2**
10. $\vdash \Psi \supset \Box(\sim a_i \vee \sim a_j)$	*RP*
11. $\vdash \Psi \supset \Box \sim (a_i \wedge a_j)$	*RP*

■

Probidade: Em toda implementação que satisfizer Ψ, se forem feitos pedidos dos dois processos, cada um será autorizado (mas não ao mesmo tempo — vide propriedade de exclusão mútua acima).

Prova:

1. $\vdash \Box(p_i \supset \Diamond a_i) \supset (\Diamond p_i \supset \Diamond\Diamond a_i)$	**T4**
2. $\vdash \Box(p_i \supset \Diamond a_i) \supset (\Diamond p_i \supset \Diamond a_i)$	*RP* (**T2**)
3. $\vdash \Psi \supset (\Diamond p_i \supset \Diamond a_i)$	*RP*
4. $\vdash \Psi \supset (\Diamond p_j \supset \Diamond a_j)$	*RP*
5. $\vdash \Psi \supset [(\Diamond p_i \wedge \Diamond p_j) \supset (\Diamond a_i \wedge \Diamond a_j)]$	*RP*
6. $\vdash \Box\Psi \supset \Box[(\Diamond p_i \wedge \Diamond p_j) \supset (\Diamond a_i \wedge \Diamond a_j)]$	*I*$\Box$
7. $\vdash \Psi \supset \Box[(\Diamond p_i \wedge \Diamond p_j) \supset (\Diamond a_i \wedge \Diamond a_j)]$	**T1**

■

Podemos estender os resultados desse exemplo da seguinte maneira: toda implementação que satisfizer Ψ (isto é, a conjunção das propriedades), apresentará um comportamento *seguro*, isto é, os dois processos nunca serão autorizados a usar o recurso ao mesmo tempo, e *probo*, isto é, o pedido de um processo não será negligenciado para sempre.

IV.4 A Lógica Temporal Linear de Primeira Ordem (LTLPo)

A lógica temporal linear pode ser estendida a partir de sua base proposicional a uma lógica de predicados de primeira ordem de modo análogo a como é feito no caso clássico.

IV.4.1 Sintaxe

Aos símbolos do alfabeto da $\mathcal{L}_{LTL_P}$ (itens 1, 2, 3 e 4) acrescentamos a lista de símbolos abaixo para obtermos $\mathcal{L}_{LTLPo}$, a linguagem da LTLPo:

5. Operador clássico: $=$ (*igualdade*);

6. Quantificadores (de primeira ordem): $\forall$ (*quantificador universal*) e $\exists$ (*quantificador existencial*) — só serão aplicados às variáveis individuais globais (vide abaixo);

7. Símbolos de funções n-árias $f_i^n (i \geq 1, n \geq 0)$; f_i^0 é chamado uma *constante individual* e é escrito também como a_i;

8. Símbolos de predicados nários $p_i^n (i \geq 1, n \geq 0)$; p_i^0 é chamado uma *constante proposicional*;

9. Variáveis (individuais): $F_i^0, i \geq 1$, é escrito também como $x_i, i \geq 1$;

Observações:

i) O conjunto de símbolos composto das variáveis e constantes divide-se em 2 subconjuntos, o conjunto dos *símbolos locais* e o dos *símbolos globais*. Os *símbolos globais* possuem uma interpretação uniforme sobre todo o universo e não mudam seus valores ou significados de um estado para outro. Os *símbolos locais*, por outro lado, podem assumir diferentes significados e valores em diferentes estados do universo. Para o objetivo pretendido, raciocinar sobre programas, os únicos símbolos locais que interessam são variáveis individuais e constantes proposicionais locais e os símbolos globais serão de todos os tipos.

ii) Nos itens 7 a 9, o i como subscrito é usado para se fazer distinção entre vários símbolos iguais e o n como superscrito refere-se ao número de argumentos aos quais a função ou predicado se aplica.

Usando esses símbolos, definimos recursivamente três classes de expressões: *termos*, *fórmulas atômicas* e *fórmulas bem-formadas*:

1. Termos — são construídos a partir de constantes individuais e variáveis às quais são aplicadas funções. A aplicação deve obedecer as restrições de número e tipo dos argumentos:

   a) cada variável individual $x_i, i \geq 1$ é um termo;

b) para todo $i \geq 1$ e $n \geq 0$, se $t_1, t_2, \ldots, t_n$ são termos e f_i^n é um símbolo de função n-ária, então $f_i^n(t_1, t_2, \ldots, t_n)$ é um termo;

2. Fórmulas Atômicas — consistem de proposições e predicados (incluindo o operador =) aplicado a termos de tipos apropriados:

   a) V e F são fórmulas atômicas;

   b) para todo $i \geq 1$ e $n \geq 0$, se $t_1, t_2, \ldots, t_n$ são termos e p_i^n é um símbolo de predicado n-ário, então $p_i^n(t_1, t_2, \ldots, t_n)$ é uma fórmula atômica;

   c) se t_1 e t_2 são termos, então $(t_1 = t_2)$ é uma fórmula atômica

3. Fórmulas Bem-formadas — são obtidas a partir das fórmulas atômicas às quais são aplicadas os conectivos, os operadores clássicos, a quantificação sobre variáveis individuais globais e os operadores temporais:

   a) toda fórmula atômica é uma fórmula bem-formada;

   b) se A é uma fórmula bem-formada, então (A) também é uma fórmula bem-formada;

   c) se A, B e C são fórmulas bem-formadas, então $\sim A$, $A \supset B$, $A \wedge B$, $A \vee B$ e $A \equiv B$ também são fórmulas bem-formadas;

   d) se x é uma variável individual global e A é uma fórmula bem-formada, então $\forall x\, A$ e $\exists x\, A$ são fórmulas bem-formadas;

   e) se A e B são fórmulas bem-formadas, então $\Box A$, $\Diamond A$, $\circ A$ e $A \mathcal{U} B$ também são fórmulas bem-formadas.

Tomaremos as convenções e definições dos operadores lógicos e suas prioridades. Diremos que uma fórmula da $\mathcal{L}_{LTLP_o}$ é *fechada* se ela não contém variáveis globais livres. Se $x_1, \ldots, x_n$ são todas variáveis globais livres de alguma fórmula A, então a fórmula $\forall x_1, \ldots, \forall x_n A$ é chamada o *fecho universal* de A.

IV.4.2 Estrutura Temporal de Primeira Ordem (K)

Estendendo a noção de estrutura temporal dada anteriormente, obtemos a estrutura temporal de primeira ordem. Uma estrutura temporal de primeira ordem K para a $\mathcal{L}_{LTLP_o}$ é uma tripla (S, ξ, W) tal que:

- **S** é uma estrutura para a $\mathcal{L}_{LTLPo}$ (vide definição de estrutura **S** para a LPPo, na subseção A.2.2);
- ξ é uma avaliação de variáveis globais com respeito a **S**, isto é, uma atribuição de um elemento de **S** a toda variável global;
- **W** é uma seqüência infinita de *estados*, $\mathbf{W} = s_0 s_1 s_2 \cdots$, onde cada s_i é uma avaliação de variáveis locais com respeito a **S**, isto é, uma atribuição de um elemento de $|\mathbf{S}|$ a cada variável local.

IV.4.3 Semântica

Para quaisquer **K**, **S**, ξ e $s_i \in \mathbf{W}$ definimos um valor $\mathbf{S}^{(\xi,s_i)}(t) \in |\mathbf{S}|$ para cada termo t e um valor $\mathbf{S}^{(\xi,s_i)}(A) \in \{\mathbf{f},\mathbf{v}\}$ para toda fórmula atômica exatamente como na lógica clássica (apêndice A):

- $\mathbf{S}^{(\xi,s_i)}(x) = \xi(x)$ para toda variável global x;
- $\mathbf{S}^{(\xi,s_i)}(a) = s_i(a)$ para toda variável local a;
- $\mathbf{S}^{(\xi,s_i)}(f(t_1,\ldots,t_n)) = \mathbf{S}(f)(\mathbf{S}^{(\xi,s_i)}(t_1),\ldots,\mathbf{S}^{(\xi,s_i)}(t_n))$ para todo símbolo f de função n-ária;
- $\mathbf{S}^{(\xi,s_i)}(p(t_1,\ldots,t_n)) = \mathbf{v}$ se, e somente se, $(\mathbf{S}^{(\xi,s_i)}(t_1),\ldots,\mathbf{S}^{(\xi,s_i)}(t_n)) \in \mathbf{S}(p)$ para todo símbolo p de predicado n-ário (diferente de $=$);
- $\mathbf{S}^{(\xi,s_i)}(t_1 = t_2) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi,s_i)}(t_1) \stackrel{\mathbf{S}}{=} \mathbf{S}^{(\xi,s_i)}(t_2)$.

Observação: As convenções tipográficas de referência às interpretações do apêndice A (subseção A.2.3) são estendidas naturalmente para a LTLPo.

$\mathbf{S}^{(\xi,s_i)}$ agora faz o papel de s_i no caso da LTLp e podemos definir indutivamente o $\mathbf{K}_i(F) \in \{\mathbf{v},\mathbf{f}\}$ para todo $\mathbf{K} = (\mathbf{S},\xi,\mathbf{W})$, toda fórmula F e $i \in \mathbb{N}_0$. Informalmente, $\mathbf{K}_i(F)$ significa "o valor verdade de F no estado s_i":

1. $\mathbf{K}_i(A) = \mathbf{S}^{(\xi,s_i)}(A)$ para toda fórmula atômica;
2. $\mathbf{K}_i(\sim A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{f}$;

3. $\mathbf{K}_i(A \wedge B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{v}$ e $\mathbf{K}_i(B) = \mathbf{v}$;

4. $\mathbf{K}_i(A \vee B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{v}$ ou $\mathbf{K}_i(B) = \mathbf{v}$;

5. $\mathbf{K}_i(A \supset B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A) = \mathbf{f}$ ou $\mathbf{K}_i(B) = \mathbf{v}$;

6. $\mathbf{K}_i(A \equiv B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_i(A \supset B) = \mathbf{v}$ e $\mathbf{K}_i(B \supset A) = \mathbf{v}$;

7. $\mathbf{K}_i(\Box A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_j(A) = \mathbf{v}$ para todo $j \geq i$;

8. $\mathbf{K}_i(\Diamond A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_j(A) = \mathbf{v}$ para algum $j \geq i$;

9. $\mathbf{K}_i(\circ A) = \mathbf{v}$ se, e somente se, $\mathbf{K}_{i+1}(A) = \mathbf{v}$;

10. $\mathbf{K}_i(A\mathcal{U}B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo k, $i \leq k < j$;

11. $\mathbf{K}_i(\forall x A) = \mathbf{v}$ se, e somente se, $\mathbf{K}'_i(A) = \mathbf{v}$ para toda estrutura temporal $\mathbf{K}' = (\mathbf{S}, \xi', \mathbf{W})$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x;

12. $\mathbf{K}_i(\exists x A) = \mathbf{v}$ se, e somente se, $\mathbf{K}'_i(A) = \mathbf{v}$ para alguma estrutura temporal $\mathbf{K}' = (\mathbf{S}, \xi', \mathbf{W})$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x;

IV.4.4 Um Exemplo da LTLPo

Sejam p um símbolo de predicado binário e f um símbolo de função binária. Seja a estrutura temporal de primeira ordem $\mathbf{K} = (\mathbf{S}, \xi, \mathbf{W})$, com $|\mathbf{S}| = \mathbb{N}_0$, $\mathbf{S}(p) = <$, isto é,

$$\mathbf{p}(\mathbf{x}, \mathbf{y}) = \mathbf{v} \Leftrightarrow (\mathbf{x} > \mathbf{y})$$

e $\mathbf{S}(f) = +$, isto é,

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathbf{x} + \mathbf{y}$$

e os valores da variável local a conforme a matriz da figura IV.4.

Da estrutura $\mathbf{K}$, obtemos:

- Seja t o termo $f(y, a)$, então $\mathbf{K}_i(t) = \mathbf{y} + s_i(a)$. Assim, temos que $\mathbf{K}_0(t) = \mathbf{y} + s_0(a) = \mathbf{y} + 1$, $\mathbf{K}_1(t) = \mathbf{y} + 0 = \mathbf{y}$, $\mathbf{K}_2(t) = \mathbf{y} + 3$ e $\mathbf{K}_3(t) = \mathbf{y} + 2$;

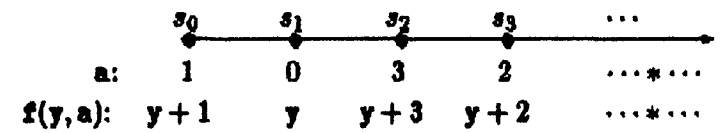

Figura IV.4: Matriz representando uma estrutura temporal de primeira ordem K

- Seja A a fórmula atômica $p(x, f(y,a))$, então $\mathbf{K}_i(A) = \mathbf{v} \Leftrightarrow x < y + s_i(a)$. Assim, temos que $\mathbf{K}_0(A) = \mathbf{v} \Leftrightarrow x < y + 1$, $\mathbf{K}_1(A) = \mathbf{v} \Leftrightarrow x < y$, $\mathbf{K}_2(A) = \mathbf{v} \Leftrightarrow x < y + 3$ e $\mathbf{K}_3(A) = \mathbf{v} \Leftrightarrow x < y + 2$;

- Seja B a fórmula $\forall y A$, isto é, $\forall y p(x, f(y,a))$, então temos que $\mathbf{K}_0(B) = \mathbf{v} \Leftrightarrow x < y+1$ para todo $y \in \mathbb{N}_0$, $\mathbf{K}_1(B) = \mathbf{v} \Leftrightarrow x < y$ para todo $y \in \mathbb{N}_0$, $\mathbf{K}_2(B) = \mathbf{v} \Leftrightarrow x < y + 3$ para todo $y \in \mathbb{N}_0$ e $\mathbf{K}_3(B) = \mathbf{v} \Leftrightarrow x < y + 2$ para todo $y \in \mathbb{N}_0$;

- Seja C a fórmula bem-formada $\circ B$, isto é, $\circ \forall y p(x, f(y,a))$, então $\mathbf{K}_0(C) = \mathbf{K}_1(B)$, $\mathbf{K}_1(C) = \mathbf{K}_2(B)$ e $\mathbf{K}_2(C) = \mathbf{K}_3(B)$;

- Seja D a fórmula bem-formada $\exists x C$, isto é, $\exists x \circ \forall y p(x, f(y,a))$, então

  $\mathbf{K}_0(D) = \mathbf{v} \Leftrightarrow$ existe $x \in \mathbb{N}_0$ tal que $x < y$ para todo $y \in \mathbb{N}_0$,

  $\mathbf{K}_1(D) = \mathbf{v} \Leftrightarrow$ existe $x \in \mathbb{N}_0$ tal que $x < y + 3$ para todo $y \in \mathbb{N}_0$,

  $\mathbf{K}_2(D) = \mathbf{v} \Leftrightarrow$ existe $x \in \mathbb{N}_0$ tal que $x < y + 2$ para todo $y \in \mathbb{N}_0$.

  temos que $\mathbf{K}_0(D) = \mathbf{f}$ desde que, para $y = 0$ não existe $x \in \mathbb{N}_0$ tal que $x < y$ e $\mathbf{K}_1(D) = \mathbf{K}_2(D) = \mathbf{v}$ pois, se escolhemos $x = 0 \in \mathbb{N}_0$, as condições são satisfeitas.

IV.4.5 Definições

Após definir a "verdade de uma fórmula em um estado", definimos as noções semânticas restantes. As definições de fórmulas válidas numa estrutura K e fórmula válida são idênticas às dadas para as fórmulas da LTLp:

Definição IV.4.5.1 Uma fórmula A da $\mathcal{L}_{LTLP_\sigma}$ *segue de um conjunto* $\mathcal{F}$ *de fórmulas fechadas*, se $\models_{\mathbf{K}} A$ para todo K tal que $\models_{\mathbf{K}} B$ para todo $B \in \mathcal{F}$.

Definição IV.4.5.2 Uma fórmula A da $\mathcal{L}_{LTLP_o}$ *segue de um conjunto* $\mathcal{F}$ *de fórmulas*, denotando-se por $\mathcal{F} \models A$, se A segue do conjunto de fechos universais de todas as fórmulas de $\mathcal{F}$.

IV.4.6 O Sistema Formal Σ_{LTLP_o}

A LTLPo é uma extensão feita à LTLp. Na axiomatização da LTLPo, usamos Σ_{LTLp}, o sistema formal da LTLp, e acrescentamos alguns esquemas de axiomas e regras de inferência. Chamaremos esse sistema formal de Σ_{LTLP_o}.

Axiomas

Os esquemas de axiomas do Σ_{LTLP_o} consistem em todos os esquemas de axiomas **A1–A10** do Σ_{LTLp}, acrescentando

A11. $\vdash \sim \exists x A \equiv \forall x \sim A$

A12. $\vdash \forall x A \supset A_x(t)$ onde t é qualquer termo substituível em A (t é globalmente livre para x em A.)

A13. $\vdash \forall x \Box A \supset \Box \forall x A$

A14. $\vdash \forall x \circ A \supset \circ \forall x A$

A15. $\vdash x = x$

A16. $\vdash x = y \supset (A \supset A_x(y))$ se A não contiver operadores temporais.

Um termo t é dito *globalmente livre para* x em A se a substituição de t por todas as ocorrências livres de x em A:

a) não crie novas ocorrências ligadas de variáveis (globais), e

b) não crie novas ocorrências de variáveis locais no escopo de um operador modal.

Nos axiomas **A11—A16**, x é qualquer variável individual global. Os axiomas **A11** e **A12** provêm do cálculo de predicados: **A11** define $\exists$ como o dual de $\forall$ e **A12**

é o axioma da instanciação. O axioma **A13** é conhecido como a fórmula de Barcan conectando os dois operadores universais, $\forall$ e $\Box$. **A14** é a fórmula de Barcan para o operador $\circ$. **A15** afirma a *reflexividade* da igualdade. O axioma **A16** afirma a propriedade da *substitutividade* da igualdade.

Regras de Inferência

Todas as regras **R1–R3** do Σ_{LTLp}, acrescentando

R4. Se $\vdash A \supset B$ então $\vdash A \supset \forall x B$ se não houver nenhuma ocorrência livre de x em A
(*Inserção do* $\forall$ — $I\forall$)

IV.4.7 Consistência do $\Sigma_{LTLP\circ}$

Seja A uma fórmula e $\mathcal{F}$ um conjunto de fórmulas. Afirmamos que:

Se $\mathcal{F} \vdash A$ *então* $\mathcal{F} \models A$

ou seja, admitindo que $\mathcal{F}$ é $\Sigma_{LTLP\circ}$,

O sistema $\Sigma_{LTLP\circ}$ *é consistente.*

Prova: A prova segue por indução, à semelhança da prova de consistência do Σ_{LTLp}, assumindo-se a derivação de A a partir de $\mathcal{F}$ ($\mathcal{F} \vdash A$):

1. A é um esquema de axioma do $\Sigma_{LTLP\circ}$. Provamos, no apêndice B, a validade dos esquemas de axiomas **A1–A16**. Em todos os casos, $\mathcal{F} \models A$.
2. $A \in \mathcal{F}$. Nesse caso, temos trivialmente que $\mathcal{F} \models A$.
3. A é uma conclusão de uma das regras de Σ_{LTLp}: as regras **R1–R3** são tratadas exatamente como na prova da consistência de Σ_{LTLp}, restando somente **R4**:
(Consistência de $I\forall$) Vamos assumir como hipótese de indução $\mathcal{F} \models A \supset B$. Seja **K** uma estrutura temporal, $\models_{\mathbf{K}} C$ para todos os fechamentos universais C de fórmulas de $\mathcal{F}$, daí $\models_{\mathbf{K}} A \supset B$. Vamos supor que $\mathbf{K}_i(A \supset \forall x B) = \mathbf{f}$ para algum $i \in \mathbb{N}_0$, isto é,

$\mathbf{K}_i(A) = \mathbf{v}$ e $\mathbf{K}_i(\forall x B) = \mathbf{f}$. Então há algum $\mathbf{K}'$ com $\mathbf{K}'_i(B) = \mathbf{f}$ e $\mathbf{K}'$ difere de $\mathbf{K}$ no máximo no valor de ξ para x. Isso significa também que $\models_{\mathbf{K}'} C$ para todo C como acima e, portanto, $\models_{\mathbf{K}'} A \supset B$. Desde que x não possui nenhuma ocorrência livre em A, temos que $\mathbf{K}'_i(A) = \mathbf{v}$, e, portanto, $\mathbf{K}'_i(B) = \mathbf{v}$. Isso é uma contradição, logo $\mathbf{K}_i(A \supset \forall x B) = \mathbf{v}$ para todo i, isto é, $\models_{\mathbf{K}}(A \supset \forall x B)$ e, portanto, $\mathcal{F} \models A \supset \forall x B$.

■

IV.4.8 Incompleteza do Σ_{LTLPo}

SZALAS *et alia* (1988) demonstram que a LTLPo é incompleta no sentido de que o conjunto de todas as fórmulas logicamente válidas não é recursivamente enumerável. Sendo assim, o conjunto de teoremas de Σ_{LTLPo}, que é recursivamente enumerável, não pode ser igual ao conjunto das fórmulas válidas. A técnica utilizada nessa demonstração consiste em codificar a computação de uma máquina de Turing $\mathcal{I}$ como uma seqüência temporal de estados. Em seguida, é construída uma fórmula $f_{laço}$ da LTLPo tal que $\models f_{laço}$ se, e somente se, $\mathcal{I}$ entra em laço (*loop*). Desde que o problema de checar se uma máquina de Turing entra em laço não é recursivamente enumerável (é o complemento do problema da parada) e $f_{laço}$ é uma fórmula temporal contendo apenas os símbolos de igualdade e constantes, temos que o conjunto das fórmulas válidas da LTLPo não é recursivamente enumerável. O artigo de SZALAS *et alia* (1988) também traz outro resultado interessante: conjuntos finitos podem ser caracterizados por fórmulas da LTLPo, mas não por fórmulas da LPPo (apêndice A).

IV.4.9 Regras Derivadas

Exibimos algumas regras derivadas que serão úteis nas demonstrações adiante. As demonstrações dessas regras podem ser encontradas em MANNA (1981).

1. *INST — Regra da Instanciação*

$$\frac{\vdash A(x)}{\vdash A_x(t)B}$$

Onde t é qualquer termo globalmente livre para x em A.

2. *I∃ — Inserção do* ∃

$$\frac{\vdash A \supset B}{\vdash \exists x A \supset B}$$

Onde x não está livre em B.

3. *Regras* $\forall\forall$

a) $\dfrac{\vdash A \supset B}{\vdash \forall x A \supset \forall x B}$ b) $\dfrac{\vdash A \equiv B}{\vdash \forall x A \equiv \forall x B}$

4. *Regras* $\exists\exists$

a) $\dfrac{\vdash A \supset B}{\vdash \exists x A \supset \exists x B}$ b) $\dfrac{\vdash A \equiv B}{\vdash \exists x A \equiv \exists x B}$

5. *RE — Regra da Equivalência*

Seja A' o resultado da substituição de uma ocorrência de uma subfórmula v_1 em A por v_2. Então

$$\frac{\vdash v_1 \equiv v_2}{\vdash A \equiv A'}$$

IV.4.10 Teoremas

Nessa subseção, são apresentados alguns teoremas obtidos no Σ_{LTLPo}. Suas demonstrações encontram-se em (MANNA, 1981).

T22. $\vdash (\forall x \Box A) \equiv (\Box \forall x A)$

T23. $\vdash (\exists x \Diamond A) \equiv (\Diamond \exists x A)$

T24. $\vdash (\forall x \circ A) \equiv (\circ \forall x A)$

T25. $\vdash (\exists x \circ A) \equiv (\circ \exists x A)$

IV.5 A Especificação Formal de SDs com a LTLPo

Para exemplificar o uso da LTLPo na especificação formal de SDs, consideraremos um problema padrão do domínio de protocolos: a especificação de um protocolo de comunicação de *bit* alternado (PBA).

IV.5.1 O Protocolo de *Bit* Alternado (PBA)

O protocolo de *bit* alternado proporciona um serviço confiável de transferência de mensagens sobre um meio de transmissão não-confiável que liga dois pontos. Um meio de transmissão é não-confiável se ele pode perder, duplicar ou adulterar (corromper) mensagens. Entretanto, há uma probabilidade não-nula de transmissão de mensagens bem-sucedida, isto é, se uma determinada mensagem for enviada repetidamente, então pelo menos uma delas atingirá seu destino (CHEN *et alia*, 1983).

Seja o sistema da figura IV.5. Entre os ambientes 1 e 2 só existem canais de comunicação não confiáveis. Isso é o caso, por exemplo, de duas redes locais localizadas distantes entre si e que se comuniquem utilizando um meio de transmissão de longas distâncias como satélite, ondas de rádio, telefone, etc., formando uma rede de grande área (*Wide Area Network* — WAN) (CRICHLOW, 1988). A tarefa do PBA é entregar mensagens do ambiente 1 para o ambiente 2 e vice-e-versa. Nesse exemplo, será especificado apenas *metade* do PBA: o serviço de transferência de mensagens do ambiente 1 para o ambiente 2, somente. A outra metade é semelhante, invertendo-se os ambientes de envio e recebimento.

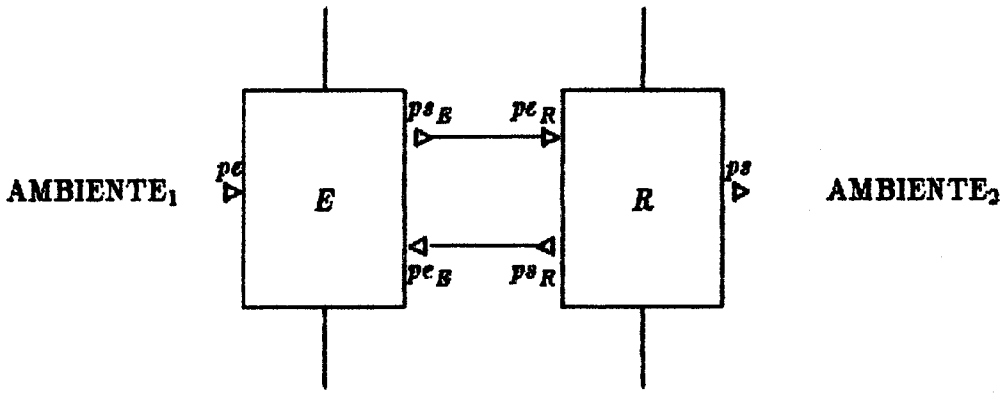

Figura IV.5: Esquema do Protocolo de *Bit* Alternado (PBA)

No decorrer desse exemplo será assumido a existência de portas (BALZER, 1971; aqui representadas por "$\triangleright$" e "$\triangleleft$") de recebimento de mensagens pe e pe_E em E e pe_R em R, a existência de portas de envio de mensagens ps e ps_R em R e ps_E em E e canais

de comunicação ligando essas portas (aqui representadas por linhas horizontais ligando portas) estabelecidos, conforme a figura IV.5.

Quando o processo E (estação de *Envio*) recebe uma mensagem m pela porta pe, isso é entendido como uma solicitação feita pelo ambiente 1 para entregar a mensagem ao ambiente 2. Ao receber essa mensagem, E obtém um número b de um bit (valores 0 ou 1) da seguinte forma: quando é recebida a primeira mensagem, é atribuído a b o valor 0 ou 1, arbitrariamente; quando as mensagens seguintes são recebidas, o valor de b é alternado — se b tinha valor 0 na mensagem anterior, com a nova mensagem o valor de b será 1 e vice-e-versa (daí o nome protocolo de *bit alternado*).

Após E obter o valor de b, ele o anexa à mensagem, compondo um pacote $\langle m, b\rangle$. Esse pacote é enviado através de ps_E por um meio de transmissão não-confiável que pode perder, duplicar, ou adulterar $\langle m, b\rangle$. Por isso, após enviar o pacote, o processo E espera por uma confirmação (com o mesmo valor de b) enviada de R que chegará através de pe_E: se o tempo de espera expirar e a confirmação não for recebida, E torna a enviar o mesmo pacote $\langle m, b\rangle$ para R.

Se E enviar repetidamente o pacote $\langle m, b\rangle$, então pelo menos um pacote chegará a R através de pe_R. Quando o pacote chega a R, seus dois componentes são separados. Comparando o valor do bit b então recebido com o valor do *bit* do pacote recebido anteriormente (no caso de ser a primeira vez, o valor do *bit* anterior é indeterminado, mas forçosamente diferente de 0 e 1). Se os valores forem diferentes, conclui-se que a mensagem m é recebida pela primeira vez e deve ser entregue ao ambiente 2 (atualizando o valor do *bit* anterior), através da porta ps. Se os valores forem iguais, então não é a primeira vez que a mensagem é recebida e, portanto, a mensagem já foi entregue ao ambiente 2. Em ambos os casos, R envia para E, através de ps_R, uma confirmação com o valor do *bit* do pacote $\langle m, b\rangle$ recebido.

O processo E enviará o pacote $\langle m, b\rangle$ para R através de ps_E até que seja recebido uma confirmação de valor igual a b através de pe_E. Como o meio de transmissão entre as portas ps_E e pe_R é não-confiável, pode ser que o pacote se perca. Se ele for perdido, a confirmação seguramente não chegará, pois R só confirma os pacotes recebidos. Neste caso, o pacote tornará a ser enviado. Pela probabilidade não-nula de transmissão bem-

sucedida de mensagens, se o pacote (m, b) for enviado de E através de ps_E repetidamente, pelo menos uma vez R o receberá através de pe_R.

O processo R envia uma confirmação para E, através de ps_R, se, e somente se, ele recebe um pacote (m, b). Se a transmissão da confirmação é bem-sucedida (E recebe a confirmação através de pe_E), E pára de enviar (m, b) pois foi confirmado que a mensagem m foi entregue ao ambiente 2. O processo E então vai esperar por outra mensagem m' do ambiente 1. Entretanto, se a transmissão da confirmação não for bem-sucedida, E não receberá a confirmação e tornará a enviar o pacote. Isso sendo feito repetidamente, acarretará o envio repetido da confirmação e pelo menos uma vez a confirmação chegará a E, e o processo E deixará de enviar o pacote (m, b).

IV.5.2 Uma Especificação Formal para o PBA

Sejam os domínios

$$\text{MENSAGENS} = \{m | mensagem(m)\},$$

$$\text{PORTAS-ENTRADA} = \{pe, pe_R, pe_E\},$$

$$\text{PORTAS-SAÍDA} = \{ps_E, ps, ps_R\} \text{ e}$$

$$\text{BINÁRIO} = \{0, 1\}, \text{BINÁRIO} \subset \text{MENSAGENS}$$

representando, respectivamente, as mensagens, as portas de entrada, as portas de saída e os valores binários. Sejam ainda os predicados

$$enviado : \text{MENSAGENS} \times \text{PORTAS-SAÍDA} \mapsto \{\text{v}, \text{f}\}$$

tal que $enviado(m, ps_E)$ representa o fato que a mensagem $m \in$ MENSAGENS foi enviada através da porta $ps_E \in$ PORTAS-SAÍDA,

$$recebido : \text{MENSAGENS} \times \text{PORTAS-ENTRADA} \mapsto \{\text{v}, \text{f}\}$$

tal que $recebido(m, pe)$ representa o fato que a mensagem $m \in$ MENSAGENS foi recebida através da porta $pe \in$ PORTAS-ENTRADA.

No decorrer do exemplo (especificação e provas) será assumido a seguinte quantificação:

$$\forall m \in \text{MENSAGENS} \; \forall b \in \text{BINÁRIO}$$

significando que as variáveis m (mensagem) e b (*bit*) não possuem aspectos temporais, isto é, serão interpretadas uniformemente no tempo (vide observação i) da seção IV.4.1).

Especificamos formalmente o protocolo enunciando propriedades que o nó E, o nó R e os canais de comunicação devem possuir para o funcionamento correto do PBA.

Estação de Envio (Nó E)

O nó E deve possuir as seguintes propriedades:

- *Propriedade 1:* Entre o ambiente 1 e o nó E não há problemas com a transmissão de mensagens pois é uma rede local em uma área geográfica restrita. Quando o ambiente 1 envia alguma mensagem m, esta chega sem problemas a E. Quando isso ocorre é porque o ambiente 1 está solicitando a E que entregue, através do meio de transmissão não-confiável, a mensagem m ao ambiente 2. Após E receber a mensagem m do ambiente 1, é calculado o *bit* b levando em conta seu valor anterior (quando do envio de outra mensagem $m' \in \text{MENSAGEM}, m' \neq m$), pois deve-se alternar entre 0 e 1. O predicado $pacote(\langle m, b\rangle)$ significa que à mensagem m foi justaposto um *bit* b, sujeito à restrição anterior. Pode-se formalizar isso por

$$e_1 : \Box(recebido(m, pe) \supset pacote(\langle m, b\rangle))$$

  interpretando-se da seguinte maneira: sempre é o caso que, se a mensagem m for recebida por E através de pe, então será justaposto a m um *bit* b e as duas coisas (m e b) formarão um pacote $\langle m, b\rangle$.

- *Propriedade 2:* De posse do pacote $\langle m, b\rangle$, o nó E o envia, no próximo instante, através de ps_E. Isso sempre será verificado, e a fórmula seguinte pode representar essa propriedade:

$$e_2 : \Box(pacote(\langle m, b\rangle) \supset \circ enviado(\langle m, b\rangle, ps_E))$$

- *Propriedade 3:* Seja $\sqcup$ um operador temporal, chamado "*until* fraco", tal que, se $A \sqcup B$, não é necessário que $\Diamond B$ seja verificado (axioma **A10**), isto é, $A \sqcup B \equiv \Box A \vee (A \mathcal{U} B)$ — se $A \sqcup B$, então A é doravante verdade ou $A \mathcal{U} B$ é verdade (o que obriga $\Diamond B$ a ser verdade). Usamos esse operador para expressar a propriedade que o

pacote $\langle m, b\rangle$ será intermitentemente enviado por E até (*until* fraco) que seja recebido uma confirmação de valor b do nó R:

$$e_3 : \Box[(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E)) \sqcup recebido(b, pe_E)].$$

De e_3 e considerando a definição de $\sqcup$ em termos de $\Box$ e $\mathcal{U}$ dada anteriormente, temos

$$e_3' : \Box[\Box(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E)) \vee$$
$$(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E)) \mathcal{U} recebido(b, pe_E)].$$

- *Propriedade 4:* Sempre é verdade que, se for recebido a confirmação através de pe_E, então o nó E deve parar, para sempre, de enviar o pacote $\langle m, b\rangle$. Isso pode ser expresso por

$$e_4 : \Box(recebido(b, pe_E) \supset \Box \sim enviado(\langle m, b\rangle, ps_E))$$

- *Propriedade 5:* O pacote $\langle m, b\rangle$ será, para sempre, intermitentemente enviado através de ps_E se, e somente se, nunca for o caso de o pacote $\langle m, b\rangle$ ser intermitentemente enviado através de ps_E até que seja recebido a confirmação através de pe_E. Isto quer dizer que as subfórmulas da disjunção e_3' são mutuamente exclusivas, podendo ser formalizado por e_5:

$$e_5 : \Box(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E)) \equiv$$
$$\Box \sim (enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E)) \mathcal{U} recebido(b, pe_E))$$

- *Propriedade 6:* Sempre é o caso de o pacote $\langle m, b\rangle$ ser intermitentemente enviado através de ps_E até que seja recebido a confirmação através de pe_E se, e somente se, não for o caso de o pacote $\langle m, b\rangle$ ser para sempre intermitentemente enviado através de ps_E. Esta propriedade decorre do fato que o envio intemitente do pacote $\langle m, b\rangle$ através de ps_E é verificado até que seja recebido a confirmação através de pe_E, e pode ser formalizado por e_6:

$$e_6 : \Box((enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E)) \mathcal{U} recebido(b, pe_E)) \equiv$$
$$\sim \Box(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E))$$

Verificando a Estação de Envio (E)

Pelas propriedades do nó E formalmente especificadas, podemos, na presença de algumas hipóteses adicionais, chegar a uma conclusão indesejável. Seja Υ a conjunção das propriedades de E.

Como primeira hipótese, vamos supor que a estação E receba uma mensagem m do ambiente 1, significando que está sendo feito uma solicitação para a entrega da mensagem m ao ambiente 2. Pode-se expressar isso por h_1:

$$h_1 : \Diamond recebido(m, pe).$$

Vamos supor, ainda, que a confirmação nunca chegue a E (isso é plausível, já que nada foi explicitamente afirmado a esse respeito), o que pode ser formalizado por h_2:

$$h_2 :\sim \Diamond recebido(b, pe_E)$$

Dados Υ, h_1 e h_2, podemos afirmar que o pacote $\langle m, b\rangle$ será enviado infinitamente freqüentemente pela porta ps_E.

Prova:

1. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset \sim \Diamond recebido(b, pe)$	*RP*
2. $\vdash \sim \Diamond recebido(b, pe) \supset$ $\sim[(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond(enviado(\langle m, b\rangle, ps_E)) \mathcal{U} recebido(b, pe_E)]$	**A10**, *RP*
3. $\vdash \Box \sim \Diamond recebido(b, pe) \supset$ $\Box \sim[(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond(enviado(\langle m, b\rangle, ps_E)) \mathcal{U} recebido(b, pe_E)]$	$\Box\,\Box$
4. $\vdash \sim \Diamond recebido(b, pe) \supset$ $\Box \sim[(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond(enviado(\langle m, b\rangle, ps_E)) \mathcal{U} recebido(b, pe_E)]$	**A1 e T2**
5. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset$ $\Box \sim[(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond(enviado(\langle m, b\rangle, ps_E)) \mathcal{U} recebido(b, pe_E)]$	*RP* (1 e 4)
6. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset \Box(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond(enviado(\langle m, b\rangle, ps_E))$	*RP* (e_5 e 5)
7. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset \Box\Box(enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond(enviado(\langle m, b\rangle, ps_E))$	**T1**
8. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset \Box(\Diamond enviado(\langle m, b\rangle, ps_E) \supset \Diamond\circ\Diamond enviado(\langle m, b\rangle, ps_E))$	**T4**
9. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset \Box(\Diamond enviado(\langle m, b\rangle, ps_E) \supset \circ\Diamond enviado(\langle m, b\rangle, ps_E))$	**T16 e T2**
10. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset (\Diamond enviado(\langle m, b\rangle, ps_E) \supset \Box\Diamond enviado(\langle m, b\rangle, ps_E))$	**A8**
11. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset \Diamond enviado(\langle m, b\rangle, ps_E)$	$RP(h_1, e_1, e_2)$
12. $\vdash \Upsilon \wedge h_1 \wedge h_2 \supset \Box\Diamond enviado(\langle m, b\rangle, ps_E)$	*RP* (10 e 11)

■

A linha 12 pode ser lida da seguinte forma: para qualquer mensagem m e qualquer *bit* b, se for satisfeito Υ e supondo o recebimento de m por E (h_1) e o não-recebimento da confirmação, então o pacote $\langle m, b\rangle$ será enviado infinitamente freqüentemente. Isso é uma

conclusão indesejável: um protocolo de comunicação no qual ocorra essa situação não será de utilidade alguma. Devemos prosseguir na especificação formal dos demais componentes do PBA para que seja possível investigar a plausibilidade de h_2.

Estação de Recebimento (Nó R)

O nó R deve possuir as seguinte propriedades:

- *Propriedade 1:* Sempre quando R receber o pacote $\langle m, b\rangle$ de E, é separado mensagem e *bit*. A mensagem deve ser entregue ao ambiente 2, obedecendo à restrição de só entregá-la uma única vez, pois o ambiente 1 deseja que *uma*, e somente uma, mensagem m seja transmitida para o ambiente 2. Seja $entregue(m, ps)$ o predicado que expresse isso, isto é, ele é verificado quando a mensagem m é entregue ao ambiente 2 via ps, sujeito à restrição de só fazê-lo uma única vez. O *bit* b deve ser enviado de volta para E, via ps_R, na forma de uma confirmação. A fórmula abaixo pode expressar isso:

  $$r_1 : \Box[recebido(\langle m, b\rangle, pe_R) \equiv (\circ entregue(m, ps) \wedge \circ\circ enviado(b, ps_R))].$$

  Os operadores $\circ$ foram usados apenas para dar uma idéia de seqüencialidade das ações, determinando, também, a unicidade do estado que satisfaz cada um dos predicados.

Verificando a Estação de Recebimento (R)

De r_1 pode-se deduzir que o número de vezes que o pacote $\langle m, b\rangle$ for recebido é igual ao número de vezes que a confirmação será enviada:

Prova:

1. $\vdash r_1 \supset \Box(recebido(\langle m, b\rangle, pe_R) \supset \circ\circ enviado(b, ps_R))$ — *RP*
2. $\vdash r_1 \supset \Box\Box(recebido(\langle m, b\rangle, pe_R) \supset \Diamond enviado(b, ps_R))$ — **T1, T10 e T2**
3. $\vdash r_1 \supset \Box(\Diamond recebido(\langle m, b\rangle, pe_R) \supset \Diamond enviado(b, ps_R))$ — **T4 e T2**
4. $\vdash r_1 \supset \Box\Diamond recebido(\langle m, b\rangle, pe_R) \supset \Box\Diamond enviado(b, ps_R))$ — **A2**

■

A linha 4 pode ser lida da seguinte maneira: se for verificado r_1, então, se for recebido o pacote $\langle m, b\rangle$ de pe_R infinitamente freqüentemente, então a confirmação será

enviada infinitamente freqüentemente através de ps_R.

Os Canais de Comunicação

Os canais de comunicação são indispensáveis ao funcionamento do PBA. Por isso, faz-se necessário especificar seu comportamento. Os canais de comunicação entre E e R são não-confiáveis e suas propriedades devem refletir essa característica:

- *Propriedade 1:* Como o meio de transmissão possui uma probabilidade não-nula de transmissão de mensagens, se um número ilimitado de mensagens $\langle m, b\rangle$ idênticas é enviado através de ps_E, então não apenas uma, mas um número ilimitado de mensagens $\langle m, b\rangle$ idênticas será recebido por pe_R. Podemos formalizar essa propriedade da seguinte maneira:

$$c_1 : \Box\Diamond enviado(\langle m, b\rangle, ps_E) \supset \Box\Diamond recebido(\langle m, b\rangle, pe_R)$$

  Essa propriedade segue da definição de meio de transmissão não-confiável, por indução matemática: desde que um número ilimitado de mensagens idênticas é enviado de E, pelo menos uma delas, por exemplo, a mensagem x, atingirá R (a propriedade da probabilidade não-nula). Como o número de mensagens após x é também ilimitado, pelo menos uma dessas mensagens chegará em R. Esse mesmo processo continua indefinidamente (CHEN *et alia*, 1983).

- *Propriedade 2:* O canal de comunicação que une as portas ps_R e pe_E goza da mesma propriedade descrita anteriormente. Logo, temos

$$c_2 : \Box\Diamond enviado(b, ps_R) \supset \Box\Diamond recebido(b, pe_E)$$

Seja Γ a conjunção de c_1 e c_2.

IV.5.3 A Verificação do PBA

Após a especificação formal completa do PBA, podemos re-examinar a conclusão indesejável a que chegamos na verificação da estação de envio E. Considerando o restante da especificação, podemos concluir o oposto, isto é $\Diamond recebido(b, pe_E)$.

