UNIVERSIDADE FEDERAL DE SANTA CATARINA PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

UMÁ PROPOSTA DE SISTEMA BASEADO NO CONHECIMENTO PARA APLICAÇÕES TEMPO-REAL UTILIZANDO O ENFOQUE SÍNCRONO

TESE SUBMETIDA À UNIVERSIDADE FEDERAL DE SANTA CATARINA PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS, ÁREA ENGENHARIA ELÉTRICA, ÁREA DE CONCENTRAÇÃO SISTEMAS DE INFORMAÇÃO.

Celso Antônio ALVES KAESTNER

Florianópolis, 20 de outubro de 1993.

UMA PROPOSTA DE SISTEMA BASEADO NO CONHECIMENTO PARA APLICAÇÕES TEMPO-REAL UTILIZANDO O ENFOQUE SÍNCRONO

Celso Antônio Alves Kaestner

ESTA TESE FOI JULGADA ADEQUADA PARA A OBTENÇÃO DO TÍTULO DE

DOUTOR EM CIÊNCIAS

ÁREA ENGENHARIA ELÉTRICA, ÁREA DE CONCENTRAÇÃO SISTEMAS DE INFORMAÇÃO, E APROVADA EM SUA FORMA FINAL PELO PROGRAMA DE PÓS-GRADUAÇÃO.

já tem

JEAN-MARIE FARINES, Dr.Ing.
(Orientador)

ROBERTO DE SOUZA SALGADO, Ph.D.
(Coordenador do Curso)

BANCA EXAMINADORA

JEAN-MARIE FARINES, Dr.Ing.
(Presidente)

SHEILA REGINA MURGEL VELOSO, D.Sc.
(Relator)

GUILHERME BITTENCOURT, Dr.Rer.Nat.

JONI DA SILVA FRAGA, Dr.

JOSÉ EDUARDO CURY, Dr. d'État

MALIK GHALLAB, Dr. d'État

A meus pais Dalila e Egon

Agradecimentos

Agradeço à Capes pelo suporte financeiro recebido em forma de bolsa ao longo do curso de doutorado.

Ao Centro Federal de Educação Tecnológica do Paraná e à Pontifícia Universidade Católica do Paraná, pelo integral apoio.

Aos professores do Curso de Pós-graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina, pelos ensinamentos recebidos.

Ao Dr. Malik Ghallab, e aos demais pesquisadores do *Laboratoire d'Automatique et d'Analyse des Systèmes*, em Toulouse, pela acolhida durante meu estágio neste laboratório.

Ao pessoal do Laboratório de Controle e Microinformática da Universidade Federal de Santa Catarina, onde mais que colegas, encontrei amigos.

Ao meu orientador, Prof. Jean-Marie Farines, pela orientação, interesse e dedicação durante o desenvolvimento deste trabalho.

Aos amigos de Florianópolis, de Toulouse e de Curitiba pela cumplicidade.

Às minhas filhas Melissa, Camila e Tatiana, pela compreensão nos momentos em que estive ausente.

À Cláudia, pelo amor, pelo companheirismo, e por tornar tão agradável esta etapa de minha vida.

Conteúdo

Sumário

A utilização de técnicas de Inteligência Artificial em Sistemas Tempo-Real vem se tornando uma necessidade devido a crescente complexidade dos sistemas envolvidos. No entanto muitas das técnicas até agora utilizadas não contemplam aspectos primordiais referentes ao problema do Tempo-Real: o determinismo e reatividade que devem caracterizar as aplicações, e a garantia de atendimento às restrições temporais impostas.

Este trabalho apresenta uma proposta para a geração de Sistemas Baseados no Conhecimento para aplicações Tempo-Real. O sistema é fundamentado em um formalismo bem definido, construído sobre uma extensão da lógica temporal proposicional. O caráter paraconsistente desta lógica permite o tratamento automatizado de questões relativas à inconsistência e incompletude da Base de Conhecimentos do sistema, bem como a representação de informações imprecisas e incertas.

Para a Representação do Conhecimento utiliza-se o paradigma "Sistema Baseado em Regras". As regras são posteriormente transformadas em um conjunto de fórmulas lógicas, e sua aplicação é então feita a partir dos valores de um conjunto finito de atributos.

O enfoque temporal utilizado é síncrono, no qual o agente computacional deve responder instantaneamente aos estímulos provenientes do ambiente. O tempo é considerado diretamente a partir da ocorrência de eventos que promovem a comunicação entre o ambiente e o agente computacional.

A correspondência entre o formalismo proposto e uma classe restrita de Sistemas Dinâmicos é estabelecida. Isto permite a inclusão natural de outras formas de representação, tais como funções e sistemas dinâmicos qualitativos.

Uma metodologia e um ambiente para o desenvolvimento de sistemas também são propostos. A obtenção de um mecanismo de execução eficiente (autômato) é feita através do pré-tratamento prévio e completo de toda a Base de Conhecimentos do sistema. Foram desenvolvidos algoritmos e estruturas de dados para uma implementação desta sistemática.

Entre as aplicações visadas por este trabalho incluem-se especialmente as áreas de Controle de Processos, Automação Industrial e Robótica.

Abstract

The use of Artificial Intelligence techniques in Real-Time Systems becames more and more a necessity due to the increasing complexity of these systems. Meanwhile, the current techniques usually do not address important features of these systems: determinism, reactivity and the satisfaction of the required temporal conditions.

This work presents a new approach for real-time rule-based systems generation. The system is substantiated in a rigorous formalism, constructed over an extention of the propositional temporal logic. The paraconsistent character of this logic permits the automatizated treatment of some questions about incomplete and inconsistent information in the system knowledge-base, as well as the representation of incertain / imprecise information.

The paradigm "Rule-Based Production System" is used for knowledge representation. Afterwards the rules are translated as logical formulas and its application is based on the values of a finite set of attributs.

The synchronous temporal approach is used, which considers that the computational agent must react instantaneously to the stimulus received from the environment. Time is considered simply as a sequence of the events that promote the communication between the environment and the computational agent.

The relation between the proposed formalism and a restricted class of Dynamical Systems is established. It permits the easy inclusion of others forms of knowledge representation, as qualitative functions and dynamical systems.

A design methodology and a developping environment are also proposed. The obtention of a efficient execution mechanism (automata) is done by a complete pretreatment of the entire knowledge-base of the system. Data structures and algorithms to implement the proposed strategy were also developed.

The main areas of application of the system are: Process Control, Industrial Automation and Robotics.

Capítulo 1

Introdução

A construção de ferramentas de programação fundamentadas em modelos matemáticos formais constitui uma preocupação crescente em Informática. Têm sido exemplos correntes desta preocupação os estudos sobre a formalização da programação seqüencial e dos processos paralelos (e.g. as lógicas de Floyd-Hoare, as álgebras de processos), o λ-cálculo, a programação em lógica e a teoria de tipos [BENVENISTE 91a].

Em alguns domínios, no entanto, só recentemente esta meta começa a ser atingida:

- na Inteligência Artificial (IA), onde a maioria dos sistemas existentes - Sistemas Especialistas, núcleos geradores (*kernels*), etc - não são completamente formalizados, por causa da deficiência de expressividade na lógica adotada, geralmente a clássica, ou de problemas ligados diretamente à aplicação. Inconsistência e incompletude em bases de conhecimentos, o raciocínio não-monotônico, e as dificuldades na representação de condições temporais, são os exemplos mais conhecidos de problemas que, apesar das inúmeras soluções propostas, ainda se constituem em dificuldades reais à construção de sistemas inteligentes [LAFFEY 88];

- nos Sistemas Tempo-Real (STR), onde historicamente predominam as soluções específicas (*ad-hoc*), que visam atender aos requisitos de tempo do sistema concentrando-se no problema de performance do mesmo. Neste domínio uma das formalizações que se destacou nestes últimos anos utiliza o enfoque síncrono, que é baseado em conceitos matemáticos claros e elegantes oriundos da teoria dos sistemas dinâmicos [BENVENISTE 91b].

A interseção destes dois domínios constitui o objeto de interesse deste trabalho. A ampliação da área de aplicação dos STR em direção às áreas onde predominam as atividades cognitivas e o conseqüente aumento de sua complexidade, tem levado ao uso de técnicas de IA na sua construção [LAFFEY 88], [WRIGHT 86], [STANKOVIC 88]. Surge assim uma nova classe de sistemas, cujos componentes serão aqui denominados Sistemas Baseados no Conhecimento para aplicações Tempo-Real (SBCTR). Estes sistemas, além de levar em conta a natureza tempo-real das aplicações, estão muitas vezes sujeitos a outras limitações, como e.g. as restrições de *hardware* nos sistemas embutidos, o que os distinguem dos Sistemas Baseados no Conhecimento (SBC) tradicionais.

Em um sentido mais restrito a aplicação dos resultados deste trabalho se destina à análise e ao tratamento de pequenas e médias aplicações, em que se procuram soluções simples porém de alta confiabilidade.

1.1 Por que utilizar Sistemas Baseados no Conhecimento para aplicações Tempo-Real ?

Os SBCTR podem ser vistos como um novo tipo de Sistema Tempo-Real, que se apresenta como o mais adequado em situações onde a carga de conhecimento se torna importante [GHALLAB 93], e a quantidade de informações fornecida ao usuário deve ser reduzida. Neste caso o SBCTR age como um elemento auxiliar para a avaliação da situação ou ajuda à condução (*situation assessment*), atuando de forma a fornecer ao usuário uma visão simplificada do problema [JAKOB 90]. Uma evolução natural desta conduta consiste na retirada do usuário do laço de execução do sistema, criando "sistemas autônomos inteligentes" [HAYES-ROTH 90].

As situações acima se refletem diretamente em algumas áreas de aplicação visadas por este trabalho: o Controle Inteligente de Sistemas Contínuos, o Controle de Sistemas a Eventos Discretos e a Programação para a Robótica.

Para aumentar a confiabilidade e garantir a segurança de plantas industriais complexas, o uso da Teoria de Controle de Sistemas Contínuos muitas vezes não basta. Eventuais intervenções dos operadores são de importância crucial, em particular quando o sistema está sujeito a contingências, como falhas em componentes, operação em condições severas, mudanças bruscas de comportamento, adaptação a novos parâmetros de trabalho, etc. Seria desejável a utilização de um sistema que fosse capaz de atuar tão eficientemente quanto os especialistas humanos nestas situações. Um sistema de controle que incorpore algumas destas características é um exemplo típico de SBCTR.

No caso de Sistemas a Eventos Discretos (SED) existe um fator complicador: a inexistência de uma teoria bem sedimentada que fundamente o tratamento destes sistemas, ao contrário do que ocorre no caso anterior (Sistemas Contínuos). Um resumo das principais abordagens propostas para o tratamento dos SED pode ser encontrado em [SILVA 92]. A condução dos SED por meio de heurísticas e regras de conduta pode se apresentar como uma alternativa prática à solução de problemas reais, indicando assim a aplicabilidade do paradigma SBCTR neste caso.

As aplicações em Robótica se constituem numa das áreas de atuação mais características dos SBCTR, pois estão sujeitas a fortes restrições temporais e a exigências de funcionamento autônomo inteligente. Entre as muitas tarefas envolvidas neste processo podem ser citadas: a fusão e integração dos dados provenientes de sensores, os controles direcionais e de atuadores, e os problemas de planejamento e execução das tarefas pré-programadas. Ressalta-se que neste caso são mais evidentes as restrições sobre o *hardware* utilizado na implementação.

1.2 A proposta do trabalho

O objetivo deste trabalho é o de propor um modelo formal, uma metodologia e um ambiente para o desenvolvimento de Sistemas Baseados no Conhecimento para aplicações Tempo-Real, utilizando como forma de representação do conhecimento o paradigma "Sistema Baseado em Regras", tratando a questão "Tempo-Real" segundo o enfoque síncrono, e visando aplicações nas áreas de Controle de Processos, Automação Industrial e Robótica.

No capítulo 2 trata-se da conceituação e modelagem dos STR, e da apresentação da abordagem síncrona para o desenvolvimento destes sistemas. Em seguida introduz-se o paradigma SBCTR e seus principais requisitos; o atendimento a estes requisitos dentro do escopo do trabalho também é discutido.

A definição de um formalismo matemático destinado a fundamentar a construção de SBCTR é o objetivo do capítulo 3.

Vários cálculos proposicionais de complexidade crescente são desenvolvidos, culminando na definição de uma lógica temporal proposicional paraconsistente, que inclui, além dos conectivos lógicos clássicos, extensões obtidas com o auxílio da teoria dos autômatos.

Mostra-se ainda a correspondência entre os modelos destes cálculos e a relação entrada / saída de um sistema baseado em regras reativo, bem como as suas potencialidades para expressar condições e restrições temporais.

O tratamento formal também permite a inclusão natural de outros elementos, tais como funções e sistemas dinâmicos qualitativos, de grande importância para o desenvolvimento dos SBCTR nas áreas de aplicação consideradas.

Neste capítulo são finalmente apresentados algoritmos e estruturas de dados que permitem a utilização efetiva do formalismo. O processo fundamental envolvido consiste na pré-compilação total da base de conhecimentos do sistema: as regras fornecidas como entrada são transformadas em estruturas equivalentes (autômatos), de execução extremamente eficiente, para atuação direta com o ambiente.

Os principais passos necessários à construção de um SBCTR dentro dos fundamentos teóricos propostos são apresentados no capítulo 4.

Para a descrição do sistema adota-se uma linguagem fundamentada especialmente na decomposição do sistema em módulos, cada um dos quais é composto por um conjunto de regras. Propriedades Tempo-Real do sistema podem ser diretamente incluídas no sistema, de forma a se garantir o atendimento às especificações em qualquer condição prevista de funcionamento.

Um procedimento especial de compilação permite a obtenção, a partir da descrição do problema feita nesta linguagem, de uma estrutura de execução altamente eficiente (autômato), que pode ser diretamente empregado para a operação definitiva frente ao ambiente.

Finalmente alguns exemplos ilustrativos da aplicação do formalismo também são apresentados.

No capítulo 5 são indicadas as principais conclusões e perspectivas do trabalho, destacando-se como principais contribuições:

- a construção de um núcleo gerador e a proposta de uma metodologia de desenvolvimento de SBCTR baseados em um formalismo matemático bem definido, fundamentado em lógica temporal e que segue o enfoque síncrono;
- a consideração automática das questões relativas a consistência e completude pelo tratamento paraconsistente dado ao formalismo; o mesmo tratamento permite ainda que se incorporem ao sistema de modo formalizado representações de elementos incertos e imprecisos;

- o manuseio pelo sistema de construções temporais complexas construídas com o auxílio de autômatos;
- a incorporação de propriedades tempo-real a verificar diretamente no formalismo;
- o tratamento uniforme dado às regras que compõem o sistema e às funções e sistemas dinâmicos qualitativos; e
- a geração de um autômato para a execução, de acordo com o enfoque síncrono, o que garante o determinismo e a reatividade desejados nas aplicações.

Salienta-se ainda que este trabalho propõe a "Síntese Automática de Programas", pela geração automática de código executável a partir das especificações dadas no formato de regras, e do uso de mecanismos de verificação das propriedades exigidas do sistema.

Capítulo 2

A abordagem síncrona nos Sistemas Baseados no Conhecimento para aplicações Tempo-Real

Este capítulo tem por objetivos básicos fixar o paradigma SBCTR dentro do enfoque síncrono e apresentar os principais requisitos destes sistemas.

Inicialmente é estabelecido o conceito de Sistema Tempo-Real adotado ao longo do trabalho. Em seguida apresenta-se o enfoque síncrono para o desenvolvimento destes sistemas, com uma indicação das diversas linguagens existentes e dos paradigmas utilizados.

A aplicabilidade de técnicas de IA em STR é discutida na seqüência; conceitos fundamentais sobre Sistemas Baseados no Conhecimento são apresentados, e em seguida analisa-se o problema crucial de performance oriundo da integração entre SBC e STR. Para ilustrar este ponto apresenta-se sinteticamente o sistema KHEOPS, uma ferramenta empregada para o desenvolvimento de sistemas segundo o paradigma Sistema Baseado em Regras, e destinada a aplicações tempo-real.

A seguir são apresentados os principais requisitos dos SBCTR e estabelecidos os que serão considerados primordialmente no contexto deste trabalho.

Conclui-se o capítulo com um esboço da proposta, que visa a construção de um gerador de SBCTR segundo o enfoque síncrono, dentro dos fundamentos expostos.

2.1 Conceitos básicos sobre Sistemas Tempo-Real

Definições

Muitas definições de STR podem ser encontradas na literatura - ver por exemplo [LAFFEY 88], [DODHIAWALA 89], [YOUNG 82], [KOPETZ 90] [MOK 90], [ANDSLEY 90], [KOPETZ 92] e uma apresentação resumida destas em [KAESTNER 91]. Nestas definições destaca-se a relação entre a parte computacional do sistema e seu ambiente, que estão conectados por sinais de interface (estímulos e respostas) e relacionados por restrições de ordem temporal (ver figura 2.1).

Denominar-se-á *Sistema Reativo* (SR) um sistema que mantém interação contínua com seu ambiente, reagindo aos estímulos provenientes do mesmo e fornecendo, após o processamento, as respostas adequadas a estes estímulos.

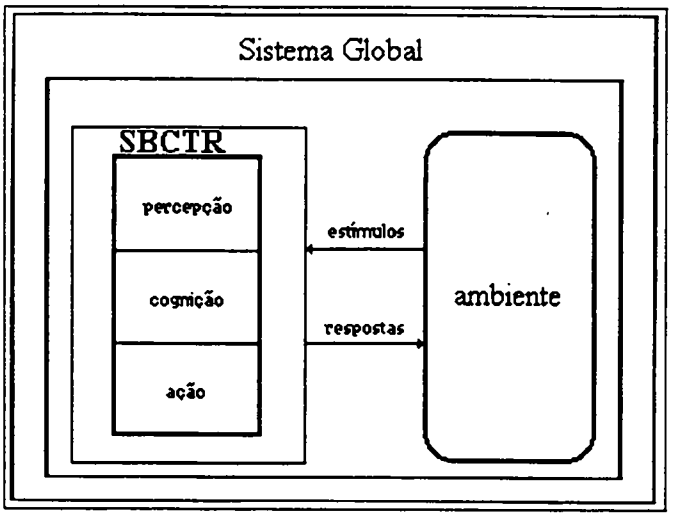

Figura 2.1: O conceito de STR adotado

A seqüência de comunicação e processamento que ocorre entre um estímulo e uma resposta se denomina uma *reação*. Cabe ressaltar que a ocorrência das reações (e dos estímulos / respostas que as compõem) não é determinada pelo projeto do sistema, mas depende unicamente do ambiente [BENNETT 88].

Um *Sistema Tempo-Real* (STR), por sua vez, é um sistema reativo sujeito adicionalmente a restrições de ordem temporal.

A violação de uma restrição temporal ocasiona uma *falha*. A cada falha está associada um custo; a partir desta visão relativa à segurança, os STR podem ser classificados em duas categorias:

- sistemas *soft real-time*, onde o custo de cada falha é comparável aos benefícios decorrentes da operação normal do sistema [1]; e
- sistemas *hard real-time*, onde as falhas tem um custo muito superior ao da operação normal do sistema; o não atendimento a uma restrição temporal pode significar em algumas situações a sua paralisação completa e definitiva [2].

Os STR podem ainda ser classificados em dois grupos, segundo o enfoque de implementação adotado [KOPETZ 92]:

- sistemas *de melhor esforço* (*"best effort"*), nos quais se supõe que a disponibilidade de recursos é insuficiente para o atendimento de todas as condições operacionais de projeto, de forma que a melhor opção é escolhida em função das probabilidades de ocorrência de eventos e de prioridades associadas às situações;
- sistemas ditos *de resposta garantida* (*"guaranteed reponse"*), onde para toda condição de carga de trabalho ou regime de falha devem ser previstas condições de operação

[1] E.g. nos sistemas transacionais de consulta a cadastros, não se considera que o sistema tenha falhado irremediavelmente se certo tempo limite de resposta não é atendido.

[2] E.g. num sistema de auxílio à condução em uma planta industrial, onde o não atendimento a uma restrição temporal pode trazer conseqüências catastróficas para o sistema.

consoantes com as especificações do projeto; é nesta categoria que se encontram as aplicações que se pretende atender neste trabalho.

A busca de um formalismo para a modelagem, especificação, implementação, e prova de propriedades dos STR tem sido objeto de muitos trabalhos recentes. Entre estes podem ser citados o formalismo TTM/RTTL (Timed Transition Models / Real-Time Temporal Logic) [OSTROFF 89], as álgebras de processos temporizados [NICOLLIN 91], os sistemas de transição temporizados [HERZINGER 91] e os autômatos temporizados [ALUR 91a] [3], além de trabalhos caracterizados por seguir um enfoque síncrono, como é o caso das linguagens ESTEREL [BERRY 83], SIGNAL [BOUSSINOT 91] e LUSTRE [HALBWACHS 91].

Nenhum dos formalismos citados se preocupa especificamente com a integração de técnicas de IA nos STR. Este trabalho se concentra na aplicação do enfoque síncrono para o desenvolvimento de STR que envolvam a utilização de IA.

A correção dos Sistemas Tempo-Real

Um STR pode também ser visto como um Sistema Dinâmico a Eventos Discretos [HAYES-ROTH 90], [OSTROFF 89], no qual um conjunto de variáveis internas determina o *estado* corrente do sistema. Para expressar a evolução que ocorre durante a execução, associa-se uma série temporal de valores a cada variável .

As mudanças no estado do sistema se denominam *eventos*, que ocorrem em instantes de tempo discretos (sem duração). Estímulos e respostas são considerados *eventos externos*, e propiciam a comunicação entre o componente computacional do sistema e o ambiente [DASARATHY 85].

A evolução de um STR é em geral determinada pela história passada, a partir de condições temporais sobre os estados. Supõe-se que cada estado armazena as informações necessárias para determinar a evolução futura do sistema (*fusion-closed hypothesis* [ALUR 91a]). Em muitos casos os valores das variáveis ficam invalidados ao longo do tempo, o que pode ocasionar a produção de resultados incorretos.

Como visto anteriormente, a correção de um STR depende não apenas do seu funcionamento logicamente correto (correção funcional ≡ *correctness*), mas também do instante em que os resultados são produzidos (correção temporal ≡ *timeliness*).

As abordagens temporais assíncrona e síncrona

Um Sistema Tempo-Real pode ser decomposto em uma série de *processos*, que executam funções específicas; estes processos são executados em paralelo, e interagem para a sincronização e a troca de informações.

A interação entre os processos pode ser considerada segundo duas abordagens [KAESTNER 91]:

- o enfoque *assíncrono*, em que os processos concorrentes evoluem a velocidades relativas indeterminadas; e
- o enfoque *síncrono*, em que a evolução se dá de forma cadenciada e bem determinada.

[3] Um resumo de vários formalismos empregados pode ser encontrado em [ALUR 91b].

No caso síncrono considera-se que nenhum tempo ocorre entre a execução das ações atômicas em um processo; não é possível para um componente permanecer inativo (*idle*) por qualquer instante de tempo. Já no caso assíncrono os processos podem permanecer inativos, incluindo instantes de tempo entre suas ações elementares. Consequentemente, no que diz respeito ao aspecto temporal, a resposta à uma solicitação em um sistema assíncrono é inerentemente não-determinista; nos sistemas síncronos, ao contrário, o comportamento temporal é totalmente previsível.

2.2 O enfoque síncrono para os Sistemas Tempo-Real

As limitações da abordagem assíncrona

O tratamento clássico dos STR é feito seguindo a abordagem assíncrona: os processos são geralmente implementados sobre um núcleo ou executivo tempo-real, e sua execução é feita segundo alguma estratégia de escalonamento pré-definida, em função dos eventos recebidos e de prioridades a eles associadas (*event-driving scheduling*).

A garantia de atendimento às restrições temporais não pode ser estabelecida sem que se conheça completamente o mecanismo de escalonamento do sistema e sua carga em execução [MOK 90], [JAHANIAN 86]. Usualmente utiliza-se a preempção - suspensão temporária de um processo antes de seu término - para o atendimento de eventos mais prioritários. Este mecanismo torna ainda mais evidente o não-determinismo implícito da abordagem, já que o instante de fim de um processo nunca é conhecido exatamente: a qualquer instante o processo pode ser interrompido para o atendimento de outra solicitação.

Os inconvenientes da abordagem assíncrona são resumidos a seguir: as manipulações do tempo são pouco precisas e pouco claras; o atendimento dos requisistos temporais depende diretamente do *software* e do *hardware* utilizados; o não-determinismo imposto se opõe ao determinismo natural das aplicações; e finalmente são introduzidos problemas de tamanho, complexidade e eficiência, em especial para as aplicações de pequeno e médio portes, onde se buscam soluções rudimentares porém eficazes [BERRY 87]. Em particular, com os constantes progressos nas áreas de *hardware* e *software* parece um contrasenso especificar e desenvolver STR limitados a uma determinada tecnologia: a abordagem síncrona vai ao encontro da flexibilidade exigida.

A abordagem síncrona

A **abordagem síncrona** foi estabelecida inicialmente por Pnueli [BERRY 83] para os Sistemas Reativos. Vários formalismos e linguagens especificamente adaptados a estes sistemas têm sido recentemente desenvolvidos, destacando-se as linguagens síncronas ESTEREL [BERRY 83], [BERRY 89], [BOUSSINOT 91], LUSTRE [HALBWACHS 91], [CASPI 87] e SIGNAL [BENVENISTE 90], [GAUTIER 87], [LeGUERNIC 91], o sistema de composição de autômatos REX [KAELBLING 88], a linguagem para especificação composicional e verificação de controladores de estado finito CSML [CLARKE 91], além do formalismo de especificação StateCharts [DZIERGOWSKI 90].

A principal característica exigida para um SR é seu determinismo: com efeito ele deve reagir sempre da mesma forma à mesma seqüência de entradas. Este determinismo pode ser estendido a um STR, considerando que ele deve também manter o determinismo em relação ao fluxo de eventos a que está sujeito o sistema. Uma conseqüência imediata

do determinismo é que estes sistemas são inerentemente mais fáceis de analisar, verificar e depurar. Determinismo, porém, não implica seqüencialidade: muitos STR podem ser decompostos em subsistemas concorrentes que cooperam de forma determinista.

O princípio de base do enfoque síncrono considera a existência de Sistemas Reativos ideais, onde as respostas são fornecidas de modo absolutamente síncrono em relação às entradas, numa reação instantânea. Considera-se portanto que o processamento **não consome tempo.**

Esta hipótese, denominada "hipótese do sincronismo", pode ser reescrita de outra forma: toda reação é executada sem que se considere a chegada de nenhum outro estímulo após o início do ciclo. Resumidamente:

Hipótese do Sincronismo ≡
Tempo de Resposta Nulo ≡
Atomicidade de Cada Reação

Esta hipótese é verificada se o sistema computacional é suficientemente rápido para concluir o processamento exigido por uma reação antes do início da próxima reação. Portanto, embora a performance não seja o único elemento exigido de um STR, é um requisito necessário também nesta abordagem.

Além da preservação do determinismo são também objetivos da abordagem síncrona:

- o reagrupamento num mesmo formalismo de todos os aspectos reativos do sistema: sincronização, comunicação, controle. O princípio de base deve portanto ser encarado num sentido amplo: assim como as reações são instantâneas, também o são a comunicação e a sincronização entre os processos, o que garante uma visão unificada de todos os processos e do ambiente;
- a utilização de técnicas formais para a especificação, implementação e verificação; e
- a produção de um código de execução eficiente, pela eliminação de perdas durante a execução (*run-time overheads*).

As linguagens síncronas ESTEREL e SIGNAL

A linguagem ESTEREL [BERRY 83], [BERRY 89], [BOUSSINOT 91] segue o paradigma "linguagem imperativa", que, além dos comandos de uma linguagem imperativa convencional, inclui comandos que determinam a execução paralela de processos.

O compilador ESTEREL, seguindo o princípio de base das linguagens síncronas, produz um autômato para a execução a partir da linguagem de entrada. Desta forma, embora a linguagem permita a definição de processos paralelos, um código seqüencial é gerado, e o escalonamento é feito em tempo de compilação. Isto também garante o determinismo desejado nas aplicações.

A geração de um autômato para execução é também conveniente por facilitar a satisfação da hipótese do sincronismo, a partir do tempo de transição maximal do autômato gerado, e por permitir a verificação de propriedades.

Uma descrição resumida e um exemplo característico de aplicação da linguagem ESTEREL são apresentados no anexo A.

A linguagem SIGNAL [BENVENISTE 90], [GAUTIER 87], [LeGUERNIC 91] segue o paradigma "linguagem tipo fluxo de dados" (*data-flow language*). Um programa em SIGNAL é formado por um conjunto de equações que devem ser verificadas e mantidas coerentes durante a execução.

A linguagem procura utilizar conceitos semelhantes aos empregados na Teoria de Controle de Sistemas Contínuos, como, por exemplo, a divisão de um sistema em blocos e sua interconexão por sinais.

Assim como em ESTEREL, um código seqüencial é gerado, permitindo a fácil verificação da hipótese do sincronismo, além da preservação do determinismo.

Um mecanismo especial, denominado "cálculo de relógios" serve como instrumento formal de verificação. As propriedades desejadas do sistema podem ser diretamente codificadas em SIGNAL, e sua satisfação decorre do resultado da análise efetuada sobre o sistema segundo este cálculo.

O anexo A apresenta uma descrição resumida e um exemplo de utilização da linguagem SIGNAL.

2.3 O problema do Tempo-Real nos Sistemas Baseados no Conhecimento

Alguns conceitos básicos sobre Sistemas Baseados no Conhecimento

Neste contexto utilizar-se-á o termo "Sistema Baseado no Conhecimento" para designar a classe de sistemas que utilizam conhecimentos sobre um domínio restrito para a resolução de problemas típicos neste mesmo domínio, de uma forma que seria considerada inteligente se realizada por um ser humano [RICH 83].

De uma maneira geral os SBC são constituídos por dois componentes básicos:

- uma *Base de Conhecimentos*, que descreve de uma maneira conveniente o conhecimento disponível sobre o problema; e
- um *Mecanismo de Inferência*, que permite a dedução de novos conhecimentos a partir dos já estabelecidos.

Muitos são os paradigmas empregados para a representação do conhecimento e a inferência, como e.g. a lógica, as redes semânticas, os sistemas baseados em regras de produção e os *frames* [BARR 86b].

Discutir-se-á aqui em especial o paradigma "Sistema Baseado em Regras" (SBR). Nestes sistemas a Base de Conhecimentos pode ser conceitualmente particionada em dois subconjuntos: (a) a *Memória de Regras*, que descreve os conhecimentos operatórios necessários à resolução do problema; e (b) a *Memória de Trabalho*, que descreve o estado corrente do problema, usualmente por meio de um conjunto de *atributos* e de valores a eles associados.

O Mecanismo de Inferência, por vezes também denominado *Motor de Inferência*, atua da seguinte forma: a partir do estado corrente do problema são determinadas as regras aplicáveis; uma ou várias destas regras são utilizadas (*disparadas*), modificando o estado do problema; este ciclo é repetido até que se encontre a solução desejada.

Os SBR também podem conter outros elementos, tais como interfaces especiais para o usuário, para o engenheiro do conhecimento e para o especialista, módulos especiais para

a justificativa da linha de inferência utilizada, etc. Uma descrição mais detalhada pode ser encontrada em [WATERMAN 86].

Caracterização do problema tempo-real nos SBC

É fato bem conhecido que os sistemas de IA são inerentemente ineficientes [McDERMOTT 78] [FORGY 82], [GHALLAB 88]. Esta ineficiência se reflete praticamente em todos os paradigmas utilizados para a construção de SBC, e em particular nos SBR.

As principais causas da ineficiência dos SBR podem ser identificadas:

- o acesso associativo dos dados - filtragem, em oposição ao acesso direto na programação clássica; e
- o não-determinismo do controle de execução - em oposição ao controle explícito feito diretamente pelo programador.

Nos SBR a etapa de filtragem (*pattern-matching*), i.e., a determinação das regras aplicáveis a uma situação dada, pode ocupar mais de 90 % do tempo de execução do sistema [McDERMOTT 78], [FORGY 82], [GHALLAB 81], [GHALLAB 88]. Vários algoritmos têm sido propostos para melhorar a performance desta etapa (ver por exemplo [FORGY 82], [MIRANKER 87], [GHALLAB 84], [GARNOUSSET 88] para os algoritmos e [ALBERT 91] para uma análise de sua complexidade média). Entretanto a simples aplicação destes algoritmos não é suficiente para garantir a performance exigida de um SBCTR. Resta a questão do atendimento às restrições temporais e da previsibilidade através do controle da inferência: o número de disparos de regras necessários para se atingir uma solução não é, a priori, limitado. Esta capacidade, que leva a considerar um tempo de reação imperativamente limitado, é uma condição essencial para se falar de SBCTR [LAFFEY 88].

Apesar deste obstáculo vários SBCTR têm sido desenvolvidos e empregados em muitas aplicações reais. O anexo B apresenta uma descrição de diversos SBCTR, bem como da sua forma de atendimento aos principais requisitos exigidos de um Sistema Tempo-Real.

Algumas novas soluções são possíveis. Em particular no caso de sistemas decisionais fechados [GHALLAB 82] - onde o conjunto de regras aplicáveis é conhecido a priori e as inferências são feitas sobre atributos também pré-determinados, uma possibilidade consiste no tratamento prévio e total da base de conhecimento do sistema. Embora este pré-tratamento tenha um custo exponencial na etapa de construção do sistema, ele traz benefícios efetivos quando da execução: nesta etapa um tempo de reação maximal pode ser garantido. O sistema KHEOPS, que é apresentado resumidamente a seguir e o sistema RUM [BONISSONE 90] funcionam segundo esta abordagem.

Uma solução: o sistema KHEOPS

O sistema KHEOPS [GHALLAB 84], [PHILLIPE 89] desenvolvido no LAAS/CNRS [4] foi proposto como uma nova ferramenta para a construção de SBC, destinada prioritari-

[4] LAAS/CNRS: *Laboratoire d'Automatique et d'Analyse des Systèmes / Centre National de la Recherche Scientifique*, Toulouse, França.

amente às aplicações tempo-real. KHEOPS foi utilizado em diversas aplicações, entre as quais pode-se citar: o sistema SIAD para a supervisão e diagnóstico de falhas no sistema de propulsão dos trens de subúrbio de Paris [GHALLAB 86]; GELIS, um sistema para auxílio à condução de um processo de manufatura de produtos cerâmicos [IPPOLITO 90]; VA-LAB, para a validação de resultados de análises médicas [PHILLIPE 89]; e no sistema de controle direcional do movimento lateral de um veículo autônomo no contexto do projeto *Prometheus* [BRUNESSAUX 92]. O sistema é escrito em LISP, pode atuar diretamente nesta linguagem ou gerar um código executável em "C".

O sistema KHEOPS procura resolver o problema de previsibilidade a partir do uso de técnicas de compilação e geração de procedimentos, que tranformam um conjunto de regras proposicionais em uma rede de decisão otimizada, de comportamento determinista.