Prova:

1. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Box\Diamond enviado(\langle m, b\rangle, ps_E)$	RP^*
2. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Box\Diamond recebido(\langle m, b\rangle, pe_R)$	RP (1 e c_1)
3. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Box\Diamond enviado(b, ps_R)$	RP^*
4. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Box\Diamond recebido(b, pe_E)$	RP (3 e b_2)
5. $\vdash \Box\Diamond recebido(b, pe_E) \supset \Diamond recebido(b, pe_E)$	**A3**
6. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Diamond recebido(b, pe_E)$	RP (4 e 5)
7. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \sim\Diamond recebido(b, pe_E)$	RP (h_2)
8. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \supset \sim\sim\Diamond recebido(b, pe_E)$	RP (6 e 7)
9. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \supset \Diamond recebido(b, pe_E)$	RP (8)

* – de provas anteriores ■

A linha 9 pode ser lida da seguinte maneira: para qualquer mensagem m e qualquer *bit* b, se for satisfeito Υ, Γ e r_1 e dado h_1, então eventualmente será recebida a confirmação através de pe_E. O recebimento da confirmação prova a *correção* do protocolo, pois se a confirmação foi recebida, é porque ela foi enviada da estação R. Se ela foi enviada da estação R, é porque R recebeu anteriormente o pacote $\langle m, b\rangle$ e o decompôs: a mensagem m foi entregue ao ambiente 2 e o *bit* b foi enviado de volta para E como uma confirmação. Dessa forma, prova-se que o PBA funciona corretamente: se uma implementação satisfizer Υ, Γ e r_1 e se acontecer h_1, então todas as mensagens m recebidas do ambiente 1 serão entregues ao ambiente 2 e será recebido uma confirmação disso.

Na propriedade c_3', usou-se o operador $\sqcup$ como uma maneira mais fraca de expressar uma interação (o envio intermitente do pacote $\langle m, b\rangle$) da qual não se sabia se terminava ou não. Com o resultado da prova anterior, podemos afirmar que o operador $\sqcup$ pode ser reescrito como $\mathcal{U}$:

Prova:

1. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Diamond recebido(b, pe_B)$	RP^*
2. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\Diamond recebido(b, pe_B) \supset \Diamond\Box \sim enviado(\langle m, b\rangle, ps_B)$	RP (e_4 e **T4**)
3. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Diamond\Box \sim enviado(\langle m, b\rangle, ps_B)$	RP (1 e 2)
4. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset \Diamond enviado(\langle m, b\rangle, ps_B)$	$RP(h_1, e_1, e_2)$
5. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\sim\Box \sim enviado(\langle m, b\rangle, ps_B) \wedge \sim\Box\Diamond enviado(\langle m, b\rangle, ps_B)$	RP(3 e 4,**A1**)
6. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\sim\Box(\sim enviado(\langle m, b\rangle, ps_B) \vee \Diamond enviado(\langle m, b\rangle, ps_B))$	RP(5,**T8**)
7. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\Diamond(enviado(\langle m, b\rangle, ps_B) \wedge \Box \sim enviado(\langle m, b\rangle, ps_B))$	RP(6,**A1**)
8. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\Diamond(enviado(\langle m, b\rangle, ps_B) \wedge \Box\Box \sim enviado(\langle m, b\rangle, ps_B))$	RP(7,**T1**)
9. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\Diamond(enviado(\langle m, b\rangle, ps_B) \wedge \bigcirc\Box \sim enviado(\langle m, b\rangle, ps_B))$	RP(8,**A6**)
10. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\Diamond(enviado(\langle m, b\rangle, ps_B) \wedge \sim\bigcirc\Diamond enviado(\langle m, b\rangle, ps_B))$	RP(9,**A1** e **A4**)
11. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\Diamond \sim(enviado(\langle m, b\rangle, ps_B) \supset \bigcirc\Diamond enviado(\langle m, b\rangle, ps_B))$	RP(10)
12. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\sim\Box(enviado(\langle m, b\rangle, ps_B) \supset \bigcirc\Diamond enviado(\langle m, b\rangle, ps_B))$	RP(10)
13. $\vdash \Upsilon \wedge \Gamma \wedge r_1 \wedge h_1 \wedge h_2 \supset$ $\Box(enviado(\langle m, b\rangle, ps_B) \supset \bigcirc\Diamond enviado(\langle m, b\rangle, ps_B)) \mathcal{U} recebido(b, pe_B))$	RP(12 e e_6)

* – de provas anteriores ■

IV.6 Mais Alguns Operadores Temporais

Existem mais alguns operadores temporais que podem ser úteis para aplicações particulares. Consideremos brevemente alguns deles:

A**atnext**B: "A irá verificar-se no próximo instante que B for verificado" (operador *at next* ou *primeira vez*);

A**while**B: "A é verificado enquanto B for verificado (no futuro)" (operador *while*);

A**unless**B: "Se há um ponto no tempo no qual B é verificado, então A é verificado até aquele ponto ou então A verifica-se permanentemente" (operador *unless* ou $\mathcal{U}$ *fraco*);

A**before**B: "Se B for verificado em algum momento no futuro, então A é verificado antes disso" (operador *before* ou *de precedência*).

O significado formal preciso destes operadores é dado pela definição da função semântica $\mathbf{K}_i$ para tais fórmulas:

1. $\mathbf{K}_i(A\mathbf{atnext}B) = \mathbf{v}$ se, e somente se, $\mathbf{K}_j(B) = \mathbf{f}$ para todo $j > i$ ou $\mathbf{K}_k(A) = \mathbf{v}$ para o menor $k > i$ com $\mathbf{K}_k(B) = \mathbf{v}$;

2. $\mathbf{K}_i(A\mathbf{while}B) = \mathbf{v}$ se, e somente se, $[\mathbf{K}_j(B) = \mathbf{f}$ para algum $j > i$ e $\mathbf{K}_k(A) = \mathbf{K}_k(B) = \mathbf{v}$ para todo k, $i < k < j]$ ou $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k > i$;

3. $\mathbf{K}_i(A\mathbf{unless}B) = \mathbf{v}$ se, e somente se, $[\mathbf{K}_j(B) = \mathbf{v}$ para algum $j > i$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo k, $i < k < j]$ ou $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k > i$;

4. $\mathbf{K}_i(A\mathbf{before}B) = \mathbf{v}$ se, e somente se, para todo $j > i$ com $\mathbf{K}_j(B) = \mathbf{v}$ existe algum k, $i < k < j]$, com $\mathbf{K}_k(A) = \mathbf{v}$;

Esses operadores podem ser expressos pelos operadores $\circ$, $\Diamond$, $\Box$ e $\mathcal{U}$ dados anteriormente. As equivalências são mostradas abaixo e demonstradas no apêndice B.

$$A\mathbf{atnext}B \equiv \circ(\sim B \mathcal{U}(A \wedge B) \vee \Box \sim B)$$

$$A\mathbf{while}B \equiv \circ(\sim B \mathcal{U}(A \wedge B) \vee \Box A)$$

$$A\mathbf{unless}B \equiv \circ(A \mathcal{U} B \vee \Box A)$$

$$A\mathbf{before}B \equiv \circ(\sim(A \vee B) \mathcal{U}(A \wedge \sim B) \vee \Box \sim(A \vee B))$$

Essas equivalências mostram que os novos operadores não são extensões próprias da linguagem $\mathcal{L}_{LTL_P}$. Eles poderiam ter sido igualmente utilizados como uma base lingüística para a definição da $\mathcal{L}_{LTL_P}$.

Deve-se chamar a atenção ao fato de que os operadores **atnext**, **unless**, **while** e **before** têm uma propriedade em comum: eles não possuem nenhum aspecto "existencial". Isto quer dizer que, por exemplo, em $A\mathbf{unless}B$, não há necessariamente um ponto no tempo no qual B se verifica. Esses operadores são chamados operadores *fracos*. O operador $\mathcal{U}$, como foi dito na sua definição semântica como observação, é, ao contrário destes, um operador *forte*, pois necessariamente deve haver um ponto no tempo no qual, se $A \mathcal{U} B$, B seja verificado.

IV.7 Propriedades Inexprimíveis na LTL

Apesar do grande repertório de aspectos que podem ser estudados com a LTL nas suas duas versões, podemos encontrar na literatura, algumas propostas de extensão a essas lógicas devido sua incapacidade de exprimir algumas propriedades:

IV.7.1 Estendendo a LTLp

WOLPER (1983) prova que há propriedades de seqüências que não são exprimíveis na LTLp, mas são facilmente exprimíveis em linguagens baseadas em expressões regulares. Um exemplo de tal propriedade é a de que um determinado evento deva ocorrer exatamente a cada n passos da computação e pode ou não acontecer nos outros passos. Isto poderia, por exemplo, expressar que um processo deve verificar a ocorrência de uma determinada condição a cada n passos da execução.

Um outro exemplo simples é a propriedade *ímpar*(p), afirmando que uma determinada proposição p tem que ser verdade em todo estado ímpar de uma seqüência. Uma fórmula como

$$p \wedge \Box(p \supset \circ \sim p) \wedge \Box(\sim p \supset \circ p)$$

não expressa *ímpar*(p), pois ela não é satisfeita pela seqüência onde p é sempre verdade, que certamente satisfaz *ímpar*(p). WOLPER (1983) demonstra que *ímpar*(p) não é exprimível na LTLp.

WOLPER (1983) apresenta uma lógica temporal linear proposicional estendida (ETL — *Extended Temporal Logic*), que pode expressar qualquer propriedade de uma seqüência definível por uma gramática linear à direita (*right linear grammar*). Para isso, é dado um método para definir um operador temporal correspondendo a qualquer gramática linear à direita. É mostrado como estender a axiomatização da LTLp no sentido de incluir esses operadores e a axiomatização extendida é provada como sendo completa.

IV.7.2 A Especificação de Sistemas de Transmissão de Mensagens

KOYMANS (1987), usando a lógica de Kamp (uma lógica modal mais geral do que a LTL), prova que certos tipos de *buffers* ilimitados não podem ser especificados, tanto na

versão proposicional como na de primeira ordem. Esse resultado é, portanto, válido para a LTLp e LTLPo, pois estas lógicas são um particular da lógica de Kamp.

O autor, então, propõe o raciocínio sobre a n-ésima ocorrência de uma proposição e a adição da quantificação sobre tais números, sendo, para essa extensão, provado que muitos sistemas, incluindo *buffers* ilimitados podem ser especificados. Entretanto, são apresentados fortes argumentos apoiando um segundo resultado de inexpressividade, afirmando que esta adição não resolve o problema para sistemas de transmissão de mensagens que preservam a ordem e podem perder mensagens.

IV.8 Conclusões

Após a exposição e uso das LTLs, chega-se às seguintes conclusões:

1. Na LTL, o conjunto de estados $\mathbf{E} = \{s_0, s_1, s_2, \ldots\}$ juntamente com a relação ρ de acessibilidade imediata, descrevem *uma* execução do SD. Entretanto um SD pode possuir mais de uma execução possível, dependendo de seus aspectos não-determinísticos (por exemplo, atrasos ou perdas de mensagens). Nesse caso, um mesmo SD possui vários conjuntos de estados $\mathbf{E}, \mathbf{E}', \ldots$ com as respectivas relações de acessibilidade imediata. Quando se especifica formalmente um SD na LTL, fica implícito que as fórmulas se referem a *todas* as possíveis execuções do sistema.

2. *Escolhendo o Nível de Abstração* — A escolha do nível de abstração a ser usado e os aspectos a serem estudados é a atribuição de quem especifica, atendendo a critérios de adequação ao problema, facilidade de uso e interesses do projeto. No exemplo da LTLp, abstraiu-se o SD de forma a representar somente o recebimento/envio de mensagens. Como a transmissão de mensagens tem um papel decisivo no funcionamento correto de SDs, essa abstração foi adequada para o exemplo. Em outras situações, pode ser adequado especificar o controle da execução (o ponto de cada processo onde a execução está) ou valores de variáveis. O melhor a ser feito é começar com uma abstração de alto nível que pareça razoável aos propósitos e, à medida que forem encontradas dificuldades, mudar o nível de abstração e refazer a especificação.

3. *Sobre as Provas nas LTLs* — A intuição de quem conduz as provas ora ajuda, ora atrapalha. Muitas vezes, o que é intuitivo (em termos temporais) não é possível

ser provado. Isso é devido ao fato de nossa intuição do tempo ser mais abrangente do que as noções temporais da LTL, que apenas relaciona eventos na escala linear do tempo. Pensamos em eventos que ocorrem *durante* algum tempo, eventos que ocorrem n vezes, eventos que ocorrem a cada 5 minutos, etc. Essas noções são difíceis ou impossíveis de exprimir na LTL. No exemplo de especificação formal com a LTLPo, a prova final de correção foi feita informalmente porque, da maneira como a LTLPo foi axiomatizada, não havia como se referir ao passado de modo formal: o Σ_{LTLPo} não formalizou a capacidade que temos de falar sobre o passado. Poderia ter sido usado um sistema formal que utilizasse operadores que "olhassem" para o passado (LICHTENSTEIN *et alii*, 1985), entretanto, optou-se por um sistema formal mais conhecido e mais intuitivo, às custas da expressividade reduzida. Por outro lado, a intuição, às vezes, nos faz descobrir erros em especificações devido a conclusões inesperadas (não-intuitivas) encontradas nas demonstrações.

4. Os exemplos de especificação formal usando as LTLs mostram uma forma de utilização da LTL: o estudo do comportamento de um modelo. Dado uma descrição informal do problema, são estabelecidas propriedades desejáveis que uma solução (implementação) deve possuir. Raciocina-se sobre essas propriedades, face alguma hipótese, e inferimos afirmações sobre seu comportamento.

5. *Propriedades "Escondidas"* — Quando se cria constantes proposicionais ou predicados para representar algo (um fato ou evento) descrito informalmente, é comum não se formalizar os interrelacionamentos implícitos na descrição informal. Isto é o caso, por exemplo, de, em um SD composto por apenas dois processos e que não haja a "bufferização" de mensagens, usar p para representar o fato que uma mensagem m foi enviada por um processo e usar q para representar o fato que a mensagem m foi recebida por outro processo. Para descrever o fato que a mensagem enviada é eventualmente recebida, pode-se usar

$$p \supset \Diamond q \qquad (+)$$

Entretanto, há outros relacionamentos menos óbvios (mas igualmente importantes), decorrentes do *significado* de p e q e de fatos implícitos na descrição informal, tais como

$$\Box(p \supset \sim q) \qquad (*+)$$

$$\sim q \mathcal{U} p \qquad (\dagger)$$

$$\Box(p \supset \circ(\sim p \mathcal{U} q)) \qquad (\ddagger)$$

afirmando que o recebimento e o envio de uma mensagem são mutuamente exclusivos ($\star\star$), uma mensagem não é recebida até que seja enviada ($\dagger$) e após o envio de uma mensagem, não haverá outro envio até que seja recebido a mensagem ($\ddagger$). Essas fórmulas, que podem se chamar de propriedades "escondidas", são difíceis ou até impossíveis de serem deduzidos de ($\star$). Muitas vezes acontece de esses relacionamentos "escondidos" serem imprescindíveis nas provas e descobre-se que houve uma sub-especificação do projeto. Na especificação formal do PBA, as propriedades 5 e 6 da estação de envio (e_5 e e_6) formalizam fatos subjacentes à propriedade 3 (e_3).

6. *Formalizando o Informal* — A leitura da descrição informal, o entendimento do problema e posterior especificação formal é um processo extremamente complexo e delicado pois o uso de um ou outro termo na descrição informal ou o uso de um ou outro conectivo ou operador na formalização muda radicalmente a especificação formal. Isso já é sentido na Lp e LPPo e é complicado ainda mais pelos aspectos temporais a serem também formalizados. Sugere-se que, nessa fase de projeto, haja um estreito relacionamento entre as pessoas que escrevem a descrição informal e as que fazem a especificação formal, para que quaisquer dúvidas possam ser resolvidas.

7. Na especificação formal do PBA usou-se o operador $\sqcup$ ("*until* fraco") em uma das fórmulas (e_3). Isso foi feito porque queríamos expressar que uma interação (o envio intermitente de uma mensagem) era feita indefinidamente ou um número finito de vezes. Lançou-se, portanto, mão do operador $\sqcup$ para expressar a ignorância que se tinha quando da formalização. Entretanto, foi provado que a interação era limitada, e foi obtida uma fórmula na qual o operador $\sqcup$ foi substituído por $\mathcal{U}$. Isso ilustra a utilidade do processo de verificação na especificação formal.

8. Na verificação da estação de envio E, mostrou-se uma das grandes utilidades da formalização: o raciocínio formal sobre a especificação face algumas hipóteses. Isso corresponde à pergunta intuitiva que é feita quando verifica-se um programa, um algoritmo ou um projeto: "e se *isso* acontecer?", com a diferença de que na lógica temporal verifica-se *formalmente* as conseqüências das hipóteses acrescidas à especificação.

IV.9 Notas Bibliográficas

A sintaxe da LTLp e LTLPo é baseada em MANNA (1974) e MANNA *et alia* (1981a). A semântica das LTLs foi retirada de KRÖGER (1987), bem como as definições de estrutura temporal (K) e estrutura temporal de primeira ordem. O exemplo de obtenção de valor-verdade da LTLp é de KRÖGER (1987), e também as definições e teoremas, que sofreram algumas adaptações.

O sistema formal Σ_{LTLp} é adaptado de KRÖGER (1987) e MANNA (1981). A prova de consistência foi baseada em KRÖGER (1987). As regras derivadas e teoremas são de MANNA (1981). O exemplo de especificação formal de SDs com a LTLp foi inspirado em um exemplo de MANNA *et alia* (1984), usando o modelo de computação distribuída adaptado de VELOSO (1985).

Na seção da LTLPo, o exemplo de obtenção de valor-verdade é retirado de KRÖGER (1987). O sistema Σ_{LTLPo} é o de MANNA (1981) assim como as regras derivadas e teoremas. A prova de consistência foi adaptada de KRÖGER (1987). O exemplo de especificação formal com a LTLPo (Protocolo de *Bit* Alternado — PBA) foi adaptado de CHEN *et alia* (1983), onde aparece uma especificação formal em lógica de predicados de primeira ordem. Uma especificação do PBA também aparece em SCHWARTZ *et alia* (1981), mas sua especificação é muito complexa para permitir uma verificação manual.

As provas dos exemplos de especificação formal são uma aplicação dos sistemas formais às propriedades. Entretanto, alguns sistemas de provas foram propostos: MANNA *et alia* (1983) propõem um sistema de prova temporal abstrato cuja parte dependente do programa possui uma interface de alto nível para a linguagem de programação que se deseja utilizar. ABADI *et alia* (1985) apresentam um sistema de provas para a LTLp baseado em resolução não-clausal. Nesse artigo, são descritos dois variantes desse sistema, um dos quais é para a LTLp com os operadores $\Box$, $\Diamond$ e $\circ$.

Alguns métodos de especificação formal que usam a LTL têm sido propostos. OWICKI *et alia* (1982) propõem um método de prova formal baseado na LTLp. O método demonstra como usar propriedades invariantes na prova de propriedades de vida. LAMPORT (1983) propõe um método para a especificação de módulos de programas em um

programa concorrente. MANNA *et alia* (1984) usam a LTLp na especificação e síntese (automática) da parte sincronizadora de processos em CSP. Mais recentemente, LAMPORT (1989) discorre sobre um método no qual a LTLPo é usada para especificar propriedades de vida.

A LTL vem sendo utilizada também na formalização da semântica de linguagens/programas concorrentes. PNUELI (1979) originalmente fez esse uso. Baseando-se na LTLPo, LAMPORT (1985) axiomatiza a semântica de uma linguagem de programação concorrente de modo composicional. VELOSO (1985) utiliza a LTLPo na descrição de propriedades de comandos (de uma linguagem para SDs) visando uma semântica axiomática.

Alguns resultados concernentes a LTL podem ser encontrados na bibliografia pesquisada. SISTLA *et alia* (1985) consideram a complexidade de satisfazibilidade e determinação de verdade em uma estrutura particular finita para diferentes LTLp (com diferentes operadores). KRÖGER (1987) demonstra a completeza de um sistema formal para a LTLp chamado Σ_{TA} com os operadores $\Box$, $\Diamond$, $\circ$ e **atnext**. GABBAY *et alii* (1980) demonstram a completeza de um sistema chamado DUX(II) semelhante ao Σ_{LTLp}.

Capítulo V

Lógica Temporal de Tempo Ramificado (LTTR)

A Lógica Temporal Linear de Tempo Ramificado (*Branching Time Temporal Logics*; RESCHER *et alia* (1971), LAMPORT (1980)) considera o tempo subjacente ao seu modelo possuindo uma natureza ramificada, em forma de árvore: a cada momento o tempo pode se dividir em caminhos alternados representando possíveis futuros. Neste sistema de lógica temporal, as modalidades refletem a natureza ramificada do tempo, permitindo a quantificação sobre possíveis futuros.

Várias lógicas foram propostas cujo modelo semântico considerava o tempo de forma ramificada. Algumas destas lógicas são o sistema K_b de RESCHER *et alia* (1971), o sistema *UB* (*Unified System of Branching Time*) de BEN-ARI *et alii* (1983), as lógicas CTL (*Computation Tree Logic*) e CTL^F (*Fair Computation Tree Logic*) (CLARKE *et alii*, 1986) e a lógica CTL* (EMERSON *et alia*, 1986). Uma característica da LTTR é a quantificação de fórmulas no que diz respeito aos ramos da estrutura do tempo em forma de árvore. Essa quantificação prefixa uma fórmula temporal linear, afirmando sua validade em todos os *caminhos*, isto é, os ramos da estrutura em árvore, ou em algum (alguns) deles.

Esse capítulo é dividido da seguinte maneira: na primeira seção fazemos uma introdução à versão ramificada da lógica temporal. Em seguida, apresentamos a LTTR: sintaxe, algumas definições, a estrutura temporal ramificada **R**, semântica, exemplos de obtenção de valor-verdade, alguns teoremas e demonstrações e propomos um sistema formal para a LTTR. Continuando, demonstramos sua consistência, consideramos sua completeza, exibimos algumas regras derivadas e teoremas e usamos a LTTR para especificar

formalmente e verificar um SD. Depois são mostradas algumas lógicas de tempo ramificado encontradas na literatura, é feito uma comparação entre a LTL e a LTTR e o capítulo se encerra com as seções de conclusões e de notas bibliográficas.

V.1 Introdução

A LTTR considera o tempo como tendo uma estrutura ramificada, em forma de árvore. A figura V.1 mostra um exemplo de como a história do mundo pode ser representada numa estrutura de tempo ramificado. Nesse exemplo, o mundo poderia ter seguido o curso da história (seqüência de estados) dado pela seqüência $s_0 s_1 s_3 s_6 \cdots$ ou $s_0 s_1 s_2 s_5 \cdots$. Na figura V.1 estão representadas todas as histórias.

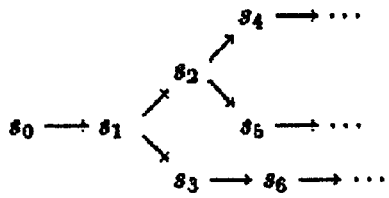

Figura V.1: Exemplo de representação ramificada do tempo

Na LTTR, o tempo é modelado de forma semelhante à LTL: é discreto e infinito, com um menor elemento (vide capítulo IV). A relação básica de acessibilidade R é a mesma da LTL, afirmando, na representação para este modelo da figura V.1, que se $R(s, s')$ é verificado, então, a partir do estado s, podemos chegar ao estado s', seguindo sempre o sentido das setas. As três características mencionadas podem ser formalmente expressas pelas fórmulas IV.1 (discreção), IV.2 (infinitude) e IV.3 (existência de um menor elemento) da LTL (cf. capítulo IV).

A relação de acessibilidade imediata $\rho : \mathbf{E} \times \mathbf{E}$, onde $\mathbf{E} = \{s_0, s_1, s_2, \ldots\}$ é o conjunto de todos os estados, é definida à semelhança do que foi feito na LTL. Em termos da representação gráfica da figura anterior, $\rho(s, s')$ é verificado se entre s e s' houver uma, e somente uma, seta. A definição de ρ em termos de R é feita da mesma forma do capítulo anterior (fórmula IV.4).

A estrutura ramificada do tempo pode ser formalizada através da relação ρ, se relaxarmos a definição feita na LTL (fórmula IV.5), não considerando mais ρ como uma função. Mais formalmente,

$$\forall s_i \exists s_j \{\rho(s_i, s_j)\} \tag{V.1}$$

é verdade, afirmando, para todo s_i, a existência de (pelo menos) um estado s_j ρ-acessível de s_i. Segundo isso, de um estado s, a história pode seguir para $s', s'', \ldots$ desde que $\rho(s, s'), \rho(s, s''), \ldots$ sejam verificados.

V.2 A Lógica Temporal de Tempo Ramificado (LTTR)

A LTTR é uma extensão do cálculo proposicional. Possuindo os mesmos operadores temporais da lógica temporal linear ($\Box$, $\Diamond$, $\circ$ e $\mathcal{U}$), a LTTR permite ainda que uma fórmula seja prefixada por um *quantificador de caminhos* (*caminho* é um "ramo" da estrutura do tempo em forma de árvore), de forma que, à interpretação usual das fórmulas, são acrescentadas considerações a respeito de sua validade em todos ou em algum (alguns) caminho(s).

Os quantificadores de caminho da LTTR são:

- $\forall_c F$ – a fórmula F é verificada em *todos* os caminhos;
- $\exists_c F$ – a fórmula F é verificada em *algum* (ou *alguns*) caminho(s).

Esses quantificadores refletem a natureza ramificada do tempo subjacente à LTTR, permitindo expressar propriedades verificadas em todas as (possíveis) histórias de um mundo ($\forall_c$) e propriedades verificadas em alguma (possível) história de um mundo ($\exists_c$) analisado.

V.2.1 Sintaxe

As fórmulas da LTTR são construídas a partir dos símbolos da LTLp (itens 1–5, subseção IV.2.1), acrescentando:

6. Quantificadores de caminho: $\forall_c$ (*em todos os caminhos*) e $\exists_c$ (*em algum caminho*);

Por questões de simplicidade, também denotaremos as constantes proposicionais por $p, q, r, \ldots$. Usando esse alfabeto, definimos indutivamente as fórmulas atômicas, as fórmulas de estado e as fórmulas de caminho. Deve ser ressaltado que a definição das fórmulas da LTTR poderia ter sido dada pelo item 3 somente (acrescentando o subitem 2.c). Porém, por questões semânticas — para fazer distinção entre o valor-verdade de uma fórmula em um determinado estado e o valor-verdade de uma fórmula em um caminho — optou-se por classificar as fórmulas em fórmulas de estado e de caminho:

1. Fórmulas Atômicas — são os construtos mais simples da LTTR, a partir dos quais as demais fórmulas (de estado e de caminho) são formadas. As fórmulas atômicas são definidas exatamente como na LTLp (subseção IV.2.1);

2. Fórmulas de Estado — representadas genericamente por $\epsilon, \epsilon_1, \epsilon_2, \ldots$, são as fórmulas cujo valor-verdade é determinado em relação a um estado s, consistindo nas fórmulas com estrutura proposicional e nas fórmulas prefixadas com os quantificadores de caminho:

   a) toda fórmula atômica v é uma fórmula de estado;

   b) se ϵ_1 e ϵ_2 são fórmulas de estado, então $\sim \epsilon_1$ e $\epsilon_1 \wedge \epsilon_2$ também são fórmulas de estado;

   c) se γ é uma fórmula de caminho (vide definição abaixo), então $\forall_c \gamma$ e $\exists_c \gamma$ são fórmulas de estado;

3. Fórmulas de Caminho — representadas genericamente por $\gamma, \gamma_1, \gamma_2, \ldots$, são as fórmulas cujo valor-verdade é determinado em relação a um caminho c, consistindo nas fórmulas de estado e nas fórmulas cujo principal operador é temporal:

   a) toda fórmula de estado ϵ é uma fórmula de caminho;

   b) se γ_1 e γ_2 são fórmulas de caminho, então $\sim \gamma_1$ e $\gamma_1 \wedge \gamma_2$ também são fórmulas de caminho;

   c) se ψ_1 e ψ_2 são fórmulas quaisquer (de estado ou de caminho), então $\Box\psi_1$, $\Diamond\psi_1$, $\circ\psi_1$ e $\psi_1 \mathcal{U} \psi_2$ também são fórmulas de caminho;

Podemos representar a sintaxe da LTTR com um diagrama BNF, como mostrado pela figura V.2.

$$\epsilon ::= \upsilon \mid \sim\epsilon \mid \epsilon_1 \wedge \epsilon_2 \mid \forall_c\gamma \mid \exists_c\gamma$$

$$\gamma ::= \epsilon \mid \sim\gamma \mid \gamma_1 \wedge \gamma_2 \mid \Box\psi \mid \Diamond\psi \mid \circ\psi \mid \psi_1 \mathcal{U} \psi_2$$

$$\psi ::= \epsilon \mid \gamma$$

Figura V.2: Diagrama BNF para a sintaxe da LTTR

Observações:

i) As abreviações da Lp (apêndice A) definem os demais operadores $\vee$, $\supset$ e $\equiv$, acrescentando a abreviação para os quantificadores de caminho

$$\sim\forall_c\sim\gamma \quad \text{é abreviado por} \quad \exists_c\gamma$$

ii) Estendendo a lista de prioridades da LTLp (capítulo IV), temos:

$\forall_c$ e $\exists_c$ têm maior prioridade que os demais conectivos e operadores;

Usaremos chaves, colchetes e parênteses para conferir mais clareza às fórmulas e nos casos onde houver ambigüidades.

V.2.2 Definições e Notações

Para prosseguirmos com a apresentação da LTTR, algumas definições e o estabelecimento de uma notação são necessários.

Definição V.2.2.1 Os *estados* da LTTR, definidos exatamente como na LTL (vide subseção IV.2.2), são mapeamentos das fórmulas atômicas $\upsilon \in \mathcal{V}$ nos valores-verdade **v** e **f**. Representando uma possível situação do mundo (vide capítulo III), os estados serão referenciados genericamente pelas letras $s, t, u, \ldots$. Será usada, alternativamente, a notação $s_i, i \in \mathbb{N}_0$, quando se quizer fazer referência a estados específicos.

Definição V.2.2.2 O conjunto $\mathbf{E} = \{s, t, \ldots\}$ representa a reunião de todos os estados. Aqui, R, a relação básica de acessibilidade (capítulo III) não é uma relação total.

Definição V.2.2.3 Um *caminho* $c = stu\ldots$ é uma seqüência não-vazia de estados $s \in \mathbf{E}$ tal que os estados estão relacionados dois a dois pela relação ρ de acessibilidade imediata

(capítulo IV). Os caminhos, também chamados de *seqüências*, serão representados genericamente pelas letras $c, d, e, \cdots$. Quando se quiser fazer referência a caminhos específicos, será usado a notação $c_i, i \in \mathbb{N}_0$.

Definição V.2.2.4 O conjunto $\mathbf{C} = \{c, d, \ldots\}$ representa a reunião de todos os caminhos ou seqüências.

Intuitivamente, um caminho c descreve *uma* possível história do mundo analisado (um ramo da árvore) e o conjunto **C** reúne *todas* as possíveis histórias de um mundo (toda a árvore). Nas demais definições, $c \in \mathbf{C}$ é um caminho tal que $c = stu \cdots$:

Definição V.2.2.5 A operação *primeiro* é uma função definida sobre seqüências

$$primeiro : \mathbf{C} \mapsto \mathbf{E}$$

que nos dá o primeiro estado da seqüência: $primeiro(c) = s$.

Definição V.2.2.6 A operação *suc* (*sucessor* — c^{suc}) é definida sobre seqüências

$$\mathbf{C}^{suc} \mapsto \mathbf{C}$$

e nos dá o sufixo da seqüência a partir de seu segundo estado: $c^{suc} = tu \cdots$.

Definição V.2.2.7 Os sufixos de uma seqüência são definidos recursivamente da seguinte forma:

$$c^0 = c$$
$$c^{n+1} = (c^n)^{suc}$$

V.2.3 Estrutura Temporal Ramificada (R)

Uma estrutura temporal ramificada **R** para a linguagem $\mathcal{L}_{LTTR}$ é a dupla (E,C), onde

- $\mathbf{E} = \{s, t, u, \ldots\}$ é um conjunto infinito de estados (definição V.2.2.2);
- $\mathbf{C} = \{c, d, e, \ldots\}$ é um conjunto infinito de caminhos (definição V.2.2.4).

Algumas restrições são feitas ao conjunto **C** de caminhos. Este deve ser

a) *fechado a sufixos* — se $c \in \mathbf{C}$, então $c^{suc} \in \mathbf{C}$ (LAMPORT, 1980);

b) *fechado a fusões* — se $q_1 s c_1$ e $q_2 s c_2 \in \mathbf{C}$ (onde q_i é um prefixo de caminho, s é um estado e c_i é um sufixo de caminho, $i = 1, 2$) então $q_1 s c_2, q_2 s c_1 \in \mathbf{C}$, isto é, se qs é o prefixo de um caminho e sc é um sufixo de outro caminho, então qsc, a fusão desses caminhos, também é um caminho (EMERSON *et alia*, 1986);

c) *fechado a limites* — se $q_1 c_1, q_1 q_2 c_2, q_1 q_2 q_3 c_3, \ldots \in \mathbf{C}$ então o caminho infinito (que é o "limite" dos prefixos $q_1, q_1 q_2, q_1 q_2 q_3, \ldots$) $q_1 q_2 q_3 \cdots \in \mathbf{C}$ (EMERSON *et alia*, 1986);

EMERSON (1983) demonstra que um conjunto $\mathbf{C}$ de caminhos é R-gerável (isto é, existe uma relação binária total $R : \mathbf{E} \times \mathbf{E} \mapsto \mathbf{C}$, tal que $\mathbf{C}$ consiste precisamente das seqüências infinitas $s_0 s_1 s_2 \cdots$ de estados de $\mathbf{E}$ para os quais $R(s_i, s_{i+1})$ é verificado) se, e somente se, ele for fechado a sufixos, a fusões e a limites.

Sejam p e q duas proposições quaisquer que assumem valores **v** ou **f** nos diversos estados da árvore e s um estado qualquer da árvore. As fórmulas temporais prefixadas pelos quantificadores de caminho possuem o seguinte significado (intuitivo):

$\forall_c \Box p$ é verificado no estado s se, e somente se, p é verdade em todos os estados dos caminhos que começam em s (incluindo s);

$\forall_c \Diamond p$ é verificado em s se, e somente se, em todos os caminhos que começam em s, houver pelo menos um estado no qual p seja verificado (podendo ser no próprio s);

$\forall_c \bigcirc p$ é verificado em s se, e somente se, p é verdade em todo sucessor imediato de s;

$\forall_c (p \mathcal{U} q)$ é verificado em s se, e somente se, em todos os caminhos começando em s, p é verificado até que q seja verificado;

$\exists_c \Box p$ é verificado no estado s se, e somente se, houver um caminho começando em s no qual p é verdade em todos os seus estados;

$\exists_c \Diamond p$ é verificado no estado s se, e somente se, houver um caminho começando em s no qual p é verdade em algum de seus estados;

$\exists_c \bigcirc p$ é verificado em s se, e somente se, p é verificado em algum sucessor imediato de s;

$\exists_c (p \mathcal{U} q)$ é verificado em s se, e somente se, em algum dos caminhos começando em s, p é verificado até que q seja verificado.

Algumas fórmulas de caminho e seus significados (intuitivos) são:

$\Box\forall_d p$ é verificado no caminho d se, e somente se, em todos os estados $s \in d$, for verificado $\forall_c p$, isto é, em todos os caminhos começando por estados de d, p é verificado em seu primeiro estado, ou, equivalentemente, p é verificado em todos os estados de d;

$\Diamond\exists_c p$ é verificado no caminho d se, e somente se, em algum estado $s \in d$, for verificado $\exists_c p$, isto é, em algum caminho começando por $s \in d$, p é verificado em seu primeiro estado (s).

V.2.4 Semântica

Para toda estrutura temporal ramificada $\mathbf{R} = (\mathbf{E}, \mathbf{C})$, todo estado $s \in \mathbf{E}$, todo caminho $c \in \mathbf{C}$, $i, j \in \mathbb{N}_0$, toda fórmula de estado ϵ e toda fórmula de caminho γ, definimos indutivamente os valores-verdade $\mathbf{R}_c(\gamma) \in \{\mathbf{v}, \mathbf{f}\}$ e $\mathbf{K}_s(\epsilon) \in \{\mathbf{v}, \mathbf{f}\}$ cujos significados pretendidos são, respectivamente, "o valor-verdade de γ no caminho c" e "o valor-verdade de ϵ no estado s":

1. $\mathbf{K}_s(v) = s(v)$ para $v \in \mathcal{V}$;
2. $\mathbf{K}_s(\sim\epsilon) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\epsilon) = \mathbf{f}$;
3. $\mathbf{K}_s(\epsilon_1 \wedge \epsilon_2) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\epsilon_1) = \mathbf{v}$ e $\mathbf{K}_s(\epsilon_2) = \mathbf{v}$;
4. $\mathbf{K}_s(\forall_c\gamma) = \mathbf{v}$ se, e somente se, $\mathbf{R}_c(\gamma) = \mathbf{v}$ para todo $c \in \mathbf{C}$ tal que $primeiro(c) = s$;
5. $\mathbf{K}_s(\exists_c\gamma) = \mathbf{v}$ se, e somente se, $\mathbf{R}_c(\gamma) = \mathbf{v}$ para algum $c \in \mathbf{C}$ tal que $primeiro(c) = s$;
6. $\mathbf{R}_c(\epsilon) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\epsilon) = \mathbf{v}$, onde $s = primeiro(c)$;
7. $\mathbf{R}_c(\sim\gamma) = \mathbf{v}$ se, e somente se, $\mathbf{R}_c(\gamma) = \mathbf{f}$;
8. $\mathbf{R}_c(\gamma_1 \wedge \gamma_2) = \mathbf{v}$ se, e somente se, $\mathbf{R}_c(\gamma_1) = \mathbf{v}$ e $\mathbf{R}_c(\gamma_2) = \mathbf{v}$;
9. $\mathbf{R}_c(\Box\epsilon) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\epsilon) = \mathbf{v}$, para todo $s \in c$;
10. $\mathbf{R}_c(\Box\gamma) = \mathbf{v}$ se, e somente se, $\mathbf{R}_d(\gamma) = \mathbf{v}$, onde $d = c^i$ para todo $i \geq 0$;
11. $\mathbf{R}_c(\Diamond\epsilon) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\epsilon) = \mathbf{v}$, para algum $s \in c$;

12. $\mathbf{R}_c(\Diamond\gamma) = \mathbf{v}$ se, e somente se, $\mathbf{R}_d(\gamma) = \mathbf{v}$, onde $d = c^i$ para algum $i \geq 0$;

13. $\mathbf{R}_c(\circ\epsilon) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\epsilon) = \mathbf{v}$, onde $s = primeiro(c^1)$;

14. $\mathbf{R}_c(\circ\gamma) = \mathbf{v}$ se, e somente se, $\mathbf{R}_d(\gamma) = \mathbf{v}$, onde $d = c^1$;

15. $\mathbf{R}_c(\epsilon_1 \mathcal{U} \epsilon_2) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\epsilon_2) = \mathbf{v}$, onde $s = primeiro(c^j)$ para algum $j \geq 0$, e $\mathbf{K}_t(\epsilon_1) = \mathbf{v}$, onde $t = primeiro(c^i)$ para todo $0 \leq i < j$;

16. $\mathbf{R}_c(\gamma_1 \mathcal{U} \gamma_2) = \mathbf{v}$ se, e somente se, $\mathbf{R}_d(\gamma_2) = \mathbf{v}$, onde $d = c^j$ para algum $j \geq 0$, e $\mathbf{R}_e(\gamma_1) = \mathbf{v}$, onde $e = c^i$ para todo $0 \leq i < j$;

Observações:

i) As definições $\mathbf{R}_c(op\,\epsilon)$ e $\mathbf{R}_c(op\,\gamma)$, $op \in \{\Box, \Diamond, \circ\}$ e $\mathbf{R}_c(\epsilon_1 \mathcal{U} \epsilon_2)$ e $\mathbf{R}_c(\gamma_1 \mathcal{U} \gamma_2)$ são consistentes, considerando ϵ como fórmula de caminho.

ii) Os quantificadores de caminho aplicados a fórmulas sem operadores temporais são "inócuos" (sem efeito), isto é, afirmam a mesma idéia da fórmula sem o quantificador. Isso é devido aos fatos de uma fórmula atemporal (sem operadores temporais) obter seu valor-verdade com respeito a um único estado e da avaliação de uma fórmula de estado em um caminho depender apenas de seu primeiro estado. Por exemplo, se uma fórmula atemporal p é verificada em um estado s, $\mathbf{K}_s(p) = \mathbf{v}$, então, em todos os caminhos (ou em algum caminho) que começarem por s, p é verificado, $\mathbf{R}_d(p) = \mathbf{v}$ para todo (ou algum) d tal que $primeiro(d) = s$, isto é, $\mathbf{K}_s(\forall_c p) = \mathbf{K}_s(\exists_c p) = \mathbf{v}$.

V.2.5 Um Exemplo da LTTR

Seja a árvore de estados da figura V.3. Neste exemplo, queremos fazer referência aos estados e caminhos de forma específica, por isso os rotularemos com s_i e c_j, $i, j \in \mathbb{N}_0$. A estrutura temporal ramificada $\mathbf{R} = (\mathbf{E}, \mathbf{C})$ obtida é tal que $\mathbf{E} = \{s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, \ldots\}$ e $\mathbf{C} = \{c_0, c_1, c_2, \ldots\}$. Os caminhos $c_j, j = 0, \ldots, 3$ são tais que $c_0 = s_0 s_1 s_3 s_5 \cdots$, $c_1 = s_0 s_1 s_3 s_6 \cdots$, $c_2 = s_0 s_1 s_3 s_7 \cdots$ e $c_3 = s_0 s_2 s_4 \cdots$, devendo-se notar que $primeiro(c_j) = s_0, j = 0, \ldots, 3$, isto é, os caminhos exibidos são aqueles cujo primeiro estado é s_0.

Vamos supor a atribuição dos valores-verdade v e f a três proposições p, q e r em cada um dos estados da árvore da figura V.3 conforme a tabela V.1.

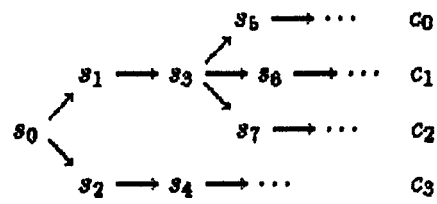

Figura V.3: Exemplo de árvore de estados

	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	⋯
p	f	f	f	f	v	f	f	v	⋯ * ⋯
q	v	v	v	v	f	f	f	f	⟨f para sempre⟩
r	f	f	f	f	v	v	v	v	⟨v para sempre⟩

Tabela V.1: Valores de p, q e r em cada estado

De posse dos caminhos e dos valores-verdade, obtemos:

- Seja A a fórmula $\Diamond r$. Então $\mathbf{R}_{c_j}(A) = \mathbf{v}, j = 0, \ldots, 3$ pois $\mathbf{K}_{s_i}(r) = \mathbf{v}, i = 4, 5, \ldots$ para algum $s_i \in c_j$. Intuitivamente, vemos que os caminhos c_j possuem pelo menos um estado no qual r é verificado;

- Seja B a fórmula $\forall_c q \mathcal{U} r$. Então $\mathbf{K}_{s_0}(B) = \mathbf{v}$, pois $\mathbf{R}_{c_j}(q \mathcal{U} r) = \mathbf{v}$ para todo $c_j \in \mathbf{C}$ tal que $primeiro(c_j) = s_0$. De fato, $\mathbf{R}_{c_j}(q \mathcal{U} r) = \mathbf{v}$, pois para $j = 0, \ldots, 2$, temos que $\mathbf{K}_{s_i}(r) = \mathbf{v}$, $s_i = primeiro(c_j^3)$ e $\mathbf{K}_{s_k}(q) = \mathbf{v}$, $s_k = primeiro(c_j^m)$, $0 \leq m < 3$; e para $i = 3$, temos que $\mathbf{K}_{s_i}(r) = \mathbf{v}$, $s_i = primeiro(c_j^2)$ e $\mathbf{K}_{s_k}(q) = \mathbf{v}$, $s_k = primeiro(c_j^m)$, $0 \leq m < 2$;

- Seja C a fórmula de caminho $\Diamond p$. Então $\mathbf{R}_{c_j}(C) = \mathbf{f}, j = 1, 2$, pois $\mathbf{K}_{s_i}(p) = \mathbf{f}$ para todo $s_i \in c_j$, e $\mathbf{R}_{c_j}(C) = \mathbf{v}, j = 2, 3$, pois $\mathbf{K}_{s_k}(p) = \mathbf{v}$ para algum $s_k \in c_j$ (s_7 e s_4, respectivamente). Daí temos que $\mathbf{K}_{s_0}(\forall_c \Diamond p) = \mathbf{f}$ e $\mathbf{K}_{s_0}(\exists_c \Diamond p) = \mathbf{v}$.