Uma base de conhecimento KHEOPS é formada por um conjunto finito de regras, no formato clássico (*IF* <*condição*> *THEN* <*ação*>), que executam inferências sobre um conjunto também finito de atributos. A partir de atributos fornecidos como entrada (*espaço de entrada*) o motor de inferência KHEOPS deduz o valor de outros atributos (*espaço de saída*).

A inferência é monotônica: nenhum disparo de regra pode reverter uma conclusão já obtida. Com isto a ordem com que se efetuam os disparos deixa de ser relevante, o que facilita o processo de compilação. Por outro lado surge um novo problema: a presença de situações de inconsistência e incompletude. A inconsistência provém do conflito entre um fato conhecido e a conclusão de uma regra disparada. A impossibilidade de determinar de forma precisa todas as saídas a partir das entradas caracterizam a incompletude. Estas questões serão analisadas mais precisamente no capítulo 3.

No sistema KHEOPS estas situações são identificadas e cabe ao programador a tomada de medidas visando a sua eliminação, se isto for julgado necessário.

A compilação é feita como segue:

1. um conjunto de valores é fixado para os atributos de entrada;
2. todos os possíveis encadeamentos de regras são efetuados, as ações correspondentes são executadas;
3. estados idênticos (i.e. que têm os mesmos valores definidos para todos os atributos) são agrupados;
4. repetem-se os passos de 1 a 3 até que nenhum novo estado seja gerado; e finalmente
5. a estrutura obtida é otimizada e compactada, pela eliminação de redundâncias e estabelecimento de uma ordem ótima de avaliação.

A finitude da rede pode ser assegurada já que se considera que os atributos assumem valores sobre domínios finitos: para um atributo que assume valores sobre um domínio simbólico esta finitude é imediata; no entanto para um atributo numérico um procedimento de discretização deve ser empregado. Neste último caso o sistema KHEOPS utiliza um mecanismo de limiares: inicialmente um número finito de intervalos é obtido a partir das regras; os valores de atributos numéricos podem ser então convertidos para um domínio discreto, constituído por estes intervalos; a relação de pertinência do valor do atributo em cada intervalo determina diretamente a conversão. A figura 2.2 ilustra este mecanismo. Nesta

figura *temperatura*, *pressão* e *estado* são atributos sobre os quais impõem-se as condições presentes nas premissas das regras. Os valores de comparação (limiares) também são indicados.

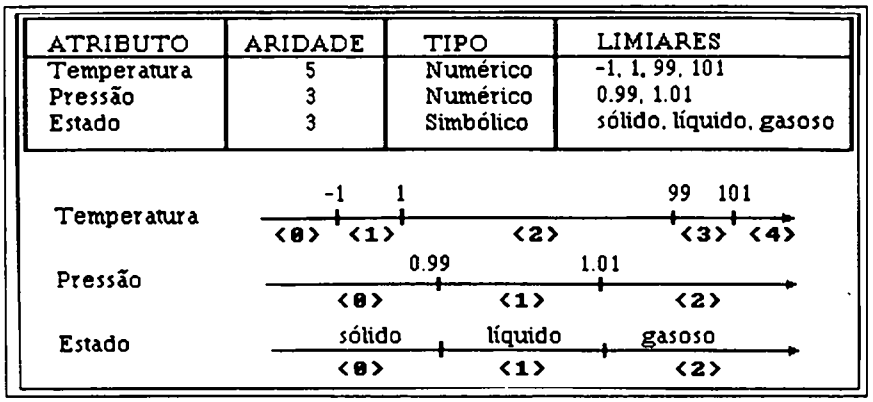

Figura 2.2: O mecanismo de limiares do sistema KHEOPS

Um limitante do sistema KHEOPS refere-se a exigência de monotonicidade adotada no seu modelo de base. Com efeito, como se verá no ítem 2.4, as aplicações tempo-real são eminentemente não-monotônicas: a sucessiva chegada de eventos obriga a considerar alterações nos valores dos dados. Igualmente limitante é o fato que a execução da rede dedutiva gerada pelo KHEOPS é atemporal, i.e., desconsidera-se formalmente a evolução histórica do sistema.

Estes problemas são tratados pela consideração de *evoluções restritas*: valores de atributos deduzidos em ciclos (i.e., execuções de procedimento de inferência do KHEOPS) anteriores são utilizados como entrada do ciclo corrente. Na sintaxe KHEOPS denota-se por $\%a$ (respectivamente $@a$) o valor do atributo de saída (respectivamente de entrada) a no ciclo precedente de inferência. A consideração de evoluções restritas permite a construção de condições temporais complexas relacionando os atributos. Além disto, outras condições temporais para o KHEOPS foram desenvolvidas, tais como as indicadas na figura 2.3. Estas condições são testadas a partir de um sinal de relógio pré-definido (e.g. segundo, minuto, ciclo, etc).

Um exemplo típico de uso do KHEOPS é apresentado no anexo C.

A aplicabilidade do enfoque síncrono em Sistemas Baseados no Conhecimento para aplicações Tempo-Real

É evidente que em situações onde a carga de conhecimento utilizada se torna importante o emprego do paradigma "Sistema Baseado em Regras" facilita em muito a descrição do problema e a especificação do controle.

Neste contexto o sistema KHEOPS apresenta uma resposta às principais questões ligadas a viabilidade de construção dos SBCTR: o acesso às condições das regras é direto (não há operação de filtragem na etapa de execução) e o não-determinismo é apenas aparente (o percurso na rede é determinista e um tempo maximal de travessia pode ser determinado).

Embora não tenha sido desenvolvido segundo o princípio de base das linguagens síncronas, o sistema KHEOPS pode ser considerado como atuando dentro desta abordagem, seja de-

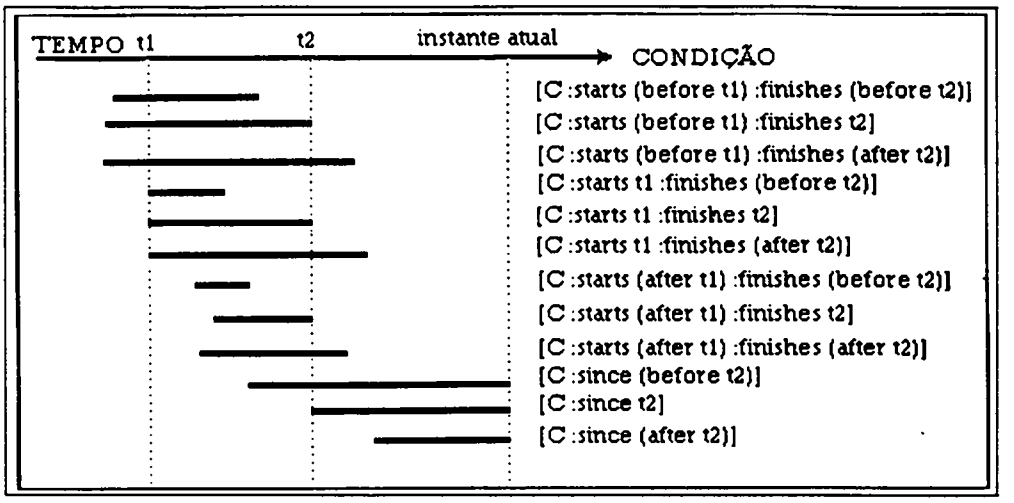

Figura 2.3: Semântica das condições temporais presentes no sistema KHEOPS

vido à forma reativa através da qual o sistema fornece respostas em saída a partir das entradas em cada ciclo, seja pela forma que caracteriza a estrutura executável obtida pela compilação.

Muitos dos fundamentos do sistema KHEOPS serão considerados neste trabalho. No entanto algumas observações podem ser feitas sobre o sistema:

- o processo de dedução não é completamente formalizado; embora baseado numa extensão da lógica proposicional [PHILLIPE 89], o processo de compilação do KHEOPS é correto mas não completo (a cada folha da árvore gerada pelo compilador KHEOPS corresponde um modelo da teoria lógica subjacente, mas a recíproca não é verdadeira);
- os problemas de inconsistência e incompletude das Bases de Conhecimento são apenas identificados pelo KHEOPS, e não sofrem tratamentos adicionais;
- o tratamento temporal no sistema não é construído sobre uma base formal; apenas indica-se a semântica dos operadores temporais;
- não existem procedimentos para a verificação automática de propriedades tempo-real; e
- em relação a abordagem síncrona, apenas as evoluções restritas e as condições obtidas quando se utiliza como unidade de medida o ciclo (que equivale na terminologia dos SR a noção de reação) podem ser consideradas coerentes com o enfoque. Isto decorre do fato que a aplicação das condições temporais nos sistemas síncronos deve ser feita sobre qualquer evento, e não sobre um sinal particular (como o segundo, etc).

Este trabalho apresenta uma proposta alternativa para a geração de SBCTR segundo o enfoque síncrono, que visa resolver as questões acima citadas.

2.4 Requisitos e principais características dos Sistemas Baseados no Conhecimento para aplicações Tempo-Real

Esta seção apresenta alguns dos requisitos necessários à construção de SBCTR, em especial no que se refere à representação do conhecimento e ao processo de inferência utilizado.

2.4.1 Requisitos referentes ao Tempo-Real

O objetivo principal de um Sistema Tempo-Real é a manutenção da conduta do sistema dentro dos limites de tempo impostos pelo ambiente, o que corresponde a manter a correção funcional e temporal do sistema.

Em conseqüência os SBCTR devem atender, em primeiro lugar, aos seguintes requisitos que caracterizam o seu funcionamento [KAESTNER 91]:

1. previsibilidade (determinismo): é a principal condição requerida de um STR, afirma Stankovic [STANKOVIC 88], [ANDSLEY 90]: "A propriedade mais importante de um STR é sua previsibilidade, i.e., seu comportamento funcional e temporal deve ser tão determinista quanto necessário para satisfazer as especificações do sistema". Este requisito está ligado diretamente a confiabilidade operacional exigida do STR: correção funcional e temporal só podem ser efetivamente garantidas em um sistema previsível [YOUNG 82];

2. correção temporal: caracteriza o atendimento às restrições temporais impostas; nos sistemas visados por este trabalho isto significa que o sistema deve ser capaz de fornecer ao ambiente de forma determinista a resposta mais aceitável que esteja disponível no instante limite imposto pelas restrições de tempo;

3. reatividade: pela própria modelagem, um SBCTR está em constante comunicação com o ambiente, de forma que o tratamento da entrada e saída dos dados deve merecer atenção especial. Em vários casos, a comunicação externa deve ser feita também com outros SBCTR, por razões de distribuição ou redundância [LEINWEBER 87], [HAYES-ROTH 90].

Além disto o tempo de reação do sistema deve ser coerente com o exigido pela aplicação, satisfazendo as restrições temporais impostas. Como citado anteriormente o desempenho constitui o principal empecilho para a construção de SBCTR, devido a natureza freqüentemente combinatória e a complexidade inerente às técnicas de IA utilizadas.

Os requisitos acima apresentados devem ser incorporados aos SBCTR, já que não são usualmente considerados no desenvolvimento de um SBC tradicional. As consequências desta integração sobre a estrutura de conhecimento e sobre os mecanismos de inferência em um SBR são analisadas em [KAESTNER 91] e resumidas a seguir.

2.4.2 Requisitos quanto à representação de conhecimento

Serão analisadas a seguir várias dimensões do conhecimento especialmente envolvidas no tratamento de um problema tempo-real.

Representação da variável tempo

O tempo é a variável mais importante presente nas aplicações tempo-real, pois distingue estas aplicações das demais, e deve se integrar convenientemente à metodologia do projeto, à linguagem de programação utilizada e às técnicas empregadas para especificação e verificação do sistema.

A possibilidade de representação e de raciocínio sobre eventos e fatos atuais, passados e futuros, bem como o seqüenciamento de suas ocorrências é uma funcionalidade fundamental e indispensável a exigir de um SBCTR.

A inclusão do tempo como uma dimensão suplementar ao conhecimento utilizado nos SBCTR permite o emprego de heurísticas dependentes do tempo, a indicação de dependências temporais entre os elementos, como e.g. dados históricos, análise de tendências, etc [MOORE 90], [PERKINS 90].

A representação do tempo necessita em primeiro lugar da definição do modelo temporal adotado para o sistema. O enfoque síncrono utiliza naturalmente o domínio do tempo discreto, determinado a partir das ocorrências de eventos.

Uma consequência deste tratamento é o caráter multiforme do tempo: todo evento dá origem a uma temporização baseada nos instantes de sua ocorrência. Assim os quantificadores e relações temporais, que devem ser aplicados sobre os atributos para compor a estrutura das regras e limitar o escopo temporal de sua aplicação (ver figura 2.4), devem ter sua temporização baseada nos eventos.

```
escopo de atuação:=   <quantificador>
  temporal            <relação temporal>
                      <designação temporal>

onde:
<quantificador>     := ALWAYS
                       ONCE
                       JUST
<relação temporal>  := AT
                       DURING
                       BEFORE
                       AFTER

<designação temporal> := <valor temporal>
                         <função temporal>
```

Figura 2.4: Exemplo de quantificadores e relações temporais

De particular importância são as restrições temporais impostas sobre os eventos externos do sistema, que impõem limitações (e.g. máximo, mínimo, de duração) sobre suas ocorrências. Entre estas restrições a mais importante é a que fixa o prazo limite de uma resposta (*deadline*) [JAHANIAN 86], [MOK 90].

Representação de informações imprecisas e incertas

Os conceitos de impreciso e incerto podem ter diferentes significados segundo o contexto. Aqui como em [DUBOIS 80] estes termos indicam dois pontos de vista antagônicos da

imperfeição da informação sobre uma realidade. O impreciso (também o difuso) concerne ao valor da informação, e o incerto é relativo a sua conformidade para com a realidade.

O tratamento deste tipo de informação nos SBCTR provém da necessidade de reconhecer e de tratar dados de validade imprecisa ou decrescente, devido a ruídos, medidas inconsistentes provenientes de diferentes fontes, etc [LAFFEY 88]. Além disto como em qualquer SBR o conhecimento descrito pelo especialista pode conter uma incerteza inerente associada [MOORE 90].

Cabe ainda salientar que o emprego de alguma forma de raciocínio aproximado permite uma maior flexibilidade no tratamento do problema da geração da resposta mais aceitável para uma restrição temporal imposta [DECKER 90], [LESSER 88], [DURFEE 88].

As principais técnicas de representação do impreciso / incerto [5] utilizam métodos quantitativos, que associam números ou índices associados aos atributos e às regras. Vários métodos são utilizados e se diferenciam quanto à semântica dada à representação numérica, como e.g. os conjuntos difusos [ZADEH 78], os fatores de certeza [SHORTLIFFE 76], e a probabilidade condicional [KUENG-CHI 90]. Em todos os casos é também fundamental garantir a propagação correta e eficiente da informação imprecisa e incerta, o que garantirá sua correta aplicação ao problema [BONISSONE 90].

Conhecimento profundo (*deep-knowledge*)

O conhecimento necessário à solução de um problema pode ser distinguido entre associativo (*shallow*) e profundo ou baseado em modelos (*deep*). O conhecimento associativo corresponde ao normalmente utilizado nos SBC: heurísticas, regras de decisão, etc. Por outro lado, o conhecimento profundo representa conhecimento formado por relações, funções ou equações diferenciais, que interrelacionam grandezas físicas (e.g. a lei de Ohm), e descrevem a dinâmica de um processo. Este conhecimento é portanto seguramente válido na modelagem escolhida [MOORE 90].

A integração entre estas duas formas de conhecimento é muito importante para os SBCTR e tem sido recentemente muito utilizada em aplicações específicas, especialmente na área de Controle de Processos [D'AMBROSIO 87].

Chamada de procedimentos externos

Os SBCTR não funcionam isoladamente. Em sistemas complexos pode ser conveniente a utilização de algoritmos para a realização de tarefas especiais tais como a compressão de dados, o tratamento de sinais, a extração de características, o tratamento de entrada / saída, etc [LAFFEY 88], [WRIGHT 86]. Novamente esta situação se encontra com mais freqüência no caso de aplicações em controle, onde muitos algoritmos são bem conhecidos e podem facilmente ser utilizados, através da chamada de procedimentos [D'AMBROSIO 87], [WRIGHT 86].

Metaconhecimento

O termo metaconhecimento designa o conhecimento utilizado para aplicação do conhecimento ao nível do problema disponível em um SBC. É o responsável pela determinação do contexto de aplicação das fontes de conhecimento (formadas por conjuntos de regras no

[5] Para um resumo destas técnicas ver [KUENG-CHI 90].

caso dos SBR) e permite que outros componentes do sistema, como por exemplo o mecanismo de inferência ou um gerenciador de conhecimento, determinem a aplicabilidade deste conhecimento [MOORE 90].

O emprego do metaconhecimento é importante nos SBCTR, já que forma a base do mecanismo de foco de atenção (*focus-of-attention*) do sistema, de forma similar ao que ocorre com o especialista humano. A ação do SBCTR é dirigida pelos dois objetivos seguintes [MOORE 90]:

1. a manutenção de uma vigilância sobre o estado global do sistema, de forma a estar atento a ocasionais eventos prioritários; e

2. uma concentração no sub-problema mais imediato a resolver, que passa a ter maiores recursos disponíveis e pode ser mais rapidamente solucionado.

As exigências de representação de conhecimento em um SBCTR, apresentadas acima, permitem concluir que estes sistemas possuem inerentemente uma complexidade maior que os SBC tradicionais, exigindo uma multiplicidade de formas de representação do conhecimento, tornando necessário o uso de técnicas convenientes para o manuseio de cada forma de representação, em cada componente funcional do sistema.

2.4.3 Requisitos quanto aos mecanismos de inferência

Tão importante quanto a representação do conhecimento é sua utilização. Os próximos ítens analisam as necessidades de um SBCTR referentes ao processo de inferência.

Manutenção da verdade (*truth maintenance*)

O requisito de manutenção da verdade está relacionado à garantia de um conhecimento consistente durante o processo de inferência. Este problema provém da não-monotonicidade inerente aos SBCTR, pois em aplicações tempo-real os dados presentes no sistema não permanecem estáticos durante a execução, mas são freqüentemente modificados, seja pelo aparecimento de novos dados provenientes do ambiente, ou por perda de validade do próprio dado [LAFFEY 88].

Do ponto de vista estático o processo de manutenção da verdade tem por objetivo modificar a parte dos dados dependente de uma mudança efetuada. Esta modificação pode ocorrer, por exemplo, quando as hipóteses que levaram a uma conclusão são refutadas.

O uso da técnica de revogação (*backtracking*), onde o sistema retorna a um estado anterior conhecido, exige que se recalcule por diversas vezes as mesmas asserções, o que é impraticável em um STR.

Os sistemas de "manutenção da verdade" (TMS: *truth-maintenance systems*) buscam encontrar uma solução para este problema. Vários sistemas são disponíveis para este fim, como por exemplo o TMS [DOYLE 79], e o ATMS [DeKLEER 86a].

Outra possibilidade é a utilização de uma ferramenta teórica não-convencional para o tratamento do problema. Nesta linha situam-se as lógicas *defaults* [REITER 80], [ETHERINGTON 87], as lógicas não-monotônicas [McDERMOTT 78] e autoepistêmicas [THAYSE 90], e mais recentemente a lógica paraconsistente [COSTA 89], [COSTA 90], que será empregada neste trabalho.

Tratamento de hipóteses

Diretamente relacionada aos requisitos de foco de atenção e de manutenção da verdade está a exigência do gerenciamento de hipóteses, utilizadas como teste para a solução do subproblema corrente.

O gerenciamento correto de hipóteses deve permitir a utilização de mecanismos de inferência em encadeamento misto (*forward / backward*), o que pode reduzir consideravelmente o espaço de busca [D'AMBROSIO 87], [BONISSONE 90], [DECKER 90], [HAYES-ROTH 90].

O tratamento de hipóteses é especialmente útil em problemas de diagnóstico, onde o conjunto de hipóteses para o problema pode ser definido de forma adequada "a priori" [RICH 83].

Manuseio de situações prioritárias

A habilidade do sistema de estar atento aos eventos provenientes do ambiente corresponde ao requisito de reatividade (*responsiveness*) de um STR [DODHIAWALA 89].

Sob o ponto de vista do processo de inferência, um SBCTR deve ser capaz de tratar as mensagens provenientes do ambiente de acordo com sua prioridade: quanto mais urgente for a situação, mais rapidamente o sistema deve perceber sua importância, executar o raciocínio e tomar as ações indicadas [LAFFEY 88], [HAYES-ROTH 90].

Nos SBCTR a inferência é caracterizada pela constante modificação dos dados fora do controle do próprio mecanismo de inferência, o que ocasiona uma necessidade de manter a coerência do raciocínio. Como este é um processo custoso, não devem ser poupados esforços no sentido de diminuir a complexidade desta operação, seja pela restrição dos fatos considerados (foco de atenção) seja pelo controle da prioridade dos eventos atendidos em cada instante.

2.4.4 Garantia de resposta

O atendimento à correção funcional e temporal pode significar, nos SBCTR, o fornecimento da melhor resposta possível, mesmo com perda de qualidade, dentro das restrições de tempo impostas [LAFFEY 88]. Isto deve ocorrer mesmo na presença de situações de pico de carga cognitiva ou de taxa máxima de falha do sistema, pois assume-se que o mesmo dispõe de recursos para atender a todas estas situações (i.e. o sistema é totalmente determinista) [KOPETZ 92].

Para obter uma tal resposta, o SBCTR deve ser capaz de raciocinar sobre critérios de aceitação de uma solução, e ser capaz de avaliar o tempo necessário para a execução dos planos elaborados.

Se a obtenção da solução ótima - de acordo com o plano correspondente - não é possível dentro do tempo disponível, um compromisso adequado entre o instante de disponibilidade de uma solução e sua qualidade deverá ser encontrado, resguardada sempre a segurança do sistema frente a situações críticas [LESSER 88].

Este compromisso tempo x qualidade pode ser analisado sob três pontos de vista: completude (alguns aspectos da solução são ignorados, nem todos os objetivos são atendidos,

nem todos os sub-problemas são resolvidos); precisão (alguns elementos são determinados apenas de forma aproximada); e certeza (algumas evidências não são consideradas).

Em problemas complexos uma série de diferentes formas de aproximação podem ser necessárias para que se atinja a solução adequada [LESSER 88]. Três grandes classes de aproximações podem ser identificadas:

1. estratégias de busca aproximada, que exploram uma região mais reduzida do espaço de busca do problema;

2. aproximações sobre os dados, com o emprego de uma visão mais abstrata dos mesmos, o que fornece um espaço de pesquisa mais reduzido; e

3. aproximações sobre o conhecimento aplicado, o que permite uma pesquisa mais rápida no espaço de estados do problema.

A definição da etapa (a priori ou durante a inferência) e forma (ver os três itens acima) com que se darão as aproximações em um problema deve ser definida no projeto do SBCTR. Podem-se fixar ainda as três condições necessárias para uma boa aproximação [LESSER 88], [DECKER 90], [DURFEE 88]:

1. as aproximações devem ter um efeito bem definido sobre a solução, em termos do tempo, precisão e certeza, de maneira que se possa determinar claramente se a solução satisfaz ou não o objetivo;

2. as aproximações devem ser bem formadas: o raciocínio aproximado e o exato devem ser integrados dando uma continuidade às soluções; e

3. o raciocínio aproximado deve ser consistente com o raciocínio exato. As aproximações não devem jamais eliminar elementos que estão presentes na solução ótima.

2.4.5 Requisitos e necessidades complementares de um Sistema Baseado no Conhecimento para aplicações Tempo-Real

Vários outros requisitos e necessidades complementares dos SBCTR podem ser indicados além dos expostos anteriormente.

Quanto à representação de conhecimento e mecanismos de inferência:

- mecanismos convenientes para a aquisição e edição do conhecimento, preferencialmente com o uso de ícones, que permitem uma representação natural dos componentes do sistema [MOORE 90].

- mecanismos para explanação do raciocínio efetuado, que devem incluir também referências temporais; e

- mecanismos para a verificação da consistência das bases de conhecimento e prova da correção e completude do processo de inferência.

Outras condições necessárias podem ainda estar presentes:

- a necessidade de integração dos SBCTR com sistemas STR convencionais existentes;
- restrições quanto ao *hardware* disponível para execução;
- a distribuição, que pode ser uma imposição do ambiente ou uma exigência de projeto.

2.5 Esboço preliminar da proposta

Este trabalho visa apresentar alternativas para solucionar algumas das limitações apresentadas na utilização dos Sistemas Baseados em Regras para aplicações Tempo-Real.

O formalismo de base

O ponto de partida é a adoção da *abordagem síncrona* como princípio de base para a solução do problema tempo-real, o que simplifica sobremaneira a questão e facilita sua formalização. Utilizando o princípio comum ao KHEOPS e às demais linguagens síncronas (ESTEREL, SIGNAL, ...), i.e., a compilação total e a geração de um código determinista, buscou-se um tratamento matemático mais formal, construído com base no estudo da teoria de modelos subjacente à lógica que fundamenta a proposta.

O processo de compilação consiste na busca de todos os modelos da teoria lógica formada pelas regras que compõem a base de conhecimentos do sistema, o que determina o seu comportamento - e o da saída - para todas as situações possíveis da entrada. São portanto eliminados os problemas de controle e encadeamento de regras, pela pré-compilação total e prévia da Base de Conhecimentos do sistema.

Visando contemplar também o caso temporal, optou-se por utilizar uma lógica temporal proposicional, linear em relação ao passado - o que possibilita a fixação clara da semântica dos operadores temporais - e arborescente em relação ao futuro - o que permite a codificação direta na linguagem de propriedades tempo-real no sistema.

Como se mostrará a seguir, o formalismo proposto também permite a solução de outros problemas - tais como a análise de situações de inconsistência, etc - pelo tratamento paraconsistente dado à lógica.

A representação do conhecimento

Serão considerados de forma direta os seguintes requisitos:

- representação da variável tempo e de condições temporais, sempre consideradas sobre eventos - elementos primitivos na proposta; neste ponto cabe ressaltar que, de acordo com o enfoque síncrono, as primitivas temporais devem ser passíveis de aplicação a qualquer tipo de evento e não apenas a um sinal particular de temporização;
- integração de conhecimento profundo e procedural, pelo uso de técnicas advindas do Cálculo Qualitativo [IWASAKI 89];
- dados imprecisos e incertos poderão ser utilizados, como conseqüência marginal do tratamento paraconsistente dado à lógica empregada.

Mecanismos de inferência

Serão tratados os seguintes problemas:

- o uso de uma lógica temporal paraconsistente permitirá o tratamento do problema de manutenção da verdade no sistema;
- os mecanismos de modularização propostos permitem o tratamento dos problemas de aplicação das fontes de conhecimento e foco de atenção.

Verificação

O tratamento lógico adotado permite a incorporação direta e natural de propriedades tempo-real ao sistema, facilitando sobremaneira a tarefa de verificação de propriedades nos sistemas gerados.

A satisfação dos requisitos especificamente tempo-real é feita de acordo com o **enfoque síncrono**: sua adoção garante o atendimento à previsibilidade exigida de um SBCTR, bem como a reatividade instantânea às mudanças do ambiente externo.

Quanto ao problema crucial de desempenho e de garantia de resposta, é de responsabilidade do programador a verificação do atendimento da hipótese do sincronismo no sistema gerado.

A figura 2.5 apresenta um resumo dos requisitos mencionados neste capítulo e indica sua forma de atendimento, dentro deste trabalho.

2.6 Conclusão

Neste capítulo foram apresentados os problemas advindos da integração

$$IA + STR = SBCTR$$

dentro do enfoque síncrono.

O ponto inicial da discussão refere-se à real possibilidade desta integração, já que é fato conhecido que, do ponto de vista temporal, as técnicas de IA são inerentemente ineficientes.

O enfoque síncrono para os STR foi apresentado como uma nova possibilidade de tratamento do problema, indo em direção aos requisitos de determinismo, correção temporal e reatividade exigidos destes sistemas.

O sistema KHEOPS foi apresentado como exemplo de SBCTR já largamente utilizado em aplicações reais, o que comprova a possibilidade da integração, desde que sejam utilizados os tratamentos convenientes.

Em seguida foram analisados os principais requisitos exigidos dos SBCTR, no que se refere especificamente ao problema tempo-real, à representação do conhecimento e aos mecanismos de inferência, dentro do paradigma "Sistema Baseado em Regras".

Finalmente apresentou-se o esboço preliminar da proposta de trabalho, que será detalhada nos próximos capítulos, e cujo objetivo final é a construção de um gerador de SBCTR que siga o enfoque síncrono.

REQUISITO	FORMA DE ATENDIMENTO
Quanto ao tempo-real	
previsibilidade	enfoque síncrono
correção temporal	verificação da hipótese do sincronismo
reatividade	por eventos (enfoque síncrono)
Quanto à Representação do Conhecimento	
representação do tempo	condições temporais por autômatos
impreciso / incerto	indiretamente por lógica paraconsistente
conhecimento profundo e procedural	cálculo qualitativo
metaconhecimento	não considerado
Quanto aos Mecanismos de Inferência	
manutenção da verdade	lógica paraconsistente
tratamento de hipóteses	desnecessário (o cálculo de modelos inclui)
situações prioritárias	não considerado (enfoque síncrono)
Quanto à Garantia de Resposta	
verificação da hipótese do sincronismo	

Figura 2.5: Requisitos considerados e forma de atendimento na proposta

Capítulo 3

O formalismo matemático

O objetivo deste capítulo é propor uma estrutura formal que permita modelar, especificar, analisar, implementar e verificar as propriedades tempo-real de SBCTR que seguem o enfoque síncrono. O interesse primordial neste tipo de sistema está nos aspectos ligados à evolução entrada / saída de um sistema que evolui de forma reativa frente a seu ambiente, e na expressão das condições temporais associadas a esta evolução. Como elementos de base para a formalização utilizar-se-ão extensões da lógica temporal proposicional e a teoria dos autômatos.

Inicialmente será estudado o **caso atemporal**. O ponto de partida é a definição de uma lógica proposicional, adequada para o caso onde as variáveis tomam valores sobre domínios finitos. Como será visto este formalismo permite a integração natural de mecanismos oriundos do Cálculo Qualitativo, sobre o qual se baseia a representação do conhecimento profundo no sistema.

Os problemas de inconsistência e incompletude de um conjunto de expressões lógicas (uma *teoria*) são abordados em seguida. Propõe-se para a lógica subjacente ao sistema um tratamento *paraconsistente*, permitindo que se desenvolvam teorias inconsistentes mas não-triviais.

Parte-se então para o estudo do **caso temporal**. Inicialmente o problema da expressão de condições temporais é tratado. As condições temporais compatíveis com o enfoque síncrono são obtidas com o auxílio de elementos da teoria dos autômata, atendendo à expressividade desejada do sistema. Em seguida propõe-se a partição de toda teoria temporal num subconjunto de fórmulas que contêm apenas condições sobre o passado, e o outro subconjunto que inclui as condições temporais sobre o futuro. No primeiro caso, mostra-se que as fórmulas podem ser traduzidas na forma de um sistema dinâmico sujeito a restrições, o que facilita sua implementação. Já as condições temporais sobre o futuro expressam situações de funcionamento desejadas do SBCTR, estando ligadas a propriedades tempo-real do sistema.

Neste capítulo o estudo será fundamentalmente semântico: o interesse principal é o cálculo de modelos dos conjuntos de fórmulas lógicas que formam a base de conhecimentos em cada problema. Entretanto em alguns casos são apresentados os tratamentos axiomáticos das lógicas correspondentes, como um elemento adicional de caracterização.

Finalmente são também apresentados os **algoritmos** e as **estruturas de dados** que implementam esta geração. A estrutura de dados básica utilizada é o *hipercubo*, e os algoritmos utilizados têm base na interpretação geométrica destes elementos.

Parte I

O Caso Atemporal

Nas seções que se seguem serão desenvolvidos dois cálculos proposicionais.

O primeiro deles é uma formalização de uma lógica proposicional extendida, que foi definida sob a forma equivalente de "dedução natural" na construção do sistema KHEOPS [PHILLIPE 89].

O segundo é uma apresentação de um cálculo proposicional paraconsistente, que segue a linha das lógicas anotadas como definidas em [BLAIR 88] e [COSTA 90].

A principal contribuição nesta parte do trabalho refere-se à formalização da questão relativa à integração entre teorias lógicas, funções e relações qualitativas, dentro dos cálculos expostos.

3.1 O cálculo $P\mathcal{D}_A$

Seja $A = (a_1 a_2 \ldots)$ um conjunto finito e ordenado de símbolos proposicionais, cujos elementos serão chamados *atributos* [1]. A cada $a \in A$ está associado um domínio finito e não vazio de valores $\mathcal{D}_a$.

Definição 3.1.1 (ARIDADE) A *aridade* de um atributo $a \in A$ é denotada por $\|\mathcal{D}_a\|$ (cardinalidade do conjunto $\mathcal{D}_a$). Sem perda de generalidade os elementos de $\mathcal{D}_a$ de cardinalidade $\|\mathcal{D}_a\| = n + 1$ serão associados de forma simplificada a naturais $0, 1, \ldots n \in \mathbf{N}$.

Definição 3.1.2 (ESTADO) Seja

$$\mathcal{D}_A = \prod_{a \in A} \mathcal{D}_a \ .$$

O conjunto das atribuições $\mathcal{W} : A \longrightarrow \mathcal{D}_A$ é chamado conjunto de *estados* sobre os atributos A [2]. Um estado $w \in \mathcal{W}$ especifica uma atribuição (no sentido de designação de um valor a uma variável) para os atributos em um instante. Observa-se que, como os domínios de cada atributo são finitos e o número de atributos também é finito, o conjunto de estados $\mathcal{W}$ também é finito.

Definir-se-á a seguir o cálculo proposicional atemporal sobre $\mathcal{D}_A$, que será denotado $P\mathcal{D}_A$.

3.1.1 Sintaxe

- Os símbolos primitivos de $P\mathcal{D}_A$ são:

  1. atributos $a \in A$;
  2. naturais 0, 1, ... que designam os valores possíveis de cada domínio;
  3. conectivos lógicos: $\neg$ (negação), $\rightarrow$ (implicação), *1* (verdadeiro), *0* (falso);

[1] Elementos genéricos de A serão denotados por $a, b, \ldots$ ao invés de a_k.

[2] Embora seja possível a definição de estado a partir da união dos domínios e não de seu produto cartesiano, optou-se por esta última operação devido à sua compatibilidade para com as estruturas de dados e algoritmos empregados na implementação do sistema, e que serão descritos posteriormente.

4. símbolos auxiliares: parênteses e chaves.

- O conjunto de fórmulas $\mathcal{F}$ de $P\mathcal{D}_A$ é dado por:

1. se $a \in A$ é um atributo e $i \in \mathcal{D}_a$, então $a\{i\}$ é uma fórmula atômica; 1 e 0 também são fórmulas atômicas;
2. se F_1 e F_2 são fórmulas, então $(\neg F_1)$ e $(F_1 \rightarrow F_2)$ são fórmulas;
3. toda fórmula é obtida pela aplicação das regras acima um número finito de vezes.