V.2.6 Definições e Teoremas

Sejam ψ e ϕ fórmulas quaisquer (de estado ou de caminho).

Definição V.2.6.1 Uma fórmula ψ é *válida na estrutura temporal ramificada* $\mathbf{R}$, denotando-se por $\models_{\mathbf{R}}\psi$, se $\mathbf{R}_c(\psi) = \mathbf{v}$ para todo $c \in \mathbf{C}$.

As definições de *fórmulas válidas* e *fórmulas que seguem de um conjunto* $\mathcal{F}$ *de fórmulas* são feitas à semelhança das definições A.1.3.2 e A.1.3.3, respectivamente.

O teorema seguinte afirma que, como na LTLp, se uma fórmula ψ segue de um conjunto $\mathcal{F}$ de fórmulas válidas (ϕ), então ψ também é válida:

Teorema V.2.6.1 *Se* $\mathcal{F} \models \psi$ *e* $\models \phi$ *para todo* $\phi \in \mathcal{F}$, *então* $\models \psi$.

Prova: Seja $\mathbf{R} = (\mathbf{E}, \mathbf{C})$ uma estrutura temporal ramificada. Então $\models_{\mathbf{R}}\phi$ para todo $\phi \in \mathcal{F}$ e, portanto $\models_{\mathbf{R}}\psi$. Desde que isso é verificado para todo $\mathbf{R}$, temos que $\models \psi$. ■

O teorema a seguir é semelhante ao teorema IV.2.5.4:

Teorema V.2.6.2 *Se* $\mathcal{F} \models \psi$ *então* $\mathcal{F} \models \circ\psi$ *e* $\mathcal{F} \models \Box\psi$.

Prova: Seja $\mathbf{R} = (\mathbf{E}, \mathbf{C})$ uma estrutura temporal ramificada tal que $\models_{\mathbf{R}}\phi$ para todo $\phi \in \mathcal{F}$. Então $\mathbf{R}_c(\psi) = \mathbf{v}$ para todo $c \in \mathbf{C}$. Podemos dizer também que $\mathbf{R}_d(\psi) = \mathbf{v}$, onde $d = c^i$, para todo $i \geq 0$, para todo $c \in \mathbf{C}$, isto é, $\mathbf{R}_c(\Box\psi) = \mathbf{v}$ e $\mathbf{R}_c(\circ\psi) = \mathbf{v}(i = 1)$ para todo $c \in \mathbf{C}$. Isso significa que $\mathcal{F} \models \Box\psi$ e $\mathcal{F} \models \circ\psi$. ■

O próximo teorema ilustra o fechamento a sufixos e a fusões do conjunto $\mathbf{C}$ de caminhos.

Teorema V.2.6.3 *Se* $\mathcal{F} \models \psi$ *então* $\mathcal{F} \models \forall_c\psi$.

Prova: Seja $\mathbf{R} = (\mathbf{E}, \mathbf{C})$ uma estrutura temporal ramificada tal que $\models_{\mathbf{R}}\phi$ para todo $\phi \in \mathcal{F}$ e um caminho $d \in \mathbf{C}$. $\mathbf{R}_d(\forall_c\psi) = \mathbf{v}$ se, e somente se, $\mathbf{K}_s(\forall_c\psi) = \mathbf{v}$ onde $s = \mathit{primeiro}(d)$, isto é, para todo caminho $e \in \mathbf{C}$ tal que $\mathit{primeiro}(e) = s = \mathit{primeiro}(d)$, $\mathbf{R}_e(\psi) = \mathbf{v}$. Como $\mathcal{F} \models \psi$, então $\models_{\mathbf{R}}\psi$, isto é, $\mathbf{R}_f(\psi) = \mathbf{v}$ para todo $f \in \mathbf{C}$, em particular para $e \in \mathbf{C}$ tal que $\mathit{primeiro}(e) = \mathit{primeiro}(d)$. ■

O teorema seguinte corresponde ao teorema IV.2.5.2 da LTLp:

Teorema V.2.6.4 $\psi_1, \ldots, \psi_n \models \phi$ *se, e somente se,* $\models \forall_c\Box\psi_1 \wedge \ldots \wedge \forall_c\Box\psi_n \supset \phi$.

Prova:

($\Rightarrow$) Sejam $\psi_1, \ldots, \psi_n \models \phi$ e uma estrutura temporal ramificada $\mathbf{R} = (\mathbf{E}, \mathbf{C})$. Vamos assumir que $\mathbf{R}_d(\forall_c\Box\psi_1 \wedge \cdots \wedge \forall_c\Box\psi_n \supset \phi) = \mathbf{f}$ para algum caminho $d \in \mathbf{C}$. Isso significa que $\mathbf{R}_d(\forall_c\Box\psi_1) = \cdots = \mathbf{R}_d(\forall_c\Box\psi_n) = \mathbf{v}$ e $\mathbf{R}_d(\phi) = \mathbf{f}$, ou seja $\mathbf{K}_s(\forall_c\Box\psi_1) = \cdots = \mathbf{K}_s(\forall_c\Box\psi_n) = \mathbf{v}$ e $\mathbf{K}_s(\phi) = \mathbf{f}$ onde $s = primeiro(d)$. Isso implica que $\mathbf{R}_e(\Box\psi_1) = \cdots = \mathbf{R}_e(\Box\psi_n) = \mathbf{v}$ para todo caminho $e \in \mathbf{C}$ tal que $primeiro(e) = s = primeiro(d)$ e $\mathbf{K}_s(\phi) = \mathbf{f}$, ou ainda, $\mathbf{K}_t(\psi_1) = \cdots = \mathbf{K}_t(\psi_n) = \mathbf{v}$ para todo $t \in e$ e todo caminho $e \in \mathbf{C}$ tal que $primeiro(e) = s$. Seja uma nova estrutura temporal ramificada $\mathbf{R}' = (\mathbf{E}', \mathbf{C}')$, tal que o conjunto $\mathbf{C}'$ seja composto somente pelos caminhos $c \in \mathbf{C}$ tal que $primeiro(c) = s$, isto é, $\mathbf{C}'$ é o conjunto de todos os caminhos que começam por s, devendo ser notado que $d \in \mathbf{C}'$. Então $\mathbf{R}'_{c'}(\psi_1) = \cdots = \mathbf{R}'_{c'}(\psi_n) = \mathbf{v}$ para todo $c' \in \mathbf{C}'$ e $\mathbf{R}'_d(\phi) = \mathbf{f}$, daí temos que $\models_{\mathbf{R}'}\psi_1, \ldots, \models_{\mathbf{R}'}\psi_n$ mas não $\models_{\mathbf{R}'}\phi$. Isso é uma contradição à primeira hipótese, logo, $\mathbf{R}_c(\forall_c\Box\psi_1 \wedge \cdots \wedge \forall_c\Box\psi_n \supset \phi) = \mathbf{v}$, para todo $c \in \mathbf{C}$ e como $\mathbf{R}$ e c são quaisquer, temos que $\models \forall_c\Box\psi_1 \wedge \cdots \wedge \forall_c\Box\psi_n \supset \phi$.

($\Leftarrow$) Sejam $\models \forall_c\Box\psi_1 \wedge \ldots \wedge \forall_c\Box\psi_n \supset \phi$ e uma estrutura temporal ramificada $\mathbf{R} = (\mathbf{E}, \mathbf{C})$ tal que $\models_{\mathbf{R}}\psi_1, \ldots, \models_{\mathbf{R}}\psi_n$, isto é, $\mathbf{R}_c(\psi_1) = \cdots = \mathbf{R}_c(\psi_n) = \mathbf{v}$ para todo $c \in \mathbf{C}$. Como o conjunto $\mathbf{C}$ de caminhos é fechado a sufixos, podemos afirmar que $\mathbf{R}_d(\psi_1) = \cdots = \mathbf{R}_d(\psi_n) = \mathbf{v}$, onde $d = c^i$ para todo $i \geq 0$ e todo $c \in \mathbf{C}$, ou seja, $\mathbf{R}_c(\Box\psi_1) = \cdots = \mathbf{R}_c(\Box\psi_n) = \mathbf{v}$ para todo $c \in \mathbf{C}$. Podemos também afirmar que, para todo $s \in \mathbf{E}$ e todo $e \in \mathbf{C}$ tal que $s = primeiro(e)$, $\mathbf{R}_e(\Box\psi_1) = \cdots = \mathbf{R}_e(\Box\psi_n) = \mathbf{v}$. Isto é, $\mathbf{K}_s(\forall_c\Box\psi_1) = \cdots = \mathbf{K}_s(\forall_c\Box\psi_n) = \mathbf{v}$ para todo $s \in \mathbf{E}$, logo, $\mathbf{R}_c(\forall_c\Box\psi_1 \wedge \cdots \wedge \forall_c\Box\psi_n) = \mathbf{v}$ para todo $c \in \mathbf{C}$. Como $\mathbf{R}_c(\forall_c\Box\psi_1 \wedge \ldots \wedge \forall_c\Box\psi_n \supset \phi) = \mathbf{v}$ para todo $c \in \mathbf{C}$ (hipótese), temos que $\mathbf{R}_c(\phi) = \mathbf{v}$ para todo $c \in \mathbf{C}$. Isso significa que $\models_{\mathbf{R}}\phi$ e temos o resultado desejado que $\psi_1, \ldots, \psi_n \models \phi$.

■

O teorema V.2.6.5 afirma que, como na LTLp, (teorema IV.2.5.3), a regra de inferência *Modus Ponens* (Lp — apêndice A) preserva a validade na LTTR:

Teorema V.2.6.5 *Se* $\mathcal{F} \models \psi_1$ *e* $\mathcal{F} \models \psi_1 \supset \psi_2$ *então* $\mathcal{F} \models \psi_2$.

Prova: Seja $\mathbf{R} = (\mathbf{E}, \mathbf{C})$ uma estrutura temporal ramificada tal que $\models_{\mathbf{R}} \phi$ para todo $\phi \in \mathcal{F}$. Então $\mathbf{R}_c(\psi_1) = \mathbf{R}_c(\psi_1 \supset \psi_2) = \mathbf{v}$, para todo $c \in \mathbf{C}$, e portanto $\mathbf{R}_c(\psi_2) = \mathbf{v}$ para todo $c \in \mathbf{C}$. Isso significa que $\mathcal{F} \models \psi_2$. ■

O teorema a seguir torna explícito o relacionamento entre a Lp (apêndice A) e a LTTR. A definição de *tautologia da LTTR* é feita de modo análogo à de tautologia da LTLp (definição IV.2.5.3):

Teorema 2.6.6 *Toda tautologia da LTTR é válida.*

Prova: Segue à semelhança do teorema IV.2.5.5.

O próximo teorema "transporta" para a LTTR as fórmulas válidas da LTLp:

Teorema 2.6.7 *Se A é uma fórmula válida da LTLp, então A é uma fórmula válida da LTTR.*

Prova: Seja A uma fórmula válida da LTLp. Dada uma estrutura temporal ramificada $\mathbf{R} = (\mathbf{E}, \mathbf{C})$, cada caminho $c \in \mathbf{C}$ pode ser visto como uma estrutura temporal (linear) $\mathbf{K}_c = (\mathbf{E}_c, \rho_c)$ onde $\mathbf{E}_c$ são os estados do caminho c e $\rho_c = \rho|_{\mathbf{E}_c}$ é a relação de acessibilidade imediata restrita aos estados de $\mathbf{E}_c$. Como A é válida na LTLp, $\models_{\mathbf{K}_c} A$, logo, $\models_{\mathbf{R}} A$ e, como $\mathbf{R}$ é qualquer, $\models A$, ou seja, A é uma fórmula válida da LTTR. ■

V.2.7 O Sistema Formal Σ_{LTTR}

Chamaremos de Σ_{LTTR} o sistema abaixo que formaliza a LTTR.

Axiomas

Os esquemas de axiomas que constituem o Σ_{LTTR} são:

A1. $\vdash \forall_c \sim \psi \equiv \sim \exists_c \psi$

A2. $\vdash \forall_c \psi \supset \psi$

A3. $\vdash \psi \supset \exists_c \psi$

A4. $\vdash \forall_c(\psi_1 \wedge \psi_2) \equiv \forall_c \psi_1 \wedge \forall_c \psi_2$

A5. $\vdash \forall_c(\psi_1 \supset \psi_2) \supset (\forall_c\psi_1 \supset \forall_c\psi_2)$

A6. $\vdash \forall_c\Box\psi \supset \forall_c\Box\forall_c\Box\psi$

O axioma **A1** define $\exists_c$ como o dual de $\forall_c$. **A2** é a versão LTTR para a instanciação. O axioma **A3** afirma que se ψ é verificado agora (nessa execução) então há uma execução na qual ψ é verificado. **A4** e **A5** ilustram a distributividade (forte e fraca, respectivamente) do quantificador $\forall_c$ sobre $\wedge$ e $\supset$. O axioma **A6** atesta o fechamento a sufixos e a fusões do conjunto de caminhos **C**.

Regras de Inferência

As regras de inferência do Σ_{LTTR} são:

R1. Se A é uma tautologia da LTTR, então $\vdash A$ (*Tautologia Proposicional — TP*);

R2. Se A é uma fórmula válida da LTLp, então $\vdash A$ (*Validade Temporal — VT*);

R3. Se $\vdash \psi_1 \supset \psi_2$ e $\vdash \psi_1$, então $\vdash \psi_2$ (*Modus Ponens — MP*)

R4. Se $\vdash \psi$, então $\vdash \Box\psi$ (*Inserção do* $\Box$ *— I*$\Box$);

R5. Se $\vdash \psi$, então $\vdash \forall_c\psi$ (*Inserção do* $\forall_c$ *— I*$\forall_c$);

V.2.8 Consistência do Σ_{LTTR}

Seja ψ uma fórmula e $\mathcal{F}$ um conjunto de fórmulas. Afirmamos que:

Se $\mathcal{F} \vdash \psi$ *então* $\mathcal{F} \models \psi$

ou seja, admitindo que $\mathcal{F}$ é Σ_{LTTR},

O sistema Σ_{LTTR} *é consistente.*

Prova: A prova segue por indução no tamanho da derivação de ψ a partir de $\mathcal{F}$ ($\mathcal{F} \vdash \psi$), à semelhança da prova de consistência do Σ_{LTL_P} (subseção IV.2.7):

1. ψ é um esquema de axioma do Σ_{LTTR}. Provamos a validade de todos os esquemas de axiomas **A1–A6** no apêndice C: em todos os casos, $\mathcal{F} \models \psi$.

2. $\psi \in \mathcal{F}$. Nesse caso, temos trivialmente que $\mathcal{F} \models \psi$.

3. (Consistência de *TP*) Seja A a conclusão de *TP* com a premissa satisfeita de que A é uma tautologia da LTTR. Então, pelo teorema V.2.6.6, $\models A$, isto é, $\mathcal{F} \models A$ para $\mathcal{F}$ qualquer.

4. (Consistência de *VT*) Seja A a conclusão de *VT* com a premissa satisfeita de que A é uma fórmula válida da LTLp. Então, pelo teorema V.2.6.7, $\models A$, isto é, $\mathcal{F} \models A$ para $\mathcal{F}$ qualquer.

5. (Consistência de *MP*) Seja ψ_2 a conclusão de *MP* com premissas ψ_1 e $\psi_1 \supset \psi_2$. Então, temos que $\mathcal{F} \vdash \psi_1$ e $\mathcal{F} \vdash \psi_1 \supset \psi_2$, obtendo $\mathcal{F} \models \psi_1$ e $\mathcal{F} \models \psi_1 \supset \psi_2$, pela hipótese de indução, e portanto, $\mathcal{F} \models \psi_2$, pelo teorema V.2.6.5.

6. (Consistência de $I\Box$) Seja ψ_2 a conclusão de $I\Box$ com a premissa ψ_1. Então, temos que $\mathcal{F} \vdash \psi_1$ e ψ_2 é $\Box\psi_1$, e pela hipótese de indução, $\mathcal{F} \models \psi_1$. Pelo teorema V.2.6.2 obtemos $\mathcal{F} \models \Box\psi_1$ (e também $\mathcal{F} \models \circ\psi_1$, que não nos interessa aqui) ou seja, $\mathcal{F} \models \psi_2$.

7. (Consistência de $I\forall_c$) Seja ψ_2 a conclusão de $I\forall_c$ com a premissa ψ_1. Então, temos que $\mathcal{F} \vdash \psi_1$ e ψ_2 é $\forall_c\psi_1$, e pela hipótese de indução, $\mathcal{F} \models \psi_1$. Pelo teorema V.2.6.3 obtemos $\mathcal{F} \models \forall_c\psi_1$, ou seja, $\mathcal{F} \models \psi_2$.

■

V.2.9 Completeza do Σ_{LTTR}

Na axiomatização da LTTR, foi feita uma tentativa para capturar idéias intuitivas e que seriam úteis para o objetivo pretendido, especificar formalmente e verificar SDs. Não houve uma preocupação formal com a completeza, mas EMERSON *et alia* (1984) afirmam que o problema de se dar uma axiomatização para a LTTR interpretada sobre estruturas temporais ramificadas está aberto. Entretanto, pode ser encontrado, no trabalho de BEN-ARI *et alii* (1983), uma axiomatização completa para uma lógica temporal de tempo ramificado, o sistema *UB*, de menor expressividade do que a LTTR pois não permite o aninhamento de operadores temporais (vide subseção V.4.2).

V.2.10 Regras Derivadas

Exibimos algumas regras derivadas que serão úteis nas demonstrações subseqüentes. Essas regras são demonstradas no apêndice C. Os símbolos $\psi, \psi_1, \ldots, \psi_n$ e ϕ representam fórmulas quaisquer (de estado ou de caminho).

1. *RP — Raciocínio Proposicional*

$$\frac{\vdash (\psi_1 \wedge \ldots \wedge \psi_n) \supset \phi \quad \vdash \psi_1, \ldots, \vdash \psi_n}{\vdash \phi}$$

2. *Regras* $\forall_c\forall_c$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \forall_c\psi_1 \supset \forall_c\psi_2}$ b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \forall_c\psi_1 \equiv \forall_c\psi_2}$

3. *Regras* $\forall_c\Box$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \forall_c\Box\psi_1 \supset \forall_c\Box\psi_2}$ b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \forall_c\Box\psi_1 \equiv \forall_c\Box\psi_2}$

4. *Regras* $\exists_c\exists_c$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \exists_c\psi_1 \supset \exists_c\psi_2}$ b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \exists_c\psi_1 \equiv \exists_c\psi_2}$

5. *Regras* $\exists_c\Diamond$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \exists_c\Diamond\psi_1 \supset \exists_c\Diamond\psi_2}$ b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \exists_c\Diamond\psi_1 \equiv \exists_c\Diamond\psi_2}$

V.2.11 Teoremas

Alguns teoremas que podem ser derivados no sistema Σ_{LTTR} são mostrados aqui e demonstrados no apêndice C.

T1. $\vdash \exists_c(\psi_1 \wedge \psi_2) \supset \exists_c\psi_1 \wedge \exists_c\psi_2$

T2. $\vdash \forall_c\Box \sim\psi \equiv \sim\exists_c\Diamond\psi$

T3. $\vdash \forall_c\Box\psi \supset \exists_c\Diamond\psi$

T4. $\vdash \forall_c\Box\psi \supset \forall_c\Diamond\psi$

T5. $\vdash \forall_c\circ\psi \supset \exists_c\circ\psi$

T6. $\vdash \forall_c\Box(\psi_1 \wedge \psi_2) \equiv \forall_c\Box\psi_1 \wedge \forall_c\Box\psi_2$

T7. $\vdash \exists_c\Box(\psi_1 \wedge \psi_2) \supset (\exists_c\Box\psi_1 \wedge \exists_c\Box\psi_2)$

T8. $\vdash \forall_c\Box(\psi_1 \supset \psi_2) \supset (\forall_c\Box\psi_1 \supset \forall_c\Box\psi_2)$

T9. $\vdash \forall_c\circ(\psi_1 \supset \psi_2) \supset (\forall_c\circ\psi_1 \supset \forall_c\circ\psi_2)$

T10. $\vdash \forall_c\Box\forall_c\Box\psi \equiv \forall_c\Box\psi$

T11. $\vdash \exists_c\Diamond\exists_c\Diamond\psi \equiv \exists_c\Diamond\psi$

T12. $\vdash (\forall_c\Box\psi_1 \wedge \exists_c\Diamond\psi_2) \supset \exists_c\Diamond(\psi_1 \wedge \psi_2)$

V.3 A Especificação Formal de SDs com a LTTR

A LTTR, assim como a LTL, adota a visão das intercalações (vide capítulo II) só que ao invés de uma seqüência linear de estados globais, seu modelo de SD consiste em uma *árvore* de estados globais denominada *árvore de computação* (CLARKE *et alii*, 1986) ou *árvore de execução* (BEN ARI *et alii*, 1983). Cada estado, representando a situação global do sistema, pode ter um ou mais sucessores, correspondendo aos possíveis futuros estados. Cada caminho da árvore corresponde a uma execução particular do SD — uma seqüência de estados globais descrevendo uma possível história do SD.

V.3.1 Um Exemplo

Seja o SD composto pelos processos P_1, P_2 e S, conforme a figura V.4. Cada processo $P_i, i = 1, 2$, é composto por cinco módulos (grupos de um ou mais comandos que realizam uma tarefa afim) representados por retângulos:

- RS_i: módulo de recebimento de solicitação, corresponde ao local do processo no qual o controle da execução permanece até que uma solicitação externa (por exemplo, o pedido de algum usuário ou o sinal de algum aparelho), sob a forma de uma mensagem s_i, seja recebida;
- E_i: módulo de envio no qual é enviado um pedido p_i para o processo sincronizador S;

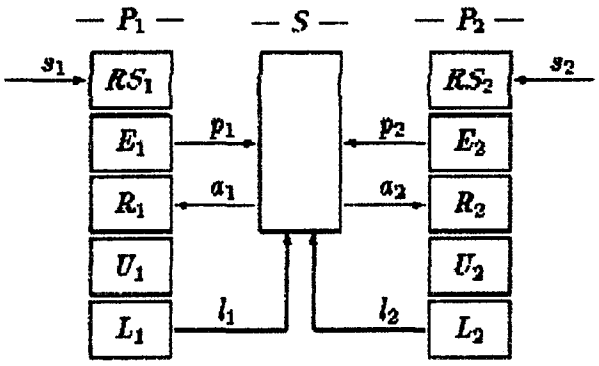

Figura V.4: SD composto por 3 processos: P_1, P_2 e S

- R_i: módulo de recbimento no qual é recebido uma autorização a_i permitindo o processo chegar ao módulo seguinte;
- U_i: módulo de uso, correspondendo a utilização de algum recurso, de forma exclusiva, pelo processo P_i (por exemplo, uso de equipamento compartilhado, acesso a informações, etc.);
- L_i: módulo de liberação, onde o processo P_i informa ao processo S que deixou de utilizar o recurso (deixou o módulo U_i), enviando a mensagem l_i.

Os processos $P_i, i = 1, 2$, funcionam de modo seqüencial, na ordem indicada pela figura V.4. P_i permanece em RS_i até que seja feito uma solicitação externa, determinado pela chegada da mensagem s_i. Após a chegada de s_i, o controle da execução move-se para o módulo E_i, no qual P_i enviará a mensagem p_i (pedido de permissão) para S e irá esperar em R_i até que S lhe envie uma mensagem a_i autorizando-o a entrar em U_i. Após um intervalo de tempo finito que o controle esteja em U_i, ele se move para L_i onde a mensagem l_i é enviada para S, informando que o processo P_i deixou U_i. Depois que o controle executa L_i ele volta para RS_i e recomeça a rotina (processos cíclicos).

Nesse exemplo assumiremos que:

1. Os processos permanecem constantemente ativos, sendo o seu comportamento o alvo de nosso estudo.

2. Os canais de comunicação são 100% confiáveis e

3. Toda mensagem enviada chega eventualmente, isto é, após um intervalo finito de tempo, ao seu destino.

V.3.2 Uma Especificação Formal

Para simplificar a notação nas fórmulas, usaremos os nomes dos módulos RS_i, E_i, R_i, U_i e L_i denotando o fato que o controle da execução está naquele módulo, e também usaremos os nomes das mensagens s_i, p_i, a_i e l_i denotando o fato que a mensagem foi *recebida* onde era esperada.

A seguir especificamos formalmente, a partir da descrição informal acima, algumas propriedades que os processos apresentam:

Processos P_i

- *Propriedade 1:* Durante todas as execuções, em cada um dos processos, o controle da execução só pode estar em um módulo a cada momento. Por exemplo, quando o controle da execução estiver em RS_i, ele não estará em nenhum dos outros módulos. Mais formalmente:

$$\pi_1 a : \forall_c \Box[RS_i \supset \sim(E_i \vee R_i \vee U_i \vee L_i)]$$

e o mesmo pode ser dito dos demais módulos:

$$\pi_1 b : \forall_c \Box[E_i \supset \sim(RS_i \vee R_i \vee U_i \vee L_i)]$$

$$\pi_1 c : \forall_c \Box[R_i \supset \sim(RS_i \vee E_i \vee U_i \vee L_i)]$$

$$\pi_1 d : \forall_c \Box[U_i \supset \sim(RS_i \vee E_i \vee R_i \vee L_i)]$$

$$\pi_1 e : \forall_c \Box[L_i \supset \sim(RS_i \vee E_i \vee R_i \vee U_i)]$$

- *Propriedade 2:* Em qualquer execução, sempre é verdade que, se uma mensagem foi recebida, então o controle da execução está no módulo onde ela (a mensagem) era esperada. Por exemplo, em qualquer execução, sempre é o caso que, se s_i foi recebida, então o controle da execução está em RS_i, o que pode ser formalizado por:

$$\pi_2 a : \forall_c \Box(s_i \supset RS_i)$$

de $\pi_2 a$, obtemos (por *RP*)

$$\pi_2 a' : \forall_c \Box[s_i \supset (RS_i \wedge s_i)]$$

e, juntamente com a tautologia $\forall_c \Box[(RS_i \wedge s_i) \supset s_i]$, obtemos

$$\pi_2 a'' : \forall_c \Box[s_i \equiv (RS_i \wedge s_i)]$$

o mesmo pode ser formalizado sobre a_i:

$$\pi_2 b'' : \forall_c \Box[a_i \equiv (R_i \wedge a_i)]$$

As fórmulas $\pi_2 a$ e $\pi_2 b$ tornam explícitos fatos subjacentes à descrição informal do exemplo.

- *Propriedade 3:* Durante qualquer execução, se o processo P_i estiver em RS_i, então ele ficará em RS_i até que uma mensagem s_i chegue. Essa propriedade pode ser representada pela fórmula abaixo:

$$\pi_3 : \forall_c \Box(RS_i \supset RS_i \mathcal{U} s_i)$$

- *Propriedade 4:* Em qualquer execução, sempre é verdade que, se a mensagem s_i for recebida, então, no próximo estado, o controle da execução estará em E_i, o que pode ser representado por

$$\pi_4 : \forall_c \Box(s_i \supset \circ E_i)$$

- *Propriedade 5:* Em qualquer execução, sempre é verdade que, se o controle da execução estiver em E_i, então, no próximo estado, o controle da execução estará em R_i, podendo ser formalizado por

$$\pi_5 : \forall_c \Box(E_i \supset \circ R_i)$$

Essa propriedade ilustra a "instantaneidade" do módulo E_i, isto é, o módulo E_i pode ser visto como uma ação atômica.

- *Propriedade 6:* Em qualquer execução, sempre é verdade que, se o controle da execução estiver em E_i, então, eventualmente, a mensagem p_i será recebida por S:

$$\pi_6 : \forall_c \Box(E_i \supset \Diamond p_i)$$

- *Propriedade 7:* Em qualquer execução, sempre é verdade que, se a mensagem a_i for recebida, então, no próximo estado, o controle da execução estará em U_i, podendo ser formalizado por:

$$\pi_7 : \forall_c \Box(a_i \supset \circ U_i)$$

- *Propriedade 8:* Seja a constante $AUTO_i$ simbolizando o fato que o processo P_i foi autorizado a entrar em seu módulo de uso U_i. Pela definição de a_i, temos que, em qualquer execução, sempre é verdade que, se a_i é recebido, então $AUTO_i$ é verificado:

$$\pi_8 : \forall_c \Box(a_i \supset AUTO_i)$$

- *Propriedade 9:* Em qualquer execução, sempre é verdade que, se o controle da execução de P_i acha-se em U_i, é porque o processo P_i foi autorizado a fazê-lo, o que pode ser formalizado por

$$\pi_9 : \forall_c \Box(U_i \supset AUTO_i)$$

De π_9 é possível deduzir π'_9:

$$\pi'_9 : \forall_c \Box(\sim AUTO_i \supset \sim U_i)$$

significando que, se o processo não foi autorizado a entrar em U_i (não for verificado $AUTO_i$), então ele (o processo) não está em U_i.

- *Propriedade 10:* Em qualquer execução, sempre é verdade que se o controle da execução estiver em U_i, então eventualmente o controle da execução estará em L_i:

$$\pi_{10} : \forall_c \Box(U_i \supset \Diamond L_i)$$

- *Propriedade 11:* Em qualquer execução, sempre é o caso que, enquanto o controle da execução estiver em U_i a mensagem l_i não será recebida:

$$\pi_{11} : \forall_c \Box(U_i \supset \sim l_i)$$

- *Propriedade 12:* Em qualquer execução, sempre é verdade que se o controle da execução estiver em L_i, então eventualmente l_i será recebido por S e no próximo estado o controle da execução estará em RS_i:

$$\pi_{12} : \forall_c \Box[L_i \supset (\Diamond l_i \wedge \circ RS_i)]$$

Processo S

- *Propriedade 1:* Em qualquer execução, sempre é verdade que, se o processo sincronizador S receber l_i, então eventualmente S autoriza P_i ou P_j a entrar em seu módulo de uso:

$$\sigma_1 : \forall_c \Box [l_i \supset \Diamond (a_i \vee a_j)]$$

- *Propriedade 2:* σ_2 (abaixo) é uma validade temporal que foi prefixada por $\forall_c \Box$ (através das regras $I\forall_c$ e $I\Box$). Com isso procurou-se abreviar as demonstrações pois é possível deduzir essa fórmula: em qualquer execução, sempre é verdade que se o processo sincronizador S receber p_i, então no próximo estado P_i recebe ou não a mensagem a_i (S autoriza ou não P_i a entrar em U_i):

$$\sigma_2 : \forall_c \Box [p_i \supset \circ (a_i \vee \sim a_i)]$$

V.3.3 Verificando a Especificação Formal

Seja Π a conjunção das propriedades acima e $i, j \in \{1,2\}, i \neq j$. Podemos investigar algumas propriedades desejáveis de um SD:

V.3.3.1 Exclusão Mútua

Conforme Π foi especificado, não é garantido a exclusão mútua. De fato, com uma hipótese adicional pode ser provado o contrário, isto é, que há um momento no qual os dois processos $P_i, i = 1,2$, estão em seus respectivos módulos de usq U_i. Sejam as seguintes fórmulas:

$h_1 : R_i \wedge R_j$

$h_2 : R_i \supset \circ a_i$

Elas não contradizem a si mesmas nem a nenhuma das propriedades do SD (Π), não sendo difícil imaginar uma execução do SD que as satisfaça. Afirmamos que, se existir alguma execução onde a conjunção de h_1 e h_2 ocorra, isto é:

$$H : \exists_c \Diamond [(R_i \wedge R_j) \wedge (R_i \supset \circ a_i)]$$

então existirá alguma execução onde eventualmente os processos estarão simultaneamente em seus módulos de uso, isto é, $\exists_c\Diamond(U_i \wedge U_j)$ é verificado.

Prova:

1. $\vdash \Pi \wedge H \supset \exists_c\Diamond(h_1 \wedge h_2)$	*TP*
2. $\vdash (R_i \supset \circ a_i) \supset (R_i \wedge R_j \supset \circ a_i \wedge \circ a_j)$	*TP*
3. $\vdash (R_i \wedge R_j) \wedge (R_i \supset \circ a_i) \supset (\circ a_i \wedge \circ a_j)$	*RP*(2)
4. $\vdash \exists_c\Diamond[(R_i \wedge R_j) \wedge (R_i \supset \circ a_i)] \supset \exists_c\Diamond\circ(a_i \wedge a_j)$	$\exists_c\Diamond$ (3) e *VT*
5. $\vdash \Pi \wedge H \supset \exists_c\Diamond(a_i \wedge a_j)$	*RP* (4 e *VT*)
6. $\vdash \Pi \wedge H \supset \exists_c\Diamond(a_i \wedge a_j) \wedge \forall_c\Box(a_i \supset \circ U_i)$	*RP* (π_7)
7. $\vdash \Pi \wedge H \supset \exists_c\Diamond[(a_i \wedge a_j) \wedge (a_i \supset \circ U_i)]$	*RP* (**T12**)
8. $\vdash \Pi \wedge H \supset \exists_c\Diamond(\circ U_i \wedge \circ U_j)$	*RP* (τ)
9. $\vdash \Pi \wedge H \supset \exists_c\Diamond\circ(U_i \wedge U_j)$	*RP* (8 e *VT*)
10. $\vdash \Pi \wedge H \supset \exists_c\Diamond(U_i \wedge U_j)$	*RP* (9 e *VT*)

■

Podemos entender a linha 10 da seguinte maneira: se existir uma execução na qual eventualmente h_1 e h_2 ocorram, então há uma execução na qual eventualmente os processos estarão simultaneamente em seus módulos de uso. Como H é plausível de acontecer, não contradizendo nenhuma propriedade de Π, então precisamos "reforçar" Π adicionando mais uma propriedade de forma que seja garantida a exclusão mútua. Seja a seguinte propriedade adicional ao processo sincronizador S:

$$\sigma_3 : \forall_c\Box[AUTO_i \supset (AUTO_i \wedge \sim AUTO_j)\mathcal{U}l_i]$$

e seja Π' a conjunção de Π com σ_3, então afirmamos que a propriedade da exclusão mútua é verificada, isto é, $\sim\exists_c\Diamond(U_i \wedge U_j)$ é verdade.

Prova:

1. $\vdash \Pi' \supset \forall_c\Box[AUTO_i \supset (AUTO_i \wedge \sim AUTO_j\mathcal{U}l_1]$	*TP* (σ_3)
2. $\vdash \Pi' \supset \forall_c\Box(U_i \supset AUTO_i)$	*TP* (π_8)
3. $\vdash \Pi' \supset \forall_c\Box[U_i \supset (AUTO_i \wedge \sim AUTO_j)\mathcal{U}l_1]$	*RP* (1 e 2)
4. $\vdash \Pi' \supset \forall_c\Box[U_i \supset (l_i \vee (AUTO_i \wedge \sim AUTO_j \wedge \circ((AUTO_i \wedge \sim AUTO_j\mathcal{U}l_1))))]$	*RP* (*VT* e 3)
5. $\vdash \Pi' \supset \forall_c\Box(U_i \supset \sim l_i)$	*TP* (π_{11})
6. $\vdash \Pi' \supset \forall_c\Box[U_i \supset (AUTO_i \wedge \sim AUTO_j \wedge \circ(AUTO_i \wedge \sim AUTO_j\mathcal{U}l_1))]$	*RP* (4 e 5)
7. $\vdash \Pi' \supset \forall_c\Box(U_i \supset \sim AUTO_j)$	*RP* (6)
8. $\vdash \Pi' \supset \forall_c\Box(\sim AUTO_j \supset \sim U_j)$	*TP* (π_9')
9. $\vdash \Pi' \supset \forall_c\Box(U_i \supset \sim U_j)$	*RP* (τ e 8)
10. $\vdash \Pi' \supset \forall_c\Box(\sim U_i \vee \sim U_j)$	*RP* (9)
11. $\vdash \Pi' \supset \forall_c\Box \sim (U_i \wedge U_j)$	*RP* (10)
12. $\vdash \Pi' \supset \sim\exists_c\Diamond(U_i \wedge U_j)$	*RP* (11 e **T2**)

■

V.3.3.2 Inanição

É necessário garantir que, em qualquer execução, se for feito uma solicitação s_i ao processo P_i, então este processo deve entrar em seu módulo de uso U_i em um intervalo finito de tempo, isto é, $\forall_c\Box(s_i \supset \Diamond U_i)$ é verificado. Isso só será possível se pusermos uma restrição adicional ao processo sincronizador S: a de que ele eventualmente autorize (envie a_i para) o processo que lhe pediu permissão (enviou p_i):

$$\sigma_4 : \forall_c\Box(p_i \supset \Diamond a_i)$$

Seja Π'' a conjunção de Π' com σ_4, então afirmamos que, no SD em estudo, nenhum processo morrerá por inanição, isto é, todas as solicitações serão eventualmente atendidas.

Prova:

1. $\vdash \Pi'' \supset \forall_c\Box(s_i \supset \circ E_i)$	$RP\,(\pi_4)$
2. $\vdash \Pi'' \supset \forall_c\Box(s_i \supset \Diamond E_i)$	$RP\,(VT$ e 1)
3. $\vdash \Pi'' \supset \forall_c\Box(E_i \supset \Diamond p_i)$	$RP\,(\pi_6)$
4. $\vdash \Pi'' \supset \forall_c\Box(s_i \supset \Diamond p_i)$	$RP\,(VT$, 2 e 3)
5. $\vdash \Pi'' \supset \forall_c\Box(p_i \supset \Diamond a_i)$	$RP\,(\sigma_4)$
6. $\vdash \Pi'' \supset \forall_c\Box(s_i \supset \Diamond a_i)$	$RP\,(VT$, 4 e 5)
7. $\vdash \Pi'' \supset \forall_c\Box(a_i \supset \circ U_i)$	$RP\,(\pi_7)$
8. $\vdash \Pi'' \supset \forall_c\Box(a_i \supset \Diamond U_i)$	$RP\,(VT, \bar{7})$
9. $\vdash \Pi'' \supset \forall_c\Box(s_i \supset \Diamond U_i)$	$RP\,(VT$, 6 e 8)

■

V.3.3.3 Bloqueio Perpétuo

No SD em estudo, os únicos módulos nos quais há a possibilidade de haver um bloqueio perpétuo seriam $R_i, i = 1,2$, isto é, os dois processos para sempre em R_i. Como os processos são seqüenciais, a única maneira de o controle da execução chegar a R_i é passando antes por E_i, e o bloqueio perpétuo seria uma execução na qual eventualmente E_i fosse verificado e, do próximo estado em diante, o processo ficasse no módulo R_i. Isto pode ser formalizado por $\exists_c\Diamond(E_i \wedge \circ\Box R_i)$. Afirmamos que, se o SD satisfizer Π'', isso jamais ocorre, isto é, $\forall_c\Box(E_i \supset \circ \sim\Box R_i)$ é verificado.

Prova:

1. $\vdash \Pi'' \supset \forall_c\Box(E_i \supset \Diamond p_i) \wedge \forall_c\Box(p_i \supset \Diamond a_i)$	$TP\,(\pi_6, \sigma_6)$
2. $\vdash \Pi'' \supset \forall_c\Box(E_i \supset \Diamond a_i)$	$RP\,(VT$ e 1)
3. $\vdash \Pi'' \supset \forall_c\Box(a_i \supset \circ U_i) \wedge \forall_c\Box(U_i \supset \sim R_i)$	$TP\,(\pi_7$ e $\pi_1 d)$
4. $\vdash \Pi'' \supset \forall_c\Box(a_i \supset \circ \sim R_i)$	$RP\,(VT$ e 3)
5. $\vdash \Pi'' \supset \forall_c\Box\Box(a_i \supset \circ \sim R_i)$	$RP\,(VT$ e 4)
6. $\vdash \Pi'' \supset \forall_c\Box(\Diamond a_i \supset \Diamond\circ \sim R_i)$	$RP\,(VT$ e 5)
7. $\vdash \Pi'' \supset \forall_c\Box(E_i \supset \Diamond\circ \sim R_i)$	$RP\,(VT$, 2 e 6)
8. $\vdash \Pi'' \supset \forall_c\Box(E_i \supset \circ\Diamond \sim R_i)$	$RP\,(VT$, 7)
9. $\vdash \Pi'' \supset \forall_c\Box(E_i \supset \circ \sim\Box R_i)$	$RP\,(VT$, 8)

■

Podemos entender a linha 9 da seguinte maneira: qualquer implementação que apresentar as propriedades Π', em todas as suas execuções, sempre será verificado que, se o controle da execução estiver em E_i, então no próximo estado, eventualmente o controle da execução não estará em R_i. Como após E_i segue-se R_i (propriedade π_5), daí segue que a espera em R_i é finita e como R_i é o único módulo onde é possível haver bloqueio perpétuo, concluímos que a implementação é isenta de bloqueio perpétuo.

V.4 Outras Lógicas de Tempo Ramificado

Vários sistemas formais para a lógica temporal de tempo ramificado têm sido propostos. Todos esses sistemas formais, com exceção do sistema K_b abaixo, têm em comum a presença do quantificador de caminhos seguido por uma combinação dos operadores temporais usuais ($\Box$, $\Diamond$, $\circ$ e $\mathcal{U}$). Em alguns desses sistemas, algumas restrições são colocadas no modo de como os operadores temporais podem ser combinados com os quantificadores de caminho.

V.4.1 O Sistema K_b

RESCHER *et alia* (1971) definem um sistema formal cujo modelo semântico considera o tempo de forma ramificada. Esse sistema, chamado de K_b, não possui uma quantificação sobre os caminhos, mas possui operadores temporais que "olham" para o passado. Esses operadores, H e P, onde Pp significa que p aconteceu alguma vez no passado e Hp significa que p tem acontecido a todo instante no passado, são usados na axiomatização do sistema K_b.

V.4.2 O Sistema *UB*

O sistema *UB* (BEN-ARI *et alii*, 1983) propõe uma unificação da LTL com a LTTR. Nele, são usados dois símbolos, um para cada modalidade: o primeiro, $\forall_c$ ou $\exists_c$, denota a quantificação sobre caminhos. O segundo, um dos operadores temporais $\Box$, $\Diamond$ ou $\circ$, denota a quantificação sobre os estados dos caminhos. Há três modalidades primitivas: $\forall_c\Box$, $\exists_c\Diamond$ e $\forall_c\circ$ e três modalidades duais são definidas: $\exists_c\Diamond\varphi \equiv\sim \forall_c\Box \sim \varphi$, $\forall_c\Diamond\varphi \equiv\sim \exists_c\Box \sim \varphi$ e $\exists_c\circ\varphi \equiv\sim\forall_c\circ \sim\varphi$.

Em outras palavras, no sistema *UB*, o quantificador de caminhos é sempre acompanhado por uma, e somente uma, ocorência dos operadores temporais $\Box$, $\Diamond$ ou $\circ$, não podendo haver aninhamentos destes operadores. Essa restrição limita, de modo significante, o poder de expressão da lógica. Por exemplo, uma propriedade associada à probidade de um sistema tal como "ao longo de algum futuro a propriedade φ ocorre infinitamente freqüentemente" pode ser formulada como $\exists_c\Box\Diamond\varphi$; entretanto essa fórmula envolve um aninhamento dos operadores temporais, violando as restrições da sintaxe do sistema *UB*, não existindo nenhuma fórmula equivalente a esta (EMERSON *et alia*, 1984).

V.4.3 CTL (*Computation Tree Logic*)

A CTL (CLARKE *et alii*, 1985; CLARKE *et alii*, 1986), à semelhança do *UB*, restringe a sintaxe de suas fórmulas: elas possuem o quantificador de caminhos seguido de um, e somente um, operador temporal. A CTL foi concebida visando sua utilização em um checador de modelos — uma ferramenta automatizada para verificar se um sistema concorrente satisfaz uma especificação formal escrita em CTL. A restrição sintática da CTL fez-se necessária para que o checador de modelos tivesse sua complexidade reduzida.

V.4.4 CTL^F (*Fair Computation Tree Logic*)

A CTL^F foi proposta por CLARKE *et alii* (1986). Nesse artigo, afirma-se que a CTL não pode expressar o fato que alguma propriedade φ deva eventualmente ser verificada em todas as execuções probas. Visando manipular a probidade e ainda obter um algoritmo de checagem de modelos eficiente, é proposto a modificação da semântica da CTL, dando origem à CTL^F. A CTL^F possui exatamente a mesma semântica da CTL, exceto que

todos os quantificadores de caminho variam sobre caminhos probos.

V.4.5 CTL*

Na CTL* (EMERSON *et alia*, 1986), um quantificador de caminhos pode prefixar uma fórmula composta por combinações sem restrições (isto é, envolvendo aninhamentos arbitrários e conectivos booleanos) dos operadores temporais usuais ($\Box$, $\Diamond$, $\circ$ e $\mathcal{U}$). EMERSON *et alia* (1984) afirmam que a CTL* engloba várias outras lógicas da literatura, entre elas o sistema $\mathcal{UB}$ e a CTL, além das lógicas de tempo ramificado propostas por LAMPORT (1980) e por GABBAY *et alii* (1980). A LTTR foi apresentada através da CTL*.

V.5 LTTR x LTL

A questão de se escolher entre a LTTR e a LTL tem muito pouco a ver com a questão filosófica da estrutura do tempo em si, o que levaria a problemas metafísicos como a questão do determinismo *versus* o livre-arbítrio. Ao invés disso, a escolha deve ser pragmaticamente baseada no tipo de SD e das propriedades que se deseja formalizar e estudar.

A LTL é o formalismo adequado para se caracterizar o conjunto de todas as seqüências de execuções que um SD gera e estudar propriedades que se verificam uniformemente para todas as seqüências de execução de um SD (BEN-ARI *et alii*, 1983). As lógicas de tempo linear são geralmente adequadas para a verificação da correção de programas concorrentes pré-existentes. Para objetivos de verificação manual, freqüentemente não se leva em conta qual caminho da execução é realmente seguido ou que um caminho particular existe porque estamos interessados tipicamente em propriedades que se verifiquem em *todos* os caminhos da execução. É satisfatório, portanto, escolher um caminho arbitrário e raciocinar sobre ele. Nestas situações, a simplicidade da LTL é um forte ponto a seu favor (EMERSON *et alia*, 1986).

A abordagem do tempo ramificado, por outro lado, considera, para um dado SD, o conjunto de todas as suas possíveis execuções. Alguns SDs possuem um aspecto não-determinístico oriundo das diferentes velocidades dos processos e de atrasos no envio de mensagens. Como resultado disso, há estados que possuirão mais de um sucessor, correspondendo ao não determinismo. Sobre esse conjunto de execuções, podemos estudar

propriedades existenciais tais como o término correto para, pelo menos, uma possível execução. Mais geralmente, podemos estudar a propriedade de que há sempre uma possível execução que atinge algum objetivo, o que, claramente, não implica em todas as execuções atingindo o mesmo objetivo.

Em suma, a escolha entre a LTL e a LTTR não deve ser feita no campo filosófico, mas deve ser ditada pelo tipo de SDs e propriedades que se queira estudar (BEN-ARI *et alii*, 1983). Entretanto, EMERSON *et alia* (1986) enumeram algumas vantagens da LTTR sobre a LTL:

- *Checagem de Modelos* – dado um SD de estados finitos, o grafo de estados globais de tal sistema pode ser visto como uma estrutra temporal ramificada (com um número grande, mas finito, de estados possíveis). O problema de checar se o SD possui uma certa propriedade reduz-se àquele de checar se a fórmula descrevendo aquela propriedade é verdade na estrutura temporal ramificada correspondente ao sistema. Em geral, parece que a checagem de modelos é mais fácil para as lógicas temporais de tempo ramificado do que para as de tempo linear.

- *Quantificando a LTL* – a LTL também sofre do problema de que, quando vemos uma LTL $\mathcal{L}$ como uma LTTR $\mathcal{B}(\mathcal{L})$, isto é, todas as fórmulas na forma $\forall_c\varphi$ onde φ é uma fórmula de $\mathcal{L}$, ela não é fechada sob negação. Se o SD estiver correto, será possível provar que uma propriedade é verificada em todas as suas execuções. Por outro lado, se o SD estiver incorreto (porque a propriedade não é verificada ao longo de alguma execução), será impossível refutar a propriedade para o sistema como um todo. Uma lógica que não pode expressar um fato como "a propriedade Ψ não se verifica ao longo de alguma execução do sistema" parece ter uma séria desvantagem.

- *Quantificação Existencial* – há situações nas quais desejamos a habilidade de afirmar explicitamente a existência de caminhos alternativos e devemos usar algum sistema de lógica de tempo ramificado. Isso provém do não-determinismo — além daquele usado para modelar o paralelismo — presente em muitos SDs. Seja uma instância do problema da exclusão mútua no qual cada processo P_i funciona como um servidor de terminal. A qualquer momento, P_i (não-deterministicamente) pode ou não receber um caractere. Um atributo chave de uma solução correta é que deva ser possível para um P_i particular permanecer em sua seção não-crítica. SNC_i para sempre

(esperando, mas nunca recebendo um caractere do teclado), enquanto outro P_j continua a receber e processar caracteres. Deve também ser possível que P_i receba um caractere e então entre em sua região de tentativa, T_i. Daí ele eventualmente entra na seção crítica SC_i onde o caractere é processado antes de P_i voltar a SNC_i. Mas, não importando o que aconteça, uma vez que P_i está em SNC_i, ou ele fica aí para sempre ou eventualmente entra em T_i. Para expressar esta propriedade, podemos usar uma fórmula da LTTR envolvendo um termo (o qual pretende-se que seja verificado sempre que P_i estiver em SNC_i) da forma $\exists_c \Box SNC_i \wedge \exists_c \Diamond T_i \wedge \forall_c(\Box SNC_i \vee \Diamond T_i)$. Essa fórmula não é possível de ser expressada na LTL (EMERSON *et alia* (1986) demonstram tal afirmação). Uma fórmula que é candidato natural, $(\Box SNC_i \vee \Diamond T_i)$, permite um modelo "degenerado", onde todas as execuções satisfazem $\Diamond T_i$ e nenhuma satisfaz $\Box SNC_i$.

Essa habilidade de quantificar existencialmente sobre caminhos é particularmente útil em aplicações tais como a síntese automática de programas a partir de especificações em lógica temporal (CLARKE *et alii*, 1986), onde é necessário especificações completas e muito precisas. É possível sintetizar, de forma bem-sucedida, uma classe de programas interessantes usando apenas a LTL (MANNA *et alia*, 1984), mas como o comentário acima indica, algum meio externo à lógica deve ser usado se desejarmos assegurar a existência de caminhos de computação alternativos.

V.6 Conclusões

Após formalizarmos a LTTR e a usarmos na especificação formal de SDs, chegamos a algumas conclusões sobre a adequação desse formalismo.

1. A LTTR oferece o formalismo para investigarmos propriedades *invariantes* ($\forall_c \Box \phi$), *inevitáveis* ($\forall_c \Diamond \phi$) e *potenciais* ($\exists_c \Diamond \phi$) (CLARKE *et alii*, 1986). As propriedades potenciais são particularmente úteis no estudo de SDs com aspectos não-determinísticos (por exemplo, aqueles SDs cujo funcionamento considera possíveis perdas/atrasos de mensagens ou aqueles SDs cuja interação com o meio externo altera seu funcionamento). Nesses SDs, é importante verificarmos se alguma execução eventualmente atinge uma situação indesejável, antecipando um resultado que só obteríamos após a implementação:

2. Encontrou-se certa dificuldade na especificação formal de processos cíclicos com uma condição especial de início. Na LTTR (assim como na LTL) não há como denotar esse instante único e especial, o início da execução do SD. O mais próximo que se consegue é algo do tipo

$$\text{INÍCIO} \supset \forall_c \Diamond \varphi$$

onde INÍCIO seria um predicado que especificaria as propriedades iniciais;

3. A especificação formal de um SD, a partir de um modelo ou de uma descrição informal nos faz ver o problema de diferentes perspectivas. Esse fato é ainda mais notável quando o formalismo é a LTTR pois as considerações sobre execuções alternativas permitem estudar propriedades interessantes (vide item 1 dessa lista);

4. A LTTR, até mais do que a LTL, carece de métodos que tornem seu uso mais cômodo. Na bibliografia pesquisada, nenhum método foi proposto que considerasse o tempo de forma ramificada;

5. Sentiu-se uma grande dificuldade em encontrar invariantes. A tarefa de construção de provas é geralmente tediosa e um certo grau de engenhosidade pode ser necessário para organizar a prova de uma maneira inteligível e simples;

6. Assim como na LTL, a elaboração de fórmulas na LTTR que representem as descrições informais (do problema ou de propriedades que a solução deva possuir) é uma importante etapa na especificação formal de SDs. Como na LTTR os quantificadores de caminho dão ainda mais expressividade ao que pode ser formalizado, eles podem ser mais um agravante a essa questão.

   Por exemplo, seja a propriedade de que a única maneira de chegarmos ao módulo M' de um processo é através do módulo M. Usando os nomes dos módulos como uma variável proposicional afirmando o lugar onde o controle da execução está, poderíamos formalizar essa propriedade por

$$\forall_c \Box(M \equiv \Diamond M') \qquad (\star)$$

   Entretanto, essa propriedade não impede que haja estados nos quais $\sim M'$ seja verificado. Usando TP, obtemos a fórmula

$$\forall_c \Box(\sim M \equiv \sim \Diamond M') \qquad (\star\star)$$

e ainda (por VT e TP)

$$\forall_c \Box(\sim M \supset \Box \sim M') \qquad (\star\star\star)$$

afirmando que, em qualquer execução sempre é verdade que, se $\sim M$ é verificado, então *nunca* M' será verdade. A fórmula $(\star\star\star)$ é conseqüência lógica de $(\star)$ mas não é conseqüência da descrição informal da propriedade, pois se o controle da execução estiver antes de M ($\sim M$ é verificado), então pode-se concluir que M' nunca será verificado. Nesse exemplo, uma alternativa seria expressar a propriedade usando uma forma mais fraca de $(\star)$, como

$$\forall_c \Diamond(M \equiv \circ M')$$

afirmando a existência de um estado em qualquer execução, no qual é feito a transição de M para M'.

7. Como o repertório de regras (regras derivadas, teoremas, axiomas e regras de inferência) aumenta consideravelmente, é necessário muita experiência para conduzir as provas.

V.7 Notas Bibliográficas

A sintaxe da LTTR é a da CTL* (EMERSON *et alia*, 1986). A semântica é uma adaptação da semântica da CTL* para o formalismo do capítulo anterior. A estrutura temporal ramificada também é uma adaptação do capítulo anterior. As noções de fórmulas de estado e de caminhos são de EMERSON *et alia* (1986), assim como as definições da seção V.2.2. O exemplo de obtenção de valor-verdade na LTTR é original, assim como as definições, teoremas e provas da seção V.2.6, que foram inspirados no capítulo anterior.

O sistema formal Σ_{LTTR} é inspirado no sistema *UB* (BEN-ARI *et alii*, 1983) e a prova de sua consistência foi baseada em KRÖGER (1987). As regras derivadas e teoremas são originais, assim como suas demonstrações. O exemplo de especificação formal de SDs com a LTTR é original — tanto a especificação formal em si como as provas.

A comparação entre a LTTR e a LTL foi feita a partir de BEN-ARI *et alii* (1983) e EMERSON *et alia* (1986). LAMPORT (1980) propõe uma lógica temporal de tempo

ramificado possuindo a quantificação universal de caminhos implícita. A partir dessa lógica, várias conclusões são obtidas, entre elas que as lógicas de tempo linear e tempo ramificado têm poderes expressivos incomparáveis e que a lógica de tempo linear é superior à lógica de tempo ramificado para se raciocinar sobre programas concorrentes. EMERSON *et alia* (1986) discordam sobre essas conclusões e demonstram como LAMPORT (1980) errou ao chegar essas conclusões.

A LTTR tem sido usada na verificação automática de sistemas concorrentes de estados finitos (isto é, com um número grande, mas finito, de estados possíveis). CLARKE *et alii* (1986) apresentam um procedimento eficiente para essa verificação. Essa técnica oferece uma alternativa prática à construção manual de provas, uma vez que essa tarefa pode ser muito trabalhosa. Um estudo da CTL* é feito por EMERSON *et alia* (1984). Nesse artigo, questões sobre a complexidade de um procedimento de decisão são investigadas.

Capítulo VI

Lógica Temporal de Conjuntos de Intercalações (LTCI)

A Lógica Temporal de Conjunto de Intercalações (LTCI — KATZ *et alia*, 1987) é a lógica temporal cujo modelo semântico é baseado em um conjunto de conjuntos de seqüências de estados (intercalações — cf. capítulo II). Possuindo características da LTL e da LTTR, considerando ainda a ordenação parcial dos eventos, essa lógica pode descrever propriedades que não podem ser descritas com a LTL ou a LTTR. A diferença em poder de expressão é qualitativa e não meramente devido a presença ou ausência de um operador temporal particular.

O principal objetivo deste capítulo é apresentar um modelo alternativo para a lógica temporal, modelo este que considera a ordenação parcial dos eventos e ainda assim permite o raciocínio sobre os estados globais de um SD. Na primeira seção deste capítulo, é feito uma breve introdução sobre a LTCI, sendo mostrado em que ela difere da LTL e da LTTR. Na segunda seção, é apresentada, formalmente, a LTCI: sintaxe, algumas definições, a estrutura temporal de conjuntos de intercalações e a semântica. Depois, são feitas algumas restrições adicionais aos modelos e tecemos considerações sobre a axiomatização da LTCI. Na terceira seção, discorremos sobre a especificação formal de SDs e as maneiras alternativas de especificar, sendo mostrado um exemplo simples. Encerrando o capítulo, as seções das conclusões e das notas bibliográficas. Neste capítulo, os teoremas e proposições são exibidos sem provas — estas podem ser encontradas no trabalho de KATZ *et alia* (1987).

VI.1 Introdução

As lógicas temporais vistas anteriormente (capítulos IV e V) possuem algo em comum: elas são interpretadas sobre os estados globais do SD, adotando a visão das intercalações (vide capítulo II) e assumindo uma intercalação completa de ações atômicas, o que gera seqüências de estados globais. LAMPORT (1985b) chama a atenção de que há uma ordem parcial "inerente" entre os eventos em uma execução de um SD. A ordem é definida pela seqüencialidade de eventos locais em um processo e os eventos de envio e posterior recebimento de uma mensagem em particular. O fecho transitivo dessas relações seqüenciais definem uma ordem parcial e todas as outras intercalações de eventos não relacionados por essa ordem são arbitrárias. As lógicas citadas, embora convenientes para expressar estados globais e provar propriedades por indução, não podem expressar propriedades decorrentes da ordem parcial subjacente, pois essa informação é perdida uma vez que as seqüências de intercalações são geradas.

Por outro lado, em uma abordagem puramente de ordenações parciais (visão espaço-tempo — capítulo II), não existe o conceito de um estado global e, por conseguinte, muitas propriedades interessantes de natureza global não podem ser expressas.

A LTCI une características de ambas as abordagens: a visão das intercalações permitirá raciocinar sobre estados globais, mas as intercalações oriundas da mesma ordenação parcial serão agrupadas de forma que cada conjunto possa ser considerado independentemente quando for necessário.

As modalidades da LTCI são baseadas nos operadores da LTTR. Uma estrutura da LTCI é, entretanto, diferente da estrutura de $\mathcal{UB}$ (BEN-ARI *et alii*, 1983), CTL e CTL* (CLARKE *et alii*, 1985; CLARKE *et alii*, 1986) pois nessas lógicas (vide capítulo V) a estrutura ramificada única de um programa inclui todos os caminhos que estão associados com qualquer execução possível, sendo comum identificar uma execução única com uma intercalação. Entretanto, na LTCI, uma execução é identificada com uma única ordenação parcial entre os eventos conforme a visão da ordenação parcial. Constrói-se o conjunto de todos os estados globais que se originam de uma ordenação parcial e usa-se uma relação no conjunto de estados globais para formar uma sub-estrutura ramificada. Uma estrutura na LTCI é um conjunto dessas sub-estruturas ramificadas, cada uma das quais

está associada com uma única execução. Para uma fórmula ser satisfeita por tal estrutura, ela deve ser satisfeita por cada uma das sub-estruturas consideradas como únicas, conforme a semântica ramificada usual (cf. capítulo V).

A LTCI herda características da LTL, da LTTR e da abordagem das ordenações parciais. Da LTL, é usada a transição entre estados globais e a propriedade de que uma fórmula é satisfeita por uma estrutura se ela é satisfeita por qualquer execução (de ordem parcial). A estrutura ramificada é do *UB* ou CTL (vide capítulo V) bem como a capacidade de escolher entre continuações alternativas. Da abordagem das ordenações parciais, é adotada a visão de uma ordenação parcial como uma entidade cuja estrutura é semanticamente significante, embora a forma que ela tome aqui seja diferente. A LTCI exige, entretanto, que duas propriedades semânticas sejam satisfeitas: a primeira é que o número de eventos na ordenação parcial (e, por conseguinte, o número de processos) seja contável. A segunda é que nenhum evento possui um número infinito de predecessores. Se não fosse assim, não seria possível assegurar a existência de caminhos que contêm cada evento de uma ordem parcial.

VI.2 A Lógica Temporal de Conjuntos de Intercalações (LTCI)

Nesta seção, definições formais da sintaxe e semântica da LTCI são dadas. Desde que a LTCI é orientada para a descrição de SDs com uma semântica de ordem parcial, ela será apresentada em dois níveis: primeiro, uma estrutura geral para a LTCI é definida como um conjunto de conjuntos de seqüências de estados. Cada conjunto Σ de seqüências contém seqüências σ com algumas restrições semânticas bem básicas. A sintaxe e a semântica da LTCI são definidas à semelhança da semântica da CTL (CLARKE *et alii*, 1986 — cf. capítulo V), com a diferença de que um conjunto de conjuntos de seqüências é usado no lugar de um único conjunto de seqüências. Então, em um segundo nível, o conjunto de estruturas é limitado àquelas que são construídas de um dado conjunto de ordenações parciais.

VI.2.1 Sintaxe

As fórmulas da LTCI são construídas a partir dos símbolos da LTLp (itens 1–5, vide capítulo IV), acrescentando:

6. Quantificadores de seqüência: $\forall_\sigma$ (*em toda seqüência*) e $\exists_\sigma$ (*em alguma seqüência*);

Os operadores temporais são, aqui, também chamados de *modalidades de seqüência*. Usando esse alfabeto, define-se recursivamente as fórmulas bem-formadas com base nas fórmulas atômicas, à semelhança da Lp (apêndice A), acrescentando o seguinte item às fórmulas bem-formadas:

d) se A e B são fórmulas bem-formadas, então $\forall_\sigma \Box A$, $\exists_\sigma \Box A$, $\forall_\sigma \Diamond A$, $\exists_\sigma \Diamond A$, $\forall_\sigma \circ A$, $\exists_\sigma \circ A$, $\forall_\sigma (A \mathcal{U} B)$ e $\exists_\sigma (A \mathcal{U} B)$ também são fórmulas bem-formadas;

Observações:

i) As abreviações da Lp (apêndice A) definem os demais conectivos $\vee$, $\supset$ e $\equiv$;

ii) Por simplicidade, foi definida uma LTCI proposicional; a extensão para uma LTCI de primeira ordem pode ser feita da maneira que foi feito na LTL;

iii) A LTCI, à semelhança da CTL (CLARKE *et alii*, 1986 — vide capítulo V), restringe o tipo de fórmula que pode aparecer após um quantificador de seqüência, não permitindo o aninhamento dos operadores temporais. Isto significa que os quantificadores de seqüência devem ser seguidos por um, e somente um, operador temporal ($\Box$, $\Diamond$, $\circ$ ou $\mathcal{U}$). Podemos, entretanto, definir uma LTCI*, semelhante à CTL* (CLARKE *et alii*, 1986 — vide capítulo V), na qual relaxaríamos essa restrição, permitindo, por exemplo, fórmulas do tipo $\forall_\sigma \Box \Diamond A$, $\forall_\sigma A$ ou $\exists_\sigma \Box (A \mathcal{U} B)$.

iv) À lista de prioridades dos operadores da LTLp acrescenta-se:

$\forall_\sigma$ e $\exists_\sigma$ têm maior prioridade que todos os outros conectivos e operadores;

VI.2.2 Notação e Definições

Visando a apresentação da LTCI, o estabelecimento de uma notação e algumas definições se fazem necessárias.

Notação: Será usada a letra grega σ para denotar uma *seqüência de estados* s_i (ou *caminho*), por exemplo $\sigma = s_0 s_1 s_2 \cdots$. À semelhança da LTTR (seção V.2.2), são definidas as operações sobre seqüências *primeiro* (dá o primeiro estado da seqüência), *sucessor* (*suc* — dá o sufixo da seqüência a partir de seu segundo estado) e os *sufixos* de uma seqüência.

Definição VI.2.2.1 Seja **V** um conjunto de eventos (vide subseção II.1.6) tal que $\mathbf{V} = \{e_0, e_1, e_2, \ldots\}$. A *relação de precedência entre eventos* $<$ ($< \subseteq \mathbf{V}^2$) é tal que $e_i < e_j$ se, e somente se, o evento e_i ocorrer antes do evento e_j. Por definição, todo evento $e \in \mathbf{V}$ precede ele mesmo, $e < e$.

A dupla $(\mathbf{V}, <)$ fornece uma ordenação parcial em um conjunto de eventos **V**. O termo *uma execução única* é tido como sinônimo de "uma ordenação parcial". A relação de precedência entre eventos representa exatamente o que a abstração de SDs permite inferir sobre a ordem de eventos. Sabendo que dois eventos e_1 e e_2 não relacionados (segundo $<$) foram executados, nenhuma conclusão adicional pode ser obtida quanto a sua ordem relativa de execução enquanto não for fornecido um relógio global. Dois eventos assim são ditos *concorrentes*.

Sempre que houver uma escolha não-determinística explícita no código de algum processo do SD, uma execução incluirá somente a escolha específica feita naquela execução. Portanto, em geral, haverá várias ordenações parciais associadas a um programa.

Um estado global não existe diretamente no modelo das ordenações parciais (visão espaço-tempo). No sentido de incorporar tais estados, para cada ordenação parcial possível $(\mathbf{V}, <)$ os seguintes termos (LAMPORT, 1985b) são definidos:

Definição VI.2.2.2 Uma *fatia* **F** é um subconjunto finito de **V** no qual verifica-se a seguinte propriedade: para cada par de eventos e_i e e_j em **V** tais que $e_i < e_j$, se $e_j \in \mathbf{F}$ então $e_i \in \mathbf{F}$.

Uma fatia $\mathbf{F} \subseteq \mathbf{V}$ é, portanto, um subconjunto finito de **V** fechado sob a relação

de precedência. O conjunto de todas as fatias definidas em $(V, <)$ será denotado por Δ.

Um *estado global* pode ser representado por uma fatia. Ele está associado a um instantâneo global (vide subseção II.1.6) que é elemento do produto cartesiano de todos os instantâneos locais resultantes após a execução dos eventos maximais em F conforme a relação $<$ (um *evento maximal* $e \in F$ é aquele que, dado qualquer $f \in F, f \neq e$, nunca é o caso que $e < f$). Além disso, em um modelo de comunicação assíncrona, para todo canal de comunicação associa-se um conjunto de mensagens enviadas nele por eventos de F e recebidas por eventos fora de F.

É possível que dois ou mais estados globais diferentes de um programa tenham exatamente os mesmos instantâneos em uma execução. Por exemplo, um laço em um programa pode fazer com que o conjunto de variáveis possuam repetidamente os mesmos valores durante a única execução. Entretanto, estados globais diferentes ocorrem porque o conjunto de eventos acumulados nas fatias apropriadas é estendido cada vez que os eventos ocorrem. Portanto, diferentes fatias podem ser caracterizadas pelo mesmo instantâneo.

Definição VI.2.2.3 A *relação básica de acessibilidade* $R : \Delta \times \Delta$ é tal que $R(s,t)$ é verificado se, e somente se, $s \subset t$ (t possui pelo menos os mesmos eventos de s).

Quando $R(s,t)$ é verificado, dizemos que s *precede* t, devendo-se notar que s e t são finitos, pela definição de Δ.

Definição VI.2.2.4 A *relação de acessibilidade imediata* $\rho : \Delta \times \Delta$ é tal que $\rho(s,t)$ é verificado se, e somente se, $R(s,t)$ for verificado e não houver r tal que $\rho(s,r)$ e $\rho(r,t)$.

Dois estados globais s e t estão relacionados por ρ se t for diferente de s pela execução de um único evento adicional. Se $\rho(s,t)$ é verificado, dizemos que s *precede imediatamente* t.

A relação ρ entre os estados globais gera uma estrutura ramificada, pois para um único estado, geralmente temos mais que um sucessor. Os possíveis sucessores alternativos são provocados pela ocorrência de eventos não-relacionados (de acordo com a ordenação parcial) em diferentes processos e não pelo não-determinismo dentro de um processo.

Segundo a visão das intercalações, olhamos para seqüências de eventos. Cada

seqüência é uma ordenação total entre (os começos dos) eventos. Portanto, entre cada um dos eventos em uma seqüência, mesmo em processos diferentes, há uma ordem. Cada evento começa a partir de um estado global do programa e termina com um novo estado global.

Definição VI.2.2.5 Uma seqüência maximal única de instantâneos globais é chamada uma *seqüência de intercalações* ou um *caminho* (maximalidade de uma seqüência significa que ela não é um prefixo próprio de outra seqüência). Uma porção finita e contígua de um caminho é chamado de um *caminho finito*.

Pode-se facilmente ver (LAMPORT, 1985b) que o conjunto de caminhos gerados por ρ é exatamente o conjunto de seqüências intercaladas que estão relacionadas a uma única execução. Deve-se salientar que a relação ρ é construída separadamente para cada execução.

Uma propriedade interessante da relação R é:

Proposição VI.2.2.1 *Se $R(r, s)$ é verificado (portanto $r \subset s$ e r e s são finitos) então há um caminho finito de r a s.*

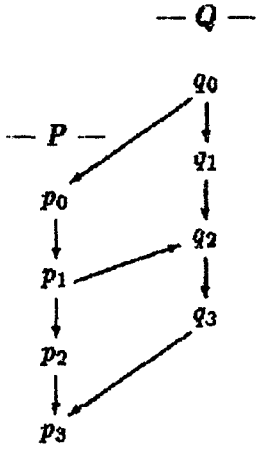

Figura VI.1: Exemplo de representação de dois processos através de eventos

VI.2.3 Um Exemplo de SD Visto pela LTCI

Seja um SD composto por dois processos P e Q. Os processos consistem em seqüências de comandos que, ao serem executados, constituirão os eventos, representados nesse exemplo por p_i e $q_i, i = 0, \ldots, 3$, correspondendo aos eventos de P e Q, respectivamente. A figura VI.1 mostra os eventos dos processos P e Q dispostos em duas colunas, uma para cada processo. Na figura, setas verticais dão a ordem dos eventos em cada processo e setas diagonais dão a ordem de eventos em processos diferentes podendo ser entendidas, por exemplo, como a execução de um comando de comunicação: a seta aponta para o evento de recebimento da mensagem e a outra extremidade dá o evento de emissão da mensagem.

$q_0 < q_1$	$q_0 < p_0$	$p_0 < p_1$
$p_1 < p_2$	$p_2 < p_3$	$p_1 < q_2$
$q_1 < q_2$	$q_2 < q_3$	$q_3 < p_3$

Tabela VI.1: Relação $<$ para os eventos do SD

No conjunto de eventos $\mathbf{V} = \{p_0, \ldots, p_3, q_0, \ldots, q_3\}$ é definida a relação $<$: $p_i < p_i, q_i < q_i, i = 0, \ldots, 3$, e também a tabela VI.1.

As fatias definidas em $(\mathbf{V}, <)$ são dadas pela tabela VI.2.

$\mathbf{F}_0 = \{q_0\}$	$\mathbf{F}_4 = \{q_0, q_1\}$	$\mathbf{F}_8 = \{p_0, p_1, q_0, q_1, q_2\}$
$\mathbf{F}_1 = \{p_0, q_0\}$	$\mathbf{F}_5 = \{p_0, q_0, q_1\}$	$\mathbf{F}_9 = \{p_0, p_1, p_2, q_0, q_1, q_2\}$
$\mathbf{F}_2 = \{p_0, p_1, q_0\}$	$\mathbf{F}_6 = \{p_0, p_1, q_0, q_1\}$	$\mathbf{F}_{10} = \{p_0, p_1, p_2, q_0, q_1, q_2, q_3\}$
$\mathbf{F}_3 = \{p_0, p_1, p_2, q_0\}$	$\mathbf{F}_7 = \{p_0, p_1, p_2, q_0, q_1\}$	$\mathbf{F}_{11} = \{p_0, p_1, p_2, p_3, q_0, q_1, q_2, q_3\}$

Tabela VI.2: Fatias construídas na relação $<$

O conjunto das fatias $\Delta = \{\mathbf{F}_0, \ldots, \mathbf{F}_{11}\}$ define a relação de acessibilidade imediata ρ da maneira mostrada pela tabela VI.3.

Considerando as fatias representando estados globais e a relação ρ, podemos construir as seguintes seqüências:

$\rho(\mathbf{F}_0, \mathbf{F}_1)$	$\rho(\mathbf{F}_0, \mathbf{F}_4)$	$\rho(\mathbf{F}_1, \mathbf{F}_2)$	$\rho(\mathbf{F}_1, \mathbf{F}_5)$	$\rho(\mathbf{F}_2, \mathbf{F}_3)$
$\rho(\mathbf{F}_2, \mathbf{F}_6)$	$\rho(\mathbf{F}_3, \mathbf{F}_7)$	$\rho(\mathbf{F}_4, \mathbf{F}_5)$	$\rho(\mathbf{F}_5, \mathbf{F}_6)$	$\rho(\mathbf{F}_6, \mathbf{F}_7)$
$\rho(\mathbf{F}_6, \mathbf{F}_8)$	$\rho(\mathbf{F}_7, \mathbf{F}_9)$	$\rho(\mathbf{F}_8, \mathbf{F}_9)$	$\rho(\mathbf{F}_9, \mathbf{F}_{10})$	$\rho(\mathbf{F}_{10}, \mathbf{F}_{11})$

Tabela VI.3: Relação ρ para as fatias do conjunto Δ

$\sigma_0 = \mathbf{F}_0\mathbf{F}_1\mathbf{F}_2\mathbf{F}_3\mathbf{F}_7\mathbf{F}_9\mathbf{F}_{10}\mathbf{F}_{11}$; $\sigma_1 = \mathbf{F}_0\mathbf{F}_1\mathbf{F}_5\mathbf{F}_6\mathbf{F}_7\mathbf{F}_9\mathbf{F}_{10}\mathbf{F}_{11}$;

$\sigma_2 = \mathbf{F}_0\mathbf{F}_1\mathbf{F}_5\mathbf{F}_6\mathbf{F}_8\mathbf{F}_9\mathbf{F}_{10}\mathbf{F}_{11}$; $\sigma_3 = \mathbf{F}_0\mathbf{F}_4\mathbf{F}_5\mathbf{F}_6\mathbf{F}_7\mathbf{F}_9\mathbf{F}_{10}\mathbf{F}_{11}$;

$\sigma_4 = \mathbf{F}_0\mathbf{F}_4\mathbf{F}_5\mathbf{F}_6\mathbf{F}_8\mathbf{F}_9\mathbf{F}_{10}\mathbf{F}_{11}$.

cada uma das quais representando uma possível história do SD da figura VI.1.

VI.2.4 Estrutura Temporal de Conjuntos de Intercalações (C)

Uma estrutura temporal de conjuntos de intercalações C para a linguagem $\mathcal{L}_{LTCI}$ é a dupla (E, I), onde

- $\mathbf{E} = \{s_0, s_1, s_2, \ldots\}$ é o conjunto de *estados*. A relação básica de acessibilidade R (definição VI.2.2.3), define uma ordenação parcial e, como na LTL, cada estado $s \in \mathbf{E}$ é um mapeamento das fórmulas atômicas aos valores-verdade **v** e **f**.
- $\mathbf{I} = \{\Sigma_0, \Sigma_1, \Sigma_2, \ldots\}$ é um conjunto de conjuntos de seqüências de estados, onde cada $\Sigma \in \mathbf{I}$, $\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \ldots\}$ é um conjunto *não-vazio* de seqüências σ fechado a sufixos e a fusões (vide capítulo V). Cada seqüência $\sigma \in \Sigma$, $\sigma = s_i s_j s_k \cdots$, é uma seqüência de estados tal que os estados estão relacionados dois a dois pela relação ρ de acessibilidade imediata (definição VI.2.2.4).

Escreve-se $\mathbf{E}(\Sigma)$ denotando o conjunto de todos os estados de E que aparecem pelo menos em uma das seqüências $\sigma \in \Sigma$.

VI.2.5 Semântica

Para toda estrutura temporal de conjuntos de intercalações $\mathbf{C} = (\mathbf{E}, \mathbf{I})$, todos os conjuntos de seqüências $\Sigma \in \mathbf{I}$ e toda fórmula F, definimos indutivamente o valor-verdade $\mathbf{C}_i(F) \in \{\mathbf{v}, \mathbf{f}\}$, para cada estado $s_i \in \mathbf{E}(\Sigma)$. $\mathbf{C}_i(F)$ significa, informalmente, "o valor-verdade de F no estado s_i":

1. $\mathbf{C}_i(v) = s_i(v)$ para $v \in \mathcal{V}$. Informalmente quer dizer "o valor-verdade de v no estado s_i é dado pela atribuição que s_i fizer a v";

2. $\mathbf{C}_i(\sim A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_i(A) = \mathbf{f}$;

3. $\mathbf{C}_i(A \wedge B) = \mathbf{v}$ se, e somente se, $\mathbf{C}_i(A) = \mathbf{v}$ e $\mathbf{C}_i(B) = \mathbf{v}$;

4. $\mathbf{C}_i(\forall_\sigma \Box A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_k(A) = \mathbf{v}$, onde s_k=*primeiro*(σ^j), para todo $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$, para todo $j \geq 0$;

5. $\mathbf{C}_i(\forall_\sigma \Diamond A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_k(A) = \mathbf{v}$, onde s_k=*primeiro*(σ^j), para todo $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$, para algum $j \geq 0$;

6. $\mathbf{C}_i(\forall_\sigma \bigcirc A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_k(A) = \mathbf{v}$, onde s_k=*primeiro*(σ^1), para todo $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$;

7. $\mathbf{C}_i(\forall_\sigma A \mathcal{U} B) = \mathbf{v}$ se, e somente se, para todo $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$, $\mathbf{C}_k(B) = \mathbf{v}$ onde s_k= *primeiro*(σ^j), para algum $j \geq 0$ e $\mathbf{C}_l(A) = \mathbf{v}$, onde s_l=*primeiro*(σ^m) para todo $m, 0 \leq m < j$;

8. $\mathbf{C}_i(\exists_\sigma \Box A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_k(A) = \mathbf{v}$, onde s_k=*primeiro*(σ^j), para algum $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$, para todo $j \geq 0$;

9. $\mathbf{C}_i(\exists_\sigma \Diamond A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_k(A) = \mathbf{v}$, onde s_k=*primeiro*(σ^j), para algum $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$, para algum $j \geq 0$;

10. $\mathbf{C}_i(\exists_\sigma \bigcirc A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_k(A) = \mathbf{v}$, onde s_k=*primeiro*(σ^1), para algum $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$;

11. $\mathbf{C}_i(\exists_\sigma A \mathcal{U} B) = \mathbf{v}$ se, e somente se, para algum $\sigma \in \Sigma$ tal que *primeiro*$(\sigma) = s_i$, $\mathbf{C}_k(B) = \mathbf{v}$ onde s_k=*primeiro*(σ^j), para algum $j \geq 0$ e $\mathbf{C}_l(A) = \mathbf{v}$, onde s_l=*primeiro*(σ^m) para todo $m, 0 \leq m < j$;

Observações:

i) Com essa semântica, estabeleceu-se um arcabouço que permitiria a introdução de um outro par de quantificadores: um expressando que uma fórmula é verdadeira em todos os conjuntos de seqüências e outro expressando a existência de um conjunto de seqüências que satisfaz uma fórmula. Entretanto, na LTCI, uma quantificação sobre todos os conjuntos de seqüências é sempre assumido, não sendo necessário escrevê-la: nas regras semânticas acima, os conjuntos Σ não são quantificados, assumindo-se que é para *todo* Σ.

ii) A semântica da LTCI guarda semelhanças com a LTTR e com a LTL. Com a LTTR, temos que o valor-verdade de uma fórmula é obtido em relação aos estados de alguma (todas) seqüência de estados (ou caminho). Com a LTL, temos que o valor-verdade de uma fórmula diz respeito a *todos* os conjuntos de seqüências (ou caminhos).

VI.2.6 Restrições Adicionais aos Modelos

Foi definido anteriormente um arcabouço para lidar com um conjunto de conjuntos de seqüências. Agora, concentramos a atenção no conjunto de estruturas geradas a partir do modelo de ordenações parciais usando a relação ρ entre estados globais (definição VI.2.2.4). A relação ρ foi construída de uma ordem parcial particular $(V, <)$ e, portanto, corresponde a uma única execução. As seqüências consideradas serão os *caminhos* e os conjuntos de seqüências serão chamados de *conjuntos de intercalações*.

Assim,

Definição VI.2.6.1 $\text{CAMINHOS}(\rho) = \{(s_0 s_1 s_2 \cdots) | \forall i \geq 0, \rho(s_i, s_{i+1})\}$.

Definição VI.2.6.2 Um caminho *aceitável* σ satisfaz a condição que para cada evento $e \in V$ existe um estado global em σ tal que (como uma fatia) contém e.

A definição anterior significa que qualquer caminho inclui todos os eventos da ordenação parcial a partir da qual ele foi gerado e não pode ignorar o próximo evento de um processo para sempre.

Definição VI.2.6.3 O conjunto dos caminhos aceitáveis de $\text{CAMINHOS}(\rho)$ é denotado

por *aceitável*(CAMINHOS(ρ)).

Proposição VI.2.6.1 *O conjunto de caminhos aceitáveis não é vazio.*

A prova dessa proposição, como dada por KATZ *et alia* (1987) é: (lembrando que, por definição, o conjunto V de eventos é enumerável) seja o caminho gerado através da repetição do seguinte procedimento: dado qualquer prefixo finito do caminho, é escolhido um evento que ainda não esteja em nenhuma fatia desse prefixo; então, estende-se a última fatia do prefixo para conter este evento — este caminho é aceitável.

Definição VI.2.6.4 Um *conjunto de intercalações* é um conjunto infinito de caminhos formado a partir de *aceitável*(CAMINHOS(ρ)), onde cada caminho finito aceitável (isto é, um caminho no qual seu último estado global não possui sucessores segundo a relação ρ) é feito infinito pela repetição de seu último estado.

Proposição VI.2.6.2 *O conjunto de caminhos que constitui um conjunto de intercalações é fechado a sufixos e a fusões.*

Desde que é concentrada a atenção em uma interpretação especial para a LTCI, a saber, as "transições globais causadas pelo modelo de ordenação parcial da execução", a propriedade semântica específica da aceitabilidade é embutida na lógica. Sob essa interpretação restrita, uma estrutura C irá conter um conjunto de conjuntos de intercalações, um para cada ordenação parcial possível de um programa.

Uma estrutura C irá satisfazer uma fórmula F, se cada um dos conjuntos de intercalações em C, tomados como um estrutura ramificada independente, satisfizer F. Portanto, uma fórmula F será considerada verdadeira em uma estrutura se F for satisfeita por cada uma das execuções. Isto é uma propriedade de "sempre falar sobre *todas as execuções*", semelhante à LTL, e em oposição à LTTR, onde é possível falar sobre a existência de uma execução (vide observação ii) da subseção VI.2.5).

Definição VI.2.6.5 Uma fórmula A é *válida na estrutura temporal de conjuntos de intercalações* $\mathbf{C} = (\mathbf{E}, \mathbf{I})$, denotando-se por $\models_{\mathbf{C}} A$, se, e somente se, $\mathbf{C}_i(A) = \mathbf{v}$ para todo $s_i \in \mathbf{E}(\Sigma)$, para todo $\Sigma \in \mathbf{I}$.

O significado intuitivo de $\models_{\mathbf{C}} A$ é que, para todo conjunto de intercalações Σ

(isto é, para toda execução), para todo estado s_i aparecendo em algum caminho de Σ, $\mathbf{C}_i(A) = \mathbf{v}$. Por exemplo, seja **C** uma estrutura de todos os conjuntos de intercalações obtidos através da execução de um programa distribuído **P** composto por dois processos P_1 e P_2. Seja F uma proposição sobre estados globais. As proposições $INÍCIO_1$ e $INÍCIO_2$ significam que o controle da execução de P_1 e P_2, respectivamente, estão antes do início do processo. Seja A a fórmula $(INÍCIO_1 \wedge INÍCIO_2) \supset \exists_\sigma \Diamond F$. Nesse caso, $\mathbf{C}_i(A) = \mathbf{v}$ significa que, se s_i é um estado que satisfaz $(INÍCIO_1 \wedge INÍCIO_2)$, então existe um caminho em Σ começando com s_i no qual há um estado que satisfaz F. Portanto, o significado de $\models_{\mathbf{C}} A$ é "para toda execução, há um caminho a partir do estado global inicial que atinge um estado satisfazendo F".

As definições de *fórmulas válidas* e *fórmulas que seguem de um conjunto $\mathcal{F}$ de fórmulas* são feitas à semelhança das definições A.1.3.2 e A.1.3.3 (apêndice A).

VI.2.7 Axiomatização da LTCI

A proposta da LTCI por KATZ *et alia* (1987) não apresenta um sistema dedutivo. Entretanto, são indicados alguns sistemas formais que poderiam ser usados como base para o Σ_{LTCI}.