- Utilizar-se-ão também as seguintes simplificações de notação:

1. as definições usuais:
   $(F_1 \vee F_2) \equiv (\neg F_1 \rightarrow F_2)$,
   $(F_1 \wedge F_2) \equiv (\neg(\neg F_1 \vee \neg F_2))$, e
   $(F_1 \leftrightarrow F_2) \equiv ((F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1))$;
2. as convenções usuais de precedência de operadores e de eliminação de parênteses;
3. $a\{i_1 \ldots i_m\} \equiv a\{i_1\} \vee \ldots \vee a\{i_m\}$; se $\mathbf{D} = \{i_1 \ldots i_m\}$ é um sub-conjunto de $\mathcal{D}_a$, pode-se escrever também $a\mathbf{D}$ ao invés de $a\{i_1 \ldots i_m\}$.

3.1.2 Semântica

Definição 3.1.3 (INTERPRETAÇÃO) Uma *interpretação* I é a associação de um elemento de $\mathcal{D}_a$ para cada atributo a. Este elemento é denominado *valor* do atributo a sob I. I pode ser considerada formalmente como uma atribuição à $(a_1 a_2 \ldots)$, dentro do domínio produto $\mathcal{D}_A = (\mathcal{D}_{a_1} \times \mathcal{D}_{a_2} \times \ldots)$. Uma interpretação nesta lógica corresponde portanto à noção de estado, como visto anteriormente.

Definição 3.1.4 (VALORAÇÃO) A toda interpretação I está associada uma *valoração* $v_I : \mathcal{F} \longrightarrow \{0, 1\}$, definida da seguinte forma:

1. sempre se tem: $v_I(1) = 1$ e $v_I(0) = 0$;
2. se a é um atributo, então:
   - $v_I(a\{i\}) = 1$ se somente se (sse) $I(a) = i$;
   - $v_I(a\{i\}) = 0$ sse $I(a) = j$, com $i \neq j$;
3. se F_1 e F_2 são fórmulas, então:
   - $v_I(\neg F_1) = 1$ sse $v_I(F_1) = 0$;
   - $v_I(F_1 \rightarrow F_2) = 1$ sse $v_I(F_1) = 0$ ou $v_I(F_2) = 1$.

Definição 3.1.5

1. (TEORIA) Um conjunto qualquer de fórmulas Γ é chamado uma *teoria* sobre o conjunto de atributos A.
2. (SATISFAÇÃO) Uma interpretação I *satisfaz* uma fórmula F, denotando-se por $I \models F$ sse $v_I(F) = 1$; se $v_I(F) = 0$ diz-se que I *falsifica* F ($I \not\models F$).

3. (MODELO) Uma interpretação I é um *modelo* para uma teoria Γ sse $I \models F$, para toda fórmula $F \in \Gamma$; o conjunto de modelos de uma teoria Γ será denotado $M(\Gamma)$.
4. (CONSEQUÊNCIA SEMÂNTICA) Se Γ é uma teoria e F é uma fórmula, diz-se que F é uma *conseqüência semântica* de Γ, e escrever-se-á $\Gamma \models F$, sse para toda interpretação I na qual $v_I(G) = 1$ para toda $G \in \Gamma$, tem-se também $v_I(F) = 1$.
5. (TAUTOLOGIA) Uma fórmula F é *válida*, i.e. uma tautologia, sse $v_I(F) = 1$ para qualquer I; neste caso escrever-se-á $\models F$.

3.1.3 Axiomatização

A axiomatização de $P\mathcal{D}_A$ inclui:

- Os axiomas do cálculo proposicional clássico:

  1. $F_1 \rightarrow (F_2 \rightarrow F_1)$;
  2. $((F_1 \rightarrow (F_2 \rightarrow F_3)) \rightarrow ((F_1 \rightarrow F_2) \rightarrow (F_1 \rightarrow F_3)))$;
  3. $((\neg F_1 \rightarrow \neg F_2) \rightarrow ((\neg F_1 \rightarrow F_2) \rightarrow F_1))$;
     para fórmulas F_1, F_2, F_3;
     e o axioma que assegura a associação de um só valor do domínio para cada atributo em cada estado:
  4. $(a\{0\} \wedge \neg a\{1 \ldots n\}) \vee \ldots \vee (a\{n\} \wedge \neg a\{0 \ldots n-1\})$, com $n+1 = \|\mathcal{D}_a\|$ e para todo $a \in A$.

- Regra de inferência (*Modus Ponens*):

$$\frac{\vdash F_1 \rightarrow F_2, \quad \vdash F_1}{\vdash F_2}$$

Utilizou-se acima o símbolo $\vdash$, que é ligado às definições de dedução e de prova, apresentadas a seguir:

Definição 3.1.6 (DEDUÇÃO) Se Γ é um conjunto de fórmulas, uma *dedução a partir de* Γ é uma seqüência $(F_1 F_2 \ldots F_n)$ tal que para todo i com $1 \leq i \leq n$ tem-se: (a) F_i é um axioma da lógica; (b) F_i é um elemento de Γ; ou (c) F_i pode ser obtida a partir de dois membros anteriores da seqüência (e.g. F_j e F_k com $j < i$, $k < i$), pela regra de inferência (*Modus Ponens*). Neste caso escreve-se $\Gamma \vdash F_n$.

Definição 3.1.7 (PROVA) Uma *prova* é uma dedução a partir de $\Gamma = \emptyset$; F_n é dito um *teorema*, e escreve-se $\vdash F_n$.

Para este cálculo proposicional, os seguintes resultados podem ser obtidos:

Proposição 3.1.1 São válidas as seguintes fórmulas, para todo $a \in A$:

1. $a\mathcal{D}_a$;
2. $\neg a\mathbf{D} \leftrightarrow a(\mathcal{D}_a \setminus \mathbf{D})$;

3. $a\mathbf{D} \vee a\mathbf{E} \leftrightarrow a(\mathbf{D} \cup \mathbf{E})$.

O segundo ítem desta proposição estabelece uma forma alternativa de definir a negação em $P\mathcal{D}_A$.

Proposição 3.1.2 Para a inferência, podem ser utilizados:

1. o *Modus Ponens* generalizado:

$$\frac{a\mathbf{D},\ \ a\mathbf{D}' \rightarrow b\mathbf{E},\ \ \mathbf{D} \subseteq \mathbf{D}'}{b\mathbf{E}}$$

2. restrição de domínio:

$$\frac{a\mathbf{D},\ a\mathbf{E}}{a(\mathbf{D} \cap \mathbf{E})}$$

Proposição 3.1.3 O sistema axiomático apresentado é correto (*sound*) (i.e. se $\Gamma \vdash F$ então $\Gamma \models F$) e completo (i.e. se $\Gamma \models F$ então $\Gamma \vdash F$) para $P\mathcal{D}_A$.

Este resultado é uma conseqüência direta da adequação e completude do cálculo proposicional clássico e do fato que o axioma (4) acima produz a restrição de unicidade desejada às interpretações.

Exemplo 3.1.1 Sejam $A = (a\ b\ c\ d)$ com $\mathcal{D}_a = \mathcal{D}_b = \{0\ 1\ 2\}$, $\mathcal{D}_c = \mathcal{D}_d = \{0\ 1\}$ e:

$$\begin{aligned}\Gamma\ = \{\ & a\{0\ 1\} \wedge b\{1\ 2\} \rightarrow c\{0\}, && (a\{2\} \wedge \neg b\{1\} \rightarrow c\{0\}) \rightarrow d\{0\},\\ & a\{1\} \wedge c\{1\} \rightarrow d\{1\}, && a\{2\} \wedge c\{1\} \rightarrow b\{1\} \vee d\{1\},\\ & \neg(a\{0\ 2\} \wedge b\{0\ 1\} \wedge d\{0\}) && \}\end{aligned}$$

Os modelos de Γ podem ser obtidos de diversas formas. Neste exemplo 36 interpretações diferentes são possíveis, correspondendo ao produto das aridades dos atributos. Estas interpretações são os modelos da teoria vazia $\Gamma_\emptyset = \emptyset$. Cada fórmula impõe uma restrição sobre as interpretações que são satisfeitas. No exemplo para que a última fórmula de Γ acima $(\neg(a\{0\ 2\} \wedge b\{0\ 1\} \wedge d\{0\}))$ seja satisfeita é necessária a eliminação de 8 interpretações (as que não a safistazem):

a	b	c	d
0	0	0	0
0	0	1	0
2	0	0	0
2	0	1	0
0	1	0	0
0	1	1	0
2	1	0	0
2	1	1	0

As outras fórmulas da teoria causam o mesmo efeito. É portanto possível obter todos os modelos de uma teoria pela enumeração extensiva das suas interpretações e pela eliminação daquelas que não satisfazem as fórmulas que compõem a teoria. Obtêm-se assim $M(\Gamma)$ para o exemplo:

modelos	a	b	c	d
m0	1	0	0	0
m1	1	1	0	0
m2	0	2	0	0
m3	1	2	0	0
m4	2	2	0	0
m5	2	0	1	1
m6	2	2	1	1

Um método eficiente para a obtenção de $M(\Gamma)$ será descrito mais adiante (ver seção 3.11).

Proposição 3.1.4 Sejam Γ_1 et Γ_2 duas teorias sobre o mesmo conjunto de atributos A; tem-se: $M(\Gamma_1 \cup \Gamma_2) = M(\Gamma_1) \cap M(\Gamma_2)$.

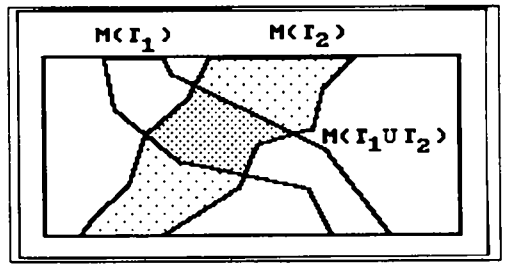

Figura 3.1: União de teorias e seus modelos

Em particular o acréscimo de novas fórmulas a uma teoria diminui o número de modelos desta teoria.

3.2 Teorias lógicas em PD_A, relações e funções

A simples definição de uma teoria lógica não é suficiente para que se obtenha a modelagem desejada de um SBCTR. Como citado anteriormente um aspecto primordial a considerar é a evolução do sistema frente ao fluxo de eventos que implementa as interações entre o mesmo e seu ambiente (reações).

Esta necessidade de modelar a reatividade do sistema será estabelecida a seguir, de forma coerente com o enfoque síncrono. Inicialmente definem-se conjuntos de atributos de entrada e de saída. As reações podem ser então consideradas da seguinte forma: a partir de valores fornecidos para os atributos de entrada obtêm-se os valores dos atributos de saída, de acordo com a teoria lógica que define o comportamento do sistema.

Definição 3.2.1 (RELAÇÃO INDUZIDA) Seja A um conjunto de atributos e $U = \{u_1\, u_2 \ldots u_p\}$, $Y = \{y_1\, y_2 \ldots y_q\}$ dois subconjuntos de A, denominados atributos de entrada e de saída. A toda teoria Γ está associada uma relação R_Γ sobre $\mathcal{D}_U \times \mathcal{D}_Y$, onde $\mathcal{D}_U = \mathcal{D}_{u_1} \times \ldots \times \mathcal{D}_{u_p}$ e $\mathcal{D}_Y = \mathcal{D}_{y_1} \times \ldots \times \mathcal{D}_{y_q}$; esta relação é chamada *relação induzida por* Γ, e é definida da seguinte forma:

$$\begin{array}{c} \text{Seja } \vec{u} \in \mathcal{D}_U,\ \vec{y} \in \mathcal{D}_Y, \quad \vec{u}\ R_\Gamma\ \vec{y} \quad \text{sse} \quad \text{existe algum} \quad m \in M(\Gamma) \\ \text{tal que} \quad m \models u_i\{(\vec{u})_i\},\ \ i = 1,p \quad \text{e} \quad m \models y_j\{(\vec{y})_j\},\ \ j = 1,q. \end{array}$$

onde $(\vec{u})_i$ representa a i-ésima coordenada de $\vec{u}$.

Deve-se notar que são aqui considerados os modelos das fórmulas lógicas completas e não os modelos parciais para as premissas e conclusões de um condicional ($IF \ldots THEN \ldots$), como é usual utilizar nos sistemas baseados em regras.

Definição 3.2.2 (PONTOS DE INCONSISTÊNCIA E DE INCOMPLETUDE) As entradas $\vec{u} \in \mathcal{D}_U$ que não estão em $Dom(R_\Gamma)$ são denominadas *pontos de inconsistência* de R_Γ. As entradas $\vec{u} \in \mathcal{D}_U$ tais que há várias saídas $\vec{y_1}, \ldots \vec{y_k}$ com $(\vec{u}\ R_\Gamma\ \vec{y_1}), \ldots (\vec{u}\ R_\Gamma\ \vec{y_k})$ são denominadas *pontos de incompletude* de R_Γ.

Alguns casos particulares podem ser considerados segundo a natureza da relação R_Γ (ver figura 3.2):

- se R_Γ é uma relação total, i.e., $Dom(R_\Gamma) = \mathcal{D}_U$ não há pontos de inconsistência;
- se R_Γ é uma função parcial $\mathcal{D}_U \not\longrightarrow \mathcal{D}_Y$, não há pontos de incompletude;
- se R_Γ é uma função (total), não há nem pontos de inconsistência nem de incompletude; para cada valor em entrada é possível se obter um único valor em saída.

Proposição 3.2.1 (RECÍPROCA) Toda relação R sobre $\mathcal{D}_U \times \mathcal{D}_Y$ (que são domínios finitos) define uma teoria Γ_R cujos modelos são os pontos de R, i.e., tal que a sua relação induzida é R.

Prova: É suficiente considerar $\Gamma_R = \{\ F_1 \vee \ldots \vee F_k\ \}$ onde as fórmulas são dadas por: $F_i = u_1\{\hat{u}_1\} \wedge \ldots \wedge u_p\{\hat{u}_p\} \wedge y_1\{\hat{y}_1\} \wedge \ldots \wedge y_q\{\hat{y}_q\}$, para todo elemento $(\hat{u}_1 \times \ldots \times \hat{u}_p \times \hat{y}_1 \times \ldots \times \hat{y}_q) \in R$.

Exemplo 3.2.1 Seja a relação $\otimes : \{0\ 1\ 2\ 3\} \times \{0\ 1\ 2\} \longrightarrow \{0\ 1\ 2\ 3\}$ definida pela tabela abaixo (para simplificar a notação escreve-se e ao invés de $\{e\}$ para os conjuntos com um só elemento):

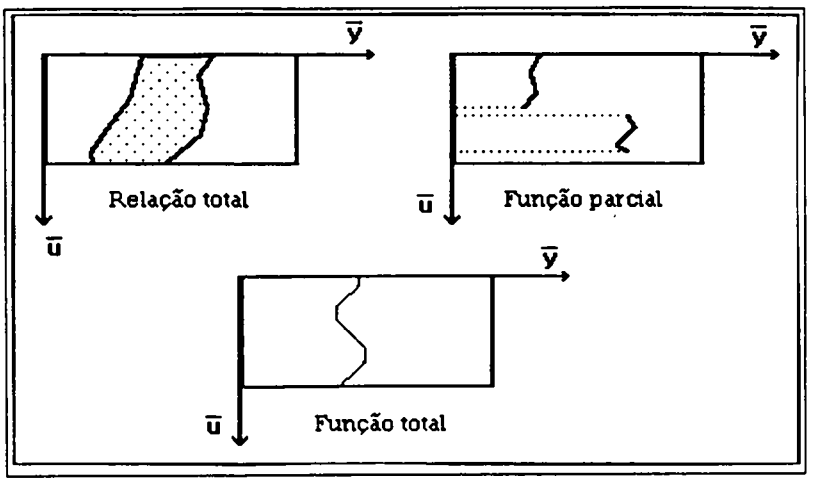

Figura 3.2: Diferentes tipos de relações

	a			
b	0	1	2	3
0	0	{1 2}	0	0
1	0	1	3	0
2	3	1	{0 2}	0

$c = a \otimes b$

É possível se escrever uma teoria $\Gamma_\otimes$ cujos modelos são justamento os pontos da relação $\otimes$. Para tanto sejam $\mathcal{D}_a = \mathcal{D}_c = \{0\ 1\ 2\ 3\}$ e $\mathcal{D}_b = \{0\ 1\ 2\}$; a teoria

$$\begin{aligned}\Gamma_\otimes \ = \{\ & (c\{0\} \wedge (a\{3\} \vee (a\{0\} \wedge b\{0\ 1\}) \vee (a\{2\} \wedge b\{0\ 2\}))) && \vee \\ & (c\{1\} \wedge a\{1\}) && \vee \\ & (c\{2\} \wedge ((a\{1\} \wedge b\{0\}) \vee (a\{2\} \wedge b\{2\}))) && \vee \\ & (c\{3\} \wedge ((a\{0\} \wedge b\{2\}) \vee (a\{2\} \wedge b\{1\}))) && \}\end{aligned}$$

tem os modelos desejados.

Exemplo 3.2.2 No caso de funções, i.e. quando a cada ponto no domínio corresponde um só ponto na imagem, a teoria pode ser dada por um conjunto de equivalências. Seja a função $\oplus : \{0\ 1\} \times \{0\ 1\} \longrightarrow \{0\ 1\}$ definida pela tabela:

	a	
b	0	1
0	0	1
1	1	0

$c = a \oplus b$

A teoria $\Gamma_\oplus$ sobre $A = (a\ b\ c)$ com $\mathcal{D}_a = \mathcal{D}_b = \mathcal{D}_c = \{0\ 1\}$:

$$\begin{aligned}\Gamma_\oplus \ = \{\ & c\{0\} \leftrightarrow\ (a\{0\} \wedge b\{0\}) \vee (a\{1\} \wedge b\{1\})\ , \\ & c\{1\} \leftrightarrow\ (a\{1\} \wedge b\{0\}) \vee (a\{0\} \wedge b\{1\})\ \}\end{aligned}$$

tem como modelos os pontos de $\Gamma_\oplus$.

3.3 Teorias lógicas em $P\mathcal{D}_A$, relações e funções qualitativas

Como mencionado no capítulo 2 no que diz respeito aos requisitos dos SBCTR, a integração entre o conhecimento heurístico e o conhecimento profundo é fundamental ao desenvolvimento dos SBCTR. Enquanto as regras e outras heurísticas fornecidas pelo especialista refletem procedimentos de operação, imitando o raciocínio humano, o objetivo da Física Qualitativa [KUIPERS 86], [IWASAKI 89] é o de fornecer equações que traduzam as leis físicas que governam o processo. Cabe salientar que esta integração esta cada vez mais presente nas aplicações, como e.g. em [STEYER 91].

A forma de representação deste conhecimento profundo que é mais facilmente integrável ao modelo proposto consiste na utilização do cálculo qualitativo [IWASAKI 89], como indicar-se-á a seguir.

No cálculo qualitativo considera-se que variáveis assumem valores dentro de domínios finitos. Por exemplo, se a e b são variáveis qualitativas de domínios $\mathcal{D}_a = \{0\ 1 \ldots \|\mathcal{D}_a\| - 1\}$ e $\mathcal{D}_b = \{0\ 1 \ldots \|\mathcal{D}_b\| - 1\}$ respectivamente, uma relação (função) qualitativa $\otimes(a, b)$ $(b = \otimes(a))$ relacionando as variáveis a e b é simplesmente uma relação (função) ordinária entre $\mathcal{D}_a$ e $\mathcal{D}_b$. Este procedimento pode ser generalizado para relações (funções) cujos domínios e imagens são constituídos por mais de uma variável.

Mostrar-se-á que teorias lógicas em $P\mathcal{D}_A$ e relações e funções qualitativas estão relacionadas da forma seguinte:

(teoria lógica em $P\mathcal{D}_A$) + (definição de atributos de entrada e saída)
$\equiv$
(relação qualitativa sobre os atributos)

A partir desta expressão, deduz-se que a integração entre as duas formas de conhecimento pode ser facilmente analisada, em especial no que se refere aos problemas de consistência e completude. No formalismo apresentado a resposta a esta questão é simples. Considera-se a seguir apenas o caso de relações e funções qualitativas (o caso de equações diferenciais qualitativas será analisado na parte correspondente ao caso temporal).

Se $f : A \longrightarrow B$ é uma função ou relação qualitativa sobre domínios finitos A e B, é possível a construção de uma teoria Γ_f como descrita na subseção precedente; sendo dado um conjunto de regras Γ_r o problema de consistência entre as duas formas de representação do conhecimento reduz-se à análise da teoria $\Gamma_r \cup \Gamma_f$.

Exemplo 3.3.1 Este exemplo ilustra a questão da integração. Seja o domínio $PQ = \{pp\ p\ m\ f\ ff\}$, com o seguinte significado:

pp	=	muito pequeno, muito fraco, muito mau, ...
p	=	pequeno, fraco, mau, ...
m	=	médio, normal, medíocre, ...
f	=	forte, elevado, bom, ...
ff	=	muito forte, muito elevado, excelente, ...

Este domínio pode ser codificado {0 1 2 3 4}; seja agora o operador $\otimes$ sobre PQ:

	a				
b	0	1	2	3	4
0	0	0	0	0	1
1	0	0	1	1	1
2	0	1	2	3	4
3	3	3	3	4	4
4	3	4	4	4	4

$a \otimes b$

A teoria correspondente a $\otimes$ é dada por:

$$\begin{aligned}\Gamma_{\otimes} = \{\ & c\{0\} \wedge a\{0\ 1\ 2\ 3\} \wedge b\{0\} \vee c\{0\} \wedge a\{0\ 1\} \wedge b\{1\} \vee \\ & c\{0\} \wedge a\{0\} \wedge b\{2\} \vee c\{1\} \wedge a\{2\ 3\ 4\} \wedge b\{1\} \vee \\ & c\{1\} \wedge a\{1\} \wedge b\{2\} \vee c\{2\} \wedge a\{2\} \wedge b\{2\} \vee \\ & c\{3\} \wedge a\{3\} \wedge b\{2\} \vee c\{3\} \wedge a\{0\ 1\ 2\} \wedge b\{3\} \vee \\ & c\{3\} \wedge a\{0\} \wedge b\{4\} \vee c\{4\} \wedge a\{4\} \wedge b\{2\} \vee \\ & c\{4\} \wedge a\{3\ 4\} \wedge b\{3\} \vee c\{4\} \wedge a\{1\ 2\ 3\ 4\} \wedge b\{4\} \vee \\ & c\{1\} \wedge a\{4\} \wedge b\{0\} \ \}\end{aligned}$$

Considere-se agora a teoria Γ_r, que representa o conhecimento de um especialista:

$$\begin{aligned}\Gamma_r = \{\ & a\{0\ 1\ 2\ 3\} \wedge b\{0\ 1\ 2\ 3\} \leftrightarrow c\{2\}\ , \quad a\{0\ 1\ 2\} \wedge b\{2\ 3\ 4\} \rightarrow c\{0\ 1\ 2\ 3\}\ , \\ & c\{0\ 1\ 2\} \rightarrow a\{0\}\ , \quad a\{0\} \rightarrow b\{2\ 3\}\ , \\ & a\{3\ 4\} \vee b\{1\ 2\} \rightarrow c\{2\}\ \}\end{aligned}$$

Os modelos de Γ_r são dados por:

modelos	a	b	c
m0	0	2	2
m1	0	3	2
m2	1	4	3
m3	2	4	3

O sistema formado por estas regras e pelo operador $\otimes$ é inconsistente, pois pode-se verificar que não há modelo de $\Gamma_{\otimes}$ que seja também um modelo de Γ_r.

Conclui-se assim que neste caso o conhecimento heurístico (do especialista) e profundo (modelo qualitativo) são inconsistentes.

Predicados sobre atributos

A sistemática apresentada permite a definição de *predicados* sobre os atributos, como um caso especial de **função qualitativa** de contra-domínio $\{0\ 1\}$. Assim, se $\mathcal{D}_a = \mathcal{D}_b = \{0 \ldots n\}$ sobre o qual se considera a relação de ordem usual, as tabelas abaixo definem os predicados "igual" e "menor".

$a = b$	a			
b	0	1	...	n
0	1	0	0...	0
1	0	1	0...	0
⋮	0	0	...	0
n	0	0	0...	1

Predicado "**igual**" $(a = b)$

$a < b$	a			
b	0	1	...	n
0	0	1	1...	1
1	0	0	1...	1
⋮	0	0	...0 1...	1
n	0	0	0...	0

Predicado "**menor**" $(a < b)$

3.4 O cálculo paraconsistente PT_A

Na seção anterior mostrou-se a relação entre os pontos de inconsistência e incompletude de uma teoria e a qualificação da relação associada (relação total, função parcial ou função global).

Embora a simples identificação destes pontos de inconsistência e incompletude já seja um passo muito importante pois permite conheçer as limitações do sistema [3], sua retirada não é tarefa simples, pois por exemplo as inconsistências podem provir de opiniões de diferentes especialistas sobre uma dada situação, o que dificulta sobremaneira sua eliminação [COSTA 89].

Outrossim, seria conveniente que o tratamento da inconsistência e incompletude fosse feito de modo automático pelo sistema, dentro do próprio formalismo de base adotado. Isto é o que ocorre por exemplo nos sistemas baseados em lógica difusa [ZADEH 78], já que neste formalismo, que se baseia na contribuição individual das regras e sua posterior composição, o problema é eliminado por ponderação [BONISSONE 90].

A situação ideal para garantir a consistência e a completude é aquela em que a relação entrada / saída é uma função total, i.e. a toda entrada corresponde uma saída bem determinada. Isto garante o fornecimento de uma resposta correspondente a todo estímulo proveniente da entrada.

O cálculo paraconsistente PT_A, que será apresentado a seguir se coloca como uma solução para resolver os problemas de incompletude e inconsistência. O termo *paraconsistente* indica que o referido cálculo admite teorias inconsistentes mas não triviais [COSTA 90]. O desenvolvimento seguirá a linha das lógicas anotadas de Blair e Subrahmanian [BLAIR 88].

A construção do cálculo PT_A (sintaxe, semântica, ...) pode ser feita de forma similar ao que foi feito para $P\mathcal{D}_A$ em seção anterior, sendo que ao invés de se utilizar a estrutura

[3] A hora atual poucos sistemas permitem esta identificação; um deles é o sistema KHEOPS já citado.

linear de conjunto dos domínios $\mathcal{D}_a$, empregar-se-á uma nova estrutura definida a seguir: um reticulado completo $\mathcal{T}_a$, para todo $a \in A$ [4].

Definição 3.4.1 (RETICULADO) Um *reticulado* é um conjunto $\mathcal{T}$ parcialmente ordenado no qual para todo par de elementos (μ, ν) existem o ínfimo $inf(\mu, \nu)$ e o supremo $sup(\mu, \nu)$ em $\mathcal{T}$. Um reticulado é dito *completo* se para todo $\mathcal{T}' \subset \mathcal{T}$, $inf(\mathcal{T}')$ e $sup(\mathcal{T}')$ existem em $\mathcal{T}$ [5].

Os elementos $\mu, \nu \in \mathcal{T}$ serão denominados *constantes de anotação*. O maior e o menor elementos de $\mathcal{T}$ serão denotados por $\top$ e $\bot$ respectivamente. Para a representação dos reticulados utilizar-se-ão os diagramas de Hasse [BURRIS 81].

Assim, considerando que A é um conjunto ordenado de atributos, e $\mathcal{T}_a$ é o reticulado associado ao atributo a, o cálculo é definido sobre

$$\mathcal{T}_A = \prod_{a \in A} \mathcal{T}_a \ .$$

3.4.1 Sintaxe

- PT_A é definido usando os seguintes símbolos primitivos:
  1. atributos $a \in A$;
  2. constantes de anotação $\mu, \nu, \ldots$ de cada reticulado domínio;
  3. conectivos lógicos: $\neg$ (negação), $\rightarrow$ (implicação), $\wedge$ (e), $\vee$ (ou), *1* (verdadeiro), *0* (falso);
  4. símbolos auxiliares: parênteses e dois pontos.
- O conjunto de fórmulas $\mathcal{F}$ de PT_A é dado por:
  1. para todo atributo $a \in A$ e $\mu \in \mathcal{T}_a$, então $(a : \mu)$ é uma fórmula atômica; *1* e *0* também são fórmulas atômicas;
  2. se F_1 e F_2 são fórmulas, então $(\neg\ F_1)$, $(F_1 \vee F_2)$, $(F_1 \wedge F_2)$ e $(F_1 \rightarrow F_2)$ são fórmulas;
  3. toda fórmula é obtida pela aplicação das regras acima um número finito de vezes.
- Utilizar-se-ão a simplificação $(F_1 \leftrightarrow F_2) \equiv ((F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1))$ e as convenções usuais de precedência de operadores e de eliminação de parênteses.

Definição 3.4.2 (LITERAL e HIPERLITERAL) Um *literal* é uma formula atômica ou sua negação. Se a é um atributo e μ é uma constante de anotação de $\mathcal{T}_a$, então a fórmula

$$\underbrace{\neg \ldots \neg}_{k \text{ vezes}}(a : \mu)$$

onde $\neg$ aparece k vezes é chamada de *hiperliteral* e será denotada por $\neg^k(a : \mu)$. Uma fórmula não atômica que também não é um hiperliteral é chamada uma fórmula *complexa*.

Definição 3.4.3 (NEGAÇÃO FORTE) Se F é uma fórmula, define-se a *negação forte* de F como:

$$(\sim F) \equiv (F \rightarrow ((F \rightarrow F) \wedge \neg(F \rightarrow F))).$$

[4] PT_A é construído a partir de uma modificação do trabalho [BLAIR 88], em que se consideram reticulados diferentes associados a cada atributo $a \in A$.

[5] Maiores detalhes sobre reticulados podem ser encontrados em [BURRIS 81], [SZASZ 71].

3.4.2 Semântica

Definição 3.4.4 (INTERPRETAÇÃO) Uma *interpretação* I é uma assinalação de um valor à $(a_1 a_2 \ldots)$, dentro do reticulado produto $(\mathcal{T}_{a_1} \times \mathcal{T}_{a_2} \times \ldots)$.

Definição 3.4.5 (VALORAÇÃO) Uma *valoração* $v_I : \mathcal{F} \longrightarrow \{0, 1\}$, está associada a toda I, como segue:

1. $v_I(1) = 1$ e $v_I(0) = 0$;
2. se a é um atributo, então:
   - $v_I((a : \mu)) = 1$ sse $I(a) \geq \mu$;
   - $v_I((a : \mu)) = 0$ sse $I(a) \ngeq \mu$;
   - $v_I(\neg^k(a : \mu)) = v_I(a : \neg^k \mu)$ onde a segunda ocorrência de $\neg$ denota uma função fixa mas arbitrária de $\mathcal{T}_a \longrightarrow \mathcal{T}_a$ que constitui o significado da negação $\neg$;
3. se F_1 e F_2 são fórmulas, então:
   - $v_I(F_1 \rightarrow F_2) = 1$ sse $v_I(F_1) = 0$ ou $v_I(F_2) = 1$;
   - $v_I(F_1 \wedge F_2) = 1$ sse $v_I(F_1) = 1$ e $v_I(F_2) = 1$;
   - $v_I(F_1 \vee F_2) = 1$ sse $v_I(F_1) = 1$ ou $v_I(F_2) = 1$;
4. se F é uma fórmula complexa (nenhuma das regras de negação acima pode ser aplicada) então $v_I(\neg F) = 1$ sse $v_I(F) = 0$.

As definições de *satisfação*, *modelo*, *fórmula válida* e *conseqüência semântica* são feitas como em $P\mathcal{D}_A$. Se $v_I(F) = 1$ para todo $F \in \Gamma$, F é satisfeita e a interpretação I é um modelo para Γ. Se $\models F$, F é válida em $P\mathcal{T}_A$. $\Gamma \models F$ sse para toda valoração em que $v_I(G) = 1$ para todo $G \in \Gamma$, tem-se $v_I(F) = 1$.

3.4.3 Caracterização da paraconsistência de $P\mathcal{T}_A$

Apresentam-se a seguir alguns resultados importantes sobre o cálculo $P\mathcal{T}_A$, que o caracterizam como paraconsistente [BLAIR 88]. O têrmo paraconsistente, como anteriormente mencionado, identifica teorias inconsistentes mas não-triviais.

Proposição 3.4.1

1. Se K é um esquema (i.e. um conjunto de axiomas) válido do cálculo proposicional positivo (i.e. que não inclui a negação), então K é válido em $P\mathcal{T}_A$;
2. para cada interpretação I e para cada atributo a existe $\mu \in \mathcal{T}_a$ tal que $v_I((a : \mu)) = v_I((a : \neg\mu)) = 1$.

Definição 3.4.6

1. A interpretação I tal que $v_I((a : \mu)) = v_I((a : \neg\mu)) = 1$ será denominada de $\neg$-*inconsistente*;

2. uma interpretação I é dita *não trivial* sse existe um atributo a e uma constante de anotação $\mu \in \mathcal{T}_a$ tal que $v_I((a : \mu)) = 0$.
3. uma interpretação I é dita *paraconsistente* sse I é $\neg$-inconsistente e não-trivial.

Proposição 3.4.2

1. $P\mathcal{T}_A$ é paraconsistente sse $\|\mathcal{T}_a\| \geq 2$ para algum $a \in A$;
2. para toda $F \in \mathcal{F}$ e para toda I tem-se:
   - $v_I((F \rightarrow F) \wedge \neg(F \rightarrow F)) = 0$;
   - $v_I(\sim F) = 1$ sse $v_I(F) = 0$;
   - $v_I(\sim F) = 0$ sse $v_I(F) = 1$.
3. $\models (a : \bot)$ para todo $a \in A$;
4. $\models ((a : \mu) \rightarrow (a : \nu))$ se $a \in A$ e $\mu \geq \nu$, com $\mu, \nu \in \mathcal{T}_a$;
5. se G é uma fórmula complexa então $\models (\neg G \leftrightarrow \sim G)$;
6. se $a \in A$ e $\mu \in \mathcal{T}_a$, tem-se $\models (\neg^k(a : \mu) \leftrightarrow (\neg^{k-1}(a : \neg\mu)))$;
7. se $v_I(F \rightarrow (a : \mu_j)) = 1$ para todo $j \in J$ (J é um conjunto de índices) então $v_I(F \rightarrow (a : \mu)) = 1$, onde $\mu = sup_{j \in J}(\mu_j)$;
8. se Q é um hiperliteral, em geral se tem $\not\models (\neg Q \leftrightarrow \sim Q)$ (em geral $\neg$ e $\sim$ não são equivalentes).

Prova: ver [BLAIR 88].

3.4.4 Axiomatização

No esquema de axiomas que se segue F_i indica uma fórmula qualquer, G_i indica uma fórmula complexa e a um atributo [BLAIR 88].

- Para fórmulas quaisquer:
  1. $(F_1 \rightarrow F_2) \rightarrow ((F_1 \rightarrow (F_2 \rightarrow F_3)) \rightarrow (F_1 \rightarrow F_3))$;
  2. $F_1 \rightarrow (F_2 \rightarrow F_1)$;
  3. $((F_1 \rightarrow F_2) \rightarrow F_1) \rightarrow F_1$;
  4. $(F_1 \wedge F_2) \rightarrow F_1$;
  5. $(F_1 \wedge F_2) \rightarrow F_2$;
  6. $F_1 \rightarrow (F_2 \rightarrow (F_1 \wedge F_2))$;
  7. $F_1 \rightarrow (F_1 \vee F_2)$;
  8. $F_2 \rightarrow (F_1 \vee F_2)$;
  9. $(F_1 \rightarrow F_2) \rightarrow ((F_3 \rightarrow F_2) \rightarrow ((F_1 \vee F_3) \rightarrow F_2))$;

- para fórmulas complexas:

1. $(G_1 \rightarrow G_2)) \rightarrow ((G_1 \rightarrow \neg G_2) \rightarrow \neg G_1)$;
2. $G_1 \rightarrow (\neg G_1 \rightarrow F)$;
3. $G \vee \neg G$;

- para atributos:

1. $(a : \bot)$;
2. $(a : \mu) \rightarrow (a : \nu)$ com $\mu \geq \nu$;
3. se $F \rightarrow (a : \mu_j)$ para todo $j \in J$, então $F \rightarrow (a : \mu)$ onde $\mu = sup_{j \in J}(\mu_j)$.