Considerando as regras de definições básicas, temos que a LTCI interpreta uma fórmula da LTTR simultaneamente sobre um conjunto de estruturas Abrahamson (uma *estrutura Abrahamson* é uma estrutura que permite os quantificadores de seqüência variar sobre um conjunto de seqüências semanticamente definido e fechado a sufixos e a fusões (KATZ *et alia*, 1987)). Portanto, um sistema formal como o *UB* (BEN-ARI *et alii*, 1983 — vide seção V.4.2) é um candidato natural para estabelecer as bases do Σ_{LTCI}, pois sua sintaxe é semelhante à da LTCI. Também podemos adaptar alguns axiomas do Σ_{LTTR} e acrescentá-los ao Σ_{LTCI}, visando aumentar sua expressividade. Alguns axiomas para o Σ_{LTCI} seriam:

A1. $\vdash \forall_\sigma \Box A \equiv \sim \exists_\sigma \Diamond \sim A$

A2. $\vdash \forall_\sigma \Box A \supset A$

A3. $\vdash A \supset \exists_\sigma \Diamond A$

A4. $\vdash \forall_\sigma\Box(A \wedge B) \equiv (\forall_\sigma\Box A \wedge \forall_\sigma\Box B)$

A5. $\vdash \forall_\sigma\Box(A \supset B) \equiv (\forall_\sigma\Box A \supset \forall_\sigma\Box B)$

A6. $\vdash \exists_\sigma\Diamond(A \wedge B) \supset (\exists_\sigma\Diamond A \wedge \exists_\sigma\Diamond B)$

A7. $\vdash \forall_\sigma\Box A \supset \exists_\sigma\Box A$

A8. $\vdash \forall_\sigma\circ(A \supset B) \supset (\forall_\sigma\circ A \supset \forall_\sigma\circ B)$

A9. $\vdash \forall_\sigma\Box(A \supset B) \supset (\exists_\sigma\Box A \supset \exists_\sigma\Box B)$

A10. $\vdash \forall_\sigma\Box A \supset \forall_\sigma\circ A \wedge \forall_\sigma\circ\forall_\sigma\Box A$

A11. $\vdash \forall_\sigma\Box A \supset A \wedge \exists_\sigma\circ\exists_\sigma\Box A$

A12. $\vdash \forall_\sigma\Box(A \supset \forall_\sigma\circ B) \supset (A \supset \forall_\sigma\Box B)$

A13. $\vdash \forall_\sigma\Box(A \supset \exists_\sigma\circ B) \supset (A \supset \exists_\sigma\Box B)$

Os axiomas **A1–A4** e **A6** são adaptações da LTTR, restritos à sintaxe da LTCI. Os demais axiomas são do sistema $\mathcal{UB}$. Sua consistência é facilmente demonstrável. Por exemplo, o axioma **A1**:

$$\begin{aligned}\mathbf{C}_i(\forall_\sigma\Box A) = \mathbf{v} &\Leftrightarrow \mathbf{C}_k(A) = \mathbf{v}, \text{ onde } s_k = \mathit{primeiro}(\sigma^j), \text{ para todo } \sigma \in \Sigma \text{ tal que} \\ &\quad \mathit{primeiro}(\sigma) = s_i, \text{ para todo } j \geq 0 \\ &\Leftrightarrow \mathbf{C}_k(A) = \mathbf{f}, \text{ onde } s_k = \mathit{primeiro}(\sigma^j), \text{ para nenhum } \sigma \in \Sigma \text{ tal que} \\ &\quad \mathit{primeiro}(\sigma) = s_i, \text{ para nenhum } j \geq 0 \\ &\Leftrightarrow \mathbf{C}_i(\sim\exists_\sigma\Diamond \sim A) = \mathbf{v}\end{aligned}$$

A demonstração da consistência dos demais axiomas segue de forma semelhante. Os axiomas apresentados são numerosos porque o relacionamento entre os quantificadores de seqüência e os operadores temporais deve ser feito explicitamente para todas as combinações (a dupla formada por um quantificador de seqüência e um operador temporal pode ser considerada como um único operador modal para o qual se dá axiomas). Os axiomas **A1–A13** são, contudo, apenas o fundamento para o Σ_{LTCI}: as adições ao Σ_{LTCI} advindas da interpretação pretendida devem, portanto, ser consideradas.

Quando se restringe o conjunto de estruturas a conjuntos de conjuntos de intercalações (definição VI.2.5.4), todas as fórmulas válidas permanecem válidas, mas fórmulas adicionais agora se tornam válidas. Por isso, é desejável acrescentar ao sistema dedutivo suficiente axiomas e regras de inferência no sentido de obter um sistema consistente e

completo. KATZ *et alia* (1987) afirmam, entretanto, que a axiomatização completa é um problema aberto.

Algumas regras de inferência para o Σ_{LTCI} são:

R1. Se A é uma tautologia da Lp, então $\vdash A$.

R2. Se $\vdash A \supset B$ e $\vdash A$, então $\vdash B$.

R3. Se $\vdash A$, então $\vdash \forall_\sigma \Box A$.

A prova de consistência dessas regras segue à semelhança do Σ_{LTTR} (subseção V.2.8).

VI.3 A Especificação Formal de SDs com a LTCI

Uma das características da LTCI é a expressão de propriedades que usam a habilidade de escolher um caminho dentre todos os caminhos que correspondem a uma única execução. Deve ser lembrado que não há maneira de determinar exatamente qual destes caminhos realmente ocorreu, se é que algum ocorreu, desde que nenhum relógio global é utilizado. Portanto, tem-se a liberdade de escolher um caminho que satisfaça a especificação. Um critério para a correção distribuída sob uma especificação temporal pode ser, portanto, a existência de um caminho em toda execução que satisfaz a propriedade desejada. Para algumas propriedades, por exemplo, a correção total de um programa distribuído, este critério é suficiente, como será, mostrado adiante. Para outros objetivos, o uso de tal abordagem depende do modo de como o programa é abstraído.

A abordagem clássica na especificação de programas paralelos é: porque a velocidade relativa das instruções de máquina não é parte da especificação, um programa pode seguramente ser considerado satisfazendo sua especificação temporal se cada intercalação dos eventos concorrentes satisfizer a especificação. Isto é natural para afirmações na LTL que são válidas quando elas são satisfeitas por todos os caminhos possíveis sob qualquer execução. No contexto da LTCI, caminhos pertencentes à mesma execução são agrupados em uma sub-estrutura. Isto sugere investigar quando propriedades de um tipo "existencial" são naturais para programas distribuídos.

VI.3.1 Abordagens Alternativas à Correção

Há duas abordagens à correção: uma afirmando que qualquer propriedade deve ser testada para *todos* os possíveis conjuntos de intercalações de qualquer execução (essa abordagem é a seguida na LTL) e outra abordagem afirmando ser suficiente mostrar que uma propriedade é verdade para pelo menos um caminho de cada execução. Vamos assumir que a validade de $\exists_\sigma L$ é verificada em uma estrutura representando um certo programa, mas $\sim \forall_\sigma L$ também seja verificada, isto é, há um caminho (diferente) que não satisfaz L e é verdade que $\exists_\sigma L \wedge \sim \forall_\sigma L$ (isto é, usando equivalência elementar, $\sim (\exists_\sigma L \supset \forall_\sigma L)$). Se alguém pode inferir que $\sim (\exists_\sigma L \supset \forall_\sigma L)$, esse alguém deve saber a ordem relativa entre eventos não relacionados, significando que ele tem em algum sentido, um relógio global.

A razão para tal situação é geralmente devido a sub-especificação do sistema pela omissão da descrição dos eventos do observador da descrição do sistema. Este observador pode ser, por exemplo, um recurso compartilhado, uma variável compartilhada ou um humano que pode observar duas impressoras de dois processos diferentes. O observador então impõe uma ordenação total sobre os eventos que compõem os dois caminhos diferentes -- um que satisfaz L e o outro que não o faz. Quando o observador é feito parte do sistema, dois caminhos assim pertencem a execuções diferentes. Então o caminho indesejável é excluído pela quantificação universal implicada sobre todas as execuções ao invés de pelo quantificador universal local $\forall_\sigma$ sobre caminhos em um único conjunto de intercalações.

A discussão acima mostra que a especificação de um SD é muito sensível à inclusão ou ausência de observadores externos ao modelo. Sua inclusão impõe uma ordem entre eventos que não eram ordenados anteriormente. Um sistema no qual todos os observadores possíveis são incluídos através da descrição de seus eventos de *interface* é chamado um *sistema completamente abstraído* (KATZ *et alia*, 1987). Um critério para um SD deste tipo satisfazer uma especificação temporal feita em termos de uma seqüência intercalada é que, para toda execução, exista um caminho que a satisfaça.

A noção de "linearização" desempenha um importante papel no estudo de algoritmos paralelos e distribuídos. Linearização significa completar a ordenação parcial até uma ordenação total que a contenha. No estudo de algoritmos distribuídos e paralelos, é comum tomar cada uma das seqüências (que representam computações no modelo de

intercalações) e intercambiar eventos não relacionados.

VI.3.2 Um exemplo

Sejam dois processos P_1 e P_2 usando um banco de dados compartilhado, e por motivos de consistência, seja requerido que as transações feitas pelos processos não sejam sobrepostas. A propriedade de segurança (exclusão mútua)

$$(INÍCIO_1 \wedge INÍCIO_2) \supset \Box \sim (SC_1 \wedge SC_2)$$

é equivalente à propriedade LTCI":

$$\forall_\sigma[(INÍCIO_1 \wedge INÍCIO_2) \supset \Box \sim (SC_1 \wedge SC_2)]$$

Mas isso força um requisito estrito que é necessário somente quando o banco de dados não é parte do sistema em si e age como um observador de fora. Se o banco de dados for embutido como um processo no sistema ao qual processos diferentes possam referir-se por comandos de comunicação, pode-se usar a propriedade mais fraca:

$$\exists_\sigma[(INÍCIO_1 \wedge INÍCIO_2) \supset \Box \sim (SC_1 \wedge SC_2)]$$

afirmando ser suficiente linearizar os eventos de tal forma que eles se comportem como se a exclusão mútua ocorresse em cada execução. Isso pode ser útil caso algumas transações ou parte delas possam se sobrepor no tempo sem causar mudanças concorrentes ao banco de dados.

VI.4 Conclusões

Um arcabouço temporal para raciocinar sobre estados globais construídos de ordenações parciais foi sugerido. A motivação maior para a LTCI é expresar rigorosamente e provar em um formalismo uniforme propriedades anteriormente expressas informalmente. Um critério de correção que é bastante natural para lidar com ordenações parciais e é muito natural para a LTCI é a linearização: para cada ordenação parcial, escolhe-se uma única ordenação total que a contém para representar a computação.

Mesmo se conjuntos de intercalações não são diretamente necessários para expressar uma propriedade (por exemplo, correção total é tradicionalmente expressa em

LTL) pode ser conveniente mudar para a LTCI. Propriedades que derivam da ordenação parcial não são, algumas vezes, interessantes em si, mas são de ajuda como um estágio intermediário na prova de outras propriedades mais comuns, como a correção total. Um sistema dedutivo que faz uso da habilidade de raciocinar sobre linearizações de ordens parciais pode ser, então, usado.

Uma interessante propriedade é que a LTCI permite usar estados globais em provas e especificações, enquanto se pensa em termos de ordenações parciais (sempre que for conveniente).

O objetivo deste capítulo foi mostrar uma forma diferente para o arcabouço da lógica temporal. Isso implica numa maneira diferente de modelar SDs e de uso da lógica temporal como ferramenta. Outras conclusões e comentários que podem ser feitos são mostrados nas subseções seguintes.

VI.4.1 Sobre a Axiomatização da LTCI

Devido a restrição sintática da LTCI (um quantificador de seqüência é seguido por um, e somente um, operador temporal) os axiomas devem formalizar todas as combinações entre eles. Isso acarreta uma formalização mais extensa do que foi feita em capítulos anteriores. Também, os axiomas apresentados não apresentam o operador $\mathcal{U}$ porque no sistema $\mathcal{UB}$ esse operador não é usado e no Σ_{LTTR} a sintaxe dos axiomas que continham esse operador não era permitida pela LTCI.

Uma interessante questão acerca do Σ_{LTCI} é que a semântica de ordem parcial subjacente pode restringir ainda mais a classe de estruturas, por exemplo, permitindo somente um número fixo de processos. Para cada uma dessas restrições, pode ser necessário acrescentar axiomas e/ou regras de inferência ao sistema dedutivo.

VI.4.2 A Expressividade da LTCI

Agrupar os caminhos em conjuntos de intercalações permite afirmar propriedades que são baseadas na ordem parcial subjacente. Seja uma propriedade Q e dois programas. No primeiro programa, em cada execução há um estado global que satisfaz Q, enquanto

no segundo há execuções nas quais Q é verificado em alguns estados e outras execuções nas quais Q não é verificado em nenhum estado. Por exemplo, seja Q a propriedade afirmando que o valor da variável x é maior do que o valor da variável y. Agora, definimos a propriedade global S que diz "o programa está no estado global inicial e os valores das variáveis x e y são zero" (formalmente, $Q : x > y$ e $S : (INÍCIO) \wedge x = 0 \wedge y = 0$). A fórmula $P : S \supset \exists_{\sigma} \Diamond Q$ expressa a propriedade "em toda execução de um dado programa começando com um estado global no qual x e y são zero, existe um caminho que atinge o estado global no qual Q é verificado". P faz distinção entre estruturas de programas nas quais P é verificado e estruturas de programas nas quais P não é verificado, não existindo uma fórmula F na LTL ou LTTR que expresse tal propriedade. (KATZ *et alia* (1988) demonstram que, com respeito à ordem parcial da seqüencialidade na execução, dois programas particulares são distinguíveis. Essa distinção pode ser feita na LTCI, mas não é possível nas outras lógicas temporais)

VI.4.3 Quantificando a LTCI

Há propriedades que são exprimíveis na LTCI e não são exprimíveis em qualquer variante das lógicas temporais linear ou ramificada. Por outro lado, uma propriedade existencial sobre todas as computações possíveis, tal como "há uma intercalação de computação que atinge um estado global no qual..." não é exprimível na LTCI porque há, na semântica da LTCI, um quantificador implícito que diz "em toda execução (de ordem parcial)" (vide observação da seção VI.2.5). Contudo, não há obstáculos à adição de outro nível de quantificadores que permitissem expressar "para toda ordenação parcial" e "há uma ordenação parcial". KATZ *et alia* (1987) propõem essa extensão à LTCI, chamando-a de LTCIq (LTCI *quantificada*), dando sua sintaxe e semântica.

VI.5 Notas Bibliográficas

O capítulo foi escrito a partir do trabalho de KATZ *et alia* (1987), um trabalho recente publicado em versão preliminar. Foram usados outros trabalhos, como os de LAMPORT (1978 e 1985b), para completar a apresentação das idéias. Os capítulos III, IV e V também foram úteis para o desenvolvimento do assunto.

Os axiomas propostos para o Σ_{LTCI} são do sistema ***UB*** (BEN-ARI *et alii*, 1983) e da LTTR (capítulo V). Outras importantes referências da subseção sobre a axiomatização são os trabalhos de CLARKE *et alii* (1985 e 1986) e de EMERSON (1983), este último investigando questões concernentes a estruturas ramificadas, estruturas Abrahamson e suas propriedades.

Capítulo VII

Conhecimento, Crença e Tempo

Lógicas do conhecimento e crença têm sido usadas na descrição de SDs, bem como de outras situações da vida real, tais como protocolos de comunicação, sistemas criptográficos, jogos, economia, bases de conhecimento e programas inteligentes. Vários trabalhos vêm sendo feitos usando essas lógicas, por exemplo HALPERN *et alia* 1984a, 1984b e 1985.

Alguns autores (FAGIN *et alia*, 1985; KRAUS *et alia*, 1988; SHOHAM, 1988) propuseram a incorporação do tempo aos sistemas formais que lidam com conhecimento e/ou crença. Essas extensões são tentativas de capturar os aspectos dinâmicos do conhecimento e da crença.

O objetivo deste capítulo é apresentar pesquisas mais recentes no contexto da especificação formal e verificação de SDs com a lógica temporal. Deve ser salientado que, devido a relativa pouca idade dessas idéias e propostas — pode-se dizer que os conceitos aqui apresentados ainda estão sendo estabelecidos, há uma preocupação menor com o formalismo do que em capítulos anteriores.

O capítulo está organizado da seguinte maneira: na primeira seção, é dado uma breve introdução aos conceitos de crença, crença comum, conhecimento e conhecimento comum e como o tempo alteraria essas noções. Na segunda seção, é apresentada a lógica do conhecimento e crença (LCC — sem aspectos temporais) proposta por KRAUS *et alia* (1988): sua sintaxe, um modelo para a LCC, a semântica, alguns teoremas e uma axiomatização para essa lógica (o sistema formal Σ_{LCC}).

Na terceira seção, é apresentado o relacionamento entre o conhecimento, crença

e tempo, sendo dadas diferentes noções de crença e é introduzida a lógica temporal de conhecimento e crença (LTCC): sua sintaxe, a estrutura temporal de conhecimento e crença, sua semântica e um sistema formal (Σ_{LTCC}) para ela. Em seguida, é dado um exemplo de especificação formal e verificação de um SD com a LTCC, sendo provada uma interessante propriedade. Finalizando, as seções de conclusões e de notas bibliográficas.

VII.1 Introdução

As lógicas modais do conhecimento e da crença vêm sendo propostas como ferramentas para a descrição de SDs. Salienta-se a questão de qual seria o melhor conceito para a análise de tais situações: o conhecimento ou a crença. Pode-se considerar que a principal diferença entre conhecimento e crença é que quando se *conhece*[1] φ, então φ *é* verdadeiro, mas quando se *crê* em φ, então φ *pode ou não* ser verdade.

HALPERN *et alia* (1984b) sugerem que, para algumas aplicações, um bom sistema deve ser capaz de "falar" sobre crença *e* conhecimento. Deseja-se expressar afirmações como "o agente x crê em p e conhece q" ou "o agente x crê que, se ele não tem conhecimento de p, então q é verdadeiro", e assim por diante. KRAUS *et alia* (1988) propõem uma lógica para vários agentes, incorporando os conceitos de crença, crença comum, conhecimento e conhecimento comum. Essa lógica será aqui chamada de *Lógica de Conhecimento e Crença* (LCC).

A noção de crença comum é mais fraca do que a noção de conhecimento comum. Mesmo quando p é verdade e é crença comum, p pode não ser conhecimento comum. Uma interessante questão é a de como a crença e o conhecimento mudam com o tempo. Em algumas circunstâncias, não se pode dizer nada não-trivial sobre o efeito da passagem do tempo no conhecimento ou crença, mas em outras circunstâncias, gostaria-se de dizer que o conhecimento nunca é perdido e que crenças não mudam sem razão. Quando é discutido a relação entre crença e tempo, conclui-se, inevitavelmente, que há mais de uma noção de crença.

[1] Um problema surge quando da tradução de "know" para o português: desde que seu significado pode ser tanto "saber" como "conhecer" ou "ter conhecimento de", deve-se adequar sua tradução ao contexto, como na sentença "se o agente x não sabe p, então ele sabe que não sabe p". Se, nesse exemplo, "know" fosse traduzido como "conhece", o entendimento seria prejudicado.

VII.2 A Lógica de Conhecimento e Crença (LCC)

Nesta seção, definições formais da sintaxe e semântica da LCC são dadas. Seja Agentes $\stackrel{def}{=}$ $\{1, 2, \ldots, n\}$ o conjunto finito de participantes.

VII.2.1 Sintaxe

Aos símbolos da lógica proposicional (Lp) (itens 1–4 — vide apêndice A), são acrescentados os operadores modais de conhecimento e crença:

5. Operadores modais de conhecimento e crença: K_x (*o agente x conhece algo*), B_x (*o agente x crê em algo*), $\mathcal{K}$ (*conhecimento comum de algo*) e $\mathcal{B}$ (*crença comum em algo*);

A definição de fórmulas (atômicas e bem-formadas) é semelhante à Lp (vide apêndice A), acrescentando à definição de fórmulas bem-formadas o seguinte item:

e) se A é uma fórmula bem-formada, então K_xA, B_xA, $\mathcal{K}A$ e $\mathcal{B}A$ também são fórmulas bem-formadas.

Define-se os seguintes operadores como abreviações:

$$\mathcal{E}A \stackrel{def}{=} \bigwedge_{x \in \text{Agentes}} K_xA \quad \text{e} \quad \mathcal{F}A \stackrel{def}{=} \bigwedge_{x \in \text{Agentes}} B_xA$$

isto é, a fórmula $\mathcal{E}p$ significa que todos os agentes conhecem p e $\mathcal{F}p$ significa que todos os agentes crêem em p. Os operadores modais $K_x, B_x, \mathcal{E}, \mathcal{F}, \mathcal{K}$ e $\mathcal{B}$ têm maior prioridade que os demais conectivos.

VII.2.2 Um Modelo para a LCC

Na LCC, usa-se a semântica de mundos possíveis para conhecimento e crença. Segundo esse modelo, o agente x tem conhecimento de p se p é verdade em todos os mundos que, de acordo com seu conhecimento, poderiam ser possíveis e o agente x crê em p se p é verdade em todos os mundos que poderiam ser possíveis conforme suas crenças. Define-se

um modelo no estilo de Kripke com duas relações binárias para cada agente: uma relação Γ correspondendo aos mundos possíveis de acordo com o conhecimento e uma relação Λ correspondendo aos mundos possíveis de acordo com a crença.

Um modelo **M** para a $\mathcal{L}_{LCC}$ é a tupla

$$\mathbf{M} = (\mathbf{E}, s, \Gamma_1, \ldots, \Gamma_n, \Lambda_1, \ldots, \Lambda_n)$$

onde:

- $\mathbf{E} = \{s_0, s_1, s_2, \ldots\}$ é o conjunto de todos os estados ou mundos possíveis. Como na LTLp (capítulo IV), cada estado $s_i \in \mathbf{E}$ é uma atribuição dos valores $\{\mathrm{v}, \mathrm{f}\}$ às fórmulas atômicas;
- $s \in \mathbf{E}$ é o estado real do mundo;
- $\Gamma_x, x \in$ Agentes é uma relação de equivalência (reflexiva, transitiva e simétrica) em **E**;
- $\Lambda_x, x \in$ Agentes é uma relação em **E** possuindo as seguintes características:
  1. Λ_x é serial, isto é, para todo $s_i \in \mathbf{E}$, há algum $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in \Lambda_x$;
  2. Λ_x está contida em Γ_x, isto é, $\Lambda_x \subseteq \Gamma_x$;
  3. Para quaisquer $s_i, s_j, s_k \in \mathbf{E}$, se $(s_i, s_j) \in \Gamma_x$ e $(s_j, s_k) \in \Lambda_x$, então $(s_i, s_k) \in \Gamma_x$.

O significado pretendido de Γ_x é que $(s_i, s_j) \in \Gamma_x$ se o conhecimento do agente x não o permitir distinguir entre s_i e s_j. Assim, a relação Γ_x divide os estados s_i de **E** em classes de equivalências $\mathbf{E}[s_i]$ e, se s_i é o "estado da mente" do agente x, então todos os $s' \in \mathbf{E}[s_i]$ são possíveis, de acordo com o conhecimento de x.

O significado pretendido de Λ_x é que $(s_i, s_j) \in \Lambda_x$ se, no estado s_i, o agente x crê que o estado s_j é um estado no qual o mundo poderia estar. Então $\Lambda_x \subseteq \Gamma_x$ porque é mais fácil crer em algo do que ter conhecimento de algo (só se tem conhecimento de coisas verdadeiras). Portanto, as crenças de alguém permitem-no distinguir entre mais estados do que seu conhecimento, e, conseqüentemente, pode haver alguns estados $s_i, s_j \in \mathbf{E}$ tais que $(s_i, s_j) \in \Gamma_x$ mas $(s_i, s_j) \notin \Lambda_x$. Por outro lado, crê-se naquilo que se tem conhecimento e, portanto, Λ_x está contida em Γ_x. O significado da terceira característica de Λ_x é que,

dados $s_i, s_j, s_k \in \mathbf{E}$, se s_i, s_j são possíveis conforme o conhecimento do agente x e quando ele está no estado s_j, ele crê que s_k é possível, então ele também crê que s_k é possível quando está em s_i, desde que ele não pode distinguir entre s_j e s_i.

São definidas duas outras relações binárias:

$$\Gamma^* \stackrel{\text{def}}{=} \left(\bigcup_{x \in \text{Agentes}} \Gamma_x \right)^+ \quad \text{e} \quad \mathrm{A}^* \stackrel{\text{def}}{=} \left(\bigcup_{x \in \text{Agentes}} \Lambda_x \right)^+$$

onde $^+$ representa o fecho transitivo.

VII.2.3 Semântica

Seja um modelo $\mathbf{M} = (\mathbf{E}, s, \Gamma_1, \ldots, \Gamma_n, \Lambda_1, \ldots, \Lambda_n)$. Para todo estado $s_i \in \mathbf{E}$, todo $i \in \mathbb{N}_0$ e toda fórmula F, define-se indutivamente o valor-verdade $\mathbf{M}_i(F) \in \{\mathbf{v}, \mathbf{f}\}$, informalmente significando "o valor-verdade de F no estado s_i":

1. $\mathbf{M}_i(v) = s_i(v)$ para $v \in \mathcal{V}$. Informalmente quer dizer "o valor-verdade de v no estado s_i é dado pela atribuição que s_i fizer a v";
2. $\mathbf{M}_i(\sim A) = \mathbf{v}$ se, e somente se, $\mathbf{M}_i(A) = \mathbf{f}$;
3. $\mathbf{M}_i(A \wedge B) = \mathbf{v}$ se, e somente se, $\mathbf{M}_i(A) = \mathbf{v}$ e $\mathbf{M}_i(B) = \mathbf{v}$;
4. $\mathbf{M}_i(K_x A) = \mathbf{v}$ se, e somente se, $\mathbf{M}_j(A) = \mathbf{v}$, para todo $j \in \mathbb{N}_0$ tal que $(s_i, s_j) \in \Gamma_x$;
5. $\mathbf{M}_i(B_x A) = \mathbf{v}$ se, e somente se, $\mathbf{M}_j(A) = \mathbf{v}$, para todo $j \in \mathbb{N}_0$ tal que $(s_i, s_j) \in \Lambda_x$;
6. $\mathbf{M}_i(\mathcal{K} A) = \mathbf{v}$ se, e somente se, $\mathbf{M}_j(A) = \mathbf{v}$, para todo $j \in \mathbb{N}_0$ tal que $(s_i, s_j) \in \Gamma^*$;
7. $\mathbf{M}_i(\mathcal{B} A) = \mathbf{v}$ se, e somente se, $\mathbf{M}_j(A) = \mathbf{v}$, para todo $j \in \mathbb{N}_0$ tal que $(s_i, s_j) \in \mathrm{A}^*$;

Observação: os demais conectivos $(\vee, \supset, \equiv)$ são definidos à semelhança da Lp (vide apêndice A).

VII.2.4 O Sistema Formal Σ_{LCC}

Chama-se de Σ_{LCC} o sistema abaixo que formaliza a LCC.

Axiomas

Os esquemas de axiomas que constituem a base do Σ_{LCC} são:

A1. $K_x(A \supset B) \supset (K_xA \supset K_xB)$ para todo $x \in$ Agentes;

A2. $K_xA \supset A$ para todo $x \in$ Agentes;

A3. $\sim K_xA \supset K_x \sim K_xA$ para todo $x \in$ Agentes;

A4. $\mathcal{K}(A \supset B) \supset (\mathcal{K}A \supset \mathcal{K}B)$;

A5. $\mathcal{K}A \supset K_xA$ para todo $x \in$ Agentes;

A6. $\mathcal{K}A \supset K_x\mathcal{K}A$ para todo $x \in$ Agentes;

A7. $\mathcal{K}(A \supset \mathcal{E}A) \supset (A \supset \mathcal{K}A)$;

A8. $B_x(A \supset B) \supset (B_xA \supset B_xB)$ para todo $x \in$ Agentes;

A9. $\sim B_xf$ para todo $x \in$ Agentes;

A10. $\mathcal{B}(A \supset B) \supset (\mathcal{B}A \supset \mathcal{B}B)$;

A11. $\mathcal{B}A \supset \mathcal{F}A$;

A12. $\mathcal{B}A \supset \mathcal{F}\mathcal{B}A$;

A13. $\mathcal{B}(A \supset \mathcal{F}A) \supset (\mathcal{F}A \supset \mathcal{B}A)$;

A14. $K_xA \supset B_xA$ para todo $x \in$ Agentes;

A15. $B_xA \supset K_xB_xA$ para todo $x \in$ Agentes;

A16. $\mathcal{K}A \supset \mathcal{B}A$;

Os axiomas **A1–A7** são referentes aos operadores de conhecimento e conhecimento comum. **A1** afirma a distributividade de K sobre a implicação lógica: se alguém sabe que A implica em B, então, se esse alguém sabe A, então ele sabe B. **A2** afirma que só se sabe coisas que são verdadeiras — este axioma é o que distingue *conhecimento* de *crença*: não se pode conhecer um fato que é falso, embora se possa crer nele. **A3** é um axioma de *introspecção*: o agente x sabe que ele não sabe. **A4** é a versão do axioma **A1**

para o conhecimento comum. **A5** afirma que, se um fato é conhecimento comum, então este fato é conhecido por cada um dos agentes. **A6** diz que, se um fato é conhecimento comum, então cada um dos agentes sabe que esse fato é conhecimento comum. **A7** afirma que, se é conhecimento comum que se A for verificado então A é conhecido por todas os agentes, então se A for verificado então A é conhecimento comum.

Os axiomas **A8–A13** dizem respeito aos operadores de crença e crença comum. **A8** afirma que, se o agente x crê que A implica em B, então, se ele crê em A, então ele crê em B. **A9** afirma que não se crê em coisas contraditórias. **A10** é a versão do **A8** para o operador de crença comum. **A11** diz que, se alguma coisa é crença comum, então todos crêem nisso. **A12** diz que, se algo é crença comum, então todos crêem que é crença comum. **A13** é uma regra de indução para a crença comum.

Os axiomas **A14–A16** descrevem a interrelação entre conhecimento e crença e conhecimento comum e crença comum. **A14** afirma que se crê em tudo que se conhece. **A15** diz que o agente x conhece suas crença: crenças são conscientes. **A16** diz que o conhecimento comum é crença comum.

Regras de Inferência

As regras de inferência do Σ_{LCC} são:

R1. Se A é uma tautologia da Lp então $\vdash A$ (*Tautologia Proposicional — TP*);

R2. Se $\vdash A$ e $\vdash A \supset B$, então $\vdash B$ (*Modus Ponens — MP*);

R3. Se $\vdash A$, então $\vdash \mathcal{K}A$ (*Generalização do Conhecimento Comum — G$\mathcal{K}$*);

A regra **R1** traz para a LCC todas as tautologias da Lp (vide apêndice A). **R3** é a regra de introdução do operador $\mathcal{K}$ de conhecimento comum.

Uma regra derivada que será usada no exemplo adiante é a regra de introdução do operador de crença comum $\mathcal{B}$. Se $\vdash A$ então $\vdash \mathcal{K}A$, por **R3**; $\vdash \mathcal{K}A \supset \mathcal{B}A$ é o axioma **A16** e, por MP, temos que $\vdash \mathcal{B}A$. Resumindo, se $\vdash A$, então $\vdash \mathcal{B}A$ é uma regra derivada, chamada de *Generalização da Crença Comum — G$\mathcal{B}$*.

Alguns teoremas que podem ser provados pelo Σ_{LCC} (KRAUS *et alia*, 1988) são:

T1. $K_x \sim A \supset \sim B_x A$ para todo $x \in$ Agentes;

T2. $B_x A \equiv K_x B_x A$ para todo $x \in$ Agentes;

T3. $\sim B_x A \equiv K_x \sim B_x A$ para todo $x \in$ Agentes;

T4. $K_x A \equiv B_x K_x A$ para todo $x \in$ Agentes;

T5. $\sim K_x A \equiv B_x \sim K_x A$ para todo $x \in$ Agentes;

T6. $B_x A \equiv B_x B_x A$ e $K_x A \equiv K_x K_x A$ para todo $x \in$ Agentes;

T7. $\sim B_x A \equiv B_x \sim B_x A$ para todo $x \in$ Agentes;

T8. $\mathcal{B} A \equiv \mathcal{B}\mathcal{B} A$;

T9. $\mathcal{K}(A \wedge B) \equiv \mathcal{K}A \wedge \mathcal{K}B$;

T10. $\mathcal{B}(A \wedge B) \equiv \mathcal{B}A \wedge \mathcal{B}B$;

VII.2.5 Consistência e Completeza do Σ_{LCC}

Pelas propriedades de Γ e Λ mostradas anteriormente, pode-se demonstrar facilmente que o Σ_{LCC} é consistente. KRAUS *et alia* (1988) provam a completeza do Σ_{LCC}.

VII.3 Conhecimento, Crença e Tempo

Pode-se estender a LCC adicionando uma relação e operadores modais (temporais) correspondente para capturar o conceito de tempo. Uma vez que o tempo é considerado, é possível investigar o que acontece quando são impostas algumas restrições ao relacionamento entre conhecimento, crença e tempo.

VII.3.1 As Diferentes Noções de Crença

Quando se fala sobre como crenças e conhecimentos mudam no tempo, deve-se distinguir entre, pelo menos, duas noções diferentes de crença. Primeiro, crença pode significar

"presteza em fazer uma aposta". "O agente x crê que choverá esta tarde" significa, operacionalmente, que ele levará consigo, de manhã, seu guarda-chuva. Se não chover, não há problema: o agente será lentamente convencido de que terá de levar de volta seu guarda-chuva sem usá-lo.

Um segundo significado é aquele no qual a crença é uma questão muito mais séria: não se pode permitir que a realidade contradiga as crenças de alguém, pois isso traria grandes perdas ou graves conseqüências, como, por exemplo, o caso de um investidor que crê na valorização de determinadas ações e, acontecendo o contrário, tem uma grande perda financeira, ou o caso de um nó em um SD que "confia" no recebimento de todas as suas mensagens enviadas — a perda de uma mensagem pode colocar-lhe em sérias dificuldades. Nesse sentido, portanto, só é permitido crer em coisas que nunca podem, sob quaisquer circunstâncias, serem mostradas como sendo falsas. Um agente não pode crer que seu pára-quedas abrirá porque ele poderia vir a saber que sua crença era errônea e ele não quer expor-se a esse risco. Portanto, um agente pode crer em coisas que ele conhece (isto é, que são verdadeiras segundo seu conhecimento) ou em coisas que não podem ser provadas como falsas[2].

Falando sobre esse último significado de crença, se o agente x crê que alguma coisa será verdade no próximo estado, então deve ser o caso de que ele tem conhecimento que não descobrirá, no próximo estado, que está errado. A fórmula VII.1 parece razoável para essa interpretação.

$$B_x \circ A \supset K_x \circ \sim K_x \sim A \text{ para todo } x \in \text{Agentes} \qquad \text{(VII.1)}$$

Se o agente x crê que, no próximo estado, A será verdadeiro, desde que ele não pode descobrir no próximo estado que está errado, ele tem todas as razões para persistir em suas crenças, isto é,

$$B_x \circ A \supset \circ B_x A \text{ para todo } x \in \text{Agentes} \qquad \text{(VII.2)}$$

Se VII.2 for aceito, pode-se aceitar

$$B \circ A \supset \circ B A \qquad \text{(VII.3)}$$

Pode-se notar que VII.1 é derivável de VII.2.

[2]Esse significado de crença está, talvez, mais próximo do significado de crenças religiosas.

No dia-a-dia, o verbo "crer" é, provavelmente, usado com um significado do tipo "presteza em apostar". Um tratamento lógico, não-numérico e convincente de tal noção é problemático. Em tal interpretação de crença, as fórmulas VII.2 e VII.3 não parecem razoáveis. Alguém pode crer (nesse significado "fraco") que choverá amanhã, mas amanhã ele não crerá mais nisso. Entretanto, se alguém acredita que amanhã A é verdadeiro, esse alguém acredita (hoje) que amanhã ele continuará a acreditar em A. Mais formalmente,

$$B_x \circ A \supset B_x \circ B_x A \text{ para todo } x \in \text{Agentes} \qquad \text{(VII.4)}$$

Contudo, tal fórmula nada diz sobre o futuro, sendo necessário alguma fórmula mais expressiva para capturar o que acontece às crenças de alguém no decorrer do tempo. Ninguém gosta de mudar suas crenças, e alguém só as mudará se for forçado a tal.

Por exemplo, alguém que colocou água para ferver, ligando o fogo, crê que o fogo está aceso e aquecendo a água. Ele continuará a crer nisso até que alguém o diga que o fogo apagou. Então ele pára de crer que a água irá ferver e tem conhecimento do oposto. Deseja-se dizer que, se o agente x crê em alguma coisa, ele continuará a crer nisso até que tenha conhecimento de que é falso. Uma maneira de dizer isso é:

$$B_x \circ A \supset \circ B_x A \vee \circ K_x \sim A \text{ para todo } x \in \text{Agentes} \qquad \text{(VII.5)}$$

A fórmula VII.6 é derivável de VII.5, mas mais fraca:

$$B_x \circ A \supset \circ B_x A \vee \circ B_x \sim A \text{ para todo } x \in \text{Agentes} \qquad \text{(VII.6)}$$

O significado dessa última fórmula é que alguém pára de crer em algo quando alguém *crer* que este algo é falso. Por exemplo, a pessoa do exemplo anterior *sabe* que o fogo apagou? Ele não viu o fogo apagado e, talvez, seu amigo estivesse brincando.

VII.3.2 A Lógica Temporal de Conhecimento e Crença

A extensão da lógica do conhecimento e crença (LCC) é aqui chamada de *Lógica Temporal de Conhecimento e Crença* (LTCC). Essa extensão foi inspirada em um problema que se desejava especificar formalmente e em idéias intuitivas acerca do relacionamento entre conhecimento, crença e tempo. Portanto, longe de ser a palavra final sobre o assunto, a LTCC é apenas *uma* maneira de se formalizar o raciocínio envolvendo aspectos dinâmicos do conhecimento e crença.

VII.3.2.1 Sintaxe

Aos símbolos da LCC, acrescenta-se os operadores temporais:

6. Operadores temporais: $\Box$ (*sempre* ou *doravante*), $\Diamond$ (*eventualmente*), $\circ$ (*próximo*) e $\mathcal{U}$ (*até*);

Na definição das fórmulas da LCC, acrescenta-se às fórmulas bem-formadas o seguinte item:

e) se A e B são fórmulas bem-formadas, então $\Box A$, $\Diamond A$, $\circ A$ e $A \mathcal{U} B$ também são fórmulas bem-formadas.

Observações:

i) Uma formalização alternativa para a sintaxe da LTCC é obtida considerando-se sobre a LTLp (vide capítulo IV) as extensões da LCC feitas sobre a Lp na seção anterior;

ii) Os operadores modais de conhecimento e crença possuem prioridade sobre todos os demais operadores e conectivos;

iii) Na sintaxe acima, pode-se expressar fórmulas puras de conhecimento, fórmulas puras de crença e fórmulas puras temporais e também pode-se expressar fórmulas relacionando conhecimento, crença e tempo (mistas) como $K_x \Box p \supset B_x \Box p$.

VII.3.2.2 Estrutura Temporal de Conhecimento e Crença

Pode-se obter uma estrutura para a semântica da LTCC adicionando à estrutura de conhecimento e crença $\mathbf{C}$ da LCC a relação de acessibilidade imediata ρ da LTL (vide capítulo IV). Mais precisamente, a estrura temporal de conhecimento e crença $\mathbf{C}^{\mathbf{T}}$ seria a tupla

$$\mathbf{C}^{\mathbf{T}} = (\mathbf{E}, s, \Gamma_1, \ldots, \Gamma_n, \Lambda_1, \ldots, \Lambda_n, \rho).$$

A relação ρ define uma ordenação total em $\mathbf{E}$ e, para cada $s' \in \mathbf{E}$, é definido, para cada Γ_x (Λ_x), $x \in$ Agentes, uma classe de equivalência representando o conhecimento (a

crença) do agente naquele instante (seu "estado da mente"); cada classe de equivalência é, conseqüentemente, totalmente ordenada por ρ. A relação ρ é serial, restringindo o tempo à sua forma linear. Define-se, também, uma relação R como sendo o fecho reflexivo e transitivo de ρ, isto é, R é uma relação binária em $\mathbf{E}$ definida da seguinte maneira: $(s,t) \in R$ se, e somente se, existir estados $s_0, \ldots, s_k$ tais que $s = s_0$, $t = s_k$ e $(s_i, s_{i+1}) \in \rho$, para $i < k$. A relação R é a mesma de capítulos anteriores (seção III.2.3 e capítulo IV) — foi dada apenas uma definição alternativa.

A estrutura temporal de conhecimento e crença $\mathbf{C^T}$ pode ser representada por um feixe de retas paralelas no qual cada reta representa a composição das relações ρ e Γ_x (Λ_x). A figura VII.1 ilustra essa idéia. Na figura, a primeira reta (de cima para

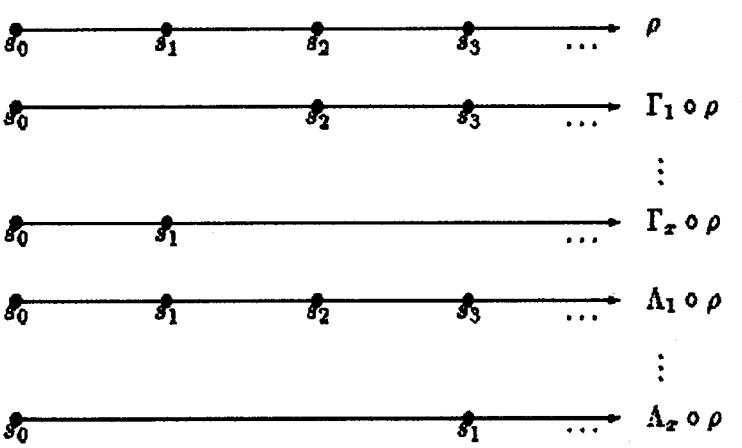

Figura VII.1: Representação de uma estrutura temporal de conhecimento e crença

baixo) simboliza o conjunto $\mathbf{E}$ ordenado pela relação ρ. A reta abaixo desta representa um exemplo de uma classe de equivalência para o estado s_0 e para a relação Γ_1, ordenada totalmente por ρ. As outras retas ilustram os demais casos. Intuitivamente, cada uma dessas retas representa o desenvolvimento do conhecimento (da crença) de cada agente no tempo.

São impostas algumas restrições à composição das relações ρ e Γ_x (Λ_x) visando limitar, de alguma forma, os modelos para a LTCC. Deseja-se, por exemplo, que a estrutura $\mathbf{C^T}$ incorpore o conceito de "não-esquecimento", isto é, que o conhecimento de um agente

não se perde. Para isso, restringe-se a interação entre o conhecimento e o tempo, de forma que esse conceito seja capturado. Em termos das relações Γ_x e ρ, isso significa que em um dado instante, o conjunto de mundos que um agente pensa (agora) que poderia possivelmente descrever a situação de amanhã contém o conjunto de mundos que ele atualmente pensa ser possível amanhã. Mais formalmente,

$$\rho \circ \Gamma_x \subseteq \Gamma_x \circ \rho \tag{VII.7}$$

Também deseja-se que a fórmula VII.3 seja válida na estrutura $\mathbf{C}^{\mathbf{T}}$, formalizando o conceito de crenças "mais sérias" (vide discussão da seção VII.3.1). Em termos das relações Λ_x e ρ, pode-se formalizar essa restrição da seguinte forma:

$$\rho \circ \Lambda_x \subseteq \Lambda_x \circ \rho \tag{VII.8}$$

Essas restrições serão sentidas na axiomatização da LTCC, pois elas irão determinar os axiomas que serão válidos nas estruturas temporais de conhecimento e crença.

VII.3.2.3 Semântica

Para toda estrutura temporal de conhecimento e crença $\mathbf{C}^{\mathbf{T}}$, todos os estados $s_i \in \mathbf{E}$, $i, j \in \mathbb{N}_0$ e toda fórmula F, define-se indutivamente o valor-verdade $\mathbf{C}_i^{\mathbf{T}}(F) \in \{\mathbf{v}, \mathbf{f}\}$, informalmente significando "o valor-verdade de F no estado s_i":

1. $\mathbf{C}_i^{\mathbf{T}}(v) = s_i(v)$ para $v \in \mathcal{V}$. Informalmente quer dizer "o valor-verdade de v no estado s_i é dado pela atribuição que s_i fizer a v";
2. $\mathbf{C}_i^{\mathbf{T}}(\sim A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_i^{\mathbf{T}}(A) = \mathbf{f}$;
3. $\mathbf{C}_i^{\mathbf{T}}(A \wedge B) = \mathbf{v}$ se, e somente se, $\mathbf{C}_i^{\mathbf{T}}(A) = \mathbf{v}$ e $\mathbf{C}_i^{\mathbf{T}}(B) = \mathbf{v}$;
4. $\mathbf{C}_i^{\mathbf{T}}(\Box A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para todo $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in R$;
5. $\mathbf{C}_i^{\mathbf{T}}(\Diamond A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para algum $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in R$;
6. $\mathbf{C}_i^{\mathbf{T}}(\circ A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in \rho$;
7. $\mathbf{C}_i^{\mathbf{T}}(A \mathcal{U} B) = \mathbf{v}$ se, e somente se, $\mathbf{C}_k^{\mathbf{T}}(B) = \mathbf{v}$, para algum $s_k \in \mathbf{E}$ tal que $(s_i, s_k) \in R$ e $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para todo $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in R$ e $(s_j, s_k) \in R$;

8. $\mathbf{C}_i^{\mathbf{T}}(K_x A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para todo $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in \Gamma_x$;

9. $\mathbf{C}_i^{\mathbf{T}}(B_x A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para todo $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in \Lambda_x$;

10. $\mathbf{C}_i^{\mathbf{T}}(\mathcal{K} A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para todo $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in \Gamma^*$;

11. $\mathbf{C}_i^{\mathbf{T}}(\mathcal{B} A) = \mathbf{v}$ se, e somente se, $\mathbf{C}_j^{\mathbf{T}}(A) = \mathbf{v}$, para todo $s_j \in \mathbf{E}$ tal que $(s_i, s_j) \in \Lambda^*$;

Alguns exemplos de fórmulas da LTCC e seus significados intuitivos são:

$K_x \circ p$: "o agente x sabe que, no próximo estado, p é verdade";

$B_x p \supset B_x \Box B_x p$: "se o agente x crê em p, então ele crê que sempre crerá em p";

$B_x p \mathcal{U} K_x \sim p$: "o agente x crê que p é verdade até que ele saiba que p é falso";

$\mathcal{K} \Box \Diamond p$: "é conhecimento comum que p é verdade infinitamente freqüentemente";

VII.3.2.4 O Sistema Formal Σ_{LTCC}

O sistema aqui utilizado para formalizar a LTCC será chamado de Σ_{LTCC} e representa *uma* tentativa de axiomatizar essa lógica. O Σ_{LTCC} é uma extensão do Σ_{LCC} (seção VII.2.4), sendo acrescidos alguns axiomas que manipulam os operadores de conhecimento e crença e os operadores temporais. Esses novos axiomas refletem as restrições impostas às estruturas temporais de conhecimento e crença, sendo escolhidos visando reunir propostas de diferentes autores (por exemplo, KRAUS *et alia*, (1988) e FAGIN *et alia*, (1985)) e também com o intuito de serem utilizados na verificação de uma especificação formal (seção VII.4).

Axiomas

Aos axiomas **A1–A16** da LCC acrescenta-se:

A17. $K_x \circ A \supset \circ K_x A$;

A18. $\mathcal{B} \circ A \supset \circ \mathcal{B} A$

Os axiomas **A17** e **A18** são os correspondentes sintáticos das restrições semânticas feitas às estruturas $\mathbf{C}^{\mathbf{T}}$ (fórmulas VII.7 e VII.8, respectivamente). O axioma **A17** afirma que, se o agente x tem conhecimento (agora) de que, no próximo instante, A será verificado, então, no próximo estado, o agente x sabe que A é verificado. **A18** afirma que, se é crença comum que no próximo estado A é verificado, então, no próximo estado, A é crença comum.

É fácil ver que os axiomas **A1–A16** continuam consistentes no Σ_{LTCC}. Os axiomas **A17** e **A18** são obviamente consistentes.

Regras de Inferência

São sugeridas duas regras de inferência a serem acrescentadas às regras **R1–R3** do Σ_{LCC}:

R4. Se A é uma fórmula válida da LTLp, então $\vdash A$ (*Validade Temporal — VT*);

R5. Se $\vdash A$, então $\vdash \Box A$ (*Inserção do* $\Box$ *— I*$\Box$);

Pode-se ver que as regras **R4** e **R5** preservam a validade em estruturas $\mathbf{C}^{\mathbf{T}}$. A regra **R4** traz para a LTCC todas as fórmulas válidas da LTLp (vide capítulo IV). **R5** introduz o operador $\Box$ em todas as fórmulas válidas da LTCC: se uma fórmula é verificada em uma estrutura $\mathbf{C}^{\mathbf{T}}$ ($\vdash A$), ela deve ser verificada em cada um de seus estados ($\vdash \Box A$).

A regra **R4** foi sugerida porque seria interessante poder dispor de todas as fórmulas válidas da LTLp. Isso aumentaria o repertório das fórmulas sobre as quais se poderia raciocinar, acrescentando à onisciência dos agentes as fórmulas temporais válidas. **R5** é uma conseqüência de se afirmar que uma fórmula é verificada em uma estrutura Kripke (como é a estrutura $\mathbf{C}^{\mathbf{T}}$).

VII.3.2.5 Consistência e Completeza do Σ_{LTCC}

Apesar de não ter sido formalmente demonstrado, pode-se ver pela definição da estrutura $\mathbf{C}^{\mathbf{T}}$ que o sistema Σ_{LTCC} é consistente. Não houve, na proposta do Σ_{LTCC}, preocupação com a completeza e na bibliografia pesquisada essa questão não é abordada.

VII.4 A Especificação Formal de SDs com a LTCC

Seja o SD como mostrado na figura VII.2 no qual três processos $P_i, i = 1, \ldots, 3$, devem utilizar (cada um) um dos recursos W ou B. O processo S é encarregado de alocar os recursos aos processos, observando que até dois processos podem usar W e até três processos podem usar B. O sistema funciona da seguinte maneira: cada processo solicita

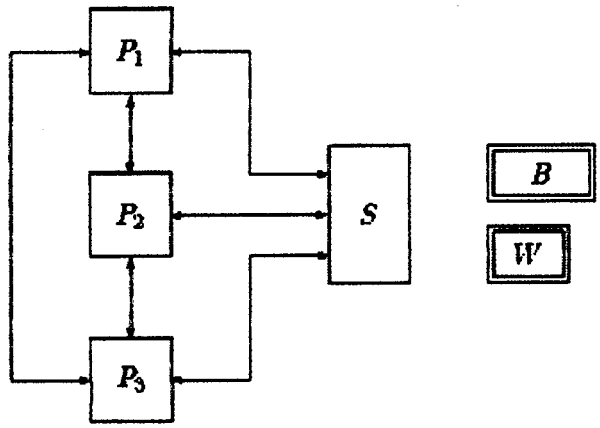

Figura VII.2: SD composto por 4 processos e 2 recursos

a S para ser conectado a um dos recursos W ou B; o processo S, obedecendo à restrição do número de usuários de cada recurso, conecta o processo ao recurso, ficando o processo, doravante, ligado ao recurso. O processo S liga os processos aos recursos segundo critérios que não importam para o exemplo.

No exemplo, é suposto que, após S conectar todos os processos, há uma "queda" no sistema e S perca a informação de quem foi conectado a cada recurso, havendo urgência dessa informação para o prosseguimento do funcionamento do SD. Então cada processo $P_i, i = 1, \ldots, 3$ deve enviar uma mensagem para S informando a qual recurso está conectado, e para isso ele pode "ver" a qual recurso os outros processos estão conectados, porém não há como saber diretamente a qual recurso ele próprio está conectado. Os processos podem trocar mensagens entre si, trocando crenças, de forma que seja informado a S, no menor tempo possível, a qual recurso cada processo está conectado.

Neste exemplo, é interessante para o presente trabalho o tipo de raciocínio usado pelos processos $P_i, i = 1, \ldots, 3$ e pelo processo S, bem como a lista de hipóteses "escondidas" no enunciado. A noção de crença comum se mostrará útil para os dois aspectos.

Para a descrição das hipóteses (escondidas ou não) e do "raciocínio" dos processos serão usadas as seguintes variáveis proposicionais básicas:

- $b_i, i = 1, \ldots, 3$ significa "o processo P_i está conectado agora ao recurso B";
- $w_i, i = 1, \ldots, 3$ significa "o processo P_i está conectado agora ao recurso W";
- $d_i, i = 1, \ldots, 3$ significa "o processo P_i envia agora uma mensagem para S informando a qual recurso ele está conectado".

Serão usadas, também, as seguintes notações:

$$\alpha_1 \stackrel{\text{def}}{=} \bigvee_{i=1,\ldots,3} b_i,$$

$$\alpha_2 \stackrel{\text{def}}{=} (b_1 \vee b_2) \wedge (b_2 \vee b_3) \wedge (b_3 \vee b_1),$$

$$\alpha_3 \stackrel{\text{def}}{=} \bigwedge_{i=1,\ldots,3} b_i$$

A fórmula α_k significa, portanto, que há pelo menos k processos usando o recurso B. Nas hipóteses e na demonstração seguintes, os subscritos i, j, k referem-se aos processos $P_i, i = 1, \ldots, 3$. Quando aparecer mais de um deles em uma mesma fórmula, será assumido que $i, j, k \in \{1, 2, 3\}, i \neq j, j \neq k, k \neq i$.

Hipóteses

São descritas, agora as hipóteses do problema.

- *Hipótese 1*: É crença comum que cada processo está conectado ao recurso B ou W. A fórmula h_1 pode representar essa hipótese:

$$h_1 : \mathcal{B}\Box(\sim b_i \equiv w_i)$$

- *Hipótese 2*: É crença comum que os processos não mudam os recursos a que estão conectados:

$$h_2 : \mathcal{B}\Box(b_i \equiv \circ b_i) \wedge \mathcal{B}\Box(w_i \equiv \circ w_i)$$

- *Hipótese 3*: É crença comum que cada processo pode "ver" o recurso ao qual está conectado *outro* processo, isto é, todos os processos crêem que podem ver a qual recurso os outros processos estão conectados:

$$h_3 : \mathcal{B}\Box(b_j \supset B_i b_j) \wedge \mathcal{B}\Box(w_j \supset B_i w_j)$$

- *Hipótese 4*: Um processo só informa a S que está conectado ao recurso B ou W quando ele crê que está conectado ao recurso B ou W, respectivamente. Para simplificar um pouco o problema, evitando o uso do operador $\mathcal{U}$ nas fórmulas, será assumido que, uma vez que um processo envie para S o nome do recurso ao qual ele esteja conectado, ele continuará enviando isso em todo instante subseqüente:

$$h_4 : \mathcal{B}\Box(B_i b_i \vee B_i w_i \equiv \circ d_i)$$

- *Hipótese 5*: O envio de uma mensagem para S pelo processo P_i ou, mais importante, o "não-envio" de uma mensagem para S, é suficientemente público para criar uma crença comum. Em outras palavras, se um processo envia ou não uma mensagem para S é imediatamente crença comum:

$$h_5 : \Box(\mathcal{B} d_i \equiv d_i) \wedge \Box(\mathcal{B} \sim d_i \equiv \sim d_i)$$

A hipótese 5 é a única que considera a realidade. Todas as outras assumiram certas crenças comuns.

- *Hipótese 6*: No estado real (s) do SD, é crença comum que pelo menos um processo está conectado ao recurso B:

$$h_6 : \mathcal{B}\alpha_1$$

Seja Φ a conjunção das hipóteses acima. Pode-se afirmar que, se nenhum processo enviar o nome de seu recurso para S no próximo instante ou no instante seguinte ao próximo instante, então todos os processos enviarão o nome de seus recursos no terceiro próximo instante. Formalmente,

$$\vdash \Phi \supset \left(\bigvee_{i=1,\ldots,3} \circ d_i \right) \vee \left(\bigvee_{i=1,\ldots,3} \circ\circ d_i \right) \vee \left(\bigwedge_{i=1,\ldots,3} \circ\circ\circ d_i \right)$$

Deve ser salientado que não se deseja provar nada sobre como os processos adquirem suas crenças ou quando os processos não enviam mensagens.

A Prova

A prova é feita em três partes principais. Primeiro, é provado que é crença comum que, no próximo estado, se é crença comum que nenhum processo enviou mensagem para S, então é crença comum que há pelo menos dois processos usando o recurso B. Na segunda parte, prova-se que é crença comum que, no próximo estado, se é crença comum que há pelo menos dois processos conectados a B, então se no segundo próximo estado ($\circ\circ$) é crença comum que nenhum processo enviou mensagem para S, então neste estado será crença comum que no próximo estado todos os processos enviarão mensagens para S.

Primeira Parte

$$\vdash h_3 \supset \mathcal{B}(w_j \supset B_i w_j) \qquad (1)$$

por TP e VT;

$$\vdash h_8 \wedge h_3 \wedge h_1 \supset \mathcal{B}(w_j \wedge w_k \supset B_i b_i) \qquad (2)$$

isso segue de (1). Agora tem-se

$$\vdash h_4 \wedge h_8 \wedge h_3 \wedge h_1 \supset \mathcal{B}\left(\sim\alpha_2 \supset \bigvee_{i=1\ldots,3} \circ d_i\right) \qquad (3)$$

por (2)

$$\vdash h_2 \supset \mathcal{B}(\alpha_2 \equiv \circ\alpha_2) \qquad (4)$$

$$\vdash h_2 \wedge h_4 \wedge h_8 \wedge h_3 \wedge h_1 \supset \mathcal{B}\circ\left(\left(\bigwedge_{i=1\ldots,3} \sim d_i\right) \supset \alpha_2\right) \qquad (5)$$

por (3) e (4).

$$\vdash h_2 \wedge h_4 \wedge h_8 \wedge h_3 \wedge h_1 \supset \circ\mathcal{B}\left(\left(\bigwedge_{i=1\ldots,3} \sim d_i\right) \supset \alpha_2\right) \qquad (6)$$

por (5) e **A18**.

$$\vdash h_2 \wedge h_4 \wedge h_8 \wedge h_3 \wedge h_1 \supset \circ\left(\left(\bigwedge_{i=1\ldots,3} \mathcal{B} \sim d_i\right) \supset \mathcal{B}\alpha_2\right) \qquad (7)$$

por (6), **A10** e **T10**.

$$\vdash h_2 \wedge h_4 \wedge h_8 \wedge h_3 \wedge h_1 \supset \mathcal{B}\circ\left(\left(\bigwedge_{i=1\ldots,3} \mathcal{B} \sim d_i\right) \supset \mathcal{B}\alpha_2\right) \qquad (8)$$

por (7), usando **T10**, **T8** e $G\mathcal{B}$, desde que todas as hipóteses começam por $\mathcal{B}$. Isto encerra a primeira parte da prova.

Segunda Parte

$$\vdash h_1 \supset \mathcal{B}(\alpha_2 \wedge w_j \supset b_i) \qquad (9)$$

por TP.

$$\vdash h_1 \supset \mathcal{B}(B_i\alpha_2 \wedge B_i w_j \supset B_i b_i) \qquad (10)$$

por (9), axiomas **A8** e **A11** e teoremas **T8** e **T10**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \supset \mathcal{B}(\mathcal{B}\alpha_2 \wedge w_j \supset \circ d_i) \qquad (11)$$

por (10), (1) e axioma **A11**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B}\left[\mathcal{B}\alpha_2 \supset \circ\left(\left(\bigvee_{i=1,\ldots,3} w_i\right) \supset \left(\bigvee_{i=1,\ldots,3} d_i\right)\right)\right] \qquad (12)$$

por (11).

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B}\mathcal{B}\alpha_2 \supset \mathcal{B}\circ\left(\left(\bigvee_{i=1,\ldots,3} w_i\right) \supset \left(\bigvee_{i=1,\ldots,3} d_i\right)\right) \qquad (13)$$

por (12) e axioma **A10**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B}\alpha_2 \supset \mathcal{B}\circ\left(\left(\bigwedge_{i=1,\ldots,3} \sim d_i\right) \supset \alpha_3\right) \qquad (14)$$

por (13) e teorema **T10**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B}\alpha_2 \supset \circ\left(\mathcal{B}\left(\bigwedge_{i=1,\ldots,3} \sim d_i\right) \supset \mathcal{B}\alpha_3\right) \qquad (15)$$

por (14) e axiomas **A18** e **A10**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B}\alpha_2 \supset \mathcal{B}\circ\left(\mathcal{B}\left(\bigwedge_{i=1,\ldots,3} \sim d_i\right) \supset \mathcal{B}\alpha_3\right) \qquad (16)$$

por (15), **T8** e **A10** e desde que todas as hipóteses começam por $\mathcal{B}$.

$$\vdash h_4 \supset \mathcal{B}\circ\left(\mathcal{B}\alpha_3 \supset \left(\bigwedge_{i=1,\ldots,3} \circ d_i\right)\right) \qquad (17)$$

pelo axioma **A11** e desde que todas as hipóteses começam por $\mathcal{B}\Box$.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B}\alpha_2 \supset \mathcal{B}\circ\left(\mathcal{B}\left(\bigwedge_{i=1,\ldots,3} \sim d_i\right) \supset \left(\bigwedge_{i=1,\ldots,3} \circ d_i\right)\right) \qquad (18)$$

por (17) e (16).

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B}\alpha_2 \supset \left(\circ \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \circ \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \circ d_i \right) \right) \tag{19}$$

por (18) e axiomas **A18**, **A10** e **T10**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B} \circ \left[\mathcal{B}\alpha_2 \supset \left(\circ \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \circ \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \circ d_i \right) \right) \right] \tag{20}$$

por (19) e desde que todas as hipóteses começam por $\mathcal{B}\Box$. Isso termina a segunda parte da prova.

Terceira Parte

Junta-se, agora, (8) e (20):

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \mathcal{B} \circ \left[\left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \left(\circ \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \circ \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \circ d_i \right) \right) \right] \tag{21}$$

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \circ \mathcal{B} \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \circ \mathcal{B} \circ \left(\left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \circ d_i \right) \right) \tag{22}$$

por (21) e axiomas **A18** e **A10**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \supset \circ \left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \circ\circ \left(\left(\bigwedge_{i=1,\ldots,3} \mathcal{B} \sim d_i \right) \supset \left(\bigwedge_{i=1,\ldots,3} \circ \mathcal{B} d_i \right) \right) \tag{23}$$

por (22) e axiomas **A18** e **A10** e teoremas **T9** e **T10**.

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \wedge h_5 \supset \left[\left(\bigwedge_{i=1,\ldots,3} \circ \sim d_i \right) \supset \left(\left(\bigwedge_{i=1,\ldots,3} \circ\circ \sim d_i \right) \supset \left(\bigwedge_{i=1,\ldots,3} \circ\circ\circ d_i \right) \right) \right] \tag{24}$$

por (23). A prova pode ser então concluída, obtendo

$$\vdash h_1 \wedge h_3 \wedge h_4 \wedge h_2 \wedge h_5 \supset \left(\left(\bigvee_{i=1,\ldots,3} \circ d_i \right) \vee \left(\bigvee_{i=1,\ldots,3} \circ\circ d_i \right) \vee \left(\bigwedge_{i=1,\ldots,3} \circ\circ\circ d_i \right) \right) \tag{25}$$

ou ainda

$$\vdash \Phi \supset \left(\left(\bigvee_{i=1,\ldots,3} \circ d_i \right) \vee \left(\bigvee_{i=1,\ldots,3} \circ\circ d_i \right) \vee \left(\bigwedge_{i=1,\ldots,3} \circ\circ\circ d_i \right) \right) \tag{26}$$

■

VII.5 Conclusões

Após a exposição feita, pode-se fazer as seguintes conclusões:

1. O interesante em unir conhecimento, crença e tempo é a capacidade de analisar a aquisição do conhecimento (e conhecimento comum) e da crença (e crença comum), bem como a mudança da crença (e crença comum) na presença de novas evidências. Essas questões foram deixadas de lado, só sendo referenciadas genericamente.

   Seria desejável um tratamento formal desses aspectos, incluindo axiomas que representassem essas novas verdades. Como exemplo, tem-se o caso da mudança de crenças dado pela fórmula $B_x\Box p \wedge K_x\Diamond \sim p \supset \circ \sim B_x\Box p$, ilustrando como alguém que antes cria na eternidade de p ($\Box p$) e tem conhecimento de que $\sim p$ é eventualmente verdade, então "perde" a crença inicial. No sistema formal apresentado (Σ_{LTCC}), investigações dessa natureza não são possíveis.

2. Na apresentação da LTCC, foi usada a LCC de KRAUS *et alia* (1988) porque eles davam um exemplo interessante de uso na especificação formal e verificação de SDs. Entretanto, poderia ter sido usada uma outra lógica de conhecimento/crença, como a de FAGIN *et alia* (1985). Outro ponto a favor da lógica de KRAUS *et alia* (1988) é que eles exploram os conceitos de crença e crença comum que são conceitos mais fracos do que o conhecimento e o conhecimento comum.

3. A falta de um tratamento formal no trabalho de KRAUS *et alia* (1988) para a incorporação do tempo ao conhecimento e à crença levou a uma investigação em outras bibliografias (FAGIN *et alia*, 1985; SHOHAM, 1988) e posterior proposta da estrutura temporal de conhecimento e crença. Entretanto, deve-se salientar que isso é apenas *um* dos modos de fazer tal relacionamento.

4. Incorporou-se o tempo ao conhecimento e à crença na sua forma linear. Contudo, nada impede de se usar o tempo em sua forma ramificada, a não ser um substancial aumento na complexidade das estruturas temporais e da semântica.

5. Os aspectos dinâmicos do conhecimento e da crença axiomatizados no Σ_{LTCC} (axiomas **A17** e **A18**) foram convenientes para o objetivo pretendido de exibir *um* sistema formal e de verificar o SD do exemplo. Entretanto, pode-se questionar a adequação

desses axiomas para outras situações. As pessoas esquecem, isto é, "perdem" conhecimento e mudam suas crenças na presença de fatos que as contradigam. A formalização desses conceitos, entretanto, parece ser problemática.

6. Teceu-se considerações informais sobre a consistência do Σ_{LTCC}, mas a sua completeza não foi investigada. FAGIN *et alia* (1985) discorrem sobre a questão de quais axiomas seriam melhor para uma lógica de conhecimento e tempo. Essa questão está aberta e o mesmo pode ser dito para a LTCC. Foi proposto um sistema formal que pareceu adequado para os objetivos que se tinha em mente (sugerir um sistema e usá-lo na verificação de uma especificação formal de um SD) — isto é apenas uma sugestão e não a palavra definitiva no assunto.

7. No estudo de SDs, é importante a idéia (conjunto de conhecimentos e crenças) que cada nó tem sobre o sistema como um todo. Essa idéia pode mudar a medida que novas mensagens são recebidas (ou deixadas de serem recebidas). A LTCC perece adequada para lidar com a especificação formal e verificação desses aspectos dinâmicos do conhecimento e crença em SDs.

8. No exemplo de especificação formal de SDs com a LTCC, usou-se apenas os operadores de crença comum $\mathcal{B}$ e de próximo estado $\circ$. Contudo, é possível ainda obter-se uma pequena amostra da complexidade das demonstrações: muitas vezes, entre dois passos subseqüentes havia muitos outros passos que foram abreviados e que, se fossem explicitados, aumentariam consideravelmente o tamanho da prova. O ganho com a expressividade tem a contrapartida do aumento de dificuldade das provas pois cresce o repertório e a complexidade das fórmulas aplicáveis nas demonstrações.

VII.6 Notas Bibliográficas

A primeira e segunda seção do capítulo foram escritas a partir do artigo de KRAUS *et alia* (1988). A terceira seção teve outras influências, como SHOHAM (1988) e FAGIN *et alia* (1985). A proposta da estrutura temporal de conhecimento e crença é adaptada de FAGIN *et alia* (1985). A figura VII.1 foi inspirada em SHOHAM (1988). A restrição VII.7 feita às estruturas $\mathbf{C}^{\mathbf{T}}$ e o axioma **A17** são de FAGIN *et alia* (1985). A restrição VII.8 e o axioma **A18** são de KRAUS *et alia* (1988). As regras de inferência são sugestões

que podem ser úteis no uso da LTCC. Não houve uma preocupação com a minimalidade do sistema e há um sentimento que a consistência, apesar de não ter sido formalmente provada, parece se verificar.

O exemplo de especificação formal de SDs com a LTCC é uma adaptação de um jogo denominado *jogo dos homens sábios* (*wise men puzzle*) aparecendo em KRAUS *et alia* (1988). A prova foi retirada da mesma referência.

As referências bibliográficas utilizadas sobre conhecimento e crença são HALPERN (1986), onde é feito uma revisão do estado da arte sobre conhecimento, HALPERN *et alia* (1984a), onde aplicou-se, originalmente, a lógica do conhecimento em SDs e o trabalho de HALPERN *et alia* (1985), onde pode ser encontrado os aspectos modais das lógicas de conhecimento e de crença. Em português, uma breve introdução à lógica do conhecimento e sua aplicação a SDs é encontrada em AMORIM *et alii* (1988). Alguns trabalhos (FAGIN *et alia*, 1985; ASHER, 1988) têm sido feitos considerando o tempo e o conhecimento/crença mas são de cunho essencialmente filosóficos ou abordam o tema de forma superficial. SHOHAM (1988) propõe uma lógica de ignorância cronológica, a *lógica de conhecimento temporal*. Entretanto, essa lógica temporal considera intervalos de tempo ao invés de estados e permite representar o conhecimento de fatos com aspectos temporais, e não representar aspectos dinâmicos do conhecimento.

Capítulo VIII

Automatizando a Lógica Temporal

Em capítulos anteriores, a lógica temporal foi utilizada para especificar e verificar SDs. Ambas as atividades foram feitas manualmente, sem a participação de qualquer recurso (quer fosse uma algoritmo, um programa ou um provador de teoremas) que as mecanizassem. É opinião de alguns gerentes de projeto, entretanto, que uma ferramenta para a especificação formal ou verificação só deva ser usada se esta oferecer recursos automatizados que dêem apoio ao processo de desenvolvimento de SDs (COHEN *et alii*, 1986). Por isso, alguns trabalhos vêm sendo feitos no sentido de mecanizar ou automatizar essas tarefas, pelo menos parcialmente.

VIII.1 Verificação Automática de SDs com a Lógica Temporal

Na abordagem tradicional da verificação de programas concorrentes, a prova que um programa satisfaz à sua especificação é construída manualmente usando os axiomas e as regras de inferência de um sistema dedutivo, tal como o Σ_{LTL_P} (capítulo IV) ou Σ_{LTTR} (capítulo V). A tarefa de construção de provas é, em geral, tediosa e um certo grau de engenhosidade se faz necessário para organizar a prova de maneira simples e inteligível. Devido a complexidade inerente de se testar a validade até para a mais simples das lógicas, provadores automáticos de teoremas não têm sido de grande ajuda.

CLARKE *et alii* (1986) defendem que a construção de provas é desnecessária para o caso de SDs de estados finitos (aqueles cuja execução pode ser representada por

uma seqüência *finita* de estados globais), propondo uma abordagem modelo-teórica que determinará mecanicamente se um sistema atende a uma especificação feita em lógica temporal.

Conforme essa abordagem, o grafo de estados globais do SD é visto como uma estrutura Kripke finita e um algoritmo, chamado de *checador de modelos*, determina se a estrutura é um modelo de uma fórmula particular, isto é, determina se o SD atende à sua especificação. Pela proposta, a especificação é feita em CTL (*Computation Tree Logic*), uma lógica temporal de tempo ramificado (vide capítulo V) e o algoritmo apresentado possui complexidade linear com o tamanho da estrutura e o tamanho da especificação.

Essa abordagem é de grande aplicabilidade, pois numerosos problemas de programação concorrente têm soluções de estados finitos, e as propriedades interessantes de muitos desses problemas podem ser expressadas na forma proposicional da lógica temporal.

VIII.2 Síntese Automática de Programas Concorrentes com a Lógica Temporal

MANNA *et alia* (1984) propõem um método de síntese automática de programas concorrentes a partir de uma especificação formal feita em LTLp (capítulo IV).

A LTLp é decidível e possui a propriedade de modelo finito, isto é, dado uma fórmula na LTLp, é decidível se essa fórmula é satisfazível, e se é satisfazível, ela possui um modelo finito. Isso é a base do método de síntese proposto: dado uma especificação na LTLp, é usado um método semelhante ao dos *tableaux* (RESCHER *et alia*, 1971; BEN-ARI *et alii*, 1983) para testar a satisfazibilidade[1] e para construir um modelo para a fórmula da especificação. Desse modelo, então, é extraído a parte sincronizadora (isto é, relativa à emissão ou recepção de mensagens) de um programa concorrente com a sintaxe semelhante à da linguagem CSP (HOARE, 1978 — vide capítulo II).

[1] Alguns autores usam o termo "satisfatibilidade" ou "satisfabilidade"

VIII.3 A Complexidade de Procedimentos de Decisão para a LTLp

SISTLA *et alia* (1985) examinam a complexidade inerente de procedimentos de decisão para a *validade*, *satisfazibilidade* e *verdade em estruturas geradas por relações binárias* para a LTLp com diferentes operadores temporais.

Estruturas geradas por relações binárias (*R*-estruturas) modelam programas concorrentes de estados finitos. O problema de se determinar a verdade em uma *R*-estrutura consiste em verificar se uma dada fórmula é verificada em um caminho começando em um estado da estrutura. Um algoritmo para resolver este problema pode ser usado para determinar se um programa concorrente falha em atender à sua especificação em alguma execução.

Usa-se a notação LTLp($\alpha_1, \ldots, \alpha_k$), onde $\alpha_1, \ldots, \alpha_k \in \{\Box, \Diamond, \circ, \mathcal{U}\}$ são operadores distintos, para indicar a LTLp restrita aos operadores temporais $\alpha_1, \ldots, \alpha_k$. Por exemplo, LTLp($\Diamond$,$\circ$) consiste na lógica temporal linear proposicional com apenas os operadores temporais $\Diamond$ e $\circ$ (SISTLA *et alia*, 1985).

VIII.3.1 A *R*-Estrutura T

Uma *R*-estrutura **T** é a dupla $(\mathbf{E}, R)$, onde:

- $\mathbf{E} = \{s, t, \ldots\}$ é o conjunto *finito* de estados (globais), descrevendo todas as possíveis situações do SD em estudo. Como em capítulos anteriores, cada $s' \in \mathbf{E}$ faz uma atribuição dos valores $\{\mathbf{v}, \mathbf{f}\}$ às fórmulas atômicas da lógica;
- $R : \mathbf{E} \times \mathbf{E}$ é uma relação binária *total*, isto é, $\forall s \in \mathbf{E}\ \exists t \in \mathbf{E}\ R(s,t)$.

Um *caminho* σ em **T** é uma seqüência infinita $\sigma = s_0 s_1 s_2 \cdots$, onde $\forall i \geq 0, s_i \in \mathbf{E}$ e $R(s_i, s_{i+1})$. Usa-se $\mathbf{T}_{\sigma,s_i}(f) \in \{\mathbf{v}, \mathbf{f}\}$ para denotar o valor-verdade da fórmula f no estado s_i do caminho σ.

VIII.3.2 A Complexidade dos Procedimentos de Decisão

O comportamento global de um SD pode ser modelado como uma *R*-estrutura: cada caminho começando do estado inicial representa uma intercalação possível das execuções dos processos individuais do sistema. Em muitos casos, os requisitos de correção de um SD podem ser expressados por uma fórmula da LTLp: o sistema estará correto se, e somente se, toda possível seqüência de execução satisfizer esta fórmula, isto é, todo caminho começando no estado inicial da *R*-estrutura correspondente satisfaz a fórmula. Por estas razões, o seguinte problema, chamado de *determinação da verdade em uma R-estrutura*, é importante na verificação de SDs de estados finitos:

> Dado uma *R*-estrutura **T**, um estado $s \in \mathbf{E}$ e uma fórmula $f \in \mathrm{LTLp}(\alpha_1, \ldots, \alpha_k)$, há um caminho σ em **T**, começando por s, tal que $\mathbf{T}_{\sigma,s}(f) = \mathbf{v}$?

Outros problemas interessantes são o da determinação da *satisfazibilidade em estruturas geradas por relações binárias*:

> Dado uma fórmula $f \in \mathrm{LTLp}(\alpha_1, \ldots, \alpha_k)$, existe alguma *R*-estrutura **T** com um estado $s \in \mathbf{E}$ na qual haja um caminho σ cujo primeiro estado seja s, tal que $\mathbf{T}_{\sigma,s}(f) = \mathbf{v}$?

e o da determinação da *validade em estruturas geradas por relações binárias*:

> Dado uma fórmula $f \in \mathrm{LTLp}(\alpha_1, \ldots, \alpha_k)$, é verdade que, para qualquer *R*-estrutura **T**, qualquer estado $s \in \mathbf{E}$ e para todos os caminhos σ cujo primeiro estado seja s, $\mathbf{T}_{\sigma,s}(f) = \mathbf{v}$?

Das definições acima, e dado uma fórmula f, conclui-se que, se f não é satisfatível, então sua negação ($\sim f$) é válida. Por outro lado, se f não é válida, então sua negação é satisfazível. Os teoremas a seguir são provados por SISTLA *et alia*, (1985):

Teorema VIII.3.2.1 Para a $\mathrm{LTLp}(\Diamond)$, os problemas de determinação da verdade em uma *R*-estrutura e da satisfazibilidade são *NP*-Completos.

Lógica	Satisfazibilidade	Validade	Verdade em uma R-Estrutura
LTLp($\Diamond$)	*NP*-Completo	*Co-NP*-Completo	*NP*-Completo
LTLp($\mathcal{U}$) LTLp($\Diamond$,$\circ$) LTLp($\Box$,$\Diamond$,$\circ$,$\mathcal{U}$)	*PESPAÇO*-Completo	*PESPAÇO*-Completo	*PESPAÇO*-Completo

Tabela VIII.1: Complexidade dos Procedimentos de Decisão para algumas LTLs

Teorema VIII.3.2.2 Para a LTLp($\Diamond$,$\circ$) e LTLp($\mathcal{U}$), os problemas de determinação da verdade em uma R-estrutura e da satisfazibilidade são *PESPAÇO*-Completos.

Pode-se organizar os resultados de SISTLA *et alia* (1985) como a tabela VIII.1.

Capítulo IX

Conclusões

Nesse trabalho, foram apresentadas algumas lógicas temporais, com uma preocupação de fazê-lo de forma didática. Na apresentação formal de cada uma das lógicas, foram dadas a sintaxe, a estrutura sobre a qual a lógica é interpretada e a semântica, além de um sistema formal cuja consistência e completeza são consideradas. Para cada lógica, foi dado um exemplo de uso na especificação formal de SDs, mostrando sua aplicabilidade a essa importante tarefa. A especificação formal foi, então, usada na verificação de algumas propriedades desejáveis (por exemplo exclusão mútua, ausência de bloqueio perpétuo e probidade) em implementações para o SD.

Ao final de cada capítulo, foram feitas conclusões específicas àquela lógica. Outras conclusões, essas de caráter geral, às quais podemos chegar são:

- *Carência de Métodos para a Especificação Formal com Lógicas Temporais* — Na bibliografia pesquisada, há uma carência de métodos para a especificação formal de programas concorrentes com a lógica temporal. OWICKI *et alia*, (1982), LAMPORT (1983a) e LAMPORT (1989), usando a LTL, foram os únicos trabalhos encontrados sobre esse assunto. A apresentação de seus métodos, no entanto, fogem ao escopo dessa dissertação. A solução encontrada para esse problema foi a prática: com a axiomatização da lógica de um lado e do outro a descrição informal do problema, começou-se a usar o formalismo, auxiliado por alguns artigos (SCHWARTZ *et alia*, (1981); LAMPORT (1983b); MANNA *et alia*, 1984) nos quais é feito algo semelhante. Nenhum método foi encontrado para a especificação formal de SDs que considerasse o tempo de forma ramificada.

- *Métodos de Dedução* — Nenhum método especial de provas foi usado nas verificações. Considerou-se uma simples aplicação do sistema formal às propriedades (fórmulas bem-formadas) da especificação. Na bibliografia pesquisada, foi encontrado (ABADI *et alia*, 1985) um sistema de prova para a LTLp, baseado em resolução não-clausal. Foi sentido, entretanto, a falta de um método para conduzir as provas — estas, às vezes, tornavam-se desnecessariamente longas e complicadas.

- *Verificação versus Especificação* — As atividades de especificação formal e verificação mostraram-se bastante interrelacionadas: enquanto é feito a especificação formal, temos o sentimento de pontos críticos do projeto que merecem ser formalmente verificados, orientando o processo posterior de verificação. Por outro lado, enquanto é feito a verificação, detectam-se erros, inconsistências ou sub-especificação o que leva a reformulação da especificação formal. Há, portanto, um *feedback* entre a especificação formal e a verificação. No exemplo de uso da LTLPo (subseção IV.5), só foram mostradas as provas úteis para o objetivo pretendido (a correção do protocolo) e a especificação formal final. No entanto, a especificação formal foi refeita várias vezes porque havia problemas coma verificação e, durante essas modificações, surgiram novas sugestões para a verificação.

  Durante as verificações temos um sentimento mais forte do uso (nas fórmulas da especificação formal) do operador $\equiv$ ao invés de $\supset$ (ou vice-e-versa), $\Diamond$ no lugar de $\circ$ (ou vice-e-versa), etc. Se for automatizada a atividade de verificação, deve-se pensar em alguma forma de interação, o usuário podendo modificar a especificação formal e ver em como mudam as conclusões.

- *Condução das Verificações* — Na verificação dos exemplos dos capítulos anteriores, foi comum abreviar vários passos nas demonstrações, sob o risco de a prova não ser compreendida de imediato por alguém estranho à atividade. Contudo, se isso não fosse feito, as demonstrações seriam muito extensas. Por exemplo, entre os passos 1 e 2 da verificação da exclusão mútua do exemplo da LTLp (subseção IV.3.3), foram abreviados os passos 1.1 a 1.4:

1. $\vdash \Psi \supset \Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i)$ — *TP* ($\psi_1 b$ e ψ_4)
1.1. $\vdash \Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i) \supset \Box[(a_i \supset \sim l_i) \wedge (a_i \supset \sim a_j \mathcal{U} l_i)]$ — **T5**
1.2. $\vdash \Psi \supset \Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i) \supset \Box[(a_i \supset \sim l_i) \wedge (a_i \supset \sim a_j \mathcal{U} l_i)]$ — *RP* (1.1)
1.3. $\vdash \Psi \supset (\Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i)) \wedge$
$\{(\Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i)) \supset \Box[(a_i \supset \sim l_i) \wedge (a_i \supset \sim a_j \mathcal{U} l_i)]\}$ — *RP* (1 e 1.2)
1.4. $\vdash (\Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i)) \wedge$
$\{(\Box(a_i \supset \sim l_i) \wedge \Box(a_i \supset \sim a_j \mathcal{U} l_i)) \supset \Box[(a_i \supset \sim l_i) \wedge (a_i \supset \sim a_j \mathcal{U} l_i)]\} \supset$
$\Box((a_i \supset \sim l_i) \wedge (a_i \supset \sim a_j \mathcal{U} l_i))$ — *MP*
2. $\vdash \Psi \supset \Box[(a_i \supset \sim l_i) \wedge (a_i \supset \sim a_j \mathcal{U} l_i)]$ — *RP* (1 e 1.4)

- *Formalizando o Informal* — Uma importante etapa na especificação formal de SDs diz respeito à elaboração de fórmulas que representem as descrições informais do problema ou das propriedades que uma solução para o problema deva possuir. Essa atividade nada mais é do que mapear uma linguagem não-formal (o português da descrição informal) em uma linguagem formal: mapear descrições informais em fórmulas de uma linguagem formal é um trabalho complexo e de caráter subjetivo pois envolve o *entendimento* do texto informal, a *apreensão* dos aspectos relevantes, a *concepção* de constantes proposicionais ou predicados e o que eles representam e seu *interrelacionamento*. As dificuldades dessa etapa não devem ser atribuídas exclusivamente ao uso do formalismo das lógicas temporais: elas surgem em qualquer sistema formal usado para esse fim. Com as lógicas temporais, entretanto, outro conceito informal de grande relevância, o relacionamento temporal, pode ser formalizado, aumentando a complexidade da atividade.

  Esta etapa exige uma interação do projetista com o autor da descrição informal (se forem pessoas diferentes) ou uma revisão da descrição informal (se forem a mesma pessoa). Quando se elabora as fórmulas, deve-se ter em mente a adequação da fórmula para as provas, buscando sua simplicidade e evitando o aninhamento de operadores temporais (vide item adiante). Mesmo estando a fórmula feita e "atendendo" à descrição informal, uma análise mais cuidadosa nas conseqüências lógicas da fórmula ou na especificação formal como um todo pode revelar contradições, conclusões inesperadas, fatos estranhos à descrição informal (que devem ser esclarecidos) ou a subespecificação (mais fatos são necessários para a prova).

- *Determinando a Duração das Verdades* — O modelo subjacente às lógicas temporais pressupõe verdades instantâneas, isto é, o valor-verdade das fórmulas é dependente de cada estado. Contudo, quando usamos constantes (ou predicados) para representar fatos ou ações não-atômicos (são verdadeiros durante *alguns* estados), surge o

problema de determinar durante quanto tempo o fato é verificado ou quanto tempo a ação leva para ser realizada. Por exemplo, seja a constante p representando o fato que o controle da execução de um processo está no módulo de recepção de mensagens. Isso complica bastante o raciocínio subseqüente, pois será necessário dizer quanto tempo leva para o módulo ser completado (por exemplo, em LTLp, $p \supset (\circ p \wedge \circ\circ p \wedge \circ\circ\circ \sim p)$ — p leva 3 estados para ser terminado). Por outro lado, se usarmos p para representar o fato que a mensagem *foi recebida*, a instantaneidade da afirmação é óbvia (por exemplo, $p \supset \circ \sim p$). Devemos explicitar a duração dessas verdades não-instantâneas, pois carregamos para as demonstrações a idéia da duração dos fatos, e se isto não for formalizado não será de utilidade alguma.

No exemplo da LTTR, foi usada a constante a_i para representar o recebimento de uma mensagem e a constante R_i para representar que o controle da execução do processo estava no módulo de recepção dessa mensagem (pelas definições informais, temos que $\forall_c \Box(a_i \supset R_i)$ e $\forall_c \Box(\sim R_i \supset \sim a_i)$ são verificados). Essa distinção foi útil nas demonstrações, pois quando queríamos precisar a recepção da mensagem (subseção V.3.3.1) usamos a_i e quando foi suficiente fazer referência ao módulo como um todo (subseção V.3.3.3) usamos R_i. Dessa forma, contornou-se o problema de precisar quantos estados é a duração do módulo R_i.

- *Da Necessidade de Provadores Automáticos* — Algumas demonstrações tornavam-se trabalhosas porque muitas vezes esquecia-se de um axioma ou teorema que simplificaria bastante a prova e ficava-se tentando provar de outra maneira. Isso foi devido ao grande número de regras aplicáveis — só com muita experiência uma pessoa guardaria tudo — para um computador isso não seria problema.

- *A Lógica Temporal sem Verificações* — A lógica temporal já é útil sem a verificação (mecânica ou manual). Escrever uma especificação formal pode ser de grande ajuda mesmo sem a verificação, porque força-se a entender precisamente o que o sistema deve fazer. Além disso, pode-se usar o formalismo da lógica temporal para organizar uma prova informal, o que pode capturar muitos erros.