- Regra de inferência (*Modus Ponens*):

$$\frac{\vdash F_1 \rightarrow F_2, \quad \vdash F_1}{\vdash F_2}$$

Apresentam-se a seguir os principais teoremas sobre PT_A [BLAIR 88]:

Proposição 3.4.3 O sistema axiomático apresentado é correto e completo para PT_A.

Proposição 3.4.4 Se T_a é finito para todo $a \in A$ então PT_A é decidível.

Proposição 3.4.5 Se T_a é finito para todo $a \in A$ e se $F \in \mathcal{F}$ é tal que $\Gamma \vdash F$ então existe um subconjunto finito Γ' de Γ tal que $\Gamma' \vdash F$ (completude fraca).

Exemplo 3.4.1 Seja $A = (a\ b\ c)$ com $T_a = \{0\ 1\}$, $T_b = \{\bot\ 0\ 1\ \top\}$ e $T_c = \{\bot\ \mu_1\ \mu_2\ \mu_3\ \mu_{12}\ \top\}$, com a estrutura de reticulado indicada à figura 3.3.

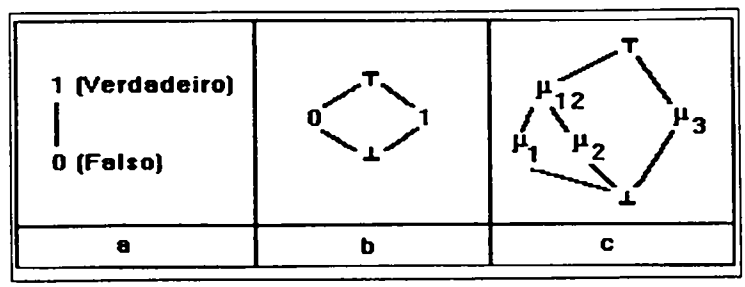

Figura 3.3: Reticulados adotados para os atributos a, b, c

Se Γ é a teoria composta pelas fórmulas $(a : 1) \wedge \neg(b : 0) \rightarrow (c : \mu_1)$, $(a : 1) \wedge (c : \mu_3) \rightarrow (c : \mu_2)$ e $\sim (a : 0)$, pode-se calcular $M(\Gamma)$ de acordo com a definição da semântica de PT_A.
Os modelos encontrados são:

modelos	a	b	c
m0	*1*	$\perp$	μ_1
m1	*1*	$\perp$	μ_{12}
m2	*1*	$\perp$	$\top$
m3	*1*	*0*	μ_1
m4	*1*	*0*	μ_{12}
m5	*1*	*0*	$\top$
m6	*1*	*1*	$\perp$
m7	*1*	*1*	μ_1
m8	*1*	*1*	μ_2
m9	*1*	*1*	μ_{12}
m10	*1*	*1*	$\top$
m11	*1*	$\top$	$\perp$
m12	*1*	$\top$	μ_1
m13	*1*	$\top$	μ_2
m14	*1*	$\top$	μ_{12}
m15	*1*	$\top$	$\top$

Dada uma teoria Γ, a ordem presente nos reticulados $\mathcal{T}_a$ induz naturalmente uma ordem $\preceq$ sobre $M(\Gamma)$ da seguinte forma:

$m_1 \preceq m_2$ sse para todo $a \in A$ tem-se $m_1(a) \leq m_2(a)$ em $\mathcal{T}_a$.

Este ordenamento pode ser utilizado para a análise de questões especulativas sobre os modelos [COSTA 89], nas quais se deseja saber da possibilidade do sistema em atender determinadas condições.

3.4.5 Casos particulares de reticulados

Nesta seção apresentam-se algumas particularizações de $P\mathcal{T}_A$, que servirão para mostrar a potencialidade e a generalidade do mesmo. Estas particularizações são obtidas como casos especiais dos reticulados $\mathcal{T}_a$, mostrando que vários cálculos podem ser considerados como casos particulares de aplicação do formalismo paraconsistente. Sem perda de generalidade considerar-se-á nesta seção que os reticulados associados a todos os atributos são idênticos, i.e., $\mathcal{T}_{a_m} = \mathcal{T}_{a_n}$ para todo $a_m, a_n \in A$. Outros casos podem ser facilmente obtidos a partir destes considerando a composição conveniente.

Para comparar sistemas lógicos será adotado o critério semântico (ver [AUDUREAU 89]), que utiliza a seguinte definição:

Definição 3.4.7 Dados dois sistemas lógicos $\mathcal{L}_1$ e $\mathcal{L}_2$ cujas interpretações estão definidas sobre a mesma classe de modelos, dir-se-á que $\mathcal{L}_1$ é *menos expressivo* que $\mathcal{L}_2$, sse para qualquer fórmula $F_1 \in \mathcal{L}_1$ existe $F_2 \in \mathcal{L}_2$ tal que:

$$\forall M,\ M \models_{\mathcal{L}_1} F_1 \text{ sse } M \models_{\mathcal{L}_2} F_2 .$$

O cálculo proposicional "clássico"

Considere-se o caso em que o reticulado $\mathcal{T}_a = \mathbf{2}, \forall a \in A$ é dado na figura 3.4, e a seguinte definição para a função negação $\neg$:

$$\begin{cases} \neg(1) = 0 \text{ e} \\ \neg(0) = 1. \end{cases}$$

Estes elementos definem o cálculo paraconsistente $P2$.

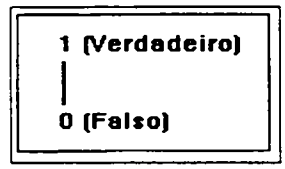

Figura 3.4: O reticulado de base para $P2$

Seja ainda o cálculo proposicional clássico PC como definido usualmente (ver por exemplo [MENDELSON 79], [HAMILTON 78]); neste caso vale o seguinte resultado:

Proposição 3.4.6 PC é menos expressivo que $P2$ (aqui se utilizam na correspondência os conectivos $\rightarrow$, $\wedge$, $\vee$ e $\sim$; note-se o emprego da negação forte $\sim$ e não de $\neg$).

Prova: Considerando a definição 3.4.7, tem-se:

- dada uma variável proposicional $p \in PC$, considere-se $(p : 1) \in P2$; se I é uma interpretação tal que $I(p) = 1$; então $I((p : 1)) = 1$ em $P2$; da mesma forma se $I(p) = 0$, tem-se $I((p : 1)) = 0$ em $P2$ (no entanto observa-se que para $I(p) = 1$ ou $I(p) = 0$ tem-se $I((p : 0)) = 1$ em $P2$);
- para as fórmulas do tipo $F_1 \rightarrow F_2$, $F_1 \wedge F_2$, $F_1 \vee F_2$ a equivalência decorre diretamente das definições de satisfação em PC e $P2$;
- para fórmulas que envolvem a negação a equivalência ocorre diretamente da proposição 3.4.2.

Cabe observar que a correspondência entre PC e $P2$ fica exata se em $P2$ se consideram apenas átomos do tipo $(a : 1)$.

O cálculo $P\mathcal{D}_A$

Seja $\mathcal{T}_a = -2^{\mathcal{D}_a}$, o reticulado reverso do reticulado das partes do conjunto $\mathcal{D}_a$, para todo $a \in A$, considerado como uma álgebra de Boole, ou seja, tendo como função negação o complemento usual de conjuntos $\neg a\{i_1 i_2 \ldots i_p\} = \mathcal{D}_a \setminus a\{i_1 i_2 \ldots i_p\}$.

De forma similar ao que foi visto no ítem anterior, uma correspondência entre $P\mathcal{D}_A$ e $P\mathcal{T}_A$ pode ser estabelecida. É fácil verificar que a pertinência do valor v_a de um atributo a ao conjunto $\mathbf{D}$ que compõem uma fórmula - definição de satisfação em $P\mathcal{D}_A$ - equivale à superioridade deste valor (tomado como conjunto atômico) em relação a $\mathbf{D}$ dentro do reticulado reverso das partes de $\mathcal{D}_a$ ($-2^{\mathcal{D}_a}$) - definição de satisfação em $P\mathcal{T}_A$. Desta forma $P\mathcal{D}_A$ é menos expressivo que $P - 2^{\mathcal{D}}_A$.

Exemplo 3.4.2 Seja a com $\mathcal{D}_a = \{0\ 1\ 2\}$; a fórmula $a\{0\ 1\}$ em $P\mathcal{D}_A$ é satisfeita para as interpretações em que a assume os valores 0 ou 1. Ora estes valores (tomados como conjuntos atômicos) verificam a noção de satisfação em $P\mathcal{T}_A$ considerando-se o reticulado $-2^{\{0\ 1\ 2\}}$ associado ao atributo a (ver figura 3.5), pois são superiores a $\{0\ 1\}$ segundo a ordem do reticulado.

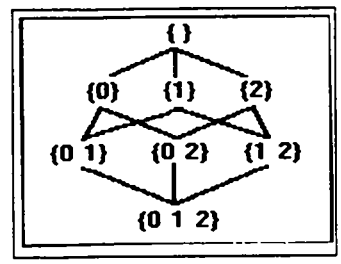

Figura 3.5: O reticulado $\mathcal{T}_a$ para $\mathcal{D}_a = \{0\ 1\ 2\}$

Outros reticulados utilizados

Alguns outros reticulados são de interesse, por darem origem a lógicas que têm aplicações importantes, tais como:

1. o reticulado 4 (ver figura 3.6), no qual a função negação é definida da seguinte forma: $\neg(1) = 0$, $\neg(0) = 1$, $\neg(\bot) = \bot$, $\neg((\top) = \top$; o cálculo $P4$ nele baseado é utilizado no estudo do problema de inconsistência em bases de conhecimentos e em procedimentos de resolução de cláusulas não-Horn [BLAIR 88], [COSTA 90].

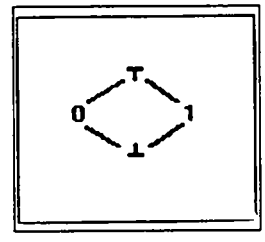

Figura 3.6: O reticulado "4"

2. o intervalo real $[0, 1]$, com a negação sendo definida por $\neg(\mu) = (1 - \mu)$, que dá origem à lógica $P[0, 1]$, que facilita o raciocínio qualitativo ou difuso [BLAIR 88];

3. o intervalo produto $CD = [0, 1] \times [0, 1]$, onde se interpreta a primeira coordenada como um grau de *crença* na proposição associada, e a segunda coordenada como um grau de *descrença* na mesma; a negação é definida por $\neg(\mu, \nu) = (\nu, \mu)$ [BLAIR 88], servindo para um tratamento formal da incerteza em Sistemas Especialistas; mostra-se a seguir um exemplo ilustrativo da aplicação deste reticulado.

Exemplo 3.4.3 Seja o conjunto de atributos $A = (t_1\ t_2\ t_3\ aux\ s_1\ s_2)$ onde t_1, t_2 e t_3 são os atributos de entrada, s_1 e s_2 são os atributos de saída e aux é uma atributo auxiliar; todos estes atributos tem para domínio o reticulado CD acima descrito; a teoria Γ abaixo representa as relações entre estes atributos em termos de crenças e descrenças de um especialista:

$$
\begin{aligned}
\Gamma = \{\ & t_1\ [0.95\ 0.00] \\
& t_1\ [0.90\ 0.00] \rightarrow s_1\ [1.00\ 0.00] \\
& t_2\ [1.00\ 0.00] \\
& t_2\ [1.00\ 0.00] \wedge aux\ [0.80\ 0.00] \rightarrow aux\ [0.80\ 0.00] \\
& aux\ [0.75\ 0.00] \rightarrow s_1\ [0.00\ 1.00] \\
& t_3\ [0.88\ 0.00] \\
& t_3\ [0.85\ 0.00] \rightarrow s_2\ [1.00\ 0.00] \quad \}
\end{aligned}
$$

Considerem-se os modelos de Γ: a primeira implicação obriga que se tenha s_1 [1.00 0.00]. Por outro lado como a partir das segunda e terceira implicações deduz-se que s_1 [0.00 1.00]. Considerando sobre $M(\Gamma)$ a ordem induzida por CD, observa-se que no menor modelo possível desta teoria s_1 [1.00 0.00]; por outro lado no maior modelo possível s_1 [1.00 1.00]. Quanto ao atributo s_2, cujo valor é deduzido unicamente em função da última implicação da teoria, tem-se s_2 [1.00 0.00] (tanto no maior quanto no menor modelos suportados).

Se a questão envolvida fosse a escolha entre as alternativas s_1 e s_2 esta última deveria ser escolhida, já que existe a possibilidade (um modelo) no qual s_1 é igualmente considerado em termos favoráveis ($C = 1$) e desfavoráveis ($D = 1$), o que não ocorre com s_2.

Parte II

O Caso Temporal

Como visto anteriormente, a definição de uma cronologia e o uso de uma representação temporal é uma característica fundamental a incorporar nos SBCTR. Os vários formalismos existentes fazem hipóteses substancialmente diferentes sobre a modelagem e computação do tempo, e.g. usando tempo denso ou discreto, linear ou ramificado [ALUR 91a]. Seguindo a proposta de fundamentar o sistema em elementos formais, propõe-se a seguir a utilização de lógicas temporais derivadas dos cálculos atemporais definidos anteriormente.

São adotadas as seguintes características para a estrutura temporal (ver figura 3.7):

- o tempo é discreto: esta abordagem é coerente com a utilização de eventos para a marcação do tempo que caracteriza o enfoque síncrono;
- o tempo possui uma origem, definida pelo estado inicial de funcionamento do sistema;
- o tempo passado é linear: o passado é conhecido e determinado;
- o tempo futuro é arborescente: o futuro é indeterminado, esta indeterminação se justifica pelas diferentes possibilidades de evolução do sistema.

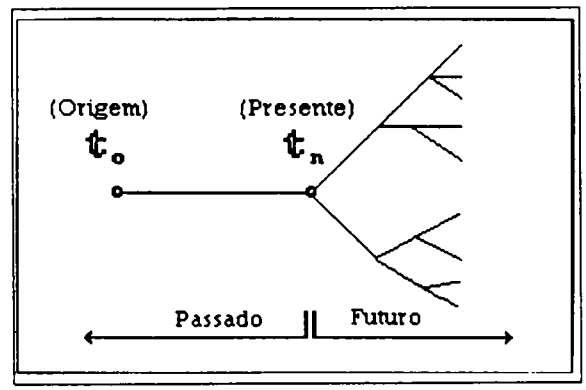

Figura 3.7: A estrutura temporal adotada

Geralmente se consideram estruturas temporais do mesmo tipo - lineares ou arborescentes - para um tratamento unilateral ou simétrico do tempo passado e futuro [BESTOUGEFF 89], [THAYSE 89]. Entretanto estruturas temporais do tipo apresentado neste trabalho se mostram mais adequadas à modelagem da evolução temporal de sistemas. Por outro lado a definição de uma lógica fundamentada neste tipo de estrutura implica certo grau de dificuldade para expressar as definições semânticas, como já observa [NEF 90].

Finalmente a lógica temporal paraconsistente tem sido considerada apenas de maneira convencional como uma extensão da forma proposicional correspondente [COSTA 86]; sua utilização apresenta o interesse de incorporar os benefícios relativos ao tratamento de situações de inconsistência e incompletude ao caso temporal.

Com o objetivo de aumentar ainda mais a expressividade da lógica adotada, utilizam-se autômatos finitos para expressar condições temporais; este tipo de enfoque não é usual, sendo mais comuns o uso de um operador "próximo" (*"next"*) generalizado [SOUZA 92], ou

o uso de gramáticas [AUDUREAU 89]. No entanto, como em [MICHEL 84], considera-se que o uso de autômatos é mais operacional, além de ser consistente com o enfoque síncrono visado para a implementação do sistema.

Os pontos acima descritos são especialmente considerados na próxima seção.

3.5 O cálculo PTD_A

Este cálculo será desenvolvido a partir do cálculo PD_A (ver seção 3.1), utilizando as mesmas definições de base e introduzindo a representação temporal.

3.5.1 Sintaxe

- Os símbolos primitivos de PTD_A são:
  - os atributos $a \in A$ e os naturais 0, 1, ... que designam os valores possíveis de cada domínio;
  - os conectivos lógicos: $\neg$ (negação), $\rightarrow$ (implicação), *1* (verdadeiro), *0* (falso), L (o último valor), P (em algum instante no passado), sX (em próximo instante possível), sF (em algum instante em um futuro possível), aF (em algum instante em todos os futuros possíveis); além de parênteses e chaves.
- As fórmulas de PTD_A são dadas pelo menor conjunto $\mathcal{F}$ que atende as condições abaixo:
  1. se $a \in A$ é um atributo e $i \in D_a$, então $a\{i\} \in \mathcal{F}$; *1* e *0* $\in \mathcal{F}$; estas fórmulas são chamadas atômicas e seu conjunto é denotado por $\mathcal{A}(\mathcal{F})$; e
  2. se F, F_1 e $F_2 \in \mathcal{F}$, então $(\neg\ F)$, $(F_1 \rightarrow F_2)$, $(L\ F)$, $(P\ F)$, $(sX\ F)$, $(sF\ F)$ e $(aF\ F) \in \mathcal{F}$.
- Além das simplificações definidas no caso atemporal (ver seção 3.1), utilizar-se-ão também as seguintes simplificações:
  1. $H\ F \equiv \neg(P\ \neg\ F)$ (em todos os instantes no passado);
  2. $aX\ F \equiv \neg(sX \neg\ F)$ (em todos os possíveis próximos instantes);
  3. $aG\ F \equiv \neg(sF \neg\ F)$ (em todos os instantes de todos os futuros possíveis);
  4. $sG\ F \equiv \neg(aF \neg\ F)$ (em todos os instantes de algum futuro possível).

3.5.2 Semântica

Definição 3.5.1 (TRAJETÓRIA) Seja W o conjunto (finito) de estados e R uma relação binária total sobre W, definidos por: $w_i\ R\ w_j$ sse w_j é um sucessor imediato de w_i. O conjunto das *trajetórias possíveis* sobre W é dado por

$C_\tau = \{\tau \in \mathbf{N} \longrightarrow W \mid \tau(i)\ R\ \tau(i+1)\}$,

i.e., pelo conjunto das funções que associam o estado correspondente a cada instante, respeitando as condições de acessibilidade entre os estados.

Definição 3.5.2 (HISTÓRIA) A *história* $h_n(\tau)$ associada à uma trajetória τ no instante $n \in \mathbf{N}$ é a sub-seqüência finita de τ: $< w_0, \ldots w_n >$. O instante w_0 é o *estado inicial* e w_n é o *estado atual* de τ.

Definição 3.5.3 (TRAJETÓRIA PARCIAL) Seja a relação $\sim_n$ sobre $\mathcal{C}_\tau$ definida por

$$\tau_1 \sim_n \tau_2 \quad sse \quad h_n(\tau_1) = h_n(\tau_2) .$$

Evidentemente $\sim_n$ é uma relação de equivalência sobre $\mathcal{C}_\tau$. Uma *trajetória parcial* de história $h_n(\tau)$ é a classe de equivalência $[\tau_n]$ de $\mathcal{C}_\tau$ associada à $h_n(\tau)$.

Definição 3.5.4 (w_n-RAMO) Toda seqüência infinita $r =< w_n, w_{n+1}, \ldots >$ tal que $w_i \; R \; w_{i+1}$, $\forall i$ é chamada um *w_n-ramo* sobre $\mathcal{W}$.

Definição 3.5.5 (EVOLUÇÃO) Seja τ_p uma trajetória parcial de história $h_n(\tau_p)$, e w_n o estado atual do sistema. Uma *evolução* de τ_p de n a $n+1$ após a recepção de w_{n+1} (o novo estado do sistema) é uma nova trajetória parcial τ_{p+1} definida por

$$\tau_{p+1} = < w_0, \ldots w_n \; w_{n+1} > \; / \sim_{n+1}$$

O estado w_{n+1} é o novo estado corrente do sistema no instante $n+1$ (ver figura 3.8).

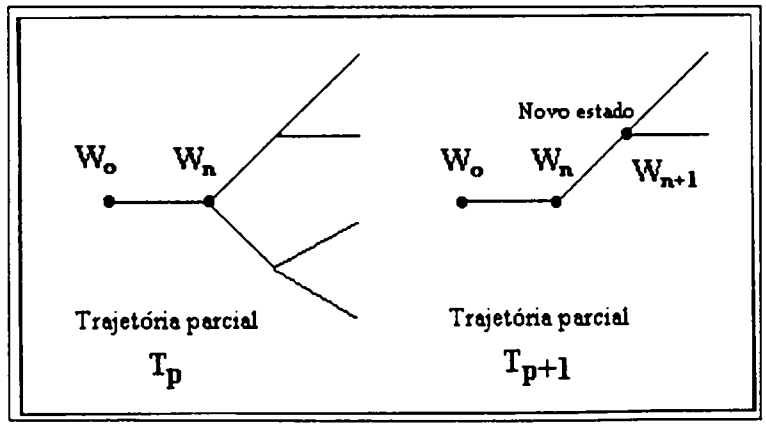

Figura 3.8: Evolução da trajetória quando da mudança de estado do sistema

Proposição 3.5.1 Tem-se sempre: $\tau_{p+1} \subset \tau_p$.

Definição 3.5.6 (INTERPRETAÇÃO) Uma *interpretação* I_{τ_p} sobre $(\mathcal{W}, R)$ restrita à trajetória parcial τ_p de história $h_n(\tau_p) = < w_0, \ldots w_n >$ é uma atribuição $I_{\tau_p} : A \times \mathcal{W} \longrightarrow \mathcal{D}_A$.

Definição 3.5.7 (VALORAÇÃO) A toda interpretação está associada uma *valoração*: $v_{I_{\tau_p}} : \mathcal{F} \times \mathcal{W} \longrightarrow \{0\ 1\}$, definida como segue:

- para toda I tem-se $v_{I_{\tau_p}}(1, w) = 1$ e $v_{I_{\tau_p}}(0, w) = 0$ em todo estado w;
- para os atributos, tem-se:

1. $v_{I_{\tau_p}}(a\{i\}, w) = 1$ sse $I_{\tau_p}(a, w) = i$, e
2. $v_{I_{\tau_p}}(a\{i\}, w) = 0$ sse $I_{\tau_p}(a, w) = j$, $i \neq j$, $\forall w \in \mathcal{W}$;

- para as fórmulas atemporais:

1. $v_{I_{\tau_p}}(\neg F, w) = 1$ sse não $v_{I_{\tau_p}}(F, w) = 0$, $\forall w \in \mathcal{W}$; e
2. $v_{I_{\tau_p}}(F_1 \rightarrow F_2, w) = 1$ sse $v_{I_{\tau_p}}(F_1, w) = 0$ ou $v_{I_{\tau_p}}(F_2, w) = 1$, $\forall w \in \mathcal{W}$;

- para as fórmulas temporais:

1. $v_{I_{\tau_p}}(LF, w_n) = 1$ sse $v_{I_{\tau_p}}(F, w_{n-1}) = 1$;
2. $v_{I_{\tau_p}}(PF, w_n) = 1$ sse $\exists w' \in h(\tau_p)$ tal que $v_{I_{\tau_p}}(F, w') = 1$;
3. $v_{I_{\tau_p}}(sXF, w_n) = 1$ sse $\exists w'$ com $w_n R w'$ e $v_{I_{\tau_p}}(F, w') = 1$;
4. $v_{I_{\tau_p}}(sFF, w_n) = 1$ sse existe um w_n-ramo $r =< w_n, w_{n_{i+1}}, \ldots >$ tal que $\exists w' \in r$ com $v_{I_{\tau_p}}(F, w') = 1$;
5. $v_{I_{\tau_p}}(aFF, w_n) = 1$ sse para todo w_n-ramo $r =< w_n, w_{n+1}, \ldots >$ tal que $\exists w' \in r$ com $v_{I_{\tau_p}}(F, w') = 1$.

As definições de teoria, satisfação, modelo e validade são feitas como em 3.1.5. Escrever-se-á $I, w_n \models F$ para indicar que a interpretação I satisfaz a fórmula F no instante w_n.

3.5.3 Axiomatização

O sistema temporal apresentado é similar aos sistemas DUX de Manna e Pnueli [AUDUREAU 89], [TALA 90] para o passado (lógica temporal linear) e a UB de Ben-Ari [AUDUREAU 89] , CTL [THAYSE 89] para o futuro (lógica temporal ramificada). O tratamento axiomático destes sistemas é apresentado em [AUDUREAU 89]. Um tratamento axiomático similar foi empregado no sistema K_b de Rescher e Urquhart [NEF 90]. Os axiomas são:

1. Os axiomas do cálculo atemporal $P\mathcal{D}_A$, incluindo a unicidade do valor de cada atributo num estado (ver 3.1.3):

$$(a\{0\} \wedge \neg a\{1 \ldots n\}) \vee \ldots \vee (a\{n\} \wedge \neg a\{0 \ldots n-1\});$$

2. para o passado (DUX [AUDUREAU 89]):

   - $H(F_1 \rightarrow F_2) \rightarrow (HF_1 \rightarrow HF_2)$;
   - $L(\neg F) \leftrightarrow \neg(LF)$;
   - $L(F_1 \rightarrow F_2) \rightarrow (LF_1 \rightarrow LF_2)$;
   - $HF \rightarrow F \wedge LHF$;
   - $H(F \rightarrow PF) \rightarrow (F \rightarrow HF)$;

3. para o futuro (UB [AUDUREAU 89]):

   - $aG(F_1 \rightarrow F_2) \rightarrow (aGF_1 \rightarrow aGF_2)$;

- $aX(F_1 \rightarrow F_2) \rightarrow (aXF_1 \rightarrow aXF_2)$;
- $aGF \rightarrow F \wedge aXF \wedge aGaXF$;
- $aG(F \rightarrow aXF) \rightarrow (F \rightarrow aGF)$;
- $aG(F_1 \rightarrow F_2) \rightarrow (sGF_1 \rightarrow sGF_2)$;
- $sGF \rightarrow F \wedge sXsGF$;
- $aGF \rightarrow sGF$;
- $aG(F \rightarrow sXF) \rightarrow (F \rightarrow sGF)$.

4. regras de inferência:

- *Modus Ponens*:

$$\frac{\vdash F_1 \rightarrow F_2, \ \vdash F_1}{\vdash F_2}$$

- necessitação:

$$\frac{\vdash F}{\vdash PF} \ e \ \frac{\vdash F}{\vdash sGF}$$

Proposição 3.5.2 O sistema $PT\mathcal{D}_A$ é correto e completo para esta axiomatização (ver [AUDUREAU 89]).

3.6 O cálculo $PT\mathcal{T}_A$

Assim como se obteve o cálculo paraconsistente $P\mathcal{T}_A$ a partir de $P\mathcal{D}_A$, é possível se construir naturalmente um cálculo paraconsistente temporal a partir de $PT\mathcal{D}_A$.

3.6.1 Sintaxe

- Os símbolos primitivos de $PT\mathcal{T}_A$ são:

  1. os atributos $a \in A$, as constantes de anotação μ, ν de cada reticulado domínio $\mathcal{T}_a$;
  2. os conectivos lógicos: $\neg$, $\rightarrow$, $\wedge$, $\vee$, 1 , 0 , L ,P , sX , sF , aF ; bem como parênteses e dois pontos.

- O conjunto de fórmulas $\mathcal{F}$ de $PT\mathcal{T}_A$ é definido da seguinte forma:

  1. se a é um atributo e se $\mu \in \mathcal{T}_a$, então $(a : \mu)$ é uma fórmula atômica; 1 e 0 também são fórmulas atômicas;
  2. se F, F_1 e F_2 são fórmulas então $(\neg F)$, $(F_1 \rightarrow F_2)$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, (LF), (PF), (sXF), (sFF) e (aFF) são fórmulas de $PT\mathcal{T}_A$.

- As mesmas simplificações de notação empregadas em 3.5.1 também são utilizadas. Da mesma forma que no cálculo paraconsistente atemporal definem-se as noções de hiperliteral (3.4.2) e de negação forte (3.4.3).

3.6.2 Semântica

A semântica de $PT\mathcal{T}_A$ é obtida a partir de uma estrutura temporal como definida anteriormente, onde permanecem válidos os conceitos de *trajetórias possíveis*, *história de uma trajetória*, *trajetória parcial*, w_n-*ramo*, *evolução*, etc. Uma interpretação restrita à trajetória parcial de história $h_n(\tau_p)$ é, neste caso, uma atribuição $I_{\tau_p} : A \times \mathcal{W} \longrightarrow \mathcal{T}_A$.

A valoração é feita da maneira usual (ver 3.4.2 e 3.5.7); e.g. para as fórmulas atômicas, se a é um atributo e μ é uma constante de anotação, tem-se:

- $v_{I_{\tau_p}}((a : \mu)) = 1$ sse $I_{\tau_p}(a) \geq \mu$;
- $v_{I_{\tau_p}}((a : \mu)) = 0$ sse $I_{\tau_p}(a) \not\geq \mu$;
- $v_{I_{\tau_p}}(\neg^k(a : \mu)) = v_{I_{\tau_p}}(a : \neg^k\mu)$, onde $\neg$ denota a função negação (arbitrária mas fixa) $\neg : \mathcal{T}_a \longrightarrow \mathcal{T}_a$.

3.6.3 Axiomatização

A axiomatização deste sistema pode ser feita combinando-se convenientemente os axiomas dos sistemas PT_A e $PT\mathcal{D}_A$. Uma axiomatização similar pode ser encontrada em [COSTA 86].

3.7 Extensão das lógicas: novos operadores temporais

Um problema bem conhecido que surge neste tipo de lógica ($PT\mathcal{D}_A$, $PT\mathcal{T}_A$) é sua falta de expressividade [6] , sendo que o exemplo mais clássico é o do operador $pair(F)$. A semântica deste operador é a seguinte: a fórmula $pair(F)$ é satisfeita para uma seqüência de estados $< w_n, w_{n+1}, \ldots >$ sse F é uma fórmula satisfeita nos estados w_n, w_{n+2}, $w_{n+4}, \ldots$, i.e., nos estados *pares*. Com esta semântica o operador $pair(F)$ não pode ser expresso na lógica descrita acima (nem para o passado linear nem para o futuro arborescente).

Uma possibilidade de resolver este problema é aumentar a lógica por meio de gramáticas ou, equivalentemente, por autômatos. Seguir-se-á aqui um caminho similar ao de M. Michel [MICHEL 84], onde autômatos não-deterministas foram empregados. Os autômatos serão utilizados para a construção de condições temporais. Na exposição que segue será considerado apenas o caso $PT\mathcal{D}_A$; o caso $PT\mathcal{T}_A$ é semelhante, desde que se considere a definição correspondente para as fórmulas atômicas.

3.7.1 Autômatos finitos para expressar operadores sobre o passado

Para operadores referentes ao passado o tempo tem um instante de início (o instante inicial w_0) e um fim (o instante atual w_n). Assim a história de uma trajetória parcial τ_p pode ser encarada como uma palavra finita sobre o alfabeto $A \longrightarrow \{0\ 1\}$; da mesma forma

[6] Problema levantado por Wolper, ver [AUDUREAU 89].

a cada fórmula F corresponde uma palavra sobre $\{0\ 1\}$ que diz se F é satisfeita (ou não) a cada instante. Equivalentemente pode-se considerar palavras sobre $2^{\mathcal{A}(\mathcal{F})}$ (o conjunto das partes de $\mathcal{A}(\mathcal{F})$), que tem como i-ésimo caracter o conjunto das fórmulas atômicas que são satisfeitas no estado w_i (ver figura 3.9).

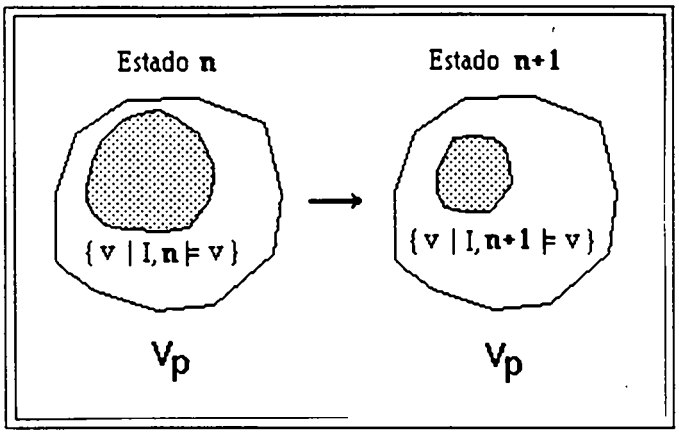

Figura 3.9: Evolução de uma interpretação temporal

Definição 3.7.1 Seja $\mathcal{X} \subset \mathcal{A}(\mathcal{F}) \cup \{0\ 1\}$ um subconjunto de fórmulas atômicas e seja $\Sigma = 2^{\mathcal{X}}$. O autômato finito [7] d define um *operador temporal* $d\ (X_1 \ldots X_n)$, para $X_i \in \mathcal{X}$, $i = 1, n$ da seguinte forma:

$d\ (X_1 \ldots X_n)$ é satisfeito no instante w_n sse o estado q do autômato d após a recepção da seqüência $h_n(\tau_p)|_{\mathcal{X}}$ é um estado terminal de d. $h_n(\tau_p)|_{\mathcal{X}}$ representa a restrição da história parcial $h_n(\tau_p)$ às fórmulas de $\mathcal{X}$.

Definição 3.7.2 (CONDIÇÃO TEMPORAL) Pode-se associar um atributo a_d a um autômato d e a um conjunto de fórmulas $\{X_1 \ldots X_n\}$ estendendo- se a interpretação I_{τ_p} a este novo atributo, considerado sobre o domínio $\mathcal{D}_d = \{0\ 1\}$, da forma seguinte:

- $I_{\tau_p}(a_d\{1\}) = 1$ sse $I_{\tau_p} \models d\ (X_1 \ldots X_n)$;
- $I_{\tau_p}(a_d\{0\}) = 1$ sse $I_{\tau_p} \not\models d\ (X_1 \ldots X_n)$.

Este atributo a_d será denominado uma *condição temporal* associada ao autômato d.

De fato sempre é possível fazer corresponder a uma fórmula F dada um atributo f de domínio $\mathcal{D}_f = \{0\ 1\}$, de forma similar ao exposto acima. Isto permite separar as fórmulas de uma teoria que contêm operadores atemporais, operadores sobre o passado e operadores sobre o futuro. Esta divisão será utilizada oportunamente.