- *Sobre o Aninhamento de Operadores Temporais* — À medida que o número de operadores temporais aumenta em uma fórmula, rapidamente torna-se difícil entender seu significado. Uma solução para este problema seria desenvolver operadores temporais

de nível mais alto cujo significado pretendido fosse o aninhamento dos operadores. Isso levaria a uma maior claridade da especificação.

Referências Bibliográficas

ABADI, M. e MANNA, Z., (1985); "Nonclausal Temporal Deduction"; *In: Proceedings*, New York, U.S.A., June 1985; Lecture Notes in Computer Science, Vol. 193; New York, Springer-Verlag, pp.1–15

ALFORD, Mack W.; LAMPORT, Leslie; MULLERY, Geoff P. (1985); "Basic Concepts"; *In: Distributed Systems: Methods and Tools for Specification — an Advanced Course*; Paul, M. e Siegert, H.J. (Eds.); Lecture Notes in Computer Science, vol.190; New York, Springer-Verlag, pp.431–479

AMORIM, Cláudio L.; BARBOSA, Valmir C.; FERNANDES, Edil S.T. (1988); *Uma Introdução à Computação Paralela e Distribuída*; Versão preliminar preparada para a VI Escola de Computação ; Campinas, São Paulo, Julho de 1988, 258p.

ANDREWS, Gregory R. e SCHNEIDER, Fred B. (1983); "Concepts and Notations for Concurrent Programming"; *ACM Computing Surveys*, 15(2):3–43, Mar. 1983

ANJART, Jean-Pierre (1985); "Issues and Tools for Protocol Specification"; *In: Distributed Systems: Methods and Tools for Specification — an Advanced Course*; Paul, M. e Siegert, H.J. (Eds.); Lecture Notes in Computer Science, vol.190; New York, Springer-Verlag, pp.481–538

ASHER, Nicholas (1988); "Reasoning About Belief and Knowledge with Self-Reference and Time"; *In: Proceedings of the 2nd. Conference on Theoretical Aspects of Reasoning About Knowledge*; March 7–9, Pacific Grove, Ca.; Moshe Y. Vardi (Ed.); Morgan Kaufmann Publishers, Inc., Los Altos, Ca., pp. 61–81

BALZER, R. M. (1971); "Ports — A Method for Dynamic Interprogramming Communication and Job Control"; *In: Proceedings AFIPS Spring Jt. Computer Conference*; Atlantic City, N. J., May 1971; U.S.A., AFIPS Press, pp. 485–489

BEN-ARI, Mordechai; PNUELI, Amir; MANNA, Zohar (1983); "The Temporal Logic of Branching Time" ; *Acta Informatica*, **20**:207–226

BOCHMANN, G. (1985); "Specification in Distributed Systems"; *In: Proceedings*, Glasgow, U.K., July 1983; Lecture Notes in Computer Science, Vol. 184; New York, Springer-Verlag, pp. 470–505

BRINCH HANSEN, Per (1975); "The Programming Language Concurrent Pascal"; *IEEE Trans. Soft. Eng.*, **SE-1**(2):199–207, June 1975

BRINCH HANSEN, Per (1978); "Distributed Processes: A Concurrent Programming Concept"; *Comm. ACM*, **21**(11):934–941, Nov. 1978

CARMO, José e SERNADAS, Amílcar (1988); "A Temporal Logic Framework for a Layered Approach to Systems Specification and Verification"; *In: Temporal Aspects in Information Systems*; C. Roland; F. Bodart; M. Leonard (Eds.); Elsevier Science Publishers B.V., IFIP, pp.31–46

CAVALLI, Ana R. e FARIÑAS-DEL-CERRO, Luis (1984); "Specification and Verification of Networks Protocols using Temporal Logic"; *Lecture Notes in Computer Science*, vol. 167; New York, Springer-Verlag New York Inc., pp. 59–73

CHANDY, K. Mani e LAMPORT, Leslie (1985); "Distributed Snapshots: Determining Global States of Distributed Systems"; *ACM Trans. Comput. Syst.*, **3**(1):63–75, Feb. 1985

CHEN, Bo-Shoe e YEH, Raymond T. (1983); "Formal Specification and Verification of Distributed Systems"; *IEEE Trans. Soft. Eng.*, **SE-9**(6):710–722, Nov. 1983

CLARKE, E. M.; BROWNE, M. C.; EMERSON, E. A.; SISTLA, A. P. (1985); "Using Temporal Logic for Automatic Verification of Finite-State Systems"; *In: Logics and Models of Concurrent Systems*; Ed.: Apt, K. R.; NATO ASI Series, Vol. F13; Berlin, Springer-Verlag, pp.3–26

CLARKE, E. M.; EMERSON, E. A.; SISTLA, A. P. (1986); "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications"; *ACM Trans. Progr. Lang. Syst.*, **8**(2):244–263, April 1986

COHEN, Bernard; HARWOOD, William T.; JACKSON, Melvyn I. (1986); *The Specification of Complex Systems*; Great Britain, Addison-Wesley Publishing Co., 143p.

CRICHLOW, Joel M. (1988); *An Introduction to Distributed and Parallel Computing*; United Kingdom, Prentice Hall International Ltd., 209p.

DIJKSTRA, Edsger W. (1975); "Guarded Commands, Non-determinacy and Formal Derivation of Programs"; *Comm. ACM*, 18(8):453–457, Aug. 1975

EMERSON, E. A. (1983); "Alternative Semantics for Temporal Logics"; *Theor. Comput. Sci.*, 26:121–130 .

EMERSON, E. A. e HALPERN, J. Y. (1986); ""Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic"; *J. ACM*, 33(1):151–178, Jan. 1986

EMERSON, E. A. e SISTLA, A. P. (1984); "Deciding Full Branching Time Logic"; *Information and Control*, 61:175–201

ENDERTON, Herbert B. (1972); *A Mathematical Introduction to Logic*; New York, Academic Press, Inc., 295p.

FAGIN, Ronald e HALPERN, Joseph (1985); "Belief, Awareness, and Limited Reasoning: Preliminary Report"; *In: Proceedings of the 9th. IJCAI*; Los Angeles, California, 1984, pp. 491–501

GABBAY, Dov; PNUELI, Amir; SHELAH, Saharon; STAVI, Jonathan (1980); "On the Temporal Analisys of Fairness"; *In: Proceedings of the 7th. Annual ACM Symposium on Principles of Programming Languages*, Las Vegas, Ne, Jan. 28–30; New York, ACM, pp.163–173

GUIDACCI DA SILVEIRA, Geraldo F. (1982); *Sur La Preuve des Sistèmes Parallèles et Distribués par La Logique Temporelle*; Thèse du Doctorat d'Etat; Université Paul Sabatier de Toulouse; France, Janvier 1982

HAILPERN, Brent (1985); "Tools for Verifying Network Protocols"; *In: Logics and Models of Concurrent Systems*; Ed.: Apt, K. R.; NATO ASI Series, Vol. F13; Berlin, Springer-Verlag, pp. 57–76

HALPERN, Joseph e MOSES, Yoram (1984a); "Knowledge and Commom Knowledge in a Distributed Environment"; Technical Report IBM, RJ 4421; San Jose, California, Oct. 1984

HALPERN, Joseph e MOSES, Yoram (1984b); "Towards a Theory of Knowledge and Ignorance"; Technical Report IBM, RJ 4448; San Jose, California, Oct. 1984

HALPERN, Joseph e MOSES, Yoram (1985); "A Guide to the Modal Logics of Knowledge and Belief"; *In: Proceedings of the 9th. IJCAI*; Los Angeles, California, 1984, pp. 480–490

HALPERN, Joseph (1986); "Reasoning About Knowledge: An Overview"; *In: Theoretical Aspects of Reasoning About Knowledge — Proceedings of the 1986 Conference*; Morgan Kaufmann Publishers, Inc., Los Altos, Ca., pp. 1–17

HAREL, David (1987); *Algorithmics: the Spirit of Computing*; Great Britain, Addison-Wesley Publishing Company, 425p.

HOARE, Charles Anthony R. (1978); "Communicating Sequential Processes"; *Comm. ACM*, 21(8):666–677, Aug. 1975

HUGHES, G. E. e CRESSWELL, M. J. (1968); *An Introduction to Modal Logic*; London, Methuen and Co. Ltd., 388p.

KATZ, Shmuel e PELED, Doron (1987); "Interleaving Set Temporal Logic"; Preliminary Version; *In: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*; Vancouver, B.C., Canada, Aug. 10–12, 1987

KELLER, Robert M. (1976); "Formal Verification of Parallel Programs"; *Comm. ACM*, 19(7):371–384, July 1976

KHEI, Huang H. (1988); *Técnicas para a Alocação Estática de Tarefas em Sistemas Distribuídos*; Tese de Mestrado; Coordenação dos Programas de Pós-Graduação em Engenharia (COPPE); Programa de Engenharia de Sistemas e Computação; Universidade Federal do Rio de Janeiro; Rio de Janeiro, Junho de 1988, 93p.

KIRNER, Cláudio e MENDES, Sueli B.T. (1988); *Sistemas Operacionais Distribuídos: Aspectos Gerais e Análise de sua Estrutura*; Rio de Janeiro, Editora Campus, 184p.

KNUTH, Donald E. (1984); *The TEX book*; Reading, Massachusetts, Addison-Wesley, 1984, 483p.

KOYMANS, Ron (1987); "Specifying Message Passing Systems Requires Extending Temporal Logic"; *In: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*; Vancouver, B.C., Canada, Aug. 10–12, 1987

KRAUS, Sarit e LEHMANN, Daniel (1988); "Knowledge, Belief and Time"; *Theor. Comput. Sci.*, **58**:155–174

KRÖGER, Fred (1987); *Temporal Logic of Programs*; EATCS Monographs on Theoretical Computer Science, vol. 8; Berlin, Springer-Verlag Berlin Heidelberg, 1987, 148p.

LAMPORT, Leslie (1977); "Proving the Correctness of Multiprocess Programs"; *IEEE Trans. Software Eng.*, **SE-3**(9):125–143, Mar. 1977

LAMPORT, Leslie (1978); "Time, Clocks, and the Ordering of Events in a Distributed System"; *Comm. ACM*, **21**(7):558–565, July 1978

LAMPORT, Leslie (1980); ""Sometime" is Sometimes "Not Never": on the Temporal Logic of Programs"; *In: Proceedings of the 7th. ACM Symposium on Principles of Programming Languages*; Las Vegas, U.S.A.

LAMPORT, Leslie (1983a); "Specifying Concurrent Program Modules"; *ACM Trans. Program. Lang. Syst.*, **5**(2):190–222, April 1983

LAMPORT, Leslie (1983b); "What Good is Temporal Logic?"; *In: Proc. of IFIP 9th. World Congress*, pp. 657–668; Paris, France, Sept. 1983

LAMPORT, Leslie (1985a); "An Axiomatic Semantics of Concurrent Programming Languages"; *In: Logics and Models of Concurrent Systems*; Ed.: Apt, K. R.; NATO ASI Series, Vol. F13; Berlin, Springer-Verlag, pp.77–122

LAMPORT, Leslie (1985b); "Paradigms for Distributed Programs"; *In: Distributed Systems: Methods and Tools for Specification — an Advanced Course*; Paul, M. e Siegert, H.J. (Eds.); Lecture Notes in Computer Science, vol.190; New York, Springer-Verlag, pp.431–479

LAMPORT, Leslie (1986); LATEX: *A Document Preparation System*; U.S.A., Addison-Wesley Publishing Company, Inc., 242p.

LAMPORT, Leslie (1989); "A Simple Approach to Specifying Concurrent Systems"; *Comm. ACM*, **32**(1):165–198

LAUER, Peter E. (1983); "Computer Systems Dossiers"; *In: Distributed Computing Systems*; Eds.: Parker,Y e Verjus, J.-P.; London, Academic Press Inc. Ltd., pp. 109–147

LEHMANN, Daniel e SHELAH, Saharon (1982); "Reasoning with Time and Chance"; *Information and Control*, **53**:165–198

LICHTENSTEIN, Orna; PNUELI, Amir; ZUCK, Lenore (1985); "The Glory of The Past"; *In: Proceedings*, New York, U.S.A., June 1985 ; Lecture Notes in Computer Science, vol. 193; Berlin, Springer-Verlag, pp. 196–218

LORIN, H. (1979); "Distributed Processing: An Assessment"; *IBM Syst. J.*, **18**(4):582–603

MAGALHÃES, Maurício Ferreira (1986); *Software para Tempo Real*; Livro-texto de apoio à I EBAI, Campinas, São Paulo, 17/02 a 01/03/86; Campinas, Editora da Unicamp, 82p.

MANNA, Zohar (1974); *Mathematical Theory of Computation*; Tokio, Japan, McGraw-Hill, Inc., 448p.

MANNA, Zohar (1978); "Is "Sometime" Sometimes Better than "Always"? Intermittent Assertions in Proving Program Correctness"; *Comm. ACM*, **21**(2):159–172, Feb. 1978

MANNA, Zohar (1981); "Verification of Sequential Programs: Temporal Axiomatization"
In: Theoretical Foundations of Programming Methodology; Eds.:Bauer, F.L.; Dijkstra, E.W.; Hoare, C.A.R.; NATO Scientific Series; D.Reidel Publ. Co., Holland

MANNA, Zohar e PNUELI, Amir (1979); "The Modal Logic of Programs" ; *In: Proceedings of the 6th International Colloquium on Automata, Languages and Programming*; Lecture Notes in Computer Science, vol. 71; Berlin, Springer-Verlag, pp. 385–409

MANNA, Zohar e PNUELI, Amir (1981a); "Verification of Concurrent Programs: the Temporal Framework"; *In: The Correctness Problem in Computer Science*; Eds.:Boyer, R.S. e Moore, J.S.; International Lecture Series in Computer Science; Academic Press, London, pp. 215–273

MANNA, Zohar e PNUELI, Amir (1981b); "Verification of Concurrent Programs: Temporal Proof Principles" ; *In: Proceedings of the Workshop on Logics of Programs*; Yorktown Heights, N.Y., U.S.A.; Lecture Notes in Computer Science, vol. 131; Berlin, Springer-Verlag, pp. 200–252

MANNA, Zohar e PNUELI, Amir (1983); "How to Cook a Temporal Proof System for Your Pet Language"; *In: Proceedings of the 10th. Annual ACM Symposium on Principles of Programming Languages*, Jan. 1983, pp. 141–154

MANNA, Zohar e WALDINGER, Richard (1978); "Is "Sometime" Sometimes Better than "Always"? — Intermittent Assertions in Proving Program Correctness"; *Comm. ACM*, **21**(2):159–172, Feb. 1978

MANNA, Zohar e WOLPER, Pierre (1984); "Synthesis of Communicating Processes from Temporal Logic Specifications"; *ACM Trans. Program. Lang. Syst.*, **6**(1):68–93, Jan. 1984

MENDELSON, Elliott (1964); *Introduction to Mathematical Logic*; New York, Van Nostrand Reinhold Company, 300p.

MULLERY, Geoff P. (1985b); "Conclusion"; *In: Distributed Systems: Methods and Tools for Specification — an Advanced Course*; Paul, M. e Siegert, H.J. (Eds.); Lecture Notes in Computer Science, vol.190; New York, Springer-Verlag, pp.539–547

NGUYEN, V.; DENNERS, A.; GRIES, D.; OWICKI, S. (1985); "Behavior: a Temporal Approach to Process Modelling"; *In: Proceedings*, New York, U.S.A., June 1985; Lecture Notes in Computer Science, Vol. 193; New York, Springer-Verlag, pp.237–254

OWICKI, Susan e LAMPORT, Leslie (1982); "Proving Liveness Properties of Concurrent Programs"; *ACM Trans. Program. Lang. Syst.*, **4**(3):455–495, July 1982

PETERSON, James L. e SILBERSCHATZ, Abraham (1983); *Operating Systems Concepts*; U.S.A., Addison-Wesley Publishing Company, 548p.

PNUELI, Amir (1977); "The Temporal Logic of Programs"; *In: Proceedings of the IEEE 18th. Annual Symposium on Foundations of Computer Science*; Providence, R. I., pp.46–57

PNUELI, Amir (1979); "The Temporal Semantics of Concurrent Programs"; *In: Proceedings of the International Conference on Semantics of Concurrent Computation*; Lecture Notes in Computer Science, vol. 70; Berlin, Springer-Verlag, pp. 1–20

PNUELI, Amir (1985); "In Transition from Global to Modular Temporal Reasoning about Programs"; *In: Logics and Models of Concurrent Systems*; Ed.: Apt, K. R.; NATO ASI Series, Vol. F13; Berlin, Springer-Verlag, pp.123–144

PRIOR, Arthur N. (1957); *Time and Modality*; Oxford, The Clarendon Press, 148p.

PRIOR, Arthur N. (1967); *Past, Present and Future*; Oxford, The Clarendon Press, 217p.

RESCHER, Nicholas e URQUHART, Alasdair (1971); *Temporal Logic*; Wien, Springer-Verlag, 273p.

SCHNEIDER, Fred B. e LAMPORT, Leslie (1985); "Paradigms for Distributed Programs"; *In: Distributed Systems: Methods and Tools for Specification — an Advanced Course*; Paul, M. e Siegert, H.J. (Eds.); Lecture Notes in Computer Science, vol.190; New York, Springer-Verlag, pp.431–479

SCHWARTZ, R. L. e MELLIAR-SMITH, P. M. (1981); "Temporal Logic Specification of Distributed System"; *In: Proceddings of the 2nd. International Conference on Distributed Computing Systems*; IEEE Computer Society Press, pp.446–454

SEGEV, Arie e SHOSHANI, Arie (1988); "Modeling Temporal Semantics"; *In: Temporal Aspects in Information Systems*; C. Roland; F. Bodart; M. Leonard (Eds.); Elsevier Science Publishers B.V., IFIP, pp.47–57

SHOHAM, Yoav (1988); *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence*; U.S.A., M.I.T. Press, 200p.

SISTLA, A.P. e CLARKE, E. M. (1985); "The Complexity of Propositional Linear Temporal Logics"; *J. ACM*, **32**(3):733–749, July 1985

SMETS, Philippe; MAMDANI, E. H.; DUBOIS, Didier; PRADE, Henri (1988); *Non-Standard Logics for Automated Reasoning*; San Diego, Academic Press Limited, 334p.

STOTTS JR., Paul D. (1982); "A Comparative Survey of Concurrent Programming Languages"; *SIGPLAN Notices*, 17(9):76–87, Sept. 1982

STROM, Rob e YEMINI, Shaula (1984); "The NIL Distributed Systems Programming Language: a Status Report"; *IBM Research Report* RC 10864; Yorktown Heights, New York, Dec. 1984

SZALAS, Andrzej e HOLENDERSKI, Leszek (1988); "Incompleteness of First-Order Temporal Logic with Until"; *Theor. Comput. Sci.*, 57:317–325

VELOSO, Sheila R. M. (1985); *Mecanismos de Transmissão de Mensagens em Sistemas Distribuídos: Um Modelo Semântico*; Tese de Doutoramento; Coordenação dos Programas de Pós-Graduação em Engenharia (COPPE); Programa de Engenharia de Sistemas e Computação; Universidade Federal do Rio de Janeiro; Rio de Janeiro, Setembro de 1985, 167p.

VIDAL SILVA, Lucy (1986); *Modelos de Implementação para o Paralelismo em Ada e RED*; Tese de Mestrado; Departamento de Informática, P.U.C.-R.J.; Rio de Janeiro, Agosto de 1986, 2 volumes, 538p.

WIRTH, Niklaus (1971); "The Programming Language Pascal"; *Acta Informatica*, 1(1):72–99

WOLPER, Pierre (1983); "Temporal Logic Can Be More Expressive"; *Information and Control*, 59:72–99

Apêndice A

Lógica Proposicional, Lógica de Predicados de Primeira Ordem e Alguns Símbolos Utilizados

Os conceitos e noções das lógicas proposicional e de primeira ordem são mostrados neste apêndice, bem como uma lista de símbolos e a notação utilizada nas regras lógicas.

A.1 Lógica Proposicional (Lp) ou Cálculo Proposicional

Uma linguagem lógica é dada por um alfabeto de símbolos e a definição de um conjunto de seqüências de símbolos, chamadas *fórmulas*, sobre esse alfabeto. A linguagem $\mathcal{L}_{Lp}$ da lógica proposicional ou cálculo proposicional é o tipo mais simples de uma linguagem lógica.

A.1.1 Sintaxe

As fórmulas da Lp são formadas a partir do seguinte alfabeto de símbolos:

1. Sinais de pontuação: "(" e ")";
2. Símbolos de valor-verdade: **V** (verdade) e **F** (falso);
3. Conectivos: $\sim$ (*não*), $\supset$ (*implicação lógica*), $\wedge$ (*e*), $\vee$ (*ou*) e $\equiv$ (*equivalência*);
4. Constantes proposicionais: $p_1, p_2, \ldots, p_n$

Por razões de simplicidade, também usaremos $p, q, r, \ldots$ para denotar constantes proposicionais. Usando esses símbolos, definimos recursivamente as *fórmulas bem-formadas* (ou simplesmente *fórmulas*) tomando por base as *fórmulas atômicas*:

1. Fórmulas Atômicas — são os construtos mais simples da Lp, a partir dos quais as fórmulas (bem-formadas) são formadas. As fórmulas atômicas, referenciadas genericamente por $v, v_1, v_2, \ldots$, são:

   a) os símbolos de valor-verdade **V** e **F** e

   b) as constantes proposicionais p_i ou $p, q, r, \ldots$;

   Chamaremos de $\mathcal{V}$ o conjunto enumerável das fórmulas atômicas.

2. Fórmulas Bem-formadas — referenciadas de modo genérico pelas meta-variáveis $A, A_1, A_2, \ldots, B, B_1, B_2, \ldots, F, \ldots$, são obtidas a partir das fórmulas atômicas, às quais são aplicadas os sinais de pontuação e os conectivos:

   a) toda fórmula atômica é uma fórmula bem-formada;

   b) se A é uma fórmula bem-formada então (A) também é uma fórmula bem-formada;

   c) se A e B são fórmulas bem-formadas, então $\sim A$, $A \supset B$, $A \wedge B$, $A \vee B$ e $A \equiv B$ também são fórmulas bem-formadas.

Observações:

i) Usando os conectivos $\sim$ e $\supset$, podemos definir os demais conectivos $\wedge$, $\vee$ e $\equiv$, isto é, os conectivos $\wedge$, $\vee$ e $\equiv$ podem ser usados como abreviações às seguintes fórmulas:

$A \wedge B$ abrevia $\sim (\sim A \supset B)$;

$A \vee B$ abrevia $\sim A \supset B$;

$A \equiv B$ abrevia $((A \supset B) \wedge (B \supset A))$

ii) Os símbolos de valor-verdade são definidos semelhantemente:

V abrevia $v_0 \vee \sim v_0$;

F abrevia $\sim$**v**

iii) Visando a simplicidade notacional, estabelecemos uma ordem de prioridade dos operadores:

$\sim$ tem maior prioridade que os demais operadores binários;

$\wedge$ e $\vee$ tem maior prioridade que $\supset$ e $\equiv$;

$\supset$ tem maior prioridade que $\equiv$;

Com essa prioridade omitiremos os sinais de pontuação que se tornarem desnecessários (incluindo os parênteses mais externos) para representar a prioridade dos operadores na sentença pretendida. Entretanto, há casos onde o uso dos sinais de pontuação é essencial para que a avaliação das fórmulas seja fiel ao pretendido pelo formalismo, evitando ambigüidades, e há outros casos onde o uso dos sinais não é essencial, mas sua presença confere mais clareza às fórmulas. Juntamente com os parênteses, usaremos as chaves ("{" e "}") e os colchetes ("[" e "]"). Podemos escrever

$$\{[A_1 \supset (A_2 \vee A_3)] \equiv ((\sim A_4) \wedge A_5)\}$$

ou

$$((A_1 \supset (A_2 \vee A_3)) \equiv ((\sim A_4) \wedge A_5))$$

ou simplesmente

$$A_1 \supset A_2 \vee A_3 \equiv \sim A_4 \wedge A_5$$

A.1.2 Semântica

A *semântica* da linguagem $\mathcal{L}_{Lp}$ é baseada no conceito de avaliação booleana: uma avaliação **B** é um mapeamento

$$\mathbf{B} : \mathcal{V} \rightarrow \{\mathbf{v}, \mathbf{f}\}$$

onde $\mathcal{V}$ é o conjunto das fórmulas atômicas e, por definição, $\mathbf{B}(\mathbf{V}) = \mathbf{v}$ e $\mathbf{B}(\mathbf{F}) = \mathbf{f}$. **B** atribui um valor **v** ou **f** a cada uma das fórmulas atômicas $v \in \mathcal{V}$. O mapeamento **B** pode ser estendido ao conjunto de todas as fórmulas:

1. $\mathbf{B}(v)$, para $v \in \mathcal{V}$, é dado;
2. $\mathbf{B}(\sim A) = \mathbf{v}$ se, e somente se, $\mathbf{B}(A) = \mathbf{f}$;
3. $\mathbf{B}(A \wedge B) = \mathbf{v}$ se, e somente se, $\mathbf{B}(A) = \mathbf{v}$ e $\mathbf{B}(B) = \mathbf{v}$;

4. $\mathbf{B}(A \vee B) = \mathbf{v}$ se, e somente se, $\mathbf{B}(A) = \mathbf{v}$ ou $\mathbf{B}(B) = \mathbf{v}$;

5. $\mathbf{B}(A \supset B) = \mathbf{v}$ se, e somente se, $\mathbf{B}(A) = \mathbf{f}$ ou $\mathbf{B}(B) = \mathbf{v}$;

6. $\mathbf{B}(A \equiv B) = \mathbf{v}$ se, e somente se, $\mathbf{B}(A \supset B) = \mathbf{v}$ e $\mathbf{B}(B \supset A) = \mathbf{v}$;

A.1.3 Definições

Definição A.1.3.1 Uma fórmula A é *válida em* $\mathbf{B}$, denotando-se por $\models_{\mathbf{B}} A$, se $\mathbf{B}(A) = \mathbf{v}$.

Definição A.1.3.2 Uma fórmula A é *válida* ou *tautologia*, denotando-se por $\models A$, se $\models_{\mathbf{B}} A$ para todo $\mathbf{B}$.

Definição A.1.3.3 Uma fórmula A *segue de* um conjunto $\mathcal{F}$ de fórmulas, denotando-se por $\mathcal{F} \models A$, se $\models_{\mathbf{B}} A$ para todo $\mathbf{B}$ tal que $\models_{\mathbf{B}} B$ para todo $B \in \mathcal{F}$.

A.1.4 O Sistema Formal Σ_{Lp}

Na formalização de uma lógica, é importante a definição de um sistema formal ou dedutivo. Chamamos de Σ_{Lp} o sistema abaixo que formaliza a Lp. No Σ_{Lp}, um conjunto de fórmulas da $\mathcal{L}_{Lp}$ é separado e chamado de conjunto dos *axiomas* de Σ_{Lp}. *Regras de Inferência* são relações R_i entre fórmulas constituindo um conjunto finito $R_1, \ldots, R_n$. Para cada R_i há um único inteiro positivo j tal que, para todo conjunto de j fórmulas e cada fórmula A, pode-se, efetivamente decidir se as j fórmulas dadas estão na relação R_i para A, e, se é assim, A é chamada de uma *conseqüência direta* das j fórmulas.

Uma *prova* no Σ_{Lp} é uma seqüência de fórmulas $A_1 \cdots A_n$ tal que, para cada i, ou A_i é um axioma de Σ_{Lp} ou A_i é uma conseqüência direta de algumas das fórmulas precedentes por virtude de uma das regras de inferência. Um *teorema* de Σ_{Lp} é uma fórmula A da $\mathcal{L}_{Lp}$ tal que há uma prova cuja última fórmula é A. Tal prova é chamada uma *prova de* A.

Uma fórmula A é uma *conseqüência* em Σ_{Lp} de um conjunto $\mathcal{F}$ de fórmulas se, e somente se, houver uma seqüência $A_1 \cdots A_n$ de fórmulas tal que $A = A_n$ e, para cada i, ou A_i é um axioma ou A_i pertence a $\mathcal{F}$ ou A_i é uma conseqüência direta por alguma regra de inferência de alguma das fórmulas precedentes na seqüência. Tal seqüência é

chamada uma *prova* (ou dedução) de A a partir de $\mathcal{F}$. Os membros de $\mathcal{F}$ são chamados de *hipóteses* ou *premissas* da prova, e é usado a notação $\mathcal{F} \vdash A$ como uma abreviação para "A é uma conseqüência de $\mathcal{F}$". Se $\mathcal{F}$ é o conjunto vazio $\emptyset$, então $\emptyset \vdash A$ se, e somente se, A é um teorema. O símbolo $\emptyset$ será omitido, escrevendo-se $\vdash A$. Portanto, $\vdash A$ é uma outra maneira de dizer que A é um teorema (MENDELSON, 1964).

Axiomas

Os esquemas de axiomas que constituem a base do Σ_{Lp} são:

A1. $\vdash A \supset (B \supset A)$

A2. $\vdash ((A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C)))$

A3. $\vdash ((\sim B \supset \sim A) \supset ((\sim B \supset A) \supset B))$

Esquema de axioma significa que as letras $A, B, C, \ldots$ podem ser substituídas por fórmulas e não somente por constantes proposicionais.

Regras de Inferência

A única regra de inferência do Σ_{Lp} é:

R1. Se $\vdash A \supset B$ e $\vdash A$ então $\vdash B$ (*Modus Ponens* — *MP*)

A.1.5 Consistência e Completeza do Σ_{Lp}

Um sistema formal é *consistente* ("sound") quando todos os seus esquemas de axiomas são válidos e todas as suas regras de inferência preservam a validade, isto é, não é possível derivar fórmulas não-válidas. Afirmamos que

O sistema Σ_{Lp} é consistente.

Um sistema formal é *completo* quando todas as fórmulas válidas são teoremas. Afirmamos que:

O sistema Σ_{Lp} é completo.

As provas dessas duas afirmações podem ser encontradas em MENDELSON (1964, pp.35–37).

A.1.6 Uma Lista de Teoremas Selecionados

Alguns teoremas do Σ_{Lp} são:

1. Leis de *de Morgan*:

   a) $\vdash \sim (A \vee B) \equiv \sim A \wedge \sim B$

   b) $\vdash \sim (A \wedge B) \equiv \sim A \vee \sim B$

2. Leis Distributivas:

   a) $\vdash A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$

   b) $\vdash A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

3. Leis Comutativas:

   a) $\vdash A \vee B \equiv B \vee A$

   b) $\vdash A \wedge B \equiv B \wedge A$

   c) $\vdash (A \equiv B) \equiv (B \equiv A)$

4. Leis Associativas:

   a) $\vdash (A \vee B) \vee C \equiv A \vee (B \vee C)$

   b) $\vdash (A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$

   c) $\vdash ((A \equiv B) \equiv C) \equiv (A \equiv (B \equiv C))$

5. Leis da Negação:

   a) $\vdash \sim\sim A \equiv A$

   b) $\vdash \sim (A \supset B) \equiv A \wedge \sim B$

   c) $\vdash \sim (A \equiv B) \equiv (A \wedge \sim B) \vee (\sim A \wedge B)$

6. Lei do Terceiro Excluído: $\vdash A \vee \sim A$

7. Lei da Contradição: $\vdash \sim (A \wedge \sim A)$

8. Lei da Contraposição: $\vdash (A \supset B) \equiv (\sim B \supset \sim A)$

9. Lei da Exportação: $\vdash ((A \wedge B) \supset C) \equiv A \supset (B \supset C)$

Os teoremas acima decorrem do próprio significado dos conectivos, sendo facilmente demonstráveis.

A.2 Lógica (ou Cálculo) de Predicados de Primeira Ordem (LPPo)

A lógica proposicional (Lp) investiga as operações lógicas $\sim$, $\wedge$, $\vee$, $\supset$ e $\equiv$ a partir de algumas fórmulas das quais nenhum detalhe adicional é dado. A *lógica de predicados de primeira ordem* (LPPo) é baseada na Lp mas, em adição a isso, examina mais de perto a estrutura das fórmulas atômicas e permite a *quantificação*.

A.2.1 Sintaxe

A linguagem da LPPo ($\mathcal{L}_{LPPo}$) constitui-se dos itens 1,2 e 3 do alfabeto da Lp mais os seguintes símbolos:

4. Operadores clássicos: $=$ (*igualdade*);

5. Quantificadores (de primeira ordem): $\forall$ (*quantificador universal*) e $\exists$ (*quantificador existencial*);

6. Símbolos de funções n-árias $f_i^n (i \geq 1, n \geq 0)$; f_i^0 é chamado uma *constante individual* e é escrito também como a_i;

7. Símbolos de predicados n-ários $p_i^n (i \geq 1, n \geq 0)$; p_i^0 é chamado uma *constante proposicional*;

8. Variáveis (individuais): $F_i^0, i \geq 0$, é escrito também como $x_i, i \geq 0$;

Usando esses símbolos, definimos recursivamente três classes de expressões: *termos*, *fórmulas atômicas* e *fórmulas bem-formadas*:

1. Termos — são construídos a partir de constantes individuais e variáveis às quais são aplicadas funções. A aplicação deve obedecer as restrições de número e tipo dos argumentos:

   a) para todo $i \geq 1$ e $n \geq 0$, se $t_1, t_2, \ldots, t_n$ são termos e f_i^n é um símbolo de função n-ária, então $f_i^n(t_1, t_2, \ldots, t_n)$ e $F_i^n(t_1, t_2, \ldots, t_n)$ também são termos;

2. Fórmulas Atômicas — consistem de proposições e predicados (incluindo o operador $=$) aplicado a termos de tipos apropriados:

   a) V e F são fórmulas atômicas;

   b) para todo $i \geq 1$ e $n \geq 0$, se $t_1, t_2, \ldots, t_n$ são termos e p_i^n é um símbolo de predicado n-ário, então $p_i^n(t_1, t_2, \ldots, t_n)$ é uma fórmula atômica;

   c) se t_1 e t_2 são termos, então $(t_1 = t_2)$ é uma fórmula atômica

3. Fórmulas Bem-formadas — são obtidas a partir das fórmulas atômicas às quais são aplicadas os conectivos, os operadores clássicos e a quantificação sobre variáveis:

   a) toda fórmula atômica é uma fórmula bem-formada;

   b) se A é uma fórmula bem-formada, então (A) também é uma fórmula bem-formada;

   c) se A, B e C são fórmulas bem-formadas, então $\sim A$, $A \supset B$, $A \wedge B$, $A \vee B$ e $A \equiv B$ também são fórmulas bem-formadas;

   d) se x é uma variável e A é uma fórmula bem-formada, então $\forall x\, A$ e $\exists x\, A$ são fórmulas bem-formadas;

Observações:

i) Em adição às abreviações da $\mathcal{L}_{Lp}$, há na $\mathcal{L}_{LPPo}$ a seguinte abreviação

$$\exists x A \text{ por } \sim \forall x \sim A$$

ii) À lista de prioridades, acrescentamos

   $\exists x$ e $\forall x$ têm maior prioridade que os demais operadores;

iii) A ocorrência de uma variável x em alguma fórmula A é *ligada* se ela aparece em alguma parte $\forall x B$ de A. Senão ela é chamada de *livre*:

iv) Se t é um termo, então $A_x(t)$ denota o resultado da substituição de t por toda ocorrência livre de x em A. Quando escrevemos $A_x(t)$, sempre assumiremos implicitamente que t não contém nenhuma variável que ocorre ligada em A (isto sempre pode ser conseguido pela substituição das variáveis ligadas de A por outras variáveis).

A.2.2 Semântica

O conceito semântico básico da LPPo é uma estrutura **S** consistindo de

- um conjunto $|\mathbf{S}| \neq \{\ \}$ (não vazio), o *universo*;
- uma função n-ária $\mathbf{S}(f) : |\mathbf{S}|^n \to |\mathbf{S}|$ para todo símbolo f de função n-ária;
- uma relação n-ária $\mathbf{S}(p) \subset |\mathbf{S}|^n$ para todo símbolo p de predicado n-ário diferente de $=$;

e uma avaliação de variáveis ξ (com respeito a **S**), que atribui algum $\xi(x) \in |\mathbf{S}|$ a toda variável x da $\mathcal{L}_{LPPo}$. Uma estrutura **S** juntamente com uma avaliação ξ define um valor $\mathbf{S}^{(\xi)}(t) \in |\mathbf{S}|$ para todo termo t:

1. $\mathbf{S}^{(\xi)}(t) = \xi(x)$ para toda variável x;
2. $\mathbf{S}^{(\xi)}(f(t_1, \ldots, t_n)) = \mathbf{S}(f)(\mathbf{S}^{(\xi)}(t_1), \ldots, \mathbf{S}^{(\xi)}(t_n))$.

Além disso, podemos definir $\mathbf{S}^{(\xi)}(A) \in \{\mathbf{v}, \mathbf{f}\}$ para toda fórmula atômica:

1. $\mathbf{S}^{(\xi)}(p(t_1, \ldots, t_n)) = \mathbf{v}$ se, e somente se, $(\mathbf{S}^{(\xi)}(t_1), \ldots, \mathbf{S}^{(\xi)}(t_n)) \in \mathbf{S}(p)$ para p diferente de $=$;
2. $\mathbf{S}^{(\xi)}(t_1 = t_2) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi)}(t_1) \stackrel{\mathbf{S}}{=} \mathbf{S}^{(\xi)}(t_2)$, onde $\stackrel{\mathbf{S}}{=}$ denota igualdade em $|\mathbf{S}|$.

Agora $\mathbf{S}^{(\xi)}$ desempenha o papel da avaliação **B** da $\mathcal{L}_{Lp}$ e pode ser estendida indutivamente a todas as fórmulas:

1. $\mathbf{S}^{(\xi)}(A)$, para fórmulas atômicas, é dado;

2. $\mathbf{S}^{(\xi)}(\sim A) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi)}(A) = \mathbf{f}$;

3. $\mathbf{S}^{(\xi)}(A \wedge B) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi)}(A) = \mathbf{v}$ e $\mathbf{S}^{(\xi)}(B) = \mathbf{v}$;

4. $\mathbf{S}^{(\xi)}(A \vee B) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi)}(A) = \mathbf{v}$ ou $\mathbf{S}^{(\xi)}(B) = \mathbf{v}$;

5. $\mathbf{S}^{(\xi)}(A \supset B) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi)}(A) = \mathbf{f}$ ou $\mathbf{S}^{(\xi)}(B) = \mathbf{v}$;

6. $\mathbf{S}^{(\xi)}(A \equiv B) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi)}(A \supset B) = \mathbf{v}$ e $\mathbf{S}^{(\xi)}(B \supset A) = \mathbf{v}$;

7. $\mathbf{S}^{(\xi)}(\forall x A) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi')}(A) = \mathbf{v}$ para todo ξ' com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x;

8. $\mathbf{K}_i(\exists x A) = \mathbf{v}$ se, e somente se, $\mathbf{S}^{(\xi')}(A) = \mathbf{v}$ para algum ξ' com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x;

A.2.3 Convenção Tipográfica para Interpretações

Quando se estuda a LPPo, é comum fazer referência às interpretações das fórmulas. Visando uma economia tipográfica, será usado `este tipo de letra` quando se quizer fazer tal alusão. Formalmente,

- $\mathbf{S}^{(\xi)}(t) \stackrel{\text{def}}{=} \mathtt{t}$
- $\mathbf{S}^{(\xi)}(f(t_1, \ldots, t_n)) \stackrel{\text{def}}{=} \mathtt{f}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$
- $\mathbf{S}^{(\xi)}(p(t_1, \ldots, t_n)) \stackrel{\text{def}}{=} \mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_n)$
- $\mathbf{S}^{(\xi)}(t_1 = t_2) \stackrel{\text{def}}{=} (\mathtt{t}_1 = \mathtt{t}_2)$
- $\mathbf{S}^{(\xi)}(x) \stackrel{\text{def}}{=} \mathtt{x}$

A.2.4 Definições

Definição A.2.4.1 Uma fórmula A da $\mathcal{L}_{LPP_o}$ é *válida em* $\mathbf{S}$, denotando-se por $\models_{\mathbf{S}} A$, se $\mathbf{S}^{(\xi)}(A) = \mathbf{v}$ para todo ξ.

Definição A.2.4.2 Uma fórmula A da $\mathcal{L}_{LPP_o}$ é *válida*, denotando-se por $\models A$, se $\models_{\mathbf{S}} A$ para todo $\mathbf{S}$.

Definição A.2.4.3 Uma fórmula A da $\mathcal{L}_{LPPo}$ *segue de um conjunto* $\mathcal{F}$ *de fórmulas*, se $\models_S A$ para todo **S** tal que $\models_S B$ para todo $B \in \mathcal{F}$.

A.2.5 O Sistema Formal Σ_{LPPo}

A LPPo é uma extensão feita à Lp. Na axiomatização da LPPo, usamos o Σ_{Lp}, o sistema formal da Lp, e acrescentamos alguns esquemas de axiomas e regras de inferência. Chamaremos o sistema formal para a LPPo abaixo de Σ_{LPPo}. As definições dadas na seção A.1.4 são também aplicáveis ao Σ_{LPPo}.

Axiomas

Todos os esquemas de axiomas **A1–A3** do Σ_{Lp}, acrescentando

A4. $\vdash \forall x A \supset A_x(t)$

A5. $\vdash x = x$

A6. $\vdash x = y \supset (A \supset A_x(y))$

Regras de Inferência

À regra de inferência **R1** do Σ_{Lp} acrescentamos

R2. Se $\vdash A \supset B$ então $\vdash A \supset \forall x B$ se não houver nenhuma ocorrência livre de x em A (*Inserção do* $\forall$ — $I\forall$)

A.2.6 Consistência e Completeza do Σ_{LPPo}

Afirmamos que:

O sistema Σ_{LPPo} *é consistente*

O sistema Σ_{LPP_0} é completo.

As provas dessas duas afirmações encontram-se em ENDERTON (1972, pp.124–139).

A.2.7 Uma Lista de Regras Derivadas Selecionadas

A notação usada nas regras derivadas é, na sua forma geral,

$$\frac{\vdash \varphi_1, \vdash \varphi_2, \ldots, \vdash \varphi_m}{\vdash \psi}$$

e significa que, se tivéssemos $\varphi_1, \varphi_2, \ldots, \varphi_m$ (as *hipóteses* da regra), então nos é permitido, por essa regra, inferir ψ (a *conclusão* da regra). Essas regras são chamadas derivadas porque sua aplicação pode ser substituída por uma seqüência apropriada de fórmulas usando apenas as regras de inferência do sistema dedutivo.

1. Regras de Hipótese

   a) Axioma de Hipótese

   $$\vdash A \supset A$$

   b) Introdução de Hipótese

   $$\frac{\vdash B}{\vdash A \supset B}$$

   c) Eliminação de Hipótese

   $$\frac{\vdash A \supset B, \vdash \sim A \supset B}{\vdash B}$$

2. Regras $\vee$

   a) Introdução do $\vee$

   i) $\dfrac{\vdash A}{\vdash A \vee B}$ ii) $\dfrac{\vdash B}{\vdash A \vee B}$

   b) Eliminação do $\vee$

   $$\frac{\vdash A \supset C, \vdash B \supset C, \vdash A \vee B}{\vdash C}$$

3. Regras $\wedge$

   a) Introdução do $\wedge$

   $$\frac{\vdash A, \vdash B}{\vdash A \wedge B}$$

b) Eliminação do $\wedge$

i) $\dfrac{\vdash A \wedge B}{\vdash A}$ ii) $\dfrac{\vdash A \wedge B}{\vdash B}$

4. Regras $\sim$

a) Introdução do $\sim$ (*Reductio ad absurdum*)

$$\frac{\vdash A \supset B, \vdash A \supset \sim B}{\vdash \sim A}$$

b) Eliminação do $\sim$

$$\frac{\vdash A, \vdash \sim A}{\vdash B}$$

5. Regras $\sim\sim$

a) Introdução do $\sim\sim$

$$\frac{\vdash A}{\vdash \sim\sim A}$$

b) Eliminação do $\sim\sim$

$$\frac{\vdash \sim\sim A}{\vdash A}$$

6. Regras $\equiv$

a) Introdução do $\equiv$

$$\frac{\vdash A \supset B, \vdash B \supset A}{\vdash A \equiv B}$$

b) Eliminação do $\equiv$

i) $\dfrac{\vdash A \equiv B}{\vdash A \supset B}$ ii) $\dfrac{\vdash A \equiv B}{\vdash B \supset A}$

7. Eliminação Generalizada do $\supset$

$$\frac{\vdash A_1, \ldots, \vdash A_n, \vdash A_1 \wedge \ldots \wedge A_n \supset B}{\vdash B}$$

8. *Modus Tollendo Ponens*

i) $\dfrac{\vdash A \vee B, \vdash \sim A}{\vdash B}$

ii) $\dfrac{\vdash A \vee B, \vdash \sim B}{\vdash A}$

9. *Modus Tollens*

$$\frac{\vdash A \supset B, \vdash \sim B}{\vdash \sim A}$$

10. Prova por Casos

$$\frac{\vdash A \supset C, \vdash B \supset C}{\vdash A \vee B \supset C}$$

11. Transitividade da Implicação

$$\frac{\vdash A \supset B, \vdash B \supset C}{\vdash A \supset C}$$

12. Transitividade da Equivalência

$$\frac{\vdash A \equiv B, \vdash B \equiv C}{\vdash A \equiv C}$$

13. Introdução do $\forall$

$$\frac{\vdash A(x)}{\vdash \forall x A(x)}$$

14. Regras do $\exists$

a) Introdução do $\exists$

$$\frac{\vdash A_x(t)}{\vdash \exists x A}$$

b) Eliminação do $\exists$

$$\frac{\vdash \exists x A, \vdash A_x(b) \supset C}{\vdash C}$$

onde b é uma constante individual não ocorrendo nem em $\exists x A$ nem em C

A.3 Símbolos

Alguns símbolos utilizados e seus respectivos significados são:

■ — símbolo indicando o fim de uma prova;

$\Rightarrow$ — meta-símbolo de conseqüência lógica, abrevia sentenças da forma "se...então..." para "...$\Rightarrow$...";

$\Leftarrow$ — meta-símbolo de implicação conversa;

$\Leftrightarrow$ — meta-símbolo significando "se e somente se", também denotado por "*sse*"

Apêndice B

Demonstrações da LTL

B.1 Validade dos Axiomas da LTL

Os esquemas de axiomas **A1—A16** dos sistemas Σ_{LTL_p} e Σ_{LTLP_o} (capítulo IV) têm a sua validade demonstrada nesta seção.

A1. $\vdash \sim\Diamond A \equiv \Box \sim A$

Prova: $\mathbf{K}_i(\sim\Diamond A) = \mathbf{v} \Leftrightarrow \mathbf{K}_i(\Diamond A) = \mathbf{f}$
$\Leftrightarrow \mathbf{K}_j(A) = \mathbf{f}$ para todo $j \geq i$
$\Leftrightarrow \mathbf{K}_j(\sim A) = \mathbf{v}$ para todo $j \geq i$
$\Leftrightarrow \mathbf{K}_i(\Box \sim A) = \mathbf{v}$

A2. $\vdash \Box(A \supset B) \supset (\Box A \supset \Box B)$

Prova: $\mathbf{K}_i(\Box(A \supset B)) = \mathbf{v} \Rightarrow \mathbf{K}_j(A \supset B) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow [\mathbf{K}_j(A) = \mathbf{f}$ ou $\mathbf{K}_j(B) = \mathbf{v}]$ para todo $j \geq 1$
$\Rightarrow \mathbf{K}_j(A) = \mathbf{f}$ para todo $j \geq i$ ou $\mathbf{K}_j(B) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_j(A) = \mathbf{f}$ para algum $j \geq i$ ou $\mathbf{K}_j(B) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_i(\Box A) = \mathbf{f}$ ou $\mathbf{K}_i(\Box B) = \mathbf{v}$
$\Rightarrow \mathbf{K}_i(\Box A \supset \Box B) = \mathbf{v}$

A3. $\vdash \Box A \supset A$

Prova: $\mathbf{K}_i(\Box A) = \mathbf{v} \Rightarrow \mathbf{K}_j(A) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_i(A) = \mathbf{v}$

A4. $\vdash \sim\circ A \equiv \circ \sim A$

Prova: $\mathbf{K}_i(\sim\circ A) = \mathbf{v} \Leftrightarrow \mathbf{K}_i(\circ A) = \mathbf{f}$
$\Leftrightarrow \mathbf{K}_{i+1}(A) = \mathbf{f}$
$\Leftrightarrow \mathbf{K}_{i+1}(\sim A) = \mathbf{v}$
$\Leftrightarrow \mathbf{K}_i(\circ \sim A) = \mathbf{v}$

A5. $\vdash \circ(A \supset B) \equiv \circ A \supset \circ B$

Prova: $\mathbf{K}_i(\circ(A \supset B)) = \mathbf{v} \Leftrightarrow \mathbf{K}_{i+1}(A \supset B) = \mathbf{v}$
$\Leftrightarrow \mathbf{K}_{i+1}(A) = \mathbf{f}$ ou $\mathbf{K}_{i+1}(B) = \mathbf{v}$
$\Leftrightarrow \mathbf{K}_i(\circ A) = \mathbf{f}$ ou $\mathbf{K}_i(\circ B) = \mathbf{v}$
$\Leftrightarrow \mathbf{K}_i(\circ A \supset \circ B) = \mathbf{v}$

A6. $\vdash \Box A \supset \circ A$

Prova: $\mathbf{K}_i(\Box A) = \mathbf{v} \Rightarrow \mathbf{K}_j(A) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_{i+1}(A) = \mathbf{v}$ (particularmente, $j = i + 1$)
$\Rightarrow \mathbf{K}_i(\circ A) = \mathbf{v}$

A7. $\vdash \Box A \supset \circ\Box A$

Prova: $\mathbf{K}_i(\Box A) = \mathbf{v} \Rightarrow \mathbf{K}_j(A) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_j(A) = \mathbf{v}$ para todo $j \geq i + 1$
$\Rightarrow \mathbf{K}_{i+1}(\Box A) = \mathbf{v}$
$\Rightarrow \mathbf{K}_i(\circ\Box A) = \mathbf{v}$

A8. $\vdash \Box(A \supset \circ A) \supset (A \supset \Box A)$

Prova: $\mathbf{K}_i(\Box(A \supset \circ A)) = \mathbf{v} \Rightarrow \mathbf{K}_j(A \supset \circ A) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow [\mathbf{K}_j(A) = \mathbf{f}$ ou $\mathbf{K}_j(\circ A) = \mathbf{v}]$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_j(A) = \mathbf{f}$ para todo $j \geq i$ ou $\mathbf{K}_{j+1}(A) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_i(A) = \mathbf{f}$ $(j = i)$ ou $\mathbf{K}_{j+1}(A) = \mathbf{v}$ para todo $j \geq i$
$\Rightarrow \mathbf{K}_i(A) = \mathbf{f}$ ou $[\mathbf{K}_i(A) = \mathbf{v}$ e $\mathbf{K}_{j+1}(A) = \mathbf{v}$ para todo $j \geq i]$
$\Rightarrow \mathbf{K}_i(A) = \mathbf{f}$ ou $\mathbf{K}_i(\Box A) = \mathbf{v}$
$\Rightarrow \mathbf{K}_i(A \supset \Box A) = \mathbf{v}$

A9. $\vdash A\mathcal{U}B \equiv B \vee (A \wedge \circ(A\mathcal{U}B))$

Prova: $\mathbf{K}_i(A\mathcal{U}B) = \mathbf{v} \Leftrightarrow \mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k, i \leq k < j$
$\Leftrightarrow [\mathbf{K}_i(B) = \mathbf{v}$ ou $\mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i+1]$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k, i \leq k < j$
$\Leftrightarrow \mathbf{K}_i(B) = \mathbf{v}$ ou $[\mathbf{K}_i(A) = \mathbf{v}$ e $[\mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i+1$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k, i + 1 \leq k < j]]$
$\Leftrightarrow \mathbf{K}_i(B) = \mathbf{v}$ ou $[\mathbf{K}_i(A) = \mathbf{v}$ e $\mathbf{K}_{i+1}(A\mathcal{U}B) = \mathbf{v}]$
$\Leftrightarrow \mathbf{K}_i(B) = \mathbf{v}$ ou $[\mathbf{K}_i(A) = \mathbf{v}$ e $\mathbf{K}_i(\circ(A\mathcal{U}B)) = \mathbf{v}]$
$\Leftrightarrow \mathbf{K}_i(B) = \mathbf{v}$ ou $\mathbf{K}_i(A \wedge \circ(A\mathcal{U}B))) = \mathbf{v}$
$\Leftrightarrow \mathbf{K}_i(B \vee (A \wedge \circ(A\mathcal{U}B))) = \mathbf{v}$

A10. $\vdash A\mathcal{U}B \supset \Diamond B$

Prova: $\mathbf{K}_i(A\mathcal{U}B) = \mathbf{v} \Rightarrow \mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo k, $i \leq k < j$
$\Rightarrow \mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i$
$\Rightarrow \mathbf{K}_i(\Diamond B) = \mathbf{v}$

A11. $\vdash \sim \exists x A \equiv \forall x \sim A$

Prova: $\mathbf{K}_i(\sim \exists x A) = \mathbf{v} \Leftrightarrow \mathbf{K}_i(\exists x A) = \mathbf{f}$

$\Leftrightarrow \mathbf{K}'_i(A) = \mathbf{f}$ para todo $\mathbf{K}'$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x

$\Leftrightarrow \mathbf{K}'_i(\sim A) = \mathbf{v}$ para todo $\mathbf{K}'$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x

$\Leftrightarrow \mathbf{K}_i(\forall x \sim A) = \mathbf{v}$

A12. $\vdash \forall x A \supset A_x(t)$ onde t é qualquer termo substituível em A.

Prova: $\mathbf{K}_i(\forall x A) = \mathbf{v} \Rightarrow \mathbf{K}'_i(A) = \mathbf{v}$ para todo $\mathbf{K}'$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x

$\Rightarrow \mathbf{K}'_i(A) = \mathbf{v}$ para $\mathbf{K}'$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \mathbf{S}^{i(i,s_i)}(t)$ e $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para y diferente de x

$\Rightarrow \mathbf{K}_i(A_x(t)) = \mathbf{K}'(A)$ para tal $\mathbf{K}'$

$\Rightarrow \mathbf{K}_i(A_x(t)) = \mathbf{v}$

A13. $\vdash \forall x \Box A \supset \Box \forall x A$

Prova: $\mathbf{K}_i(\forall x \Box A) = \mathbf{v} \Rightarrow \mathbf{K}'_j(A) = \mathbf{v}$ para todo $j \geq i$ e $\mathbf{K}'$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x

$\Rightarrow \mathbf{K}_j(\forall x A) = \mathbf{v}$ para todo $j \geq i$

$\Rightarrow \mathbf{K}_i(\Box \forall x A) = \mathbf{v}$

A14. $\vdash \forall x \circ A \supset \circ \forall x A$

Prova: $\mathbf{K}_i(\forall x \circ A) = \mathbf{v} \Rightarrow \mathbf{K}'_{i+1}(A) = \mathbf{v}$ para todo $\mathbf{K}'$ com $\xi'(y) \stackrel{\mathbf{S}}{=} \xi(y)$ para todo y diferente de x

$\Rightarrow \mathbf{K}_{i+1}(\forall x A) = \mathbf{v}$

$\Rightarrow \mathbf{K}_i(\circ forall x A) = \mathbf{v}$

A15. $\vdash x = x$

Prova: $\mathbf{K}_i(x = x) = \mathbf{v}$ desde que $\xi(x) \stackrel{\mathbf{S}}{=} \xi(x)$ e $s_i(x) \stackrel{\mathbf{S}}{=} s_i(x)$, respectivamente.

A16. $x = y \supset (A \supset A_x(y))$

Prova: $\mathbf{K}_i(x = y) = \mathbf{v}$ e $\mathbf{K}_i(A) = \mathbf{v} \Rightarrow \mathbf{K}_i(A_x(y)) = \mathbf{K}_i(A) = \mathbf{v}$

B.2 Equivalências dos Demais Operadores Temporais

Os operadores **atnext**, **while**, **unless** e **before** são demonstrados como equivalentes a fórmulas da LTL com o operador $\mathcal{U}$.

$A \textbf{atnext} B \equiv \circ(\sim B \mathcal{U} (A \wedge B) \vee \Box \sim B)$

Prova: $\mathbf{K}_i(A\mathbf{atnext}B) = \mathbf{v} \Leftrightarrow \mathbf{K}_j(B) = \mathbf{f}$ para todo $j > i$ ou $\mathbf{K}_k(A) = \mathbf{v}$ para o menor $k > i$ com $\mathbf{K}_k(B) = \mathbf{v}$

$\Leftrightarrow \mathbf{K}_j(B) = \mathbf{f}$ para todo $j > i$ ou [há um menor $k > i$ tal que $\mathbf{K}_k(A) = \mathbf{K}_k(B) = \mathbf{v}$]

$\Leftrightarrow$ [há um menor $k > i$ tal que $\mathbf{K}_k(A) = \mathbf{K}_k(B) = \mathbf{v}$ e $\mathbf{K}_j(B) = \mathbf{f}$ para todo j, $i < j < k$] ou $\mathbf{K}_j(B) = \mathbf{f}$ para todo $j > i$

$\Leftrightarrow$ [$\mathbf{K}_k(A) = \mathbf{v}$ e $\mathbf{K}_k(B) = \mathbf{v}$ para algum $k \geq i+1$ e $\mathbf{K}_j(B) = \mathbf{f}$ para todo j, $i+1 \leq j < k$] ou $\mathbf{K}_j(B) = \mathbf{f}$ para todo $j \geq i+1$

$\Leftrightarrow$ [$\mathbf{K}_k(A \wedge B) = \mathbf{v}$ para algum $k \geq i+1$ e $\mathbf{K}_j(\sim B) = \mathbf{v}$ para todo j, $i+1 \leq j < k$] ou $\mathbf{K}_j(\sim B) = \mathbf{v}$ para todo $j \geq i+1$

$\Leftrightarrow \mathbf{K}_{i+1}(\sim B \mathcal{U}(A \wedge B)) = \mathbf{v}$ ou $\mathbf{K}_{i+1}(\Box \sim B) = \mathbf{v}$

$\Leftrightarrow \mathbf{K}_{i+1}(\sim B \mathcal{U}(A \wedge B) \vee \Box \sim B) = \mathbf{v}$

$\Leftrightarrow \mathbf{K}_i(\circ(\sim B \mathcal{U}(A \wedge B) \vee \Box \sim B)) = \mathbf{v}$

$A\mathbf{while}B \equiv \circ((A \wedge B)\mathcal{U} \sim B \vee \Box A)$

Prova: $\mathbf{K}_i(A\mathbf{while}B) = \mathbf{v} \Leftrightarrow$ [$\mathbf{K}_j(B) = \mathbf{f}$ para algum $j > i$ e $\mathbf{K}_k(A) = \mathbf{K}_k(B) = \mathbf{v}$ para todo k, $i < k < j$] ou $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k > i$

$\Leftrightarrow$ [$\mathbf{K}_j(\sim B) = \mathbf{v}$ para algum $j \geq i+1$ e $\mathbf{K}_k(A) = \mathbf{v}$ e $\mathbf{K}_k(B) = \mathbf{v}$ para todo k, $i+1 \leq k < j$] ou $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k \geq i+1$

$\Leftrightarrow$ [$\mathbf{K}_j(\sim B) = \mathbf{v}$ para algum $j \geq i+1$ e $\mathbf{K}_k(A \wedge B) = \mathbf{v}$ para todo k, $i+1 \leq k < j$] ou $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k \geq i+1$

$\Leftrightarrow \mathbf{K}_{i+1}((A \wedge B)\mathcal{U} \sim B) = \mathbf{v}$ ou $\mathbf{K}_{i+1}(\Box A) = \mathbf{v}$

$\Leftrightarrow \mathbf{K}_{i+1}((A \wedge B)\mathcal{U} \sim B \vee \Box A) = \mathbf{v}$

$\Leftrightarrow \mathbf{K}_i(\circ((A \wedge B)\mathcal{U} \sim B \vee \Box A)) = \mathbf{v}$

$A\mathbf{unless}B \equiv \circ(A\mathcal{U}B \vee \Box A)$

Prova: $\mathbf{K}_i(A\mathbf{unless}B) = \mathbf{v} \Leftrightarrow$ [$\mathbf{K}_j(B) = \mathbf{v}$ para algum $j > i$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo k, $i < k < j$] ou $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k > i$

$\Leftrightarrow$ [$\mathbf{K}_j(B) = \mathbf{v}$ para algum $j \geq i+1$ e $\mathbf{K}_k(A) = \mathbf{v}$ para todo k, $i+1 \leq k < j$] ou $\mathbf{K}_k(A) = \mathbf{v}$ para todo $k \geq i+1$

$\Leftrightarrow \mathbf{K}_{i+1}(A\mathcal{U}B) = \mathbf{v}$ ou $\mathbf{K}_{i+1}(\Box A) = \mathbf{v}$

$\Leftrightarrow \mathbf{K}_{i+1}(A\mathcal{U}B \vee \Box A) = \mathbf{v}$

$\Leftrightarrow \mathbf{K}_i(\circ(A\mathcal{U}B \vee \Box A)) = \mathbf{v}$

$A\mathbf{before}B \equiv \circ(\sim(A \vee B)\mathcal{U}(A \wedge \sim B) \vee \Box \sim(A \vee B))$

Prova: $\mathbf{K}_i(A\,\mathbf{before}\,B) = \mathbf{v}$ $\Leftrightarrow$ para todo $j > i$ com $\mathbf{K}_j(B) = \mathbf{v}$ há algum k, $i < k < j$ com $\mathbf{K}_k(A) = \mathbf{v}$

$\Leftrightarrow$ [$\mathbf{K}_j(A) = \mathbf{K}_j(B) = \mathbf{f}$ para todo $j > i$] ou [$\mathbf{K}_j(A) = \mathbf{v}$ e $\mathbf{K}_j(B) = \mathbf{f}$ para algum $j > i$ e $\mathbf{K}_k(A) = \mathbf{f}$ e $\mathbf{K}_k(B) = \mathbf{f}$ para todo k, $i < k < j$]

$\Leftrightarrow$ [$\mathbf{K}_j(A) = \mathbf{v}$ e $\mathbf{K}_j(\sim B) = \mathbf{v}$ para algum $j \geq i+1$ e $\mathbf{K}_k(\sim A) = \mathbf{v}$ e $\mathbf{K}_k(\sim B) = \mathbf{v}$ para todo k, $i+1 \leq k < j$] ou [$\mathbf{K}_j(\sim A) = \mathbf{v}$ e $\mathbf{K}_j(B) = \mathbf{v}$ para todo $j \geq i+1$]

$\Leftrightarrow$ [$\mathbf{K}_j(A \wedge \sim B) = \mathbf{v}$ para algum $j \geq i+1$ e $\mathbf{K}_k(\sim A \wedge \sim B) = \mathbf{v}$ para todo k, $i+1 \leq k < j$] ou [$\mathbf{K}_j(\sim A \wedge \sim B) = \mathbf{v}$ para todo $j \geq i+1$]

$\Leftrightarrow$ [$\mathbf{K}_j(A \wedge \sim B) = \mathbf{v}$ para algum $j \geq i+1$ e $\mathbf{K}_k(\sim (A \vee B)) = \mathbf{v}$ para todo k, $i+1 \leq k < j$] ou [$\mathbf{K}_j(\sim (A \vee B) = \mathbf{v}$ para todo $j \geq i+1$]

$\Leftrightarrow$ $\mathbf{K}_{i+1}(\sim (A \vee B) \mathcal{U} (A \wedge \sim B)) = \mathbf{v}$ ou $\mathbf{K}_{i+1}(\Box \sim (A \vee B)) = \mathbf{v}$

$\Leftrightarrow$ $\mathbf{K}_{i+1}(\sim (A \vee B) \mathcal{U} (A \wedge \sim B) \vee \Box \sim (A \vee B)) = \mathbf{v}$

$\Leftrightarrow$ $\mathbf{K}_i(\bigcirc(\sim (A \vee B) \mathcal{U} (A \wedge \sim B) \vee \Box \sim (A \vee B))) = \mathbf{v}$

Apêndice C

Demonstrações da LTTR

C.1 Validade dos Axiomas do Σ_{LTTR}

Nesta seção, demonstramos a validade de cada esquema de axioma **A1–A6** do sistema Σ_{LTTR}.

A1. $\vdash \forall_c \sim\psi \equiv \sim\exists_c\psi$

Prova: $\mathbf{R}_c(\forall_c\sim\psi) = \mathbf{v} \Leftrightarrow \mathbf{K}_s(\forall_c\sim\psi) = \mathbf{v}$, onde $s = primeiro(c)$
$\Leftrightarrow \mathbf{R}_d(\sim\psi) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$
$\Leftrightarrow \mathbf{R}_d(\psi) = \mathbf{f}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$
$\Leftrightarrow \mathbf{K}_s(\exists_c\psi) = \mathbf{f}$ onde $s = primeiro(c)$
$\Leftrightarrow \mathbf{K}_s(\sim\exists_c\psi) = \mathbf{v}$ onde $s = primeiro(c)$
$\Leftrightarrow \mathbf{R}_c(\sim\exists_c\psi) = \mathbf{v}$

A2. $\vdash \forall_c\psi \supset \psi$

Prova: $\mathbf{R}_c(\forall_c\psi) = \mathbf{v} \Rightarrow \mathbf{K}_s(\forall_c\psi) = \mathbf{v}$, onde $s = primeiro(c)$
$\Rightarrow \mathbf{R}_d(\psi) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$
$\Rightarrow \mathbf{R}_e(\psi) = \mathbf{v}$ onde $primeiro(e) = s$ e $s = primeiro(c)$
$\Rightarrow \mathbf{R}_c(\psi) = \mathbf{v}$

A3. $\vdash \psi \supset \exists_c\psi$

Prova: $\mathbf{R}_c(\psi) = \mathbf{v} \Rightarrow \mathbf{R}_d(\psi) = \mathbf{v}$ para algum $d \in \mathbf{C}(d = c)$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$
$\Rightarrow \mathbf{K}_s(\exists_c\psi) = \mathbf{v}$ onde $s = primeiro(c)$
$\Rightarrow \mathbf{R}_c(\exists_c\psi) = \mathbf{v}$

A4. $\vdash \forall_c(\psi_1 \wedge \psi_2) \equiv \forall_c\psi_1 \wedge \forall_c\psi_2$

Prova: $\mathbf{R}_c(\forall_c(\psi_1 \wedge \psi_2)) = \mathbf{v} \Leftrightarrow \mathbf{K}_s(\forall_c(\psi_1 \wedge \psi_2)) = \mathbf{v}$, onde $s = primeiro(c)$

$\Leftrightarrow \mathbf{R}_d(\psi_1 \wedge \psi_2) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Leftrightarrow [\mathbf{R}_d(\psi_1) = \mathbf{v}$ e $\mathbf{R}_d(\psi_2) = \mathbf{v}]$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Leftrightarrow [\mathbf{R}_d(\psi_1) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)]$ e $[\mathbf{R}_d(\psi_2) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)]$

$\Leftrightarrow \mathbf{K}_s(\forall_c\psi_1) = \mathbf{v}$, onde $s = primeiro(c)$ e $\mathbf{K}_s(\forall_c\psi_2) = \mathbf{v}$, onde $s = primeiro(c)$

$\Leftrightarrow \mathbf{R}_c(\forall_c\psi_1) = \mathbf{v}$ e $\mathbf{R}_c(\forall_c\psi_2) = \mathbf{v}$

$\Leftrightarrow \mathbf{R}_c(\forall_c\psi_1 \wedge \forall_c\psi_2) = \mathbf{v}$

A5. $\vdash \forall_c(\psi_1 \supset \psi_2) \supset (\forall_c\psi_1 \supset \forall_c\psi_2)$

Prova: $\mathbf{R}_c(\forall_c(\psi_1 \supset \psi_2)) = \mathbf{v} \Rightarrow \mathbf{K}_s(\forall_c(\psi_1 \supset \psi_2)) = \mathbf{v}$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_d(\psi_1 \supset \psi_2) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow [\mathbf{R}_d(\psi_1) = \mathbf{f}$ ou $\mathbf{R}_d(\psi_2) = \mathbf{v}]$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow [\mathbf{R}_d(\psi_1) = \mathbf{f}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)]$ ou $[\mathbf{R}_d(\psi_2) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)]$

$\Rightarrow \mathbf{K}_s(\forall_c\psi_1) = \mathbf{f}$, onde $s = primeiro(c)$ ou $\mathbf{K}_s(\forall_c\psi_2) = \mathbf{v}$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_c(\forall_c\psi_1) = \mathbf{f}$ ou $\mathbf{R}_c(\forall_c\psi_2) = \mathbf{v}$

$\Rightarrow \mathbf{R}_c(\forall_c\psi 1 \supset \forall_c\psi_2) = \mathbf{v}$

A6. $\vdash \forall_c\Box\psi \supset \forall_c\Box\forall_c\Box\psi$

Prova: $\mathbf{R}_c(\forall_c\Box\psi) = \mathbf{v} \Rightarrow \mathbf{K}_s(\forall_c\Box\psi) = \mathbf{v}$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_d(\Box\psi) = \mathbf{v}$ para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_e(\psi) = \mathbf{v}$ onde $e = d^i$ para todo $i \geq 0$, para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_f(\psi) = \mathbf{v}$ onde $f = e^j$ para todo $j \geq 0$, onde $e = d^i$ para todo $i \geq 0$, para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_f(\psi) = \mathbf{v}$ onde $f = e^j$ para todo $j \geq 0$, para todo $g \in \mathbf{C}$ tal que $primeiro(g) = t$, onde $t = primeiro(e)$ onde $e = d^i$ para todo $i \geq 0$, para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_g(\Box\psi) = \mathbf{v}$, para todo $g \in \mathbf{C}$ tal que $primeiro(g) = t$, onde $t = primeiro(e)$ onde $e = d^i$ para todo $i \geq 0$, para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{K}_t(\forall_c\Box\psi) = \mathbf{v}$, onde $t = primeiro(\epsilon)$ onde $e = d^i$ para todo $i \geq 0$, para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_c(\forall_c\Box\psi) = \mathbf{v}$, onde $\epsilon = d^i$ para todo $i \geq 0$, para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_d(\Box\forall_c\Box\psi) = \mathbf{v}$, para todo $d \in \mathbf{C}$ tal que $primeiro(d) = s$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{K}_s(\forall_c\Box\forall_c\Box\psi) = \mathbf{v}$, onde $s = primeiro(c)$

$\Rightarrow \mathbf{R}_c(\forall_c\Box\forall_c\Box\psi) = \mathbf{v}$

C.2 Demonstrações das Regras Derivadas

Nessa seção, demonstramos as regras derivadas exibidas no capítulo V.

1. *RP — Raciocinio Proposicional*

$$\frac{\vdash (\psi_1 \wedge \ldots \wedge \psi_n) \supset \phi \quad \vdash \psi_1, \ldots, \vdash \psi_n}{\vdash \phi}$$

Prova:

1. $\vdash [(\psi_1 \wedge \ldots \wedge \psi_n) \supset \phi] \supset$ $[\psi_1 \supset (\ldots(\psi_n \supset \phi)\ldots)]$	*TP*
2. $\vdash (\psi_1 \wedge \ldots \wedge \psi_n) \supset \phi$	dado
3. $\vdash \psi_1 \supset (\ldots(\psi_n \supset \phi)\ldots)$	*MP*, 1 e 2
4. $\vdash \psi_1$	dado
5. $\vdash (\ldots(\psi_n \supset \phi)\ldots)$	*MP*, 3 e 4
$\vdots$	
$n+4$. $\vdash \phi$	*MP*, $n+2$ e $n+3$

2. *Regras* $\forall_c\forall_c$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \forall_c\psi_1 \supset \forall_c\psi_2}$

Prova:

1. $\vdash \psi_1 \supset \psi_2$	dado
2. $\vdash \forall_c(\psi_1 \supset \psi_2)$	$I\forall_c$, 1
3. $\vdash \forall_c(\psi_1 \supset \psi_2) \supset (\forall_c\psi_1 \supset \forall_c\psi_2)$	**A5**
4. $\vdash \forall_c\psi_1 \supset \forall_c\psi_2$	*MP*, 2 e 3

b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \forall_c\psi_1 \equiv \forall_c\psi_2}$

Prova:

1. $\vdash \psi_1 \equiv \psi_2$	dado
2. $\vdash \psi_1 \supset \psi_2$	*RP*
3. $\vdash \forall_c\psi_1 \supset \forall_c\psi_2$	$\forall_c\forall_c$a)
4. $\vdash \psi_2 \supset \psi_1$	*RP*
5. $\vdash \forall_c\psi_2 \supset \forall_c\psi_1$	$\forall_c\forall_c$a)
6. $\vdash \forall_c\psi_1 \equiv \forall_c\psi_2$	*RP*, 3 e 5

3. *Regras* $\forall_c\Box$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \forall_c\Box\psi_1 \supset \forall_c\Box\psi_2}$

Prova:

1. $\vdash \psi_1 \supset \psi_2$	dado
2. $\vdash \Box(\psi_1 \supset \psi_2)$	$I\Box$, 1
3. $\vdash \Box(\psi_1 \supset \psi_2) \supset (\Box\psi_1 \supset \Box\psi_2)$	*VT*
4. $\vdash \Box\psi_1 \supset \Box\psi_2$	*MP*, 2 e 3
5. $\vdash \forall_c(\Box\psi_1 \supset \Box\psi_2)$	$I\forall_c$, 4
6. $\vdash \forall_c(\Box\psi_1 \supset \Box\psi_2) \supset (\forall_c\Box\psi_1 \supset \forall_c\Box\psi_2)$	**A5**
7. $\vdash \forall_c\Box\psi_1 \supset \forall_c\Box\psi_2$	*MP*, 5 e 6

b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \forall_c\Box\psi_1 \equiv \forall_c\Box\psi_2}$

Prova:

1. $\vdash \psi_1 \equiv \psi_2$	dado
2. $\vdash \psi_1 \supset \psi_2$	*RP*
3. $\vdash \forall_c\Box\psi_1 \supset \forall_c\Box\psi_2$	$\forall_c\Box$a), 2
4. $\vdash \psi_2 \supset \psi_1$	*RP*
5. $\vdash \forall_c\Box\psi_2 \supset \forall_c\Box\psi_1$	$\forall_c\Box$a), 4
6. $\vdash \forall_c\Box\psi_1 \equiv \forall_c\Box\psi_2$	*RP*, 3 e 5

4. *Regras* $\exists_c\exists_c$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \exists_c\psi_1 \supset \exists_c\psi_2}$

Prova:

1. $\vdash \psi_1 \supset \psi_2$	dado
2. $\vdash \sim\psi_2 \supset \sim\psi_1$	*RP*
3. $\vdash \forall_c \sim\psi_2 \supset \forall_c \sim\psi_1$	$\forall_c\forall_c$
4. $\vdash \sim\forall_c \sim\psi_1 \supset \sim\forall_c \sim\psi_2$	*RP*
5. $\vdash \exists_c\psi_1 \supset \exists_c\psi_2$	*RP* e **A1**

b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \exists_c\psi_1 \equiv \exists_c\psi_2}$

Prova:

1. $\vdash \psi_1 \equiv \psi_2$	dado
2. $\vdash \psi_1 \supset \psi_2$	*RP*
3. $\vdash \exists_c\psi_1 \supset \exists_c\psi_2$	$\exists_c\exists_c$a)
4. $\vdash \psi_2 \supset \psi_1$	*RP*
5. $\vdash \exists_c\psi_2 \supset \exists_c\psi_1$	$\exists_c\exists_c$a)
6. $\vdash \exists_c\psi_1 \equiv \exists_c\psi_2$	*RP*, 3 e 5

5. *Regras* $\exists_c\Diamond$

a) $\dfrac{\vdash \psi_1 \supset \psi_2}{\vdash \exists_c\Diamond\psi_1 \supset \exists_c\Diamond\psi_2}$

Prova:

1. $\vdash \psi_1 \supset \psi_2$	dado
2. $\vdash \sim\psi_2 \supset \sim\psi_1$	*RP*
3. $\vdash \forall_c\Box \sim\psi_2 \supset \forall_c\Box \sim\psi_1$	*RP*
4. $\vdash \sim\exists_c\Diamond\psi_2 \supset \sim\exists_c\Diamond\psi_1$	**A1** e *RP*
5. $\vdash \exists_c\Diamond\psi_1 \supset \exists_c\Diamond\psi_2$	*RP*

b) $\dfrac{\vdash \psi_1 \equiv \psi_2}{\vdash \exists_c\Diamond\psi_1 \equiv \exists_c\Diamond\psi_2}$

Prova:

1. $\vdash \psi_1 \equiv \psi_2$	dado
2. $\vdash \psi_1 \supset \psi_2$	*RP*
3. $\vdash \exists_c\Diamond\psi_1 \supset \exists_c\Diamond\psi_2$	$\exists_c\Diamond$a), 2
4. $\vdash \psi_2 \supset \psi_1$	*RP*
5. $\vdash \exists_c\Diamond\psi_2 \supset \exists_c\Diamond\psi_1$	$\exists_c\Diamond$a), 4
6. $\vdash \exists_c\Diamond\psi_1 \equiv \exists_c\Diamond\psi_2$	*RP*, 3 e 5

C.3 Demonstrações dos Teoremas

Nessa seção, demonstramos os teoremas do Σ_{LTTR} mostrados no capítulo V.

T1. $\vdash \exists_c(\psi_1 \wedge \psi_2) \supset \exists_c\psi_1 \wedge \exists_c\psi_2$

Prova:

1. $\vdash (\psi_1 \wedge \psi_2) \supset \psi_1$	*RP*
2. $\vdash \exists_c(\psi_1 \wedge \psi_2) \supset \exists_c\psi_1$	$\exists_c\exists_c$
3. $\vdash (\psi_1 \wedge \psi_2) \supset \psi_2$	*RP*
4. $\vdash \exists_c(\psi_1 \wedge \psi_2) \supset \exists_c\psi_2$	$\exists_c\exists_c$
5. $\vdash \exists_c(\psi_1 \wedge \psi_2) \supset \exists_c\psi_1 \wedge \exists_c\psi_2$	*RP*

T2. $\vdash \forall_c\Box \sim\psi \equiv\sim\exists_c\Diamond\psi$

Prova:

1. $\vdash \Diamond\psi \equiv \Diamond\psi$	*VT*
2. $\vdash \exists_c\Diamond\psi \equiv \exists_c\Diamond\psi$	$\exists_c\exists_c$
3. $\vdash\sim\exists_c\Diamond\psi \equiv\sim\exists_c\Diamond\psi$	*RP*
4. $\vdash \forall_c \sim\Diamond\psi \equiv\sim\exists_c\Diamond\psi$	**A1**, *RP*
5. $\vdash \forall_c\Box \sim\psi \equiv\sim\exists_c\Diamond\psi$	*RP*, **R2**

T3. $\vdash \forall_c\Box\psi \supset \exists_c\Diamond\psi$

Prova:

1. $\vdash \Box\psi \supset \Diamond\psi$	*VT*
2. $\vdash \forall_c\Box\psi \supset \forall_c\Diamond\psi$	$\forall_c\forall_c$
3. $\vdash \forall_c\Diamond\psi \supset \Diamond\psi$	**A2**
4. $\vdash \Diamond\psi \supset \exists_c\Diamond\psi$	**A3**
5. $\vdash \forall_c\Box\psi \supset \exists_c\Diamond\psi$	*RP*, 2,3 e 4

T4. $\vdash \forall_c\Box\psi \supset \forall_c\Diamond\psi$

Prova:

1. $\vdash \Box\psi \supset \Diamond\psi$	*VT*
2. $\vdash \forall_c\Box\psi \supset \forall_c\Diamond\psi$	$\forall_c\forall_c$

T5. $\vdash \forall_c\circ\psi \supset \exists_c\circ\psi$

Prova:

1. $\vdash \circ\psi \supset \circ\psi$	*VT*
2. $\vdash \forall_c\circ\psi \supset \forall_c\circ\psi$	$\forall_c\forall_c$
3. $\vdash \forall_c\circ\psi \supset \circ\psi$	**A2**
4. $\vdash \circ\psi \supset \exists_c\circ\psi$	**A3**
5. $\vdash \forall_c\circ\psi \supset \exists_c\circ\psi$	*RP*, 2,3 e 4

T6. $\vdash \forall_c\Box(\psi_1 \wedge \psi_2) \equiv \forall_c\Box\psi_1 \wedge \forall_c\Box\psi_2$

Prova:

1. $\vdash \Box(\psi_1 \wedge \psi_2) \equiv \Box\psi_1 \wedge \Box\psi_2$	*VT*
2. $\vdash \forall_c\Box(\psi_1 \wedge \psi_2) \equiv \forall_c(\Box\psi_1 \wedge \Box\psi_2)$	$\forall_c\forall_c$
3. $\vdash \forall_c\Box(\psi_1 \wedge \psi_2) \equiv \forall_c\Box\psi_1 \wedge \forall_c\Box\psi_2$	*RP*, **A4**

T7. $\vdash \exists_c\Box(\psi_1 \wedge \psi_2) \supset \exists_c\Box\psi_1 \wedge \exists_c\Box\psi_2$

Prova:

1. $\vdash \Box(\psi_1 \wedge \psi_2) \supset \Box\psi_1 \wedge \Box\psi_2$ — *VT*
2. $\vdash \exists_c\Box(\psi_1 \wedge \psi_2) \supset \exists_c(\Box\psi_1 \wedge \Box\psi_2)$ — $\exists_c\exists_c$
3. $\vdash \exists_c\Box(\psi_1 \wedge \psi_2) \supset \exists_c\Box\psi_1 \wedge \exists_c\Box\psi_2$ — *RP*, **T1**

T8. $\vdash \forall_c\Box(\psi_1 \supset \psi_2) \supset (\forall_c\Box\psi_1 \supset \forall_c\Box\psi_2)$

Prova:

1. $\vdash \Box(\psi_1 \supset \psi_2) \supset (\Box\psi_1 \supset \Box\psi_2)$ — *VT*
2. $\vdash \forall_c\Box(\psi_1 \supset \psi_2) \supset \forall_c(\Box\psi_1 \supset \Box\psi_2)$ — $\forall_c\forall_c$
3. $\vdash \forall_c\Box(\psi_1 \supset \psi_2) \supset (\forall_c\Box\psi_1 \supset \forall_c\Box\psi_2)$ — *RP*, **A5**

T9. $\vdash \forall_c\circ(\psi_1 \supset \psi_2) \supset (\forall_c\circ\psi_1 \supset \forall_c\circ\psi_2)$

Prova:

1. $\vdash \circ(\psi_1 \supset \psi_2) \supset (\circ\psi_1 \supset \circ\psi_2)$ — *VT*
2. $\vdash \forall_c\circ(\psi_1 \supset \psi_2) \supset \forall_c(\circ\psi_1 \supset \circ\psi_2)$ — $\forall_c\forall_c$
3. $\vdash \forall_c\circ(\psi_1 \supset \psi_2) \supset (\forall_c\circ\psi_1 \supset \forall_c\circ\psi_2)$ — *RP*, **A5**

T10. $\vdash \forall_c\Box\forall_c\Box\psi \equiv \forall_c\Box\psi$

Prova:

1. $\vdash \psi \supset \psi$ — *VT*
2. $\vdash \forall_c\Box\psi \supset \forall_c\Box\psi$ — $\forall_c\Box$
3. $\vdash \Box(\forall_c\Box\psi \supset \forall_c\Box\psi)$ — *I*$\Box$
4. $\vdash \Box\forall_c\Box\psi \supset \Box\forall_c\Box\psi$ — *RP*
5. $\vdash \Box\forall_c\Box\psi \supset \forall_c\Box\psi$ — *VT*
6. $\vdash \forall_c\Box\forall_c\Box\psi \supset \forall_c\forall_c\Box\psi$ — *VT*
7. $\vdash \forall_c\forall_c\Box\psi \supset \forall_c\Box\psi$ — **A2**
8. $\vdash \forall_c\Box\forall_c\Box\psi \supset \forall_c\Box\psi$ — *RP*
9. $\vdash \forall_c\Box\psi \supset \forall_c\Box\forall_c\Box\psi$ — **A6**
10. $\vdash \forall_c\Box\forall_c\Box\psi \equiv \forall_c\Box\psi$ — *RP*

T11. $\vdash \exists_c\Diamond\exists_c\Diamond\psi \equiv \exists_c\Diamond\psi$

Prova:

1. $\vdash \Diamond\psi \supset \Diamond\psi$ — *VT*
2. $\vdash \exists_c\Diamond\psi \supset \exists_c\Diamond\psi$ — $\exists_c\exists_c$
3. $\vdash \exists_c\Diamond\psi \supset \Diamond\exists_c\Diamond\psi$ — *VT*
4. $\vdash \exists_c\Diamond\psi \supset \exists_c\Diamond\exists_c\Diamond\psi$ — **A3**
5. $\vdash \sim\exists_c\Diamond\psi \supset \sim\exists_c\Diamond\psi$ — *RP*, 2
6. $\vdash \Box \sim\exists_c\Diamond\psi \supset \Box \sim\exists_c\Diamond\psi$ — *I*$\Box$ e *VT*
7. $\vdash \Box \sim\exists_c\Diamond\psi \supset \sim\exists_c\Diamond\psi$ — *VT*
8. $\vdash \forall_c\Box \sim\exists_c\Diamond\psi \supset \forall_c \sim\exists_c\Diamond\psi$ — $\forall_c\forall_c$
9. $\vdash \forall_c\Box \sim\exists_c\Diamond\psi \supset \sim\exists_c\Diamond\psi$ — *RP*, **A2**
10. $\vdash \sim\exists_c\Diamond\exists_c\Diamond\psi \supset \sim\exists_c\Diamond\psi$ — *RP*, **T1**
11. $\vdash \exists_c\Diamond\exists_c\Diamond\psi \supset \exists_c\Diamond\psi$ — *RP*
12. $\vdash \exists_c\Diamond\exists_c\Diamond\psi \equiv \exists_c\Diamond\psi$ — *RP*, 4 e 11

T12. $\vdash (\forall_c\Box\psi_1 \wedge \exists_c\Diamond\psi_2) \supset \exists_c\Diamond(\psi_1 \wedge \psi_2)$

Prova:

1. $\vdash \forall_c\Box(\psi_1 \supset \sim\psi_2) \supset (\forall_c\Box\psi_1 \supset \forall_c\Box \sim\psi_2)$ **A5**
2. $\vdash \forall_c\Box \sim(\psi_1 \wedge \psi_2) \supset \sim(\forall_c\Box\psi_1 \wedge \sim\forall_c\Box \sim\psi_2)$ *TP*
3. $\vdash \sim\exists_c\Diamond(\psi_1 \wedge \psi_2) \supset \sim(\forall_c\Box\psi_1 \wedge \exists_c\Diamond\psi_2)$ **A1** e *RP*
4. $\vdash (\forall_c\Box\psi_1 \wedge \exists_c\Diamond\psi_2) \supset \exists_c\Diamond(\psi_1 \wedge \psi_2)$ *RP*