---

[7] Um autômato finito determinista (*uma máquina de Moore*) é uma 5-upla $d = (Q, q_0, \Sigma, f, T)$ onde:

Q : é o conjunto de estados;
q_0 : é o estado inicial;
Σ : é o alfabeto do autômato;
f : é a função de transição $f : Q \times \Sigma \longrightarrow Q$;
T : é o conjunto de estados terminais.

Nas figuras que seguem para a representação de autômatos p e q representam fórmulas atômicas, o estado inicial é marcado por um arco sem origem e os estados finais são indicados por círculos duplos.

Exemplo 3.7.1 Seja $\mathcal{X} = \{p\}$; alguns operadores representados por autômatos são mostrados na figura 3.10: (a) o autômato d_1 representa o operador Pp; (b) o autômato d_2 representa o operador Hp; (c) o autômato d_3 representa o operador $pair(p)$.

A condição correspondente a d_1 é falsa enquanto a fórmula p não foi satisfeita (estado 1 do autômato); passa a ser verdadeira a partir do instante em que p foi satisfeita (transição entre os estados 1 e 2 do autômato) e a partir deste instante esta valor permanece inalterado (estado 2 do autômato). Esta descrição corresponde exatamente à semântica de Pp. d_2 e d_3 se interpretam de forma similar.

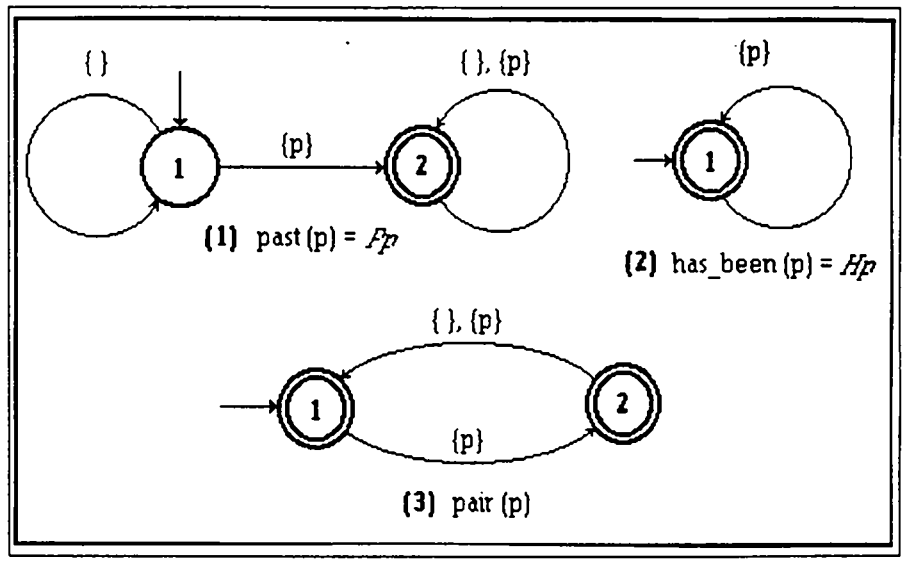

Figura 3.10: Alguns operadores e sua representação por autômatos

Exemplo 3.7.2 Seja $\mathcal{X} = \{p, q\}$; o autômato da figura 3.11 representa o operador temporal $since(p, q)$ (p é satisfeita desde que q é satisfeita).

Outros formalismos equivalentes a autômatos podem ser utilizados para a representação de condições temporais, e.g. transdutores e expressões regulares.

Para o caso dos transdutores [8] , pode-se considerar como alfabeto de saída $\{0\ 1\}$ [DOUGHERTY 88]. Vale então o resultado seguinte:

[8] Um transdutor (*máquinas de Mealy*) é uma 6-upla $t = (Q, q_0, \Sigma, \Phi, f, g)$ onde:
Q : é o conjunto de estados;
q_0 : é o estado inicial;
Σ : é o alfabeto de entrada;
Φ : é o alfabeto de saída, $\{0\ 1\}$ neste caso;
f : é a função de transição $f : Q \times \Sigma \longrightarrow Q$;
g : é a função de saída $g : Q \times \Sigma \longrightarrow \Phi$.

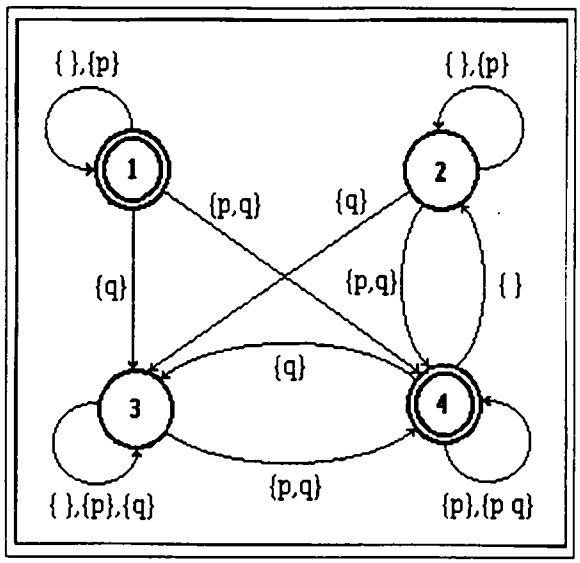

Figura 3.11: O operador temporal $since(p, q)$

Proposição 3.7.1 Se $d = (Q, q_0, \Sigma, f, T)$ define um operador temporal, a condição temporal d associada pode ser dada pelo transdutor

$t = (Q, q_0, \Sigma, \Phi, f, g)$, onde g é a função de saída dada por: $g(q, \sigma) = \begin{cases} 1 \text{ se } q \in T \\ 0 \text{ se } q \notin T \end{cases}$.

Exemplo 3.7.3 Seja $\mathcal{X} = \{p\}$; a figura 3.12 representa o atraso Lp.

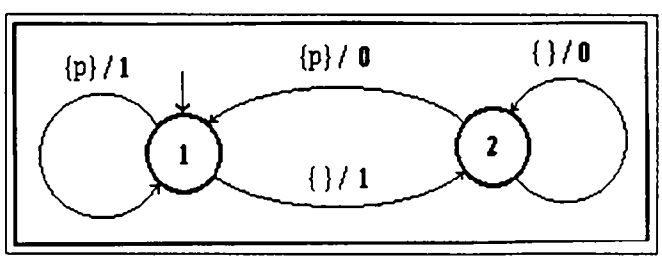

Figura 3.12: O atraso como um transdutor

Com efeito o valor emitido na saída do transdutor correponde exatamente ao estado de saída da transição correspondente: 1 para o estado 1 e 0 para o estado 2 do autômato; o estado é pois utilizado para armazenar a última ocorrência de p.

3.7.2 Autômatos para expressar operadores sobre o futuro

Assim como a lógica proposta foi estendida para o passado é possível se considerar um autômato d como agindo sobre os w_n-ramos de $\mathcal{W}$. Desta forma cada autômato d define dois operadores sd ($some$ = em alguma futura execução) e ad (all = em todas as futuras execuções) com a seguinte semântica: sd é satisfeita se existe um w_n-ramo (infinito) r tal que d aceita r; ad é satisfeita se para todo w_n-ramo r de origem em w_n, d aceita r.

Neste caso deve-se considerar os *autômatos de Buchi*, cuja verificação da aceitação é mais complexa [9].

Entretanto em muitos casos é possível se considerar prefixos de comprimento finito dos w_n-ramos, e fazer a verificação da aceitação sobre autômatos ordinários. Como considera-se aqui uma estrutura a número finito de estados, o processo de verificação de uma condição temporal futura nestes casos é sempre decidível.

Exemplo 3.7.4 Seja o autômato da figura 3.13. Este autômato define dois operadores temporais sobre o futuro: *sd* (em alguma execução futura possível F é satisfeita nos dois próximos instantes) e *ad* (em todas as futuras execuções possíveis F é satisfeita nos próximos dois instantes).

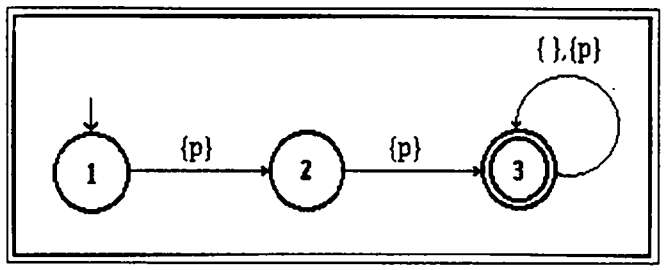

Figura 3.13: O operador nos-dois-próximos-instantes

3.7.3 Axiomatização:

A axiomatização de operadores temporais construídos com o auxílio de autômatos - na forma equivalente de gramática - se encontra descrita em [AUDUREAU 89].

Transdutores e Sistemas Dinâmicos

Na definição de uma condição temporal feita por um transdutor nada impede considerar como alfabeto de saída atributos de domínio diferente de $\{0\ 1\}$, gerando uma condição temporal mais geral: neste caso fica-se próximo de uma definição bem conhecida em Automática: os Sistemas Dinâmicos. A seção seguinte tratará esta questão.

Exemplo 3.7.5 Sejam $\mathcal{D}_a = \mathcal{D}_b = \{0\ 1\ 2\}$, e $\Sigma = \mathcal{D}_a$, $\Phi = \mathcal{D}_b$. O transdutor apresentado na figura 3.14 produz como saída o atraso unitário do atributo de entrada a. Denotar-se-á o atraso unitário por $b = a\$1$.

De forma similar ao atraso para p apresentado anteriormente (ver figura 3.12), cada estado armazena um valor possível para do atributo a no instante anterior.

[9] A construção de um autômato de Buchi que aceita uma fórmula temporal pode ser encontrada em [THAYSE 89].

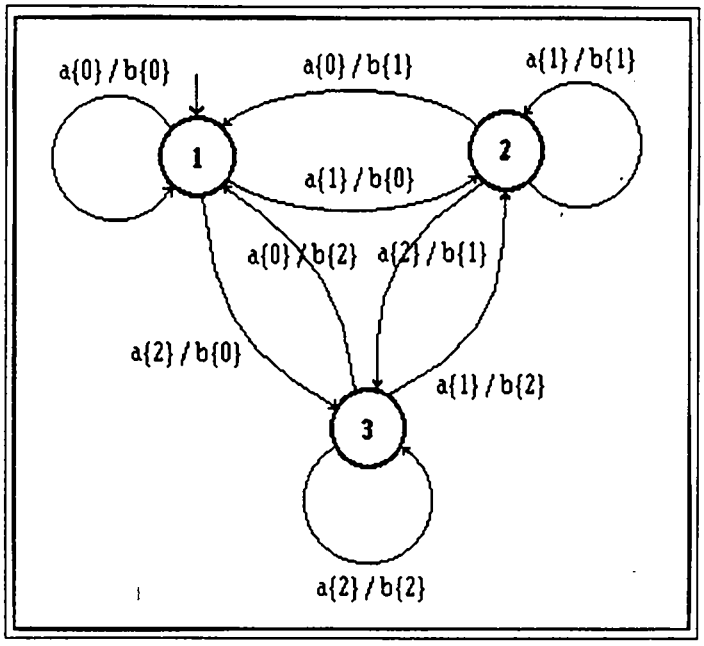

Figura 3.14: O atraso unitário $b = a\$1$

3.8 Relação entre Sistemas Dinâmicos sujeitos a Restrições e Teorias Temporais

De forma similar ao que foi visto no caso atemporal em relação às funções qualitativas (ver seção 3.3), estabelece-se nesta seção a correspondência entre os modelos de uma teoria lógica no caso temporal e as soluções de um sistema dinâmico qualitativo.

Esta correspondência pode ser resumida pela seguinte expressão:

$$\begin{gathered}(\text{teoria lógica em } PT\mathcal{D}_A) + \\ (\text{definição de atributos de entrada e saída}) + \\ (\text{limitações sobre a evolução do sistema}) \\ \equiv \\ (\text{Sistema Dinâmico sujeito a Restrições})\end{gathered}$$

Os objetivos básicos deste desenvolvimento são os seguintes:

- a incorporação direta no formalismo de Sistemas Dinâmicos Qualitativos, que comporão a base para a representação de conhecimento profundo no sistema; e
- a utilização de Sistemas Dinâmicos sujeitos a Restrições como geradores de modelos das teorias temporais associadas, o que servirá de base à implementação do sistema.

Mostrar-se-á que a mencionada equivalência inclui também as condições temporais geradas por autômatos, e é aplicável à lógica PTT_A.

3.8.1 Definições e principais conceitos sobre Sistemas Dinâmicos

Definições

Definição 3.8.1 (SISTEMA DINÂMICO SUJEITO A RESTRIÇÕES) Sejam os conjuntos disjuntos de atributos: de entrada U, de estado X, e de saída Y; um *Sistema Dinâmico sujeito a Restrições* SDR é representado por um modelo matemático formado pelas equações:

$$\begin{array}{ll} X_{i+1} = f(X_i, U_i, Y_i) & \text{(equação de estado)} \\ h(X_i, U_i, Y_i) = 0 & \text{(restrições)} \end{array}$$

que definem os valores dos atributos de saída (Y) a partir dos atributos recebidos em entrada (U), do estado atual do sistema (X) e do conjunto de restrições; f e h são funções ou relações que definem este comportamento desejado do sistema.

Uma *solução* deste sistema corresponde evidentemente a uma seqüência de atribuições de valor a X, Y e U que satisfaz às suas equações.

Este tipo de representação é similar ao utilizado na Teoria Clássica de Controle dos Sistemas Contínuos [KUO 85], [OGATA 82], e também serve de base à definição da linguagem síncrona SIGNAL [BENVENISTE 90].

A equação de estado limita as possíveis evoluções temporais do sistema, ao passo que as restrições correspondem ao relacionamento entre os atributos em um mesmo instante.

Definição 3.8.2 (SDR DE ATRASO UNITÁRIO) Um sistema dinâmico SDR cuja equação de estado é formada por equações escalares na forma $x_i^+ = x_{i+1}$ é dito *de atraso unitário*. Utilizar-se-á a notação $X = X^+ \$ 1$ para esta equação.

Um sistema dinâmico de atraso unitário pode ser implementado sob a forma de uma *máquina seqüencial de Huffman* [DOUGHERTY 88], um caso particular de transdutor, empregado classicamente na definição de circuitos seqüenciais [TAUB 84], [KOHAVI 78]. Um conjunto de fórmulas proposicionais concretizam as restrições $h(X_i, U_i, Y_i) = 0$, e podem envolver os atributos ligados ao estado presente (*present state* X) e próximo estado do sistema (*next state* X^+), ambos considerados no mesmo instante. Na figura 3.15 o bloco `1 / Z` indica a representação do atraso unitário, que corresponde à transferência do valor dos atributos do próximo estado ao estado presente, fazendo-se para tanto as atribuições necessárias.

Definição 3.8.3 (SISTEMA A NÚMERO FINITO DE ESTADOS) Considerando que o número de estados de um SDR como definido anteriormente é finito, pode-se descrever a evolução temporal de tal sistema por meio de um grafo $\mathcal{G}_\Gamma$ obtido da seguinte forma:

- os nós do grafo correspondem aos estados possíveis do sistema, i.e. aqueles que satisfazem suas restrições; e
- os arcos do grafo descrevem a relação de acessibilidade entre os estados, de acordo com a equação de estado do SDR.

Este grafo se denomina um *Sistema a número Finito de Estados* (SFE) [AUDUREAU 89] e pode ser diretamente interpretado como a estrutura modal de Kripke [HUGHES 68], [THAYSE 89] subjacente ao sistema considerado, podendo servir de base à verificação da satisfação de fórmulas temporais [10].

[10] Em geral utilizar-se-á a notação w_i para indicar o estado corrente do sistema no instante i.

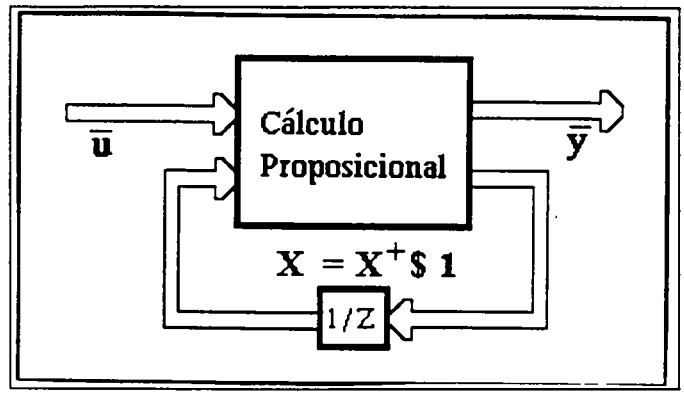

Figura 3.15: Sistema dinâmico como *máquina de Huffman*

3.8.2 *SDR* na forma clássica de um transdutor

Como mencionado anteriormente os *SDR* podem ser encarados como uma forma particular de transdutor [DOUGHERTY 88], [ARNOLD 90]. Esta seção trata de apresentar os *SDR* sob a forma utilizada classicamente na definição de trandutores.

As definições que seguem refinam o conceito de estado de um *SDR*:

Definição 3.8.4 (ESTADO RESTRITO) Denomina-se *estado restrito* a uma atribuição de valores aos atributos X de um *SDR*. Os estados restritos correspondem assim a uma restrição do conceito de estado apresentado na definição dos cálculos lógicos, quando se consideram apenas os atributos de estado X do *SDR*, o que será denotado por $w_i|_X$.

Definição 3.8.5 (EVENTOS) Os atributos que pertencem a $U \cup Y$ denominam-se os sinais observáveis do *SDR*. Um *evento* é uma atribuição de valor aos sinais observáveis. As atribuições dadas a U e Y denominam-se respectivamente eventos *de entrada* e *de saída*.

Pode-se agora considerar um *SDR* sob a forma clássica de um transdutor da seguinte maneira:

- Cada estado do transdutor corresponde a um ponto do produto cartesiano dos domínios das variáveis de estado (X), i.e. a um estado restrito;
- se no *SFE* que descreve a evolução do *SDR* há um arco indo de w_i para w_j, então há uma transição de $w_i|_X$ para $w_j|_X$, tendo para eventos de entrada $w_i|_U$ e para eventos de saída $w_i|_Y$.

3.8.3 Sistemas Dinâmicos sujeitos a Restrições para modelar Equações Diferenciais Qualitativas

Os *SDR* também podem modelar equações diferenciais qualitativas oriundas de sistemas dinâmicos contínuos [IWASAKI 89]. Neste caso a equação de estado está intimamente ligada às limitações impostas sobre a evolução do sistema dinâmico original, e.g. devido a continuidade dos domínios das variáveis ou ao relacionamento que ocorre entre atributos em função da derivada associada.

Um exemplo simplificado desta modelagem é apresentado a seguir.

Exemplo 3.8.1 (SISTEMA DE DILUIÇÃO DE PRODUTO) Seja um tanque utilizado como misturador, conforme indicado à figura 3.16. Um solvente puro (p_1) alimenta o tanque com uma vazão constante q_1; um produto p_2 com uma concentração C_2 entra no tanque a uma vazão também constante q_2. A solução de saída tem uma concentração C_3 e sai do tanque com uma vazão q_3. Como o volume V do tanque é considerado constante, deve-se ter $q_1 + q_2 = q_3$.

O taxa de acúmulo do produto p_2 no tanque é dado pela seguinte equação diferencial:

$$q_3C_3 - q_2C_2 = V\frac{dC_3}{dt}.$$

A concentração C_2 é controlada por uma válvula, de tal forma que $q_2 = K\theta$.

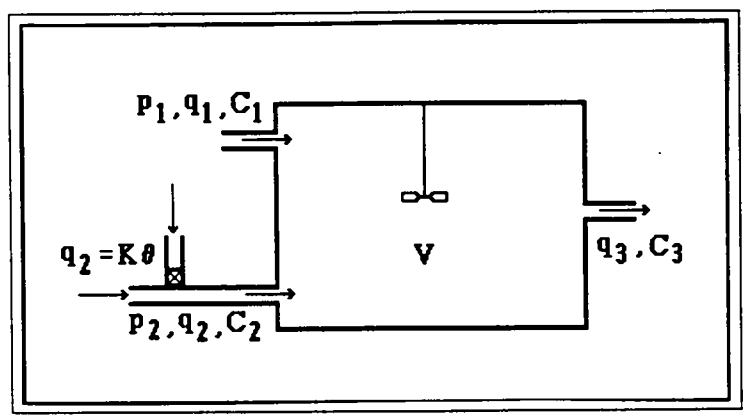

Figura 3.16: Sistema de diluição de produto

Considerem-se as variáveis qualitativas $[C_3]$, $[\frac{dC_3}{dt}] = [\dot{C}_3]$ e $[\theta]$, de domínios discretos $\mathcal{D}_{\dot{C}_3} = \mathcal{D}_{C_3} = \{-\ 0\ +\}$ e $\mathcal{D}_\theta = \{0\ +\}$ [11]. Associando-se uma variável qualitativa a cada componente da equação diferencial que rege o processo, obtem-se a seguinte equação qualitativa:

$$[\dot{C}_3] + [C_3] = [\theta].$$

Sejam ainda as seguintes regras de condução do processo de mistura, que realimentam o valor de saída $[C_3]$ para a entrada $[\theta]$:

$$[C_3]\{-\} \rightarrow [\theta]\{+\} \quad e \quad [C_3]\{+\} \rightarrow [\theta]\{0\}$$

Considerando em conjunto a equação qualitativa e as regras de condução, e supondo que "+" e "=" são operadores bem definidos sobre os domínios das variáveis, é possível se calcular os modelos (atribuições possíveis) para este sistema qualitativo:

modelos	$[C_3]$	$[\dot{C}_3]$	$[\theta]$
m1	+	−	0
m2	0	+	+
m3	0	0	0
m4	0	−	0
m5	−	+	+

[11] 0 pode significar uma faixa desejada de valores de C_3, enquanto que + e − podem significar concentrações fora da especificação.

Como mencionado acima, a evolução entre estes estados deve levar em conta a equação diferencial do sistema contínuo original. Várias são as regras a considerar; uma descrição destas regras pode ser encontrada em [IWASAKI 89]. A título de indicação em nosso exemplo não é possível a passagem direta de m1 para m5, devido à condição de continuidade da variável $[C_3]$ - não se pode passar de "−" para "+" sem cruzar o domínio "0".

A figura 3.17 mostra as evoluções possíveis para este exemplo. De forma similar ao apresentado anteriormente para teorias temporais, a estrutura obtida pode ser considerada como um SFE, o que comprova a uniformidade do enfoque em relação à introdução de equações qualitativas.

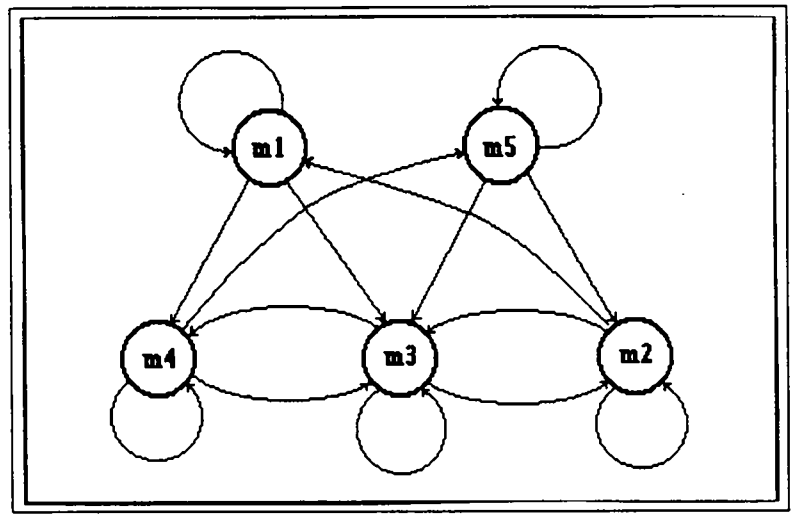

Figura 3.17: SFE que indica as evoluções possíveis para o sistema dinâmico qualitativo diluição de produto

3.8.4 Teorias temporais e Sistemas Dinâmicos sujeitos a Restrições

Consistência

Várias definições apresentadas anteriormente para os cálculos lógicos podem ser imediatamente aplicadas aos SDR, como as noções de *estado* e de *trajetória*. Em particular uma solução de um SDR é uma interpretação temporal que satisfaz suas equações:

Definição 3.8.6 (CONSISTÊNCIA ENTRE UMA INTERPRETAÇÃO TEMPORAL E UM SDR) Uma interpretação temporal I_{τ_p} é *consistente* com um sistema dinâmico SDR se para todo conjunto de atributos $U \cup X \cup Y$, a trajetória parcial τ_p é uma solução de SDR, i.e. se para todo estado $w_i, w_j \in \tau$ com $w_i R w_j$ a equação de estado e as restrições são satisfeitas.

Teorias temporais com operadores temporais sobre o passado e Sistemas Dinâmicos sujeitos a Restrições

O caso do atraso unitário

Considere-se inicialmente o caso de uma teoria temporal A_Γ que envolve apenas o operador L.

Uma fórmula do tipo $L\ p$, em que p é uma fórmula atômica, equivale diretamente ao atraso unitário $p\ \$\ 1$. Com o auxílio de atributos auxiliares podem ser estabelecidas equações do tipo $p = p^+\$1$.

As demais fórmulas da teoria se tornam então atemporais: como visto anteriormente na parte atemporal 3.2, estas fórmulas equivalem a uma relação envolvendo os atributos, que pode ser representada por $h(X_i, U_i, Y_i) = 0$.

Obtem-se desta forma um SDR de atraso unitário a partir de A_Γ. As soluções deste SDR (interpretações temporais consistentes com o sistema) correspondem diretamente aos os modelos da teoria A_Γ.

O caso geral das condições temporais

A correspondência entre os modelos de uma teoria e as soluções de um SDR pode ser estendida à toda teoria temporal P_Γ que envolve somente fórmulas referentes ao passado, inclusive as que incluem condições dadas por transdutores.

A proposição que segue estabelece este resultado:

Proposição 3.8.1 Toda condição temporal dada por um transdutor $t = (Q, q_0, \Sigma, \Phi, f, g)$ onde Σ e Φ são definidos sobre os atributos $u \in U$ e $y \in Y$ pode ser traduzida, com o auxílio de atributos auxiliares, em uma teoria temporal, ou equivalentemente, em um SDR de atraso unitário.

Prova: (algoritmo)
Seja $t = (Q, q_0, \Sigma, \Phi, f, g)$ com $\Sigma = \bigcup_i u_i\{d_i\}$, $d_i \in \mathcal{D}_{u_i}$ e $\Phi = \bigcup_j y_j\{d_j\}$, $d_j \in \mathcal{D}_{y_j}$.

1. consideram-se atributos auxiliares x^+ e x, utilizados para armazenar o próximo valor e o valor presente do estado corrente do transdutor t; estes atributos comporão a equação de estado $x_i^+ = x_{i+1}$, considerando-se $\mathcal{D}_{x^+} = \mathcal{D}_x = \{i \mid q_i \in Q\}$;
2. a partir de $f(q_l, \sigma) = q_m$ e $\sigma = u_i\{d_i\}$, obtem-se a fórmula $x^+\{m\} \leftrightarrow x\{l\} \wedge u_i\{d_i\}$;
3. a partir de $g(q_l, \sigma) = y_j\{d_j\}$ e $\sigma = u_i\{d_i\}$, obtem-se a fórmula $y_j\{d_j\} \leftrightarrow x\{l\} \wedge u_i\{d_i\}$.

Os modelos da teoria temporal P_Γ formada por estas fórmulas (*restrições*) mais a equação de estado $x = x^+\ \$\ 1$ permitem a obtenção do resultado desejado.

Embora a prova acima considere a teoria temporal sobre $PT\mathcal{D}_A$, a aplicação ao caso $PT\mathcal{T}_A$ é imediata: com efeito como são apenas consideradas fórmulas atômicas (que definem os atributos de entrada, de estado e de saída), basta considerar fórmulas do tipo $(a : \mu)$ ao invés de $a\{\mu\}$, e, para os atributos x e x^+ que relacionam os estados, e $\mathcal{T}_x = \mathcal{T}_{x^+}$ e.g. como indicado à figura 3.18.

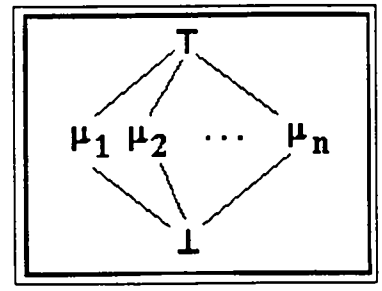

Figura 3.18: Um reticulado possível para os atributos x e x^+ em PTT_A

Observações sobre o cálculo dos modelos

Em geral não é necessário o uso da proposição acima para o cálculo dos modelos temporais que satisfazem um SDR, sendo possível encontrá-los diretamente a partir das funções de transição (f) e de saída (g) do autômato. No entanto a proposição acima mostra a coerência do enfoque em relação ao cálculo de modelos em teorias lógicas.

Outra observação importante é que a utilização explícita dos estados que compõem os autômatos de uma condição temporal como atributos ordinários em um sistema pode levar a um crescimento indesejável do número de modelos da teoria em questão. Para diminuir este problema é possível se considerar as condições temporais como variáveis booleanas de entrada no sistema, fazendo com que sua satisfação seja verificada de forma "interpretada", i.e. fora do mecanismo de cálculo de modelos do sistema.

Exemplo 3.8.2 Seja o transdutor t_1 representado à figura 3.19, que corresponde à condição temporal $y = always(u)$. A tabela abaixo resume os valores das funções de transição x^+ e de saída y associadas a este transdutor:

x^+/y	u	
x	0	1
0	1 / 0	0 / 1
1	1 / 0	1 / 0

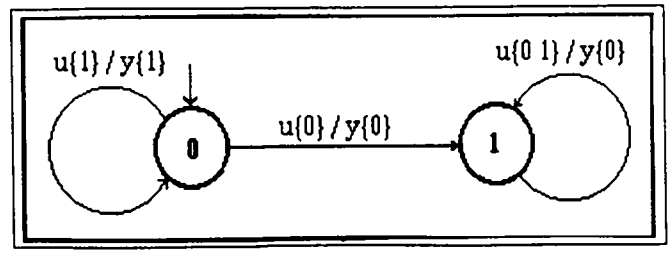

Figura 3.19: Um transdutor para $y = always(u)$

A partir desta tabela é possível construir a seguinte teoria:

$$\Gamma_1 = \{ \begin{array}{lcl} x^+\{0\} & \leftrightarrow & x\{0\} \wedge u\{1\} \\ x^+\{1\} & \leftrightarrow & u\{0\} \vee (x\{1\} \wedge u\{1\}) \\ y\{0\} & \leftrightarrow & u\{0\} \vee (x\{1\} \wedge u\{1\}) \\ y\{1\} & \leftrightarrow & x\{0\} \wedge u\{1\} \ \} \end{array}$$

Os modelos desta teoria, associados a equação de estado $x_i^+ = x_{i+1}$ correspondem ao sistema dinâmico definido a partir do transdutor t_1.

modelos	x^+	x	u	y
0	1	1	1	0
1	1	0	0	0
2	1	1	0	0
3	0	0	1	1

Exemplo 3.8.3 Seja o transdutor t_2 mostrado à figura 3.20, que representa a condição temporal $r = since(p, q)$, onde r, p e q representam fórmulas atômicas.

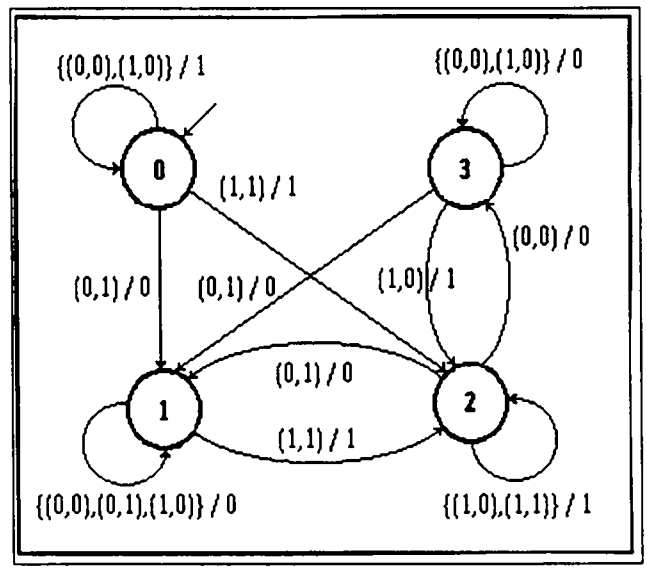

Figura 3.20: Um transdutor para $R = since(P, Q)$

A tabela com as funções de transição e de saída x^+ e r é dada por:

x^+/r	(p,q)			
x	(0,0)	(0,1)	(1,0)	(1,1)
0	0 / 1	1 / 0	0 / 1	2 / 1
1	1 / 0	1 / 0	1 / 0	2 / 1
2	3 / 0	1 / 0	2 / 1	2 / 1
3	3 / 0	1 / 0	3 / 0	2 / 1

Obtem-se de forma análoga ao caso precedente, a teoria:

$$
\begin{array}{lll}
\Gamma_2 = \{ & x^+\{0\} & \leftrightarrow (x\{0\} \wedge p\{0\} \wedge q\{0\}) \vee (x\{0\} \wedge p\{1\} \wedge q\{0\}) \\
& x^+\{1\} & \leftrightarrow (x\{0\ 1\ 2\ 3\} \wedge p\{0\} \wedge q\{1\}) \vee (x\{1\} \wedge p\{0\} \wedge q\{0\}) \vee \\
& & \quad (x\{1\} \wedge p\{1\} \wedge q\{0\}) \\
& x^+\{2\} & \leftrightarrow (x\{0\ 1\ 2\ 3\} \wedge p\{1\} \wedge q\{1\}) \vee (x\{2\} \wedge p\{1\} \wedge q\{0\}) \\
& x^+\{3\} & \leftrightarrow (x\{2\ 3\} \wedge p\{0\} \wedge q\{0\}) \vee (x\{3\} \wedge p\{1\} \wedge q\{0\}) \\
& r\{0\} & \leftrightarrow (x\{1\ 2\ 3\} \wedge p\{0\} \wedge q\{0\}) \vee (x\{0\ 1\ 2\ 3\} \wedge p\{0\} \wedge q\{1\}) \vee \\
& & \quad (x\{0\ 2\} \wedge p\{1\} \wedge q\{0\}) \\
& r\{1\} & \leftrightarrow \neg r\{0\} \quad \}
\end{array}
$$

3.8.5 Sistemas Dinâmicos sujeitos a Restrições como geradores de modelos de uma teoria temporal

O objetivo é, a partir de uma teoria temporal Γ qualquer e da definição de um conjunto de atributos de entrada, estado e saída, obter um Sistema Dinâmico sujeito a Restrições no qual as seqüências de execução (sua solução) são exatamente os modelos de Γ.

Encontra-se o SDR associado a uma teoria temporal qualquer Γ, procedendo da forma seguinte:

1. efetua-se uma partição da teoria Γ, com o uso de atributos auxiliares, em dois subconjuntos:

   - P_Γ que contém somente fórmulas atemporais e fórmulas que envolvem o passado (L, P, H e autômatos sobre o passado); e
   - F_Γ que inclui as fórmulas sobre futuros possíveis (sX, aX, sF, aF, sG, aG, e autômatos do tipo sd et ad sobre o futuro);

2. obtem-se todos os modelos de P_Γ, conforme o que foi apresentado anteriormente; a teoria P_Γ corresponde a um SDR de atraso unitário; como a equação de estado é suposta verificada por construção, as trajetórias que formam a solução deste SDR são os modelos de P_Γ;

3. verifica-se a satisfação das fórmulas temporais relacionadas aos futuros estados possíveis - o subconjunto F_Γ - sobre o SFE $\mathcal{G}_\Gamma$ oriundo de P_Γ; neste caso tem-se:

   - os nós são os modelos de P_Γ;
   - os arcos indicam a relação de acessibilidade entre os modelos, obtida a partir da equação de estado $X = X^+\$1$.

Observa-se que é possível que apenas parte do grafo $\mathcal{G}_\Gamma$ verifique todas as fórmulas da teoria: é justamente esta parte que formará os modelos da teoria temporal $\Gamma = P_\Gamma \cup F_\Gamma$.

O enfoque acima é geral; sistemas atemporais podem ser considerados como SDR estacionários, onde há um único estado que permanece inalterado durante a evolução do sistema. Isto também significa que não é imposta qualquer limitação sobre a evolução do sistema.

A seção 3.12 a seguir apresenta algoritmos e estruturas de dados convenientes para a implementação da sistemática exposta acima.

O exemplo a seguir ilustra a utilização dos SDR e dos trandutores correspondentes como geradores de modelos de teorias temporais.

Exemplo 3.8.4 Seja a teoria sobre A = (a b c d), com $\mathcal{D}_a = \mathcal{D}_b = \{0\ 1\}$, $\mathcal{D}_c = \{1\ 2\}$, $\mathcal{D}_d = \{0\ 1\}$ cuja equação de estado é $d = a\ \$\ 1$. Sejam $U = (a\ b)$, $Y = (c)$ os atributos de entrada e de saída e as fórmulas:

$$\begin{aligned}\Gamma\ = \{\ & a\{0\ 1\} \wedge b\{0\} \rightarrow c\{1\}, \quad c\{1\} \rightarrow d\{0\}, \\ & a\{0\} \wedge b\{1\} \rightarrow c\{2\}, \quad c\{2\} \wedge a\{0\} \rightarrow d\{1\}, \\ & a\{1\} \wedge sFb\{1\} \wedge c\{1\} \quad \vee\ aXd\{0\}\ \}\end{aligned}$$

O conjunto de modelos $M(\Gamma)$ pode ser obtido da seguinte forma:

1. partição $\Gamma = \Gamma_P \cup \Gamma_F$:

$$\begin{aligned}\Gamma_P\ = \{\ & a\{0\ 1\} \wedge b\{0\} \rightarrow c\{1\}, \\ & c\{1\} \rightarrow d\{0\}, \\ & a\{0\} \wedge b\{1\} \rightarrow c\{2\}, \\ & c\{2\} \wedge a\{0\} \rightarrow d\{1\}\ \}\ \text{e}\end{aligned}$$

$$\Gamma_F\ = \{\ a\{1\} \wedge sFb\{1\} \wedge c\{1\} \vee aXd\{0\}\ \}$$

2. o SDR já está na forma de atraso unitário, com equação de estado $d = a\ \$\ 1$;
3. os modelos de Γ_P são (o procedimento de cálculo será descrito a seguir 3.11):

modelos	a	b	c	d
m0	0	1	2	1
m1	1	1	2	0
m2	1	1	2	1
m3	0	0	1	0
m4	1	0	1	0
m5	1	1	1	0

4. para a verificação das fórmulas de Γ_F: apresenta-se na figura 3.21 o SFE associado à teoria Γ_P, sob a forma de um grafo. É sobre este grafo que serão verificadas as fórmulas de Γ_F, considerando-se cada nó como um mundo dentro de uma estrutura modal.

A figura 3.22 apresenta para cada sub-fórmula de Γ_F o conjunto de modelos que as satisfazem (procedimento de cálculo é apresentado em 3.12.1). A conclusão é que o SFE formado pelo grafo $\mathcal{G}_\Gamma \mid_{(m0\ m3\ m4\ m5)}$ é a estrutura que concretiza os modelos de Γ.

Este SFE pode também ser apresentado sob a forma de um transdutor, considerando que a variável de estado é d e utilizando-se a metodologia descrita anteriormente 3.8.2. A implementação de um sistema construído com base na teoria Γ e na equação de evolução $d = a\$1$ pode ser feita com base neste transdutor, de acordo com o enfoque síncrono para os STR (ver figura 3.23).

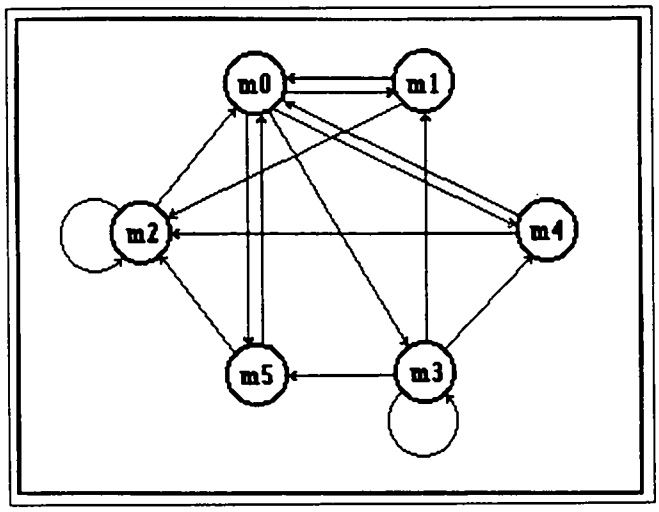

Figura 3.21: SFE associado à Γ

sub-fórmula	modelos
$a\{1\}$	m1 m2 m4 m5
$b\{1\}$	m0 m1 m2 m5
$sFb\{1\}$	m0 m1 m2 m3 m4 m5
$a\{1\} \wedge sFb\{1\}$	m1 m2 m4 m5
$c\{1\}$	m3 m4 m5
$a\{1\} \wedge sFb\{1\} \wedge c\{1\}$	m4 m5
$d\{0\}$	m1 m3 m4 m5
$aXd\{0\}$	m0 m3
$a\{1\} \wedge sFb\{1\} \wedge c\{1\} \vee aXd\{0\}$	m0 m3 m4 m5

Figura 3.22: Modelos que verificam as sub-fórmulas de Γ_F

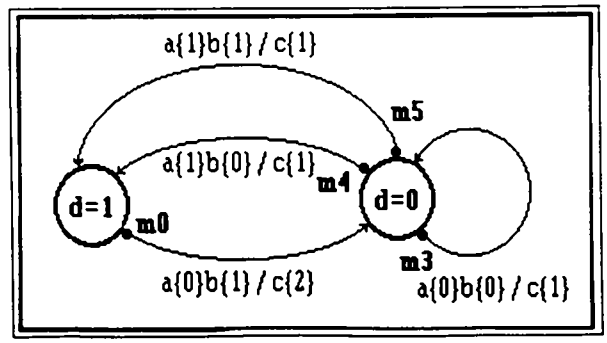

Figura 3.23: Transdutor associado a Γ

Parte III

A obtenção dos modelos de uma teoria: algoritmos e estruturas de dados

Nas seções precedentes os estudos sobre a relação entrada / saída e sobre a evolução dos sistemas foi baseado na geração dos modelos de teorias lógicas (em $P\mathcal{D}_A$, PT_A, $PT\mathcal{D}_A$, PTT_A). Nas seções seguintes são apresentados as estruturas de dados e os algoritmos que implementam esta geração.

A estrutura de dados básica utilizada para a representação dos modelos destas teorias é o *hipercubo*, aqui entendido como uma representação do domínio produto ou de um diagrama de Hasse no espaço n-dimensional [XUONG 92].

Já os algoritmos propostos para o cálculo dos modelos de uma teoria são fundamentados numa variante do método das conexões de Wallen e Bibel [BIBEL 82], [WALLEN 87].

3.9 Hipercubos e modelos

A estrutura hipercubo se baseia na representação concisa de conjuntos de dados (*set-theoretic data structures* [DOUGHERTY 88]), e na extensão das operações usuais sobre conjuntos para n-uplas destes conjuntos.

3.9.1 A estrutura de dados de base: os hipercubos

Definições

Definição 3.9.1 (HIPERCUBO) Sejam: A um conjunto de atributos e $\{a_j \mid j \in J\}$ um subconjunto de A , cujos atributos são referenciados por um conjunto de índices J; $\mathcal{D}_{a_j}$ o domínio associado a a_j; e D_{a_j} um subconjunto de $\mathcal{D}_{a_j}$. Neste texto denominar-se-á *hipercubo* (**hcubo**) h sobre um conjunto de índices J a um vetor de dimensão $\|J\|$ cuja j-ésima coordenada tem o valor D_{a_j}. Denotar-se-á $h = [D_{a_{j_1}} \ldots D_{a_{j_{\|J\|}}}]$. O conjunto J é considerado ordenado segundo a relação de ordem induzida por A.

Definição 3.9.2 (COBERTURA DE UM **hcubo**) A cada **hcubo** $h = [D_{a_{j_1}} \ldots D_{a_{j_{\|J\|}}}]$ está associado um conjunto de estados $\{a_j \mid j \in J\}$, i.e., um sub-conjunto de $\mathcal{D}_{a_{j_1}} \times \ldots \times \mathcal{D}_{a_{j_{\|J\|}}}$ da seguinte forma: $\{\vec{x} = (\ldots x_j \ldots) \in \mathcal{D}_{a_{j_1}} \times \ldots \times \mathcal{D}_{a_{j_{\|J\|}}} \mid x_j \in D_{a_j}, \forall j \in J\}$. Este conjunto de estados será chamado de *cobertura* $c(h)$ do **hcubo** h.

Exemplo 3.9.1 Sejam $A = (a\ b\ c)$, $\mathcal{D}_a = \mathcal{D}_b = \{0\ 1\ 2\}$, $\mathcal{D}_c = \{0\ 1\}$, e $J = \{0\ 1\ 2\}$. O **hcubo** $h = [\{0\ 1\ 2\}\ \{1\ 2\}\ \{0\}]$ tem por cobertura o conjunto de estados (ver figura 3.24):

$$c(h) = \{(0,1,0),(0,2,0),(1,1,0),(1,2,0),(2,1,0),(2,2,0)\}$$

No exemplo anterior pode-se constatar que os hipercubos são uma forma concisa para a representação de conjuntos simétricos de pontos.

Cabe ressaltar ainda que os **hcubo** são também adequados para o caso em que a atribuição feita por uma interpretação tem como domínio um reticulado (caso das lógicas PT_A e PTT_A), pois neste caso os domínios são vistos como conjuntos ordinários, i.e., desprovidos da relação de ordem associada.

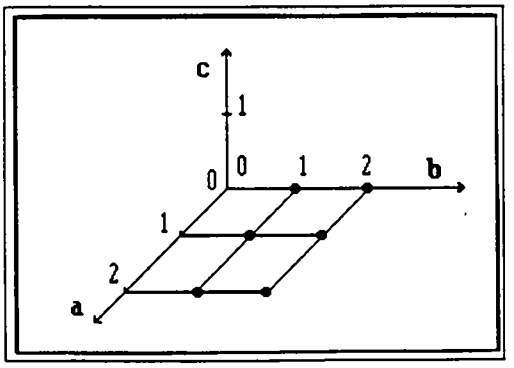

Figura 3.24: Um hipercubo e sua cobertura

Definição 3.9.3 (CONJUNTO NÃO REDUNDANTE DE HIPERCUBOS) Seja $H = \{h_k, k = 1, l\}$ um conjunto de **hcubo** de mesmos índices J, tal que para todo par (k_1, k_2) tem-se: $h_{k_1} \cap h_{k_2} = \emptyset$. Este conjunto será denominado um *conjunto não-redundante de hipercubos* (**Hcuboset**).

Definição 3.9.4 (COBERTURA E CARDINALIDADE DE UM **Hcuboset**) A *cobertura* de um **Hcuboset** H é definida por

$$c(H) = \bigcup_k c(h_k).$$

A *cardinalidade* de H (denotada $\|H\|$) é definida como o número de **hcubo** que o compõe.

Proposição 3.9.1 Sendo dada uma interpretação I sobre um conjunto de atributos A existe um **Hcuboset** tal que $c(H) = \{a_j\{m\} \mid I(a_j) = m\}$ (ou equivalentemente $v_I(a_j\{m\}) = 1$).

Definição 3.9.5 (EQUIVALÊNCIA DE **Hcuboset**) Dois **Hcuboset** H_1, H_2 são ditos *equivalentes* sse $c(H_1) = c(H_2)$.

Minimização de conjuntos de hipercubos

Dado um **Hcuboset** H, define-se o problema de minimização de H da seguinte forma: encontrar um **Hcuboset** H' equivalente tal que para todo **Hcuboset** H'' equivalente a H, tem-se: $\|H'\| < \|H''\|$. No caso da representação dos modelos de uma teoria lógica, a utilização do **Hcuboset** minimal é importante para melhorar a performance do procedimento de busca de modelos. Esta questão é similar à minimização de expressões lógicas, tratadas usualmente por métodos como os de Karnaugh, de Quine-McCluskey e o método do consenso [XUONG 92], [KOHAVI 78], [TAUB 84]. Entretanto no caso das teorias lógicas, a questão é ainda mais complexa, pois exige-se que as coberturas dos **hcubo** que compõem o **Hcuboset** não tenham pontos em comum: $\forall h_i, h_j \in H_r : c(h_i) \cap c(h_j) = \emptyset$.

Neste trabalho será utilizado um procedimento simplificado para efetuar a adição de um **hcubo** a um **Hcuboset**. A partir de um **hcubo** e de um **Hcuboset** fornecidos em

entrada o procedimento fornece um **Hcuboset** quase minimal que cobre os mesmos pontos que os elementos dados na sua entrada. Este procedimento, chamado neste texto de *adição* ($h \oplus H$), tem como inspiração o método do consenso e é descrito no anexo D.

Normalização de Hcuboset

Até este momento foram considerados **hcubo** e **Hcuboset** sobre um mesmo conjunto de índices J. Se h_1 é um **hcubo** sobre J_1 e h_2 é um **hcubo** sobre J_2, estes dois **hcubo** podem ser *normalizados*, i.e. tomados sobre um conjunto comum de índices, considerando-se para tal o conjunto $J_1 \cup J_2$ e a seguinte definição:

$$h'_1 = \begin{cases} D_{a_j} & se\ j \in J_1 \quad e \\ \mathcal{D}_{a_j} & se\ j \in J_2 \end{cases}$$

e

$$h'_2 = \begin{cases} D_{a_j} & se\ j \in J_2 \quad e \\ \mathcal{D}_{a_j} & se\ j \in J_1 \end{cases}$$

Este procedimento também pode ser empregado para normalizar os **Hcuboset** .

3.9.2 Operações sobre hipercubos e conjuntos de hipercubos

Considerando os **hcubo** (h_1...) e os **Hcuboset** (H_1...) como elementos de base, a representação dos modelos associados a cada (sub-)fórmula de uma teoria pode ser feita com o auxílio de operações elementares sobre estes elementos. Estas operações são baseadas na teoria dos conjuntos e fazem apelo à natureza geométrica dos **hcubo** e dos **Hcuboset** , que são portanto identificados às coberturas correspondentes.

As operações definidas sobre os **hcubo** e os **Hcuboset** são:

1. a união ($h_1 \cup h_2$, $h_1 \cup H_2$, $H_1 \cup H_2$);
2. a interseção ($h_1 \cap h_2$, $h_1 \cap H_2$, $H_1 \cap H_2$);
3. o complementar ($\bar{h}$, $\bar{H}$);
4. a diferença ($h_1 \setminus h_2$, $H_1 \setminus h_2$, $H_1 \setminus H_2$);
5. a projeção sobre um eixo coordenado ($proj_i\ H$); e
6. a restrição para um subconjunto de índices ($H|_{\{i=v_i \mid \forall i \in I\}}$).

A partir destas operações iniciais é possível definir outras operações mais complexas, como por exemplo, os seguintes predicados:

- $h_1 \cap H_2 \neq \{\}$? e $H_1 \cap H_2 \neq \{\}$?, que retornam *1* se as respectivas interseções são não vazias e *0* em caso contrário; e
- $h_1 \subset H_2$?, e $h_1 = H_2$?, que retornam *1* respectivamente se a inclusão ou a igualdade entre os elementos indicados ocorrem.

Estas operações são suficientes para uma representação compacta e eficiente dos modelos, como se verá nas próximas seções. Os algoritmos que implementam estas operações são descritos no anexo D.

3.10 Obtenção de modelos: o método das conexões

Para a obtenção dos modelos de uma teoria proposicional diversos métodos são possíveis, e.g. as tabelas verdade, o cálculo de seqüentes, as tabelas semânticas, etc. Os dois últimos métodos citados têm a vantagem de examinar apenas as interpretações booleanas que podem satisfazer a teoria, diminuindo sensivelmente o trabalho de verificação; procura-se fechar (i.e. obter uma proposição e sua negação) todos os ramos da árvore criada pelo cálculo dos seqüentes ou pela tabela semântica [GOCHET 90].

Uma outra possibilidade é a enumeração inicial de todos os caminhos (ramos) das fórmulas e a posterior verificação do fechamento. O célebre *princípio de resolução*, que é subjacente à linguagem Prolog e suas derivadas, pode ser considerado como sendo construído segundo este enfoque [BIBEL 82].

O método das conexões, que é empregado neste trabalho, também se fundamenta no fechamento. Porém ao contrário do que ocorre no princípio de resolução, o pré-tratamento das regras não é necessário e o método pode também ser aplicado às lógicas não-clássicas [TURNER 86], [CATACH 90].

Estas últimas características foram decisivas para sua adoção neste trabalho. Cabe lembrar que um dos objetivos é a geração extensiva de todos os modelos de uma teoria dada, da forma mais eficiente possível.

3.10.1 Fundamentação do método

Definições

A primeira etapa consiste na construção da árvore sintática (*parsing tree*) $\mathcal{A}_\Gamma$ associada à teoria Γ, que enumera os elementos sintáticos das fórmulas que compõem Γ. Os axiomas são considerados como implicitamente conectados por uma conjunção ($\wedge$) [12].

Definição 3.10.1 (POLARIDADE DE UMA FÓRMULA) A cada fórmula de uma teoria está associada uma *polaridade* da forma seguinte: uma sub-fórmula F de uma fórmula G tem a polaridade *positiva* (*1*) se ela aparece no escopo de um número par de negações explícitas ($\neg$) ou implícitas (a premissa de uma implicação $\rightarrow$). No outro caso sua polaridade é *negativa* (*0*).

Intuitivamente a polaridade associada a uma (sub-) fórmula indica se a mesma deve ou não ser satisfeita, segundo a definição indicada na semântica da lógica.

A partir do conectivo que domina cada sub-fórmula e de sua polaridade é possível a aplicação das regras de eliminação de conectivos, como no caso do cálculo por seqüentes.

[12] Ver [GOCHET 90], [BIBEL 82], [WALLEN 87].

α	α_1	α_2
$< F_1 \wedge F_2, 1 >$	$< F_1, 1 >$	$< F_2, 1 >$
$< F_1 \vee F_2, 0 >$	$< F_1, 0 >$	$< F_2, 0 >$
$< F_1 \rightarrow F_2, 0 >$	$< F_1, 1 >$	$< F_2, 0 >$
$< \neg F_1, 1 >$	$< F_1, 0 >$	
$< \neg F_1, 0 >$	$< F_1, 1 >$	
β	β_1	β_2
$< F_1 \wedge F_2, 0 >$	$< F_1, 0 >$	$< F_2, 0 >$
$< F_1 \vee F_2, 1 >$	$< F_1, 1 >$	$< F_2, 1 >$
$< F_1 \rightarrow F_2, 1 >$	$< F_1, 0 >$	$< F_2, 1 >$

Figura 3.25: Regras de eliminação de tipos primário e secundário associadas a uma fórmula

Definição 3.10.2 (REGRAS DE ELIMINAÇÃO DE CONECTIVOS) As regras de eliminação associadas à cada fórmula são de dois tipos:

1. o *tipo primário* é função do conectivo dominante e de sua polaridade: ela será do tipo α se dá origem a um prolongamento (para a satisfação da fórmula todos seus ramos devem ser satisfeitos), e do tipo β se dá lugar a uma ramificação (para a satisfação da fórmula ao menos um dos ramos deve ser satisfeito);
2. o *tipo secundário* é obtido à partir da fórmula pai da sub-fórmula: α_1, α_2 se a fórmula é descendente por prolongamento, β_1, β_2 se ela é descendente por ramificação.

As regras são apresentadas na tabela 3.25.

Intuitivamente o tipo α (respectivamente β) está associado ao meta-conectivo "e" (respectivamente "ou") na definição da noção de satisfação de uma fórmula na teoria.

Definição 3.10.3 (CAMINHO) Os *caminhos* de uma teoria Γ são posições da árvore sintática de Γ, definidos recursivamente como segue: denotar-se-á por α^k ou β^k um nó k de tipo primário α ou β de $\mathcal{A}_\Gamma$.

- base: se k_0 é a raiz de $\mathcal{A}_T$, k_0 é um caminho;
- recursão: se S é um caminho que contém o nó α^k, $(S \setminus \{\alpha^k\}) \cup \{\alpha_1^k\ \alpha_2^k\}$ é um caminho; se S é um caminho que contém o nó β^k, $(S \setminus \{\beta^k\}) \cup \{\beta_1^k\}$ e $(S \setminus \{\beta^k\}) \cup \{\beta_2^k\}$ são caminhos.

A aplicação desta definição permite a obtenção de *caminhos atômicos*, i.e., aqueles que são formados somente por fórmulas atômicas.

Definição 3.10.4 (CONEXÃO) Uma *conexão* num caminho S é um sub-caminho S' em que duas posições indicam a mesma fórmula atômica (a mesma variável proposicional e domínio) mas com polaridades distintas. Um conjunto de conexões *recobre* uma fórmula F sse todo caminho atômico obtido a partir de F contém uma conexão.

Utilização do método para a demonstração de teoremas

Proposição 3.10.1 (Andrews, Bibel) [WALLEN 87] Uma fórmula proposicional F é válida sse existe um conjunto de conexões que recobre $< F, 0 >$.

Esta proposição é uma conseqüência do seguinte resultado sobre tabelas semânticas (Smullyan): Uma fórmula proposicional é válida sse existe uma tabela semântica fechada para $< F, 0 >$.

Utilização do método para a obtenção dos modelos de uma teoria

Proposição 3.10.2 Para uma teoria Γ os modelos $M(\Gamma)$ são obtidos pelas conexões atômicas de $< \Gamma, 1 >$ (axiomas da teoria ligados por conjunção).

Análise do método

Análises do método das conexões podem ser encontradas em [GOCHET 90] e [ESCALADA 91 Salientam-se aqui as principais conclusões:

- não há redundância pois nenhuma fórmula é repetida;
- a utilização de ponteiros permite que se trabalhe com estruturas de dados eficientes;
- nenhuma pesquisa não-necessária é executada: as conexões formam o mínimo de interpretações exigidas para que se proceda a verificação;
- a complexidade é reduzida: não há necessidade de várias árvores de prova, a árvore sintática $\mathcal{A}_\Gamma$ é a estrutura única sobre a qual é realizado o procedimento de cálculo.

3.11 A obtenção de modelos no caso atemporal

Nesta seção descreve-se o procedimento desenvolvido para a obtenção dos modelos de uma teoria Γ, que tem por base o método das conexões e emprega para estruturas de dados os **hcubo** e os **Hcuboset**.

As etapas seguidas são:

1. construção da árvore sintática $\mathcal{A}_\Gamma$ da teoria Γ; a raiz de $\mathcal{A}_\Gamma$ é formada pela conjunção de todos os axiomas de Γ;
2. propagação da raiz para as folhas (*top-down*) da polaridade e dos tipos - primário, secundário - de todas as sub-fórmulas que compõem a teoria; para tanto emprega-se um procedimento recursivo cujo esquema básico é apresentado no anexo E;
3. propagação das folhas à raiz (*bottom-up*) dos conjuntos de átomos ("pontos") que satisfazem cada sub-fórmula, utilizando as operações básicas definidas sobre os **hcubo** e **Hcuboset**; uma descrição resumida desde procedimento é apresentada no anexo E;
4. o **Hcuboset** associado à raiz da teoria é a representação compacta de $M(\Gamma)$.

Salienta-se que, apesar do cálculo ser feito por extensão - com a geração direta de todos os modelos - a representação compacta fornecida pelos **hcubo** e **Hcuboset** permite que se obtenha um procedimento computacionalmente aceitável para o mesmo.

Exemplo 3.11.1 Sejam as fórmulas do exemplo 3.1.1 (A = (a b c d), $\mathcal{D}_a = \mathcal{D}_b = \{0\ 1\ 2\}$, $\mathcal{D}_c = \mathcal{D}_d = \{0\ 1\}$):

$$\begin{aligned}\Gamma_1 = \{\ & a\{0\ 1\} \wedge b\{1\ 2\} \rightarrow c\{0\}, \quad (a\{2\} \wedge \neg b\{1\} \rightarrow c\{0\}) \rightarrow d\{0\},\\ & a\{1\} \wedge c\{1\} \rightarrow d\{1\}, \quad a\{2\} \wedge c\{1\} \rightarrow b\{1\} \vee d\{1\},\\ & \neg(a\{0\ 2\} \wedge b\{0\ 1\} \wedge d\{0\}) \quad \}\end{aligned}$$

A figura 3.26 apresenta a árvore sintática deste conjunto de fórmulas. Nesta árvore as polaridades e tipos são propagados da raiz para as folhas.

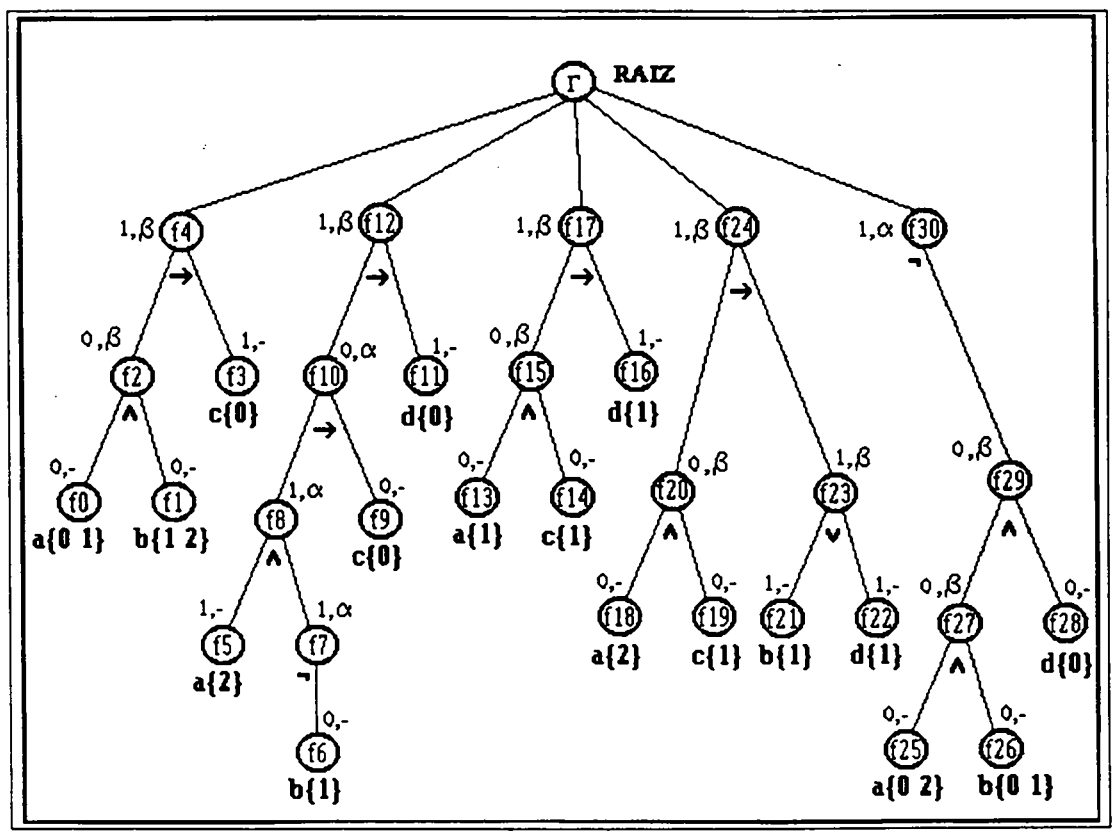

Figura 3.26: Árvore sintática, polaridades e tipos associados a Γ_1

Em seguida os **Hcuboset** que representam os modelos que satisfazem cada sub-fórmula de Γ_1 são calculados. Para as fórmulas atômicas este cálculo é direto, de acordo com a semântica da lógica correspondente. Para as fórmulas de tipo primário α o **Hcuboset** associado corresponde a interseção dos **Hcuboset** das suas fórmulas filhas; para as fórmulas de tipo β a união dos **Hcuboset** é empregada.

A figura 3.27 apresenta estes elementos; o **Hcuboset** associado à raiz de $\mathcal{A}_{\Gamma_1}$ traduz os modelos desta teoria.

fórm.	con.	pol.	t1	t2	**Hcuboset**
f0	-	0	-	β_1	{ [{2} {0 1 2} {0 1} {0 1}] }
f1	-	0	-	β_2	{ [{0 1 2} {0} {0 1} {0 1}] }
f2	$\wedge$	0	β	β_1	{ [{2} {1 2} {0 1} {0 1}] [{0 1 2} {0} {0 1} {0 1}] }
f3	-	1	-	β_2	{ [{0 1 2} {0 1 2} {0} {0 1}] }
f4	$\rightarrow$	1	β	α_1	{ [{2} {1 2} {1} {0 1}] [{0 1 2} {0} {1} {0 1}] [{0 1 2} {0 1 2} {0} {0 1}] }
f5	-	1	-	α_1	{ [{2} {0 1 2} {0 1} {0 1}] }
f6	-	0	-	α_1	{ [{0 1 2} {0 2} {0 1} {0 1}] }
f7	$\neg$	1	α	α_2	{ [{0 1 2} {0 2} {0 1} {0 1}] }
f8	$\wedge$	1	α	α_1	{ [{2} {0 2} {0 1} {0 1}] }
f9	-	0	-	α_2	{ [{0 1 2} {0 1 2} {1} {0 1}] }
f10	$\rightarrow$	0	α	β_1	{ [{2} {0 2} {1} {0 1}] }
f11	-	1	-	β_2	{ [{0 1 2} {0 1 2} {0 1} {0}] }
f12	$\rightarrow$	1	β	α_2	{ [{2} {0 2} {1} {1}] [{0 1 2} {0 1 2} {0 1} {0}] }
f13	-	0	-	β_1	{ [{0 2} {0 1 2} {0 1} {0 1}] }
f14	-	0	-	β_2	{ [{0 1 2} {0 1 2} {0} {0 1}] }
f15	$\wedge$	0	β	β_1	{ [{0 2} {0 1 2} {1} {0 1}] [{0 1 2} {0 1 2} {0} {0 1}] }
f16	-	1	-	β_2	{ [{0 1 2} {0 1 2} {0 1} {1}] }
f17	$\rightarrow$	1	β	α_3	{ [{0 2} {0 1 2} {1} {0}] [{0 1 2} {0 1 2} {0} {0}] [{0 1 2} {0 1 2} {0 1} {1}] }
f18	-	0	-	β_1	{ [{0 1} {0 1 2} {0 1} {0 1}] }
f19	-	0	-	β_2	{ [{0 1 2} {0 1 2} {0} {0 1}] }
f20	$\wedge$	0	β	β_1	{ [{0 1} {0 1 2} {1} {0 1}] [{0 1 2} {0 1 2} {0} {0 1}] }
f21	-	1	-	β_1	{ [{0 1 2} {1} {0 1} {0 1}] }
f22	-	1	-	β_2	{ [{0 1 2} {0 1 2} {0 1} {1}] }
f23	$\vee$	1	β	β_2	{ [{0 1 2} {1} {0 1} {0}] [{0 1 2} {0 1 2} {0 1} {1}] }
f24	$\rightarrow$	1	β	α_4	{ [{2} {1} {1} {0}] [{2} {0 1 2} {1} {1}] [{0 1} {0 1 2} {1} {0 1}] [{0 1 2} {0 1 2} {0} {0 1}] }
f25	-	0	-	β_1	{ [{1} {0 1 2} {0 1} {0 1}] }
f26	-	0	-	β_2	{ [{0 1 2} {2} {0 1} {0 1}] }
f27	$\wedge$	0	β	β_1	{ [{1} {0 1} {0 1} {0 1}] [{0 1 2} {2} {0 1} {0 1}] }
f28	-	0	-	β_2	{ [{0 1 2} {0 1 2} {0 1} {1}] }
f29	$\wedge$	0	β	α_1	{ [{1} {0 1} {0 1} {0}] [{0 1 2} {2} {0 1} {0}] [{0 1 2} {0 1 2} {0 1} {1}] }
f30	$\neg$	1	α	α_5	{ [{1} {0 1} {0 1} {0}] [{0 1 2} {2} {0 1} {0}] [{0 1 2} {0 1 2} {0 1} {1}] }
RAIZ	$\wedge$	1	α	-	{ [{1} {0 1} {0} {0}] [{0 1 2} {2} {0} {0}] [{2} {0 2} {1}{1}] }

Figura 3.27: Conjuntos de hipercubos associados às fórmulas de Γ_1

Exemplo 3.11.2 Seja a teoria paraconsistente Γ_2 apresentada no exemplo 3.4.1. A segunda fórmula desta teoria $((a : 1) \wedge (c : \mu_3) \rightarrow (c : \mu_2))$ tem a árvore sintática indicada à figura 3.28.

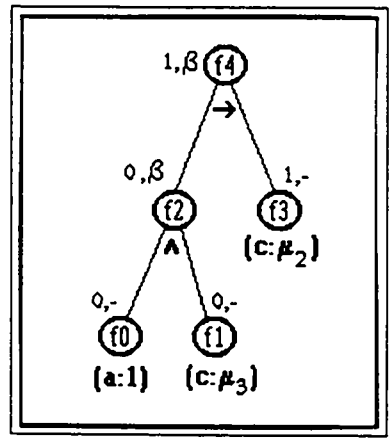

Figura 3.28: Árvore sintática associada a $(a : 1) \wedge (c : \mu_3) \rightarrow (c : \mu_2)$

Os **Hcuboset** associados a estas fórmulas são indicados na figura 3.29.

3.12 A obtenção de modelos no caso temporal

3.12.1 O método

De forma geral a obtenção dos modelos de uma teoria temporal Γ pode ser feita da seguinte forma:

1. as fórmulas de Γ são particionadas em dois conjuntos: Γ_P (fórmulas sobre o passado) e Γ_F (fórmulas sobre o futuro); esta partição é possível pela introdução de atributos auxiliares;

2. o conjunto Γ_P é transformado em um SDR de atraso unitário;

fórm.	con.	pol.	t1	t2	**Hcuboset**
f0	-	0	-	β_1	$\{[\{0\}\{\perp 01 \top\}\{\perp \mu_1\mu_2\mu_3\mu_{12} \top\}]\}$
f1	-	0	-	β_2	$\{[\{01\}\{\perp 01 \top\}\{\perp \mu_1\mu_2\mu_{12}\}]\}$
f2	$\wedge$	0	β	β_1	$\{[\{0\}\{\perp 01 \top\}\{\perp \mu_1\mu_2\mu_3\mu_{12} \top\}]$ $[\{1\}\{\perp 01 \top\}\{\perp \mu_1\mu_2\mu_{12}\}]\}$
f3	-	1	-	β_2	$\{[\{01\}\{\perp 01 \top\}\{\mu_2\mu_{12} \top\}]\}$
f4	$\rightarrow$	1	β	-	$\{[\{0\}\{\perp 01 \top\}\{\perp \mu_1\mu_2\mu_3\mu_{12} \top\}]$ $[\{1\}\{\perp 01 \top\}\{\perp \mu_1\mu_2\mu_{12} \top\}]\}$

Figura 3.29: Conjuntos de hipercubos associados às sub-fórmulas de $(a : 1) \wedge (c : \mu_3) \rightarrow (c : \mu_2)$

3. o SFE associado a este SDR é então obtido, na forma de um grafo $\mathcal{G}_\Gamma$, como já visto anteriormente:

   - os nós são os modelos da teoria Γ_P;
   - as arestas são construídas a partir da equação de estado, mantendo-se a consistência da evolução: assim se no SDR tem-se $b = a\$1$ e se em m_0 o valor de a é v, haverá uma aresta entre m_0 e m_k sse o valor de b em m_k for também v; neste caso a equação de estado é suposta verificada por construção, devido a propagação de valor representada por $b = a\$1$;

   as trajetórias deste SDR descrevem os modelos de Γ_P;

4. a satisfação das fórmulas de Γ_F é em seguida verificada sobre o grafo $\mathcal{G}_\Gamma$, considerado como um sistema a número finito de estados (SFE); a parte do grafo que não satisfaz estas fórmulas é eliminada de $\mathcal{G}_\Gamma$: o resultado é um subgrafo que traduz convenientemente os modelos temporais de Γ.

A metodologia acima descrita também pode ser empregada para SDR com outros tipos de equação de estado, desde que se garanta a satisfação da mesma à cada evolução. O grafo obtido também pode servir de base à construção de um transdutor de comportamento entrada / saída equivalente ao sistema.

A próxima seção se concentra no último passo desta seqüência, i.e. na satisfação das fórmulas temporais que envolvem o futuro. O enfoque aqui utilizado emprega a geração explícita do SFE associado a teoria temporal, e os procedimentos de verificação se baseiam em algoritmos oriundos da teoria dos grafos.

Outros enfoques são encontrados na literatura: em especial no sistema MEC [CRUBILLE 88], cada operador temporal está associado a um conjunto de equações sobre as quais se calcula um ponto fixo [ARNOLD 90], [VERNADAT 86]. O procedimento de cálculo é recursivo, tendo como base os valores de w_{sXF} e w_{aXF}, e se justifica devido à monotonicidade dos operadores envolvidos [AUDUREAU 89]. A utilização deste enfoque alternativo dentro do formalismo apresentado neste trabalho também é possível (ver [KAESTNER 92a]).

3.12.2 Construção do grafo de evolução e satisfação das fórmulas temporais

O primeiro passo consiste na geração explícita do grafo de evolução $\mathcal{G}_\Gamma$; a verificação da satisfação das fórmulas temporais será feita em seguida sobre esta estrutura.

O enfoque é similar ao empregado na linguagem CSML [CLARKE 91]. O grafo é tratado como uma estrutura de Kripke da teoria temporal, de forma que se denominará um modelo atemporal (um nó $w \in \mathcal{G}_\Gamma$) de *mundo*. A cada mundo w está associado um conjunto de sucessores ($Suc(w)$) e um conjunto de predecessores ($Pre(w)$), segundo a relação de acessibilidade determinada pela equação de estado.

De forma similar ao caso atemporal, é possível se associar a cada subfórmula F da árvore sintática de Γ_F o conjunto de mundos w_F que a satisfazem.

A definição das polaridades e tipos para os conectivos temporais (sX, aX, sF, aF, sG, aG) é direta: toda fórmula com estes conectivos tem a mesma polaridade da

fórmula pai e o seu tipo primário é definido como sendo γ, que indica ser o conectivo do tipo temporal. Da mesma forma o tipo secundário de uma fórmula filha destas fórmulas é definido como sendo γ_1. O cálculo dos mundos que satisfazem a fórmula é feito em função do conectivo temporal, segundo a definição de satisfação apresentada a seguir.

Se F e G são fórmulas, a satisfação é definida da seguinte forma:

- Para as fórmulas atemporais:

  1. $w_{\neg F} = (\text{nós}(\mathcal{G}_\Gamma) \setminus w_F)$;
  2. $w_{F \wedge G} = w_F \cap w_G$;
  3. $w_{F \vee G} = w_F \cup w_G$;
  4. $w_{F \rightarrow G} = w_{\neg F \vee G}$;

- e para as fórmulas temporais:

  1. $w_{sXF} = \{w' \in \text{nós}(\mathcal{G}_\Gamma) \mid w'Rw \text{ e } w \in w_F\}$;
  2. $w_{sFF} = \{w' \in \text{nós}(\mathcal{G}_\Gamma) \mid$ não há componente fortemente conexa de $(\mathcal{G}_\Gamma \setminus w_F)$ que pode ser atingida a partir de w' $\}$;
  3. $w_{aFF} = \{w' \in \text{nós}(\mathcal{G}_\Gamma) \mid$ todo caminho que sai de w' não atinge nenhuma componente fortemente conexa de $(\mathcal{G}_\Gamma \setminus w_F)$ $\}$.

Os algoritmos utilizados para as fórmulas temporais são apresentados no anexo F. Eles são semelhantes aos utilizados em [AHO 74], para o cálculo das componentes conexas de um grafo e para se encontrar o fechamento transitivo de uma relação.

Exemplo 3.12.1 Seja a teoria apresentada no exemplo 3.8.4. A árvore sintática para

$$\Gamma_F = \{ \quad a\{1\} \wedge sFb\{1\} \wedge c\{1\} \vee aXd\{0\} \quad \}$$

é apresentada na figura 3.30. As polaridades e tipos são propagados. Em seguida os mundos nos quais cada subfórmula é satisfeita são calculados de acordo com o método previamente apresentado, com base no SFE mostrado à figura 3.21.

Outrossim, a verificação de condições temporais dadas por autômatos atuando sobre o futuro é possível, desde que se empreguem mecanismos de busca no grafo $\mathcal{G}_\Gamma$.

Caso estas condições sejam dadas por autômatos finitos ordinários sobre prefixos dos w_n-ramos, pode-se observar que um autômato ordinário de n estados aceita uma linguagem não vazia se e somente se aceita uma palavra de comprimento $m < n$ [HOPCROFT 69]; como é possível se encontrar todos os caminhos de comprimento n no grafo $\mathcal{G}_\Gamma$, a verificação destas condições é sempre decidível.

Lembrando que uma condição temporal representa um operador d n-ário sobre as n-fórmulas que compõem a sua entrada, o mesmo procedimento descrito para os operadores temporais usuais pode ser empregado: o nó da árvore sintática correspondente à subfórmula d está associado aos mundos w_F que satisfazem esta condição temporal.

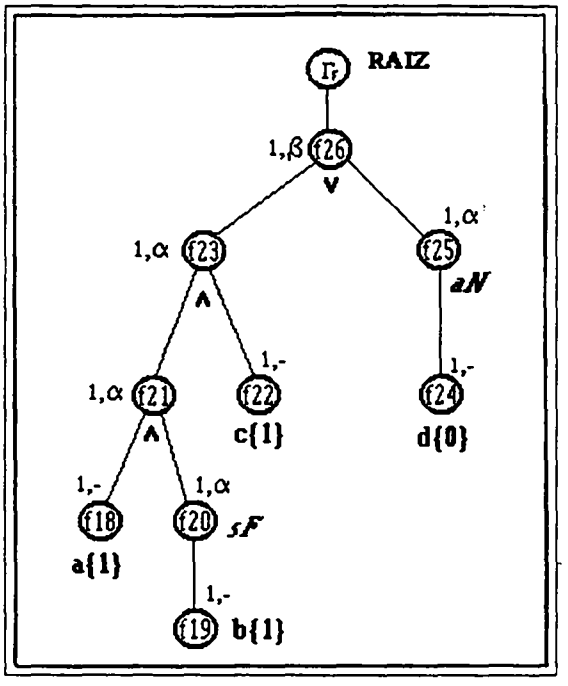

Figura 3.30: Árvore sintática, polaridades e tipos para Γ_F

3.13 Conclusão

Neste capítulo foi desenvolvido o formalismo matemático que fundamenta a construção de $SBCTR$ segundo a proposta.

Considera-se que a evolução de um $SBCTR$ reativo que segue o enfoque síncrono pode ser convenientemente traduzida por meio do cálculo de modelos da teoria lógica que compõem a Base de Conhecimentos do sistema (constituída por fórmulas lógicas), e pela definição de atributos de entrada e saída. Este tratamento exaustivo é coerente com as necessidades de determinismo, reatividade e garantia de resposta desejados neste tipo de sistema.

Assim o formalismo se baseia na definição de uma lógica temporal, com destaque para as seguintes características:

- consideração dos problemas de inconsistência e incompletude nos conjuntos de regras, pelo tratamento paraconsistente dado à lógica;
- estrutura temporal linear em relação ao passado e arborescente em relação ao futuro;
- aumento de expressividade da lógica pela utilização de condições temporais complexas -geradas com o auxílio de autômatos (para o passado linear);
- possibilidade de verificação de condições temporais sobre o futuro (o que permite a análise de propriedades); e
- inclusão natural dentro do formalismo de funções e sistemas dinâmicos qualitativos.

Quanto à implementação, mostrou-se que de maneira geral é possível a obtenção dos modelos da teoria lógica em questão como trajetórias (a solução) de um Sistema Dinâmico sujeito a Restrições, que possui a equação de estado sob a forma de um atraso unitário (máquina de *Huffmann*). Este procedimento é coerente com a geração de códigos de execução compactos e deterministas, como ocorre nas demais linguagens síncronas.

Finalmente foram apresentados os procedimentos utilizados para o cálculo de modelos dentro do formalismo anteriormente apresentado. Empregam-se procedimentos fundamentados no método das conexões de Wallen e Bibel, que trabalham basicamente sobre a árvore sintática da teoria em questão. Os algoritmos apresentados também se baseiam em operações elementares sobre conjuntos n-dimensionais, e em algoritmos oriundos da teoria dos grafos.

Capítulo 4

Uma proposta de Sistema Baseado em Regras para aplicações Tempo-Real

Como apresentado no capítulo 2, as linguagens síncronas fornecem uma nova abordagem para o tratamento de STR, na direção do atendimento aos requisitos de previsibilidade, correção temporal e reatividade exigidos por estes sistemas. O enfoque síncrono apresenta ainda outras vantagens, tais como o tratamento explícito do tempo, a facilidade de verificação, etc.

Este capítulo trata da especificação de um núcleo para a geração de SBCTR que visa incorporar os resultados e as vantagens provenientes da utilização da abordagem síncrona nos sistemas de IA, a partir do formalismo definido no capítulo anterior.

Inicialmente apresenta-se uma visão geral da metodologia proposta, na qual estão envolvidas duas etapas: a primeira de desenvolvimento, que inclui as atividades de descrição do problema e de verificação de propriedades tempo-real do sistema, e a segunda de operação, que considera a atuação definiva do SBCTR frente ao ambiente.

A representação do conhecimento no sistema se apresenta na forma de um conjunto de módulos interconectados, cada módulo sendo construído de acordo com o paradigma "Sistema Baseado em Regras". Uma descrição suscinta da linguagem de descrição proposta é apresentada.

A fim de permitir a efetiva aplicação do formalismo considera-se a definição de um procedimento, que será denominado genericamente "compilação", cuja função é produzir, a partir da linguagem de entrada do sistema, um código executável na forma de um autômato. A geração deste autômato, além de assegurar o determinismo e a reatividade desejados, permite a verificação do atendimento da "hipótese do sincronismo", que é fundamental para a correta aplicação do enfoque proposto.

Finalmente a utilização prevista para este sistema é ilustrada através de um exemplo, que trata da supervisão e auxílio à condução e de um processo industrial.

4.1 A metodologia de desenvolvimento proposta

O formalismo descrito no capítulo anterior foi fundamentado na hipótese de que a construção de um SBCTR é o resultado de duas etapas distintas: desenvolvimento e operação [BONISSONE 90].

- Na etapa de desenvolvimento exploram-se os recursos disponíveis (a nível de representação do conhecimento, mecanismos de inferência, etc) - para se encontrar a estratégia que deve ser usada na etapa de operação, assegurando que sejam obtidas as melhores respostas possíveis dentro das restrições temporais impostas.

- Na etapa de operação, o mecanismo obtido deve reagir às solicitações do ambiente de forma determinista no que se refere ao comportamento funcional e temporal.

Neste trabalho será apresentada apenas a etapa de desenvolvimento, pois é nela que se determina efetivamente o atendimento aos requisitos funcionais e temporais impostos. Quando se considera que a resposta é satisfatória, o código executável gerado pode ser empregado na operação contínua com o ambiente, enquanto persistirem mesmas condições de funcionamento.

Esta etapa envolve a descrição ou especificação do problema, feita neste caso com o auxílio de uma linguagem modular de representação do conhecimento, que pode inclusive expressar as propriedades tempo-real desejadas do sistema; isto garante o seu atendimento para toda situação prevista de funcionamento.

Um processo de compilação é o responsável pela geração do código a partir desta linguagem, cujo produto final pode ser diretamente utilizado na etapa de operação.

4.2 A representação das especificações do problema

Propõe-se que a descrição de um problema seja feita de forma modular, com base nos seguintes princípios:

- o uso extensivo de módulos, cada um dos quais representando um componente cognitivo do sistema;

- em cada módulo, este conhecimento é representado por meio de um conjunto de regras;

- o funcionamento conjunto dos módulos é obtido a partir de uma operação de composição dos módulos.

O sistema total é o resultado da composição dos módulos representados por meio de Sistemas Dinâmicos sujeitos a Restrições (SDR). Outrossim, além de facilitar a concepção, esta modularização vem em resposta a outra questão que surge quando se adota um procedimento exaustivo de geração de autômato como o proposto: o problema da "explosão combinatória", que é inerente a este tipo de cálculo.

Por outro lado a adoção do paradigma "Sistema Baseado em Regras" (SBR), como elemento principal de representação do conhecimento, se justifica diretamente por duas razões:

1. em primeiro lugar este paradigma é adequado para a descrição das heurísticas e regras de conduta comumente utilizadas nos sistemas que compõem as áreas de aplicação visadas por este trabalho [MOORE 90], [LAFFEY 88];

2. em segundo lugar é o paradigma que melhor se adapta ao formalismo matemático proposto, pois cada conjunto de regras fornecido pode ser traduzido imediatamente em uma teoria lógica correspondente. Neste caso os modelos obtidos pelos procedimentos descritos anteriormente correspondem exatamente à uma análise exaustiva de todas as entradas possíveis do sistema e ao fornecimento das saídas correspondentes, a partir da aplicação simultânea de todas as regras que compõem a teoria em questão.

Cabe ressaltar que, apesar do uso deste paradigma, o ciclo de inferência ocorre de forma substancialmente diferente da maioria dos SBR, procurando ser consistente com o princípio de funcionamento dos Sistemas Reativos:

- Todas as regras são simultaneamente aplicadas, a partir dos valores dos atributos de entrada.
- Não há qualquer estrutura especial para o armazenamento dos dados entre cada passo da inferência, (como e.g. memória de trabalho). Os únicos elementos que permitem o armazenamento do estado do sistema são os atributos do tipo atraso ($1) e as condições temporais.

4.2.1 Os princípios da modularização e da composição de módulos

A hipótese subjacente à modularização é que um sistema pode ser particionado em vários módulos que, interconectados, permitem a resolução do problema. Neste contexto considera-se que cada módulo M_m que compõe o sistema foi obtido com base no formalismo exposto no capítulo anterior, i.e., forma um SDR sobre um conjunto de atributos A_m. Considera-se também que estes conjuntos de atributos fazem parte de um conjunto global de atributos $A = \bigcup_m A_m$.

Composição de SDR

Na discussão que segue se considera, sem perda de generalidade, o caso de dois módulos.

Sejam dois SDR M e N dados pelas equações abaixo, definidos sobre os conjuntos de atributos A e B,e sejam os subconjuntos $U, X, Y \subset A$ e $V, Z, W \subset B$, de atributos de entrada, estado e saída respectivamente:

$$M : \begin{cases} X_{i+1} = f_1(X_i, U_i, Y_i) \\ h_1(X_i, U_i, Y_i) = 0 \end{cases} \qquad N : \begin{cases} Z_{i+1} = f_2(Z_i, V_i, W_i) \\ h_2(Z_i, V_i, W_i) = 0 \end{cases}$$

As seguintes considerações podem ser feitas sobre o relacionamento existente entre estes módulos:

- deve-se ter sempre $Y \cap W = \emptyset$, pois cada atributo em saída deve ser definido de forma única;
- da mesma forma sempre $X \cap Z = \emptyset$, pois os estados dos módulos devem ser independentes;
- é admitido o caso em que $U \cap V \neq \emptyset$: neste caso parte da entrada é comum aos dois módulos, que atuam em paralelo em relação a estas entradas; e finalmente

- também é admitido o caso onde $Y \cap V \neq \emptyset$, que ocorre quando os sistemas operam em série em relação a estes atributos comuns.

Os relacionamentos acima descritos podem ser formalizados através da seguinte definição:

Definição 4.2.1 (COMPOSIÇÃO DE *SDR*) O *sistema composto* de M e N, denotado por $M \circ N$ (ver figura 4.1) tem como estados $X \cup Z$, como entrada $U \cup (V \setminus Y)$ e como saída $W \cup (Y \setminus V)$, e suas equações podem ser obtidas da seguinte forma:

- a equação de estado de $M \circ N$ é obtida diretamente por justaposição das equações de estado de M e de N;
- a restrição de $M \circ N$ é a justaposição das restrições de M e N.

Os atributos $y \in Y \cap V$ são denominados *conexões* entre os módulos M e N. Quando $Y \cap V \neq \emptyset$ a definição acima permanece válida; entretanto neste caso escrever-se-á $M \prec N$, ao invés de $M \circ N$, de forma a salientar a dependência relativa de aplicação dos módulos.

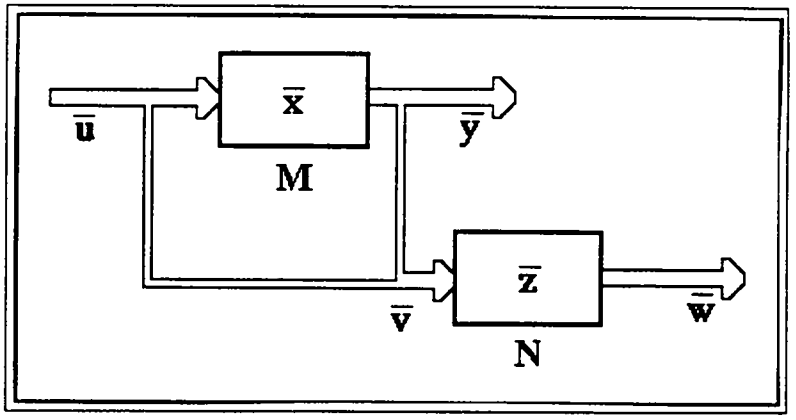

Figura 4.1: Forma geral da composição entre dois *SDR*

Nesta definição a interconexão entre os módulos foi proporcionada por atributos de mesmo nome presentes nestes módulos. Esta consideração equivale a adoção da "hipótese de unicidade de nomes", como se emprega nos formalismos de Bases de Dados [THAYSE 89].

De modo alternativo, se M e N são sistemas dinâmicos independentes, i.e. quando se tem $Y \cap V = \emptyset = U \cap V$, é possível realizar a sua composição desde que existam equações escalares do tipo $y = v$ (para as conexões em série) ou $u = v$ (para as conexões em paralelo), que serão adicionadas a $h_1(X_i, U_i, Y_i) = 0$ e a $h_2(Z_i, V_i, W_i) = 0$ para formar a restrição de $M \circ N$. Supõe-se que os domínios dos atributos mencionados em uma conexão são os mesmos (por exemplo $\mathcal{T}_y = \mathcal{T}_v$). Quando isto não ocorre uma função qualitativa pode ser utilizada para compatibilizar os domínios - por exemplo, para compatibilizar y e v basta se usar um atributo auxiliar y' de domínio $\mathcal{T}_v$, a equação $v = y'$, e uma função qualitativa fq tal que $y' = fq\ (y)$.

Compromisso explosão combinatória x tempo de reação na composição de módulos

Sejam M e N módulos sobre conjuntos de atributos $A = (\ldots a_i \ldots)$ e $B = (\ldots b_j \ldots)$, e Γ_M e Γ_N as teorias correspondentes. O número de modelos máximo de cada um deste módulos fica limitado respectivamente por $\max M(\Gamma_M) = \prod_i \|a_i\|$ e $\max M(\Gamma_N) = \prod_i \|b_j\|$.

Considerando a aplicação em seqüência dos módulos, i.e., a execução de M seguida da execução de N, apenas o cálculo de modelos acima indicado é necessário. O sistema global resultante da composição teria no caso de módulos independentes um número máximo de modelos $M(\Gamma_M) + M(\Gamma_N)$. Outrossim cada ligação entre M e N só pode reduzir ainda mais o número de modelos, pois introduz uma nova restrição ao sistema.

De modo alternativo, pode-se mostrar que se o sistema fosse considerado globalmente, i.e., sobre o conjunto de atributos $A \cup B$, o número de modelos ficaria limitado a $\max M(\Gamma_{M \cup N}) = \prod_i \|a_i\| \times \prod_i \|b_j\|$, valor muito superior ao indicado anteriormente.

Em contraposição, quanto ao tempo de reação, considerando que os transdutores correspondentes a M e N têm tempos de transição maximal t_M e t_N respectivamente, o tempo de reação do sistema composto será uma função linear de t_M e de t_N, e pode ser superior ao tempo de transição do sistema considerado de forma global.

Conclui-se que o menor número de modelos obtido pela divisão do sistema em módulos e sua composição é compensado por uma perda em termos de tempo de reação, quando da aplicação sucessiva dos autômatos obtidos para cada módulo. Como o principal problema envolvido é a redução do número de modelos a calcular, o uso da modularização se justifica não apenas como instrumento metodológico de desenvolvimento, mas como um elemento de efetivo auxílio à implementação do sistema.

Ciclo de causalidade

A aplicação irrestrita da composição pode gerar problemas, por exemplo quando um *SDR* necessita para seu funcionamento de uma entrada cujo cálculo depende de sua saída no mesmo instante. Neste caso ocorre um *ciclo de causalidade*. O resultado a seguir estabelece uma condição suficiente para que o problema não ocorra:

Proposição 4.2.1 Sejam M e N módulos de um mesmo sistema, e a relação $\prec$ sobre estes módulos. Se o fechamento reflexivo e transitivo $\prec^*$ de $\prec$ é uma relação de ordem (parcial), i.e. se verifica a propriedade antisimétrica, o sistema não contém ciclo de causalidade.

Neste caso a execução do sistema pode ser feita também de forma progressiva, de acordo com a ordem induzida por $\prec^*$. Como já mencionado anteriormente, a execução parcial e sucessiva dos módulos permite, embora com perda de eficiência, um ganho substancial em termos do tamanho do espaço de estados do problema global. O tempo de reação global do sistema permanece determinista. Com efeito, a partir do tempo máximo de reação de cada módulo e da topologia das conexões é possível se determinar o tempo máximo de reação do sistema global.

Quando se considera os *SDR* sob a forma de sistemas de transições (ver 3.8.2), o procedimento de composição acima descrito pode ser visto como um *produto síncrono* [ARNOLD 90]. Neste tipo de produto as transições do produto síncrono são obtidas como um subconjunto dos pares de transições dos autômatos componentes, só sendo admitidos

os pares que representam conexões válidas entre os módulos. De maneira similar, considerando os cálculos de processos (e.g. *CSP* [HOARE 78], *CCS* [MILNER 89]) a operação de composição pode ser vista como uma comunicação do tipo *rendez-vous* feita sobre um canal de comunicação [1]. O sistema global é sempre considerado totalmente síncrono, i.e. os processos evoluem à mesma velocidade.

4.2.2 A linguagem de representação proposta

Como anteriormente mencionado, para representar um problema propõe-se a utilização de uma linguagem fundamentada em módulos compostos de conjuntos de regras, cuja descrição informal (sintática e semântica) se apresenta a seguir.

Esta linguagem contém as características de modularidade, representação do conhecimento por meio de regras equivalentes a fórmulas lógicas, operadores temporais e representação de equações qualitativas, conforme o que foi exposto no ítem anterior e no capítulo 3.

Os comandos podem ser divididos, segundo suas características funcionais, nos seguintes grupos:

- Definição dos módulos e interconexão.

  O comando `MODULE` determina o escopo de um módulo; seus atributos de entrada e saída são respectivamente determinados pelos comandos `INPUTS` e `OUTPUTS`.

  A conexão entre os módulos é feita automaticamente por atributos de mesmo nome; no entanto uma contrução de conexão `LINK` também está prevista, permitindo a ligação entre atributos de nomes diferentes.

- Representação do conhecimento:

  - Construções clássicas:

    São compostas basicamente por regras, que dão origem às fórmulas na teoria lógica correspondente.

    A principal construção é `(RULE <nome> IF <fórmula> THEN <fórmula>)`, que corresponde a uma implicação material.

    Construções similares correspondentes aos outros conectivos definidos no formalismo podem ser incluídos, tais como `(RULE <nome> (IF <fórmula> ONLY-IF <fórmula>))`, `(RULE <nome> (<fórmula> AND <fórmula>))`, etc.

  - Construções temporais:

    Nesta categoria se enquadram: (a) o atraso unitário; (b) os operadores e quantificadores temporais; e (c) as condições temporais descritas com o auxílio de autômatos.

    O atraso unitário de um atributo a é indicado por `a$1` ; ele indica o valor deste atributo no instante imediatamente anterior.

[1] Uma descrição seguindo esta linha pode ser encontrada em [OSTROFF 89] (*CSP*, [HOARE 78]) ou [ARNOLD 90] (*CCS*, [MILNER 89]).

Operadores e quantificadores temporais são incluídos de maneira similar às regras, como e.g. (RULE <nome> (PAST <fórmula>)), ou (RULE <nome> (ALL-NEXT <fórmula>)).

O sistema prevê a implementação de um conjunto inicial de condições temporais descritas por autômatos, como e.g. BEFORE (P, Q) (P ocorre antes da ocorrência de Q), ONCE-AFTER (P, Q) (P ocorre ao menos uma vez após a ocorrência de Q), etc. Contadores também estão disponíveis, como e.g. # DURING G, que conta o número de ocorrências de F enquanto G é verdadeiro, e que pode ser utilizada em expressões do tipo (# F DURING G LT 3) [2].

Além disto, qualquer módulo implementado no sistema poderá, desde que sejam atendidas as restrições de chamada e que a saída seja booleana, ser posteriormente utilizado como condição temporal.

- Construções para tratamento qualitativo:

  Propõe-se a existência de comandos para a definição de funções e sistemas dinâmicos qualitativos, e para indicar restrições sobre a evolução de variáveis qualitativas.

  Em particular, o comando QUALITATIVE-EQUATION indica a ocorrência de uma equação, função ou sistema dinâmico qualitativo;

  DOMAIN-CONTINUITY-RESTRICTION e DERIVATIVE-RESTRICTION, indicam restrições sobre as possíveis evoluções do sistema; estes últimos comandos só poderão ser aplicados sobre domínios específicos.

  De forma similar às condições temporais, poder-se-á utilizar um módulo qualquer previamente implementado no sistema como um sistema dinâmico qualitativo.

- Ações externas:

  O comando ACTION permite a execução de procedimentos externos, aí estando incluídas as ações de entrada e saída.

- Outras declarações:

  - Facilidades de representação, tais como abreviações sintáticas, etc.

    O comando DIMENSION é utilizado para definir um atributo de tipo vetorial, cujos elementos podem ser acessados pelo índice correspondente, como na expressão sensor [2].

  - Construções auxiliares, por exemplo para a declaração INITIAL permite a fixação dos valores iniciais de atributos.

  - Declarações sobre os domínios de atributos. Os comandos DOMAIN e LATTICE definem os domínios de um atributo como sendo do tipo conjunto ou reticulado respectivamente; os reticulados utilizados mais freqüentemente poderão estar disponíveis sem necessidade de nova definição.

O exemplo que será apresentado neste capítulo ilustra a utilização destes comandos. O anexo E apresenta uma proposta preliminar de sintaxe para a linguagem de entrada.

[2] As principais condições temporais previstas para o sistema são indicadas no anexo E.

4.3 O procedimento de compilação

Descrevem-se a seguir os principais passos a serem seguidos para a geração efetiva do mecanismo de execução de um Sistema Baseado no Conhecimento para aplicações Tempo-Real, a partir da linguagem de entrada apresentada anteriormente.

O principal procedimento envolvido é o cálculo de modelos da teoria obtida a partir das regras que compõem cada módulo, segundo os algoritmos descritos no capítulo anterior, e a geração de um autômato para a execução. Como será visto no próximo item, este procedimento também permite a verificação de propriedades tempo-real sobre o sistema em desenvolvimento.

O compilador deve executar os seguintes passos:

- Para cada módulo que compõem o sistema:

  1. determinação dos atributos de entrada, de saída e de estado utilizados no módulo; os atributos de entrada e de saída são diretamente indicados pelo programador através dos comandos `INPUTS` e `OUTPUTS`, enquanto que os atributos de estado estão presentes nas expressões de atraso unitário (`a$1`);
  2. geração de todas as instâncias das regras, pela eliminação das abreviações sintáticas existentes; de forma geral estas abreviações se referem aos atributos vetoriais indicados na declaração `DIMENSION`;
  3. construção de um dicionário que inclui os atributos e seus domínios codificados; estes domínios podem também ser formados por reticulados dependendo da utilização de um formalismo paraconsistente, e servem de base para a tradução das regras à sua forma lógica equivalente;
  4. criação de uma estrutura para conversão dos valores dos atributos para seus domínios codificados, a partir do dicionário obtido no item anterior; esta estrutura é utilizada posteriormente na etapa de operação, e pode ser otimizada de acordo com o tipo do atributo em questão; para atributos simbólicos o uso de tabelas de *hashing* são convenientes; já para atributos que estão envolvidos na comparação com valores numéricos, árvores de decisão podem ser empregadas;
  5. conversão das regras de comportamento do sistema (fornecidas pelo especialista, na linguagem de entrada) sob sua forma lógica equivalente; esta tradução decorre imediatamente da sintaxe da regra e do dicionário de atributos e domínios;

     conversão análoga para as funções e sistemas dinâmicos qualitativos que compõem o módulo; também é utilizado o dicionário de atributos e domínios;
  6. identificação, análise e armazenamento das condições temporais utilizadas; o sistema deve prover um conjunto básico formado pelas condições temporais mais freqüentes, além de permitir a definição de novas condições;
  7. cálculo dos modelos da teoria formada pelas regras, de acordo com o formalismo proposto; conforme o que foi apresentado no capítulo anterior, esta tarefa envolve:

     - a separação da teoria formada pelas regras em suas partes atemporal e temporal (futuro);

- o cálculo dos modelos correspondentes à parte atemporal, segundo a noção de satisfação da lógica adotada;
- a determinação da equação de evolução e geração do SFE correspondente ao módulo; a equação de evolução se obtém diretamente dos atributos de estado;
- a verificação das parte temporal da teoria que compõe o módulo, com a restrição do SFE à parte que satisfaz as fórmulas correspondentes; e
- a construção de um autômato (transdutor) a partir do SFE definitivo.

8. determinação do tempo de transição maximal do autômato obtido; este tempo será utilizado na verificação de satisfação da hipótese do sincronismo pelo sistema.

- Para a composição de módulos:

1. determinação dos atributos de conexão entre os módulos, e análise das ligações, se for o caso;
2. estabelecimento definitivo da ordem de execução dos módulos, de acordo com as relações $\circ$ e $\prec$ identificadas a partir das conexões;
3. análise de ciclos de causalidade, conforme o que foi exposto anteriormente;
4. criação de uma estrutura para a execução do sistema composto; esta estrutura basicamente tem a função de indicar a ordem de execução dos autômatos obtidos a partir dos módulos; evidentemente a aplicação sucessiva de autômatos equivale à execução de um único autômato equivalente [LEWIS 81].
5. determinação do tempo efetivo de transição do sistema global, a partir dos tempos de transição de cada módulo e da topologia das conexões; isto permite a verificação definitiva do atendimento da hipótese do sincronismo pelo sistema global.

A estrutura composta pelo conjunto de autômatos que compõem os módulos do sistema pode ser enfim utilizada para a interação definitiva do sistema para com o ambiente.

4.4 Verificação de propriedades

Uma das etapas mais importantes no desenvolvimento de um sistema é a de verificação, cujo objetivo é provar a correção e o atendimento à especificação desejada. No caso dos STR é de particular importância a análise das especificações referentes ao comportamento temporal.

Entre os métodos de análise desenvolvidos, destacam-se [VERNADAT 86]:

- A *validação por simulação*, que tem por base a verificação das propriedades requeridas sobre certas seqüências de funcionamento do sistema. A simulação efetua uma análise apenas parcial, e portanto de pouco interesse para o tipo de sistema considerado.
- A *verificação formal*, onde o sistema é analisado em termos de propriedades requeridas sobre o conjunto formalizado de seus comportamentos. Embora mais segura

e completa, exige a descrição formalizada do sistema e das especificações, e está limitada pelo fenômeno da "explosão combinatória": em muitos casos os sistemas a verificar são muito complexos para que se torne viável proceder a análise exaustiva de seus comportamentos.

Geralmente em um sistema quando se pretende realizar a verificação são assumidas duas descrições de comportamento: uma oriunda da implementação, que descreve o sistema propriamente dito, e outra proveniente da especificação desejada ou de referência, que no caso dos STR descreve principalmente os requisitos temporais do sistema [ALUR 91a].

Nos métodos de verificação ditos *homogêneos*, as duas descrições são feitas no mesmo formalismo; nos métodos *heterogêneos* formalismos diferentes são utilizados.

O formalismo apresentado no capítulo anterior permite que sejam diretamente codificadas na linguagem proposta as propriedades tempo-real a serem satisfeitas pelo sistema, por dois motivos:

- o sistema já se encontra devidamente formalizado, pois a representação do conhecimento é feita a partir de teorias lógicas; e
- um dos mecanismos mais empregados para a verificação formal está fundamentado na satisfação de fórmulas temporais, que normalmente se referem ao comportamento futuro do sistema [VERNADAT 86], [ARNOLD 90]; o formalismo proposto permite justamente a inclusão direta deste tipo de fórmula.

4.4.1 Propriedades de um Sistema Tempo-Real

De forma resumida as propriedades de um STR podem ser classificadas em três categorias:

1. *vivacidade*: questiona se qualquer execução do sistema permitirá que se atinja um determinado estado w; propriedades deste tipo podem ser traduzidas, de uma maneira geral, por fórmulas temporais como $I \rightarrow aF\ J$ ou $I \rightarrow sF\ J$, onde I e J são fórmulas atemporais quaisquer;
2. *invariância*: questiona se em todas as execuções possíveis do sistema determinada propriedade é conservada; tais propriedades se traduzem por fórmulas temporais como $I \rightarrow aG\ J$ ou $I \rightarrow sG\ J$; e
3. *eqüidade*: questiona se todo estado possível no sistema será efetivamente alcançado; se w é um estado futuro potencial do sistema, i.e., pode ocorrer em todas as execuções futuras, então ele ocorrerá infinitamente; este tipo de propriedade não pode ser traduzida diretamente na lógica apresentada.

Uma propriedade de eqüidade pode ser expressa admitindo-se que os operadores temporais também possam ser aplicados sobre w-ramos (lógica CTL^* para o futuro [AUDUREAU 89]). Contudo o uso desta lógica ocasiona um salto na complexidade dos algoritmos de avaliação, que passam de uma complexidade linear a um problema $\mathcal{P}$-espaço completo [AUDUREAU 89]. Outra possibilidade é a consideração de restrições sobre o conjunto de estados que compõem os w-ramos sobre os quais se verifica a satisfação das fórmulas. Dado um conjunto S de

estados, um w-ramo r é dito S-eqüível se pelo menos um estado de S aparece em r infinitamente. A verificação de uma fórmula neste caso seria feita considerando apenas os w-ramos eqüíveis em relação a S. A vantagem do uso deste método é que a complexidade de avaliação permanece linear [AUDUREAU 89].

As condições temporais mais freqüentes utilizadas na verificação de propriedades tempo-real podem ser convenientemente traduzidas dentro do formalismo exposto, seja por fórmulas simples ou por meio de autômatos.

São exemplos destas condições:

- $before(F)$: a condição é satisfeita antes da ocorrência de F, equivalendo portanto à fórmula temporal $H(\neg F)$;
- $always\text{-}since(F, G)$: a condição é satisfeita se G ocorre (é satisfeita) em todos os instantes após a ocorrência de G; esta condição temporal pode ser traduzida pelo autômato visto anteriormente no exemplo 3.7.2.

De forma análoga podem ser definidas outras condições, tais como $before(F, G)$, $after(F, G)$, $since(F, G)$, $starts\text{-}at(F, G)$, $starts\text{-}before(F, G)$, $starts\text{-}after(F, G)$, $once\text{-}after(F, G)$, $once\text{-}since(F, G)$, $always\text{-}after(F, G)$, $always\text{-}since(F, G)$, $always\text{-}from\text{-}to(F, G, H)$, e $once\text{-}from\text{-}to(F, G, H)$, além de condições que envolvem a contagem de ocorrência de sinais dentro certo período temporal, como indicado na proposta de linguagem de entrada (ver 4.2.2 e o anexo F).

4.4.2 Proposta para a verificação de propriedades no sistema

A verificação de propriedades tempo-real neste sistema é feita *por checagem de modelos* (*"model checking"*), que consiste na representação de todos os possíveis comportamentos do sistema, e na verificação exaustiva das propriedades consideradas sobre esta representação [HALBWACHS 91]. Esta abordagem é compatível com o formalismo exposto, que se fundamenta justamente no cálculo exaustivo de modelos.

Desta forma a simples inclusão de propriedades a verificar como parte da descrição do problema permite garantir a sua satisfação; de acordo com os algoritmos descritos no capítulo anterior, o resultado final da compilação será então um sistema a número finito de estados (SFE) validado, cujos modelos descrevem somente as evoluções do sistema que atendem às especificações. Desta forma pode-se dizer que implementação e verificação ocorrem de forma integrada na etapa de desenvolvimento.

No caso da verificação da satisfação de uma propriedade que não foi incluída na especificação original, o procedimento consiste em verificar a inexistência de modelos para a conjunção $S \wedge \neg P$, sendo que S é o sistema obtido a partir da especificação original e que P a nova propriedade a ser verificada. Constata-se então que no contexto deste trabalho, uma abordagem homogênea também pode ser diretamente empregada para a verificação.

Considerando que o tratamento lógico adotado é proposicional, e o número de estados é finito, podem ser utilizados nesta verificação procedimentos de decisão, cujo tratamento principal é algorítmico, não sendo necessários complexos sistemas de prova.

No que diz respeito às condições temporais, como já mencionado anteriormente (3.8.4), é possível avaliá-las à parte pelo sistema, não interferindo diretamente sobre as restrições de evolução. Neste caso elas não têm participação no processo de verificação, agindo como entradas no sistema.

De forma a auxiliar o processo de verificação, um mecanismo de abstração sobre o formalismo proposto foi desenvolvido. Este mecanismo permite que se utilize apenas a parte do sistema efetivamente necessária à verificação de determinada propriedade.

4.4.3 Mecanismo de abstração proposto

De modo informal, o termo *abstração* designa o mapeamento de uma representação de base de um problema (*ground representation*) para uma nova representação mais abstrata (*abstract representation*), que preserva algumas propriedades escolhidas do modelo inicial e é mais simples de manusear [GIUNCHIGLIA 89].

As propriedades a preservar devem ser tais que facilitem a solução do problema original, e a simplicidade de manuseio usualmente significa decidibilidade e menor complexidade.

Abstração aplicada a $P\mathcal{D}_A$

De modo geral, se $\mathcal{L}_1$ e $\mathcal{L}_2$ são dois sistemas lógicos, considera-se uma abstração como um mapeamento entre $\mathcal{L}_1$ e $\mathcal{L}_2$, que envolve as linguagens destes sistemas, seus teoremas e mecanismos de inferência. Considera-se especialmente o caso em que $\mathcal{L}_2$ é uma representação "mais abstrata" que $\mathcal{L}_1$ - as TI-abstrações (*Theorem Increasing abstractions*) de [GIUNCHIGLIA 89].

A definição abaixo particulariza esta noção para cálculo atemporal $P\mathcal{D}_A$:

Definição 4.4.1 (ABSTRAÇÃO) Sejam $P\mathcal{D}_A$ e $P\mathcal{D}_{A'}$ cálculos definidos sobre os conjuntos de atributos A e A', tais que $\|A\| \geq \|A'\|$. Diz-se que $P\mathcal{D}_{A'}$ é uma *abstração* de $P\mathcal{D}_A$ se existem funções $f =< f_A, f_a, \ldots >$ tais que:

- $f_A : A \not\longrightarrow A'$ é uma função parcial injetiva; denotar-se-á $a' = f_A(a)$;
- para todo $a \in Domínio(f_A)$ tem-se $\|\mathcal{D}_a\| \geq \|\mathcal{D}_{a'}\|$;
- a todo $a \in Domínio(f_A)$ está associada uma função sobrejetiva $f_a : \mathcal{D}_a \longrightarrow \mathcal{D}_{a'}$.

Se $D \subset \mathcal{D}_a$, será utilizada a notação $f_a(D)$ para designar o conjunto $D' \subset \mathcal{D}_{a'}$ tal que $i' \in D'$ sse $i \in D$ e $f_a(i) = i'$.

Sendo dada uma abstração, a toda teoria Γ de $P\mathcal{D}_A$ corresponde uma teoria "abstrata" Γ' de $P\mathcal{D}_{A'}$ da seguinte forma:

1.-se F é uma fórmula atômica de $P\mathcal{D}_A$ de polaridade p:

- se $f_A(a) = a'$ então

$$f(aD) = \begin{cases} a' f_a(D') & \text{se } p = \text{“}+\text{”} \\ a' \mathcal{D}_{a'} \setminus f_a(\mathcal{D}_a \setminus D) & \text{se } p = \text{“}-\text{”} \end{cases}$$

- se $f_A(a)$ não está definida,

$$f(aD) = \begin{cases} 1 & \text{se } p = \text{“+”} \\ 0 & \text{se } p = \text{“–”} \end{cases}$$

2. se F é uma fórmula complexa, $f(\neg F) = \neg f(F)$, e $f(F_1 \rightarrow F_2) = f(F_1) \rightarrow f(F_2)$.

A correspondência imposta acima para as fórmulas que envolvem atributos não pertencentes a $Imagem(f_A)$, indica que estas fórmulas podem ser desprezadas na teoria abstrata, a menos que sua satisfação independa do valor deste atributo. O mesmo ocorre em relação à redução dos domínios dos atributos.

A interpretação informal da abstração descrita acima é a seguinte: alguns atributos da teoria original são desprezados (aqueles para os quais f_A não está definida), enquanto outros tem seus domínios simplificados (pela redução de domínio ocasionada pelas funções f_a).

Definição 4.4.2 (ABSTRAÇÃO DE UM MODELO) Seja um modelo m de uma teoria Γ em $P\mathcal{D}_A$ e Γ' uma abstração de Γ obtida através de f; denotar-se-á por $m(a)$ o valor do atributo a em m. A *imagem* $f(m)$ de m pela abstração f é definida por:

- se $f_A(a)$ está definida, $f(m)(a) = f_a(m(a))$;
- se $f_A(a)$ não está definida, $f(m)(a)$ também não está definida.

Evidentemente $f(m)$ pertence a mesma classe de modelos de Γ'.

(RECÍPROCA: EXPANSÃO DE UM MODELO)
De forma recíproca a todo modelo m' de Γ' corresponde um conjunto de estados em $P\mathcal{D}_A$, denotado por $f^{-1}(m')$, da seguinte forma: se m é um estado, tem-se $m \in f^{-1}(m')$ sse para todo $a \in Imagem(f_A)$ tem-se $f_a(m(a)) = m'(a')$. Note-se que de forma geral $f^{-1}(m')$ não é único pois a função de abstração não é injetiva, e não se pode garantir que os seus elementos sejam modelos de Γ.

A proposição que segue relaciona a abstração de um modelo aos modelos da teoria abstrata correspondente:

Proposição 4.4.1 Seja Γ uma teoria e Γ' uma de suas abstrações; se m é um modelo de Γ então $f(m) = m'$ para algum modelo m' de Γ'.

Prova: Como apresentado anteriormente, cada fórmula de uma teoria pode ser considerada como uma restrição sobre os possíveis modelos da teoria. A veracidade da proposição decorre do fato que uma abstração como indicado em 4.4.1 relaxa a restrição imposta sobre os modelos.

A figura 4.2 ilustra a aplicação desta proposição.

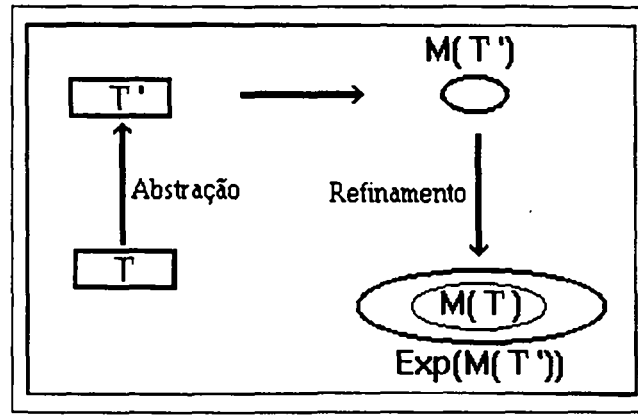

Figura 4.2: Construção abstração / cálculo de modelos proposta

Abstração como auxílio à verificação

A proposição acima permite que se simplifique o processo de verificação de uma propriedade P sobre um sistema S cujo comportamento (SFE) já foi calculado. Para tanto pode-se utilizar apenas a visão abstrata do sistema que utiliza unicamente os atributos e domínios mencionados em P. A verificação da propriedade pode ser feita sobre esta abstração de S.

Os passos a serem seguidos são:

- define-se uma abstração $P\mathcal{D}_{A'}$ de $P\mathcal{D}_A$, de acordo com os elementos (atributos, domínios) presentes em P;
- encontra-se a abstração Γ' da teoria Γ, que originou S;
- são calculados os modelos $M(\Gamma')$ de Γ', sobre os quais se procede a verificação.

Outras aplicações da abstração

O mesmo processo pode ser aplicado em conjunto com a modularização desenvolvida anteriormente. Com efeito se M e N são dois módulos, seu comportamento conjunto pode ser considerado sobre um conjunto mais simples de estados obtidos a partir de uma visão abstrata destes módulos.

O mecanismo de abstração proposto também pode ser utilizado de forma a auxiliar o cálculo de modelos de uma teoria. Neste caso procede-se da seguinte forma:

- efetuam-se os passos acima descritos, obtendo-se $M(\Gamma')$;
- calcula-se a "expansão" do conjunto $M(\Gamma')$, dada por $\bigcup_{m' \in M(\Gamma')} f^{-1}(m')$;
- volta-se à teoria original Γ, calculando os seus modelos a partir do conjunto restrito de estados encontrado no passo anterior (ver figura 4.2).

Este processo pode ser aplicado sucessivas vezes, numa composição hierárquica. A busca da melhor abstração para uma teoria, em termos da eficiência no cálculo dos modelos, deverá ser objeto de trabalho futuro.

Exemplo 4.4.1 Seja a teoria Γ sobre $A = (a\ b\ c)$, com $\mathcal{D}_a = \mathcal{D}_b = \{0\ 1\ 2\}$, $\mathcal{D}_c = \{0\ 1\}$, e formada pelo conjunto de fórmulas:

$$a\{0\ 1\} \wedge b\{0\ 2\} \rightarrow c\{1\}$$
$$a\{0\ 2\} \wedge c\{1\} \rightarrow a\{2\}$$
$$a\{1\ 2\} \wedge b\{1\} \rightarrow c\{0\}$$

Os modelos desta teoria são:

modelos	a	b	c
m0	0	1	0
m1	1	0	1
m2	1	1	0
m3	1	2	1
m4	2	1	0
m5	2	0	0
m6	2	0	1
m7	2	2	0
m8	2	2	1

Considere-se $A' = (a'\ b'\ c')$ com $\mathcal{D}_{a'} = \mathcal{D}_{b'} = \mathcal{D}_{c'} = \{0'\ 1'\}$, e as funções de abstração: $f_A : a \mapsto a', b \mapsto b', c \mapsto c'$ $f_a = f_b : 0 \mapsto 0', 1 \mapsto 1', 2 \mapsto 1'$ e $f_c : 0 \mapsto 0', 1 \mapsto 1'$.
Obtem-se a teoria "abstrata" Γ', de acordo com a definição 4.4.1:

$$a'\{0'\} \wedge b'\{0'\} \rightarrow c'\{1'\}$$
$$a'\{0'\} \wedge c'\{1'\} \rightarrow a'\{1'\}$$

A abstração da fórmula $a\{1\ 2\} \wedge b\{1\} \rightarrow c\{0\}$, é tornada trivial: com efeito, tem-se que $f_b(b\{1\}) = \{\ \}$, falsificando a premissa da implicação, que se torna uma tautologia.
Os modelos de Γ' são:

modelos	a	b	c
m0'	1'	0'	0'
m1'	1'	0'	1'
m2'	1'	1'	0'
m3'	1'	1'	1'
m4'	0'	1'	0'

Calculando-se o valor de $f^{-1}(M(\Gamma'))$ obtém-se os pontos:

modelos	a	b	c
m0	0	1	0
m1	1	0	1
m2	1	1	0
m3	1	2	1
m4	2	1	0
m5	2	0	0
m6	2	0	1
m7	2	2	0
m8	2	2	1
m9	1	0	0
m10	1	1	1
m11	1	2	0
m12	2	1	1
m13	0	2	0

Pode-se observar que todos os modelos de Γ estão em $f^{-1}(M(\Gamma'))$; entretanto os pontos m9, ... m13 não são modelos de Γ, conforme os resultados expostos nesta seção.

Abstração aplicada às outras lógicas

O raciocínio empregado no desenvolvimento acima pode ser generalizado para as outras lógicas apresentadas neste trabalho.

No caso das lógicas paraconsistentes as funções de abstração de domínio f_a sobre os atributos devem obedecer a uma restrição adicional: a ordem do reticulado "abstração" deve ser coerente com a ordem o reticulado de base: se a é um atributo tal que $f_A(a) = a'$, a função $f_a : \mathcal{T}_a \longrightarrow \mathcal{T}_{a'}$ deve obedecer a restrição seguinte: para todo $\mu, \nu \in \mathcal{T}_a$ com $\mu \leq \nu$ deve-se ter $f_a(\mu) \leq f_a(\nu)$ em $\mathcal{T}_{a'}$. O atendimento a esta restrição garante a coerência do enfoque apresentado para as lógicas $P\mathcal{T}_A$.

Já o caso das lógicas temporais pode ser tratado considerando os SDR correspondentes, encarados como geradores de modelos da teoria temporal (ver ítem 3.8.5). Neste caso a definição de abstração deve ser estendida para as equações de estado, conforme se mostra no desenvolvimento abaixo.

Considere-se o caso em que a equação de estado é formada por equações escalares na forma $b = a \ \$ \ 1$ (os domínios destes atributos são considerados idênticos). As seguintes restrições devem ser seguidas para a obtenção correta de uma abstração:

- $f_A(a)$ e $f_A(b)$ devem ser ambas definidas ou ambas indefinidas;
- no caso em que ambas estão definidas deve-se ter $f_a = f_b$.

Estas restrições garantem que a evolução traduzida pela equação de estado no sistema abstrato seja coerente com a evolução estabelecida no sistema original.

4.5 Exemplo de aplicação: supervisão e controle de um processo de fabricação de produto

A apresentação deste exemplo tem por objetivo ilustrar o uso do Núcleo de SBCTR proposto, em uma aplicação de Supervisão e Controle de Processo. Ênfase especial é dada à linguagem de descrição e aos passos realizados pelo procedimento de compilação.

Considere-se um processo químico industrial no qual estão envolvidas as três atividades seguintes: o controle de uma reação de dissolução, a neutralização de um processo de corrosão e o procedimento de carga de produto. As indicações são passadas a um operador do sistema que supervisiona todo o processo.

Cada uma destas atividades é um módulo do sistema global, respectivamente denominados "mistura", "anti-corrosão" e "carga-de-produto". Apresenta-se a seguir uma proposta para a descrição de cada um deste módulos.

Subsistema mistura

Dois produtos químicos sofrem um processo de mistura em um tanque, de forma semelhante à apresentada no exemplo 3.8.1. A quantidade de soluto que entra no tanque pode ser controlada; a concentração da mistura que sai do tanque e sua derivada podem ser medidas; os valores obtidos são utilizados para realimentar o sistema, formando um laço de controle que deve manter a concentração entre limites aceitáveis, através de uma regulagem da entrada de soluto no tanque.

O sistema dinâmico qualitativo que descreve este sub-sistema e o respectivo laço de controle podem ser codificados na linguagem proposta da seguinte forma:

```
(MODULE  mistura
         INITIAL   concentracao = <0>,
                   derivada-concentracao = <0>;
         INPUTS    concentracao, derivada-concentracao;
         OUTPUTS   teta;

  (DOMAIN         concentracao [<-> <0> <+>])
  (DOMAIN         derivada-concentracao [<-> <0> <+>])
  (DOMAIN         teta [<0> <+>])

  (QUALITATIVE-EQUATION teta = concentracao + derivada-concentracao)

  (DOMAIN-CONTINUITY-RESTRICTION  concentracao)
  (DERIVATIVE-RESTRICTION  derivada-concentracao = D concentracao)

  (RULE r1    IF    concentracao = <->
              THEN  teta = <+> )
  (RULE r2    IF    concentracao = <+>
              THEN  teta = <0> )          )
```

Os principais pontos a salientar nesta descrição e em sua compilação são os seguintes:

- os atributos `concentracao` e `derivada-concentracao` formam a entrada do módulo;

- estes atributos assumem valores conforme os domínios discretos indicados pelo comando DOMAIN; neste caso se utilizam os símbolos <+>, <0> e <-> para elementos do domínio;

- os valores destes atributos, quando recebidos do processo, são contínuos, devendo sofrer uma discretização para que possam ser empregados de forma correta no formalismo; a conversão de dá por meio de árvore de decisão, como a indicada à figura 4.3 para o atributo concentracao;

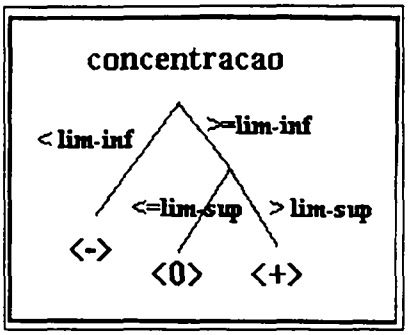

Figura 4.3: Estrutura de discretização para os valores do atributo concentracao

- o corpo do bloco é neste caso constituído pelos comandos QUALITATIVE-EQUATION, DOMAIN-CONTINUITY-RESTRICTION e DERIVATIVE-RESTRICTION;

  QUALITATIVE-EQUATION define uma restrição a ser atendida pelos valores dos atributos envolvidos; ''+'' e ''='' são funções qualitativas consideradas como predefinidas sobre o domínio [<-> <0> <+>] (uma forma de definir estas funções foi indicada em 3.3); esta restrição determina os modelos da teoria formada por esta equação, para os atributos considerados;

- DOMAIN-CONTINUITY-RESTRICTION e DERIVATIVE-RESTRICTION estabelecem limites sobre as possíveis evoluções do sistema; o símbolo D que aparece no corpo deste último comando indica uma derivação; e

- finalmente os comandos RULE indicam a forma de funcionamento do laço de controle do sistema: se concentracao é baixa (respectivamente alta) <-> (<+>), maior (menor) quantidade de soluto deve alimentar o tanque.

Os modelos encontrados a partir desta especificação determinam um SFE que descreve as possíveis evoluções do sistema; o autômato obtido a partir deste SFE foi mostrado à figura 3.17, e reproduz de forma adequada os comportamentos admissíveis do sistema, fazendo com que o controle (teta) adequado seja aplicado convenientemente em função dos valores de concentracao e derivada-concentracao recebidos.

Subsistema anti-corrosão

A reação descrita acima é considerada corrosiva; a fim de avaliar o nível de corrosão do processo, três testes de laboratório (`teste[i]`, `i = 0, 1, 2`) são feitos sobre a mistura. A partir dos resultados obtidos nos testes e de uma série de heurísticas fornecidas por especialistas, um reagente conveniente (`solucao[1]` ou `solucao[2]`) deve ser adicionado à mistura para diminuir a corrosão; resultados intermediários (`auxiliar`) também são empregados. Neste caso considera-se que as heurísticas são descritas por um conjunto de regras, conforme a descrição indicada a seguir:

```
(MODULE anti-corrosao
        DIMENSION teste[3], solucao[2];
        INPUTS    teste[0..2];
        OUTPUTS   solucao[0..1];

  (LATTICE   teste, solucao [CD] )

  (RULE r1   IF    teste[0] [0.90 0.00]
             THEN  solucao[0] [1.00 0.00] )
  (RULE r2   IF    teste[1] [1.00 0.00] AND
                   auxiliar [0.80 0.00]
             THEN  auxiliar [0.80 0.00] )
  (RULE r3   IF    auxiliar [0.75 0.00]
             THEN  solucao[0] [0.00 1.00] )
  (RULE r3   IF    teste[2] [0.85 0.00]
             THEN  solucao[1] [1.00 0.00] )   )
```

Este módulo é caracterizado pelos seguintes pontos referentes à linguagem de descrição e à sua compilação:

- o comando `DIMENSION` indica a existência dos atributos vetoriais `teste` (atributo de entrada) e `solucao` (atributo de saída); por exemplo, podem ser empregadas no corpo do módulo as referências `teste[0]`, `teste[1]` e `teste[2]`;
- o comando `LATTICE` faz com que um reticulado seja empregado para domínio dos atributos indicados; neste caso se utiliza o reticulado `[CD]`, cujos elementos são formados por pares de números entre 0 e 1, que têm por semântica respectivamente um grau de crença e de descrença na afirmação correspondente;
- de forma similar ao exemplo anterior, estruturas de conversão para domínios discretos, devem ser empregadas, fazendo com que, neste caso, valores de crença e descrença nos resultados dos testes sejam fornecidos;
- salienta-se que este módulo é completamente atemporal, fornecendo o valor do atributo em saída como reação instantânea aos valores fornecidos na entrada, independentemente do resultado occorrido na reação anterior deste mesmo módulo.

A situação descrita corresponde à apresentada anteriormente no exemplo 3.4.3, quando se apresentou o tratamento paraconsistente como meio de formalização de elementos imprecisos ou incertos. A noção de satisfação para as fórmulas presentes neste módulo correspondem portanto à sua definição na lógica paraconsistente correspondente. Os resultados fornecidos na saída também podem ser interpretados como indicado anteriormente.

Subsistema carga de produto

Enfim, um procedimento de carga de produto deve ser realizado, com base em um conjunto de especificações de comportamento. O procedimento de carga é basicamente o seguinte (ver figura 4.4):

- periodicamente caminhões tanque chegam à estação de recarga; quando o caminhão está posicionado o operador recebe um sinal "caminhão no lugar", para que possa abrir a válvula que controla o fluxo do produto;
- esta válvula pode ser aberta ou fechada por um comando de duas posições no painel de controle; para monitorar a posição da válvula dois sensores diferentes são utilizados, para maior segurança;
- os sensores fornecem as indicações "aberto", "fechado" ou "em movimento" (*barberpole*); esta última posição é mostrada no caso em que a válvula não está nem aberta nem fechada, mas num estado intermediário - que é usualmente uma posição "em movimento", mas pode eventualmente ser devido a uma falha no mecanismo de acionamento;
- quando o operador está enchendo um caminhão e um sinal "caminhão cheio" é recebido, ele deve imediatamente fechar a válvula e enviar o sinal "OK para sair" ao motorista do caminhão;
- o procedimento acima descrito deve ser seguido durante todo o dia até as 17:00 horas, quando o operador deve encerrar o preenchimento do último caminhão - se este estiver ocorrendo - e finalizar o trabalho do dia.

Durante a execução do plano básico várias falhas podem ocorrer: a falha mais crítica é o não-fechamento correto da válvula; outra falha menos crítica corresponde à sua não abertura. Quando a válvula falhar em abrir ou fechar, ou se houver uma dúvida sobre sua real posição, então o operador deve acionar a parada de emergência, considerado como o último recurso de operação.

O operador deve monitorar diversas condições temporais e agir de acordo com a situação presente:

- C1: se os dois sensores não indicam valores concordantes - i.e. um indica "aberto" e o outro indica "fechado", aquele que não indica uma posição coincidente com o valor da posição da válvula deve ser considerado em falha;
- C2: se ambos os sensores indicam a posição "em-movimento" por mais de 4 segundos, considera-se que a válvula está bloqueada e o procedimento de emergência deve ser acionado;
- C3: se algum dos sensores mostra "em-movimento" por mais de 8 segundos, o sensor em questão é considerado em falha;
- C4: se 6 segundos após a mudança da posição da válvula, o valor esperado não é indicado pelos 2 sensores, deve-se acionar o procedimento de emergência; este intervalo deve ser reduzido a 3 segundos se um dos sensores está em falha.

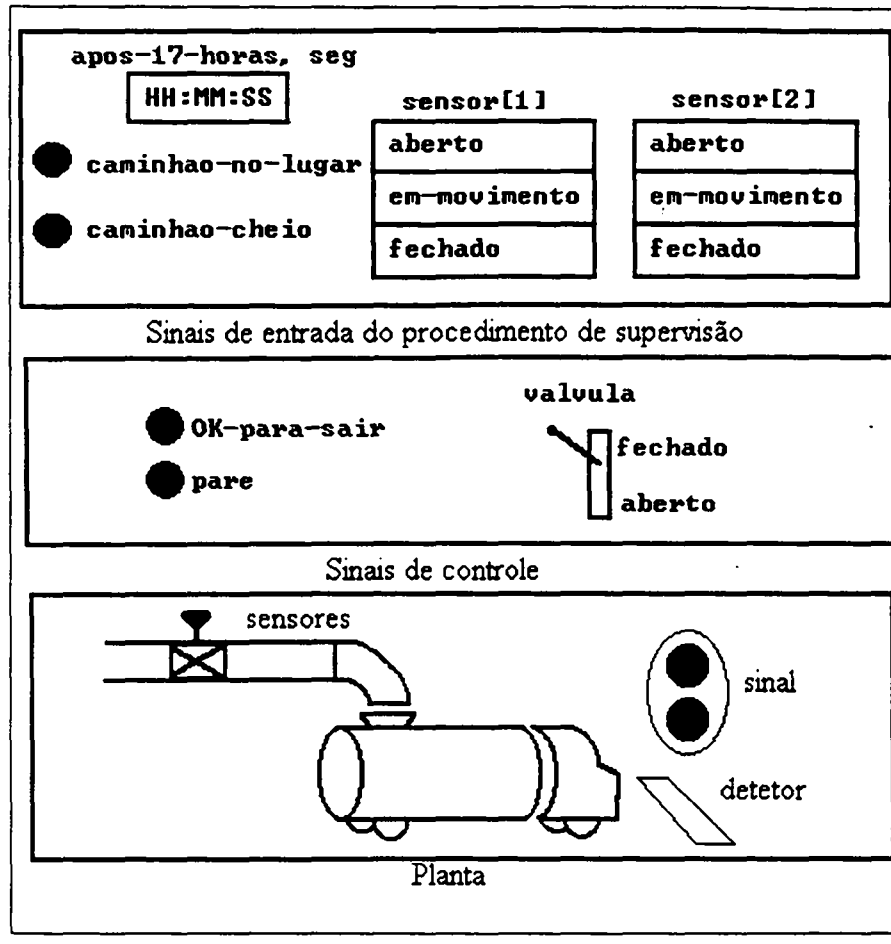

Figura 4.4: Diagrama esquemático do processo a supervisionar e controlar

Durante toda a operação se permite que apenas um sensor falhe. Nesta situação o procedimento operacional do sistema muda, mas o sistema pode continuar em operação. Já se ambos os sensores falham a operação deve ser imediatamente interrompida.

Várias construções temporais interessantes estão presentes no exemplo; há interrupções no sentido de abortar um plano (como na falha da válvula) e interrupções que devem ser armazenadas e postergadas por algum tempo (como no final da jornada diária de trabalho), bem como *time-outs* que aparecem por exemplo quando da indicação de falha de um sensor.

O subsistema pode ser codificado a partir da descrição do problema e do procedimento de operação, da seguinte forma:

```
(MODULE carga-de-produto
        DIMENSION sensor[2], estado-sensor[2];
        INITIAL   estado-sensor[0] = OK,
                  estado-sensor[1] = OK;
        INPUTS caminhao-no-lugar, caminhao-cheio,
               sensor[0..1], apos-17-horas, seg;
        OUTPUTS valvula, OK-para-sair, pare;
```

```
(RULE R-C1      IF
                sensor[i] <> sensor[i+1] AND
                sensor[i] = valvula
                THEN    estado-sensor[i+1] = falha    )
(RULE R-C2      IF
                (# seg DURING (sensor[0] = em-movimento AND
                               sensor[1] = em-movimento)) GE 4
                THEN    pare = TRUE )
(RULE R-C3      IF
                (# seg DURING sensor[i] = em-movimento) GE 8
                THEN    estado-sensor[i] = falha )
(RULE R1-C4     IF
                (# seg DURING (estado-sensor[0] = OK AND
                               estado-sensor[1] = OK AND
                               valvula <> sensor[0]  AND
                               valvula <> sensor[1])) GE 6
                THEN    pare = TRUE )
(RULE R2-C4     IF
                (# seg DURING (estado-sensor[i] = falha AND
                               valvula <> sensor[i+1])) GE 3
                THEN    pare = TRUE )
(RULE R-abrir   IF      caminhao-no-lugar
                THEN    valvula = aberto AND
                        OK-para-sair = FALSE )
(RULE R-fechar IF       caminhao-cheio
                THEN    valvula = fechado )
(RULE R1-OK-para-sair
                IF      estado-sensor[0] = OK AND
                        estado-sensor[1] = OK AND
                        sensor[0] = fechado AND
                        sensor[1] = fechado AND
                        NOT (apos-17-horas)
                THEN    OK-para-sair = TRUE )
(RULE R2-OK-para-sair
                IF      estado-sensor[i+1] = falha AND
                        sensor[i] = fechado AND
                        NOT (apos-17-horas)
                THEN    OK-para-sair = TRUE )
(RULE R1-apos-17-horas
                IF      apos-17-horas AND
                        estado-sensor[0] = OK AND
                        estado-sensor[1] = OK AND
                        sensor[0] = fechado AND
                        sensor[1] = fechado
                THEN    pare = TRUE )
(RULE R2-apos-17-horas
                IF      apos-17-horas AND
                        estado-sensor[i] = falha AND
                        sensor[i+1] = fechado
                THEN    pare = TRUE )
```

```
(RULE seguranca ALL-GOING
                (NOT sensor[0] = aberto AND
                     sensor[1] = aberto AND
                     OK-para-sair))                )
```

Algumas considerações referentes a esta descrição e à sua compilação podem ser feitas:

- DIMENSION define **sensor** e **estado-sensor** como atributos vetoriais;
- o comando INITIAL fixa o valor inicial dos atributos **estado-sensor[0]** e **estado-sensor[1]**, utilizados para o cálculo das condições temporais;
- os atributos de entrada deste subsistema são: **caminhao-no-lugar**, **caminhao-cheio**, **apos-17-horas**, **sensor[0]**, **sensor[1]**, e **seg**;

  os sinais de controle enviados ao processo são **valvula**, **OK-para-sair**, e o indicativo de parada de emergência **pare**;
- instâncias proposicionais devem ser geradas para algumas regras; por exemplo a regra R-C1 dá origem às duas instâncias proposicionais seguintes:

```
(RULE R-C1[0]  IF
               sensor[0] <> sensor[1] AND
               sensor[0] = valvula
               THEN   estado-sensor[1] = falha   )
(RULE R-C1[1]  IF
               sensor[1] <> sensor[0] AND
               sensor[1] = valvula
               THEN   estado-sensor[0] = falha   )
```

- os domínios dos atributos podem ser obtidos a partir das regras; por exemplo o atributo **valvula** tem como domínio o conjunto discreto {**aberto**, **fechado**, **em-movimento**}, construído a partir de todos as referências ao atributo em questão presentes nas regras;
- uma estrutura de conversão para os valores dos atributos em domínios codificados deve ser construída pelo compilador; no caso do atributo **valvula**, por exemplo, uma tabela de *hashing* é a estrutura mais adequada para efetuar a conversão;
- as regras devem ser convertidas à estrutura lógica equivalente; por exemplo, se os domínios dos atributos **caminhao-no-lugar**, **valvula** e **OK-para-sair** são codificados respectivamente por {0 1}, {0 1 2} e {0 1}, a regra **R-abrir** corresponderá a implicação:

$$\texttt{caminhao-no-lugar}\{1\} \rightarrow \texttt{valvula}\{0\} \wedge \texttt{OK-para-sair}\{0\}$$

- várias construções temporais estão presentes neste subsistema, como **# seg DURING (sensor[0] = em-movimento AND sensor[1] = em-movimento)) GE 4** na regra **R-C2**.

  Esta condição é satisfeita se a parte DURING permanece verdadeira durante quatro ou mais ocorrências do evento **seg**, e pode ser apresentada na forma do transdutor indicado à figura 4.5.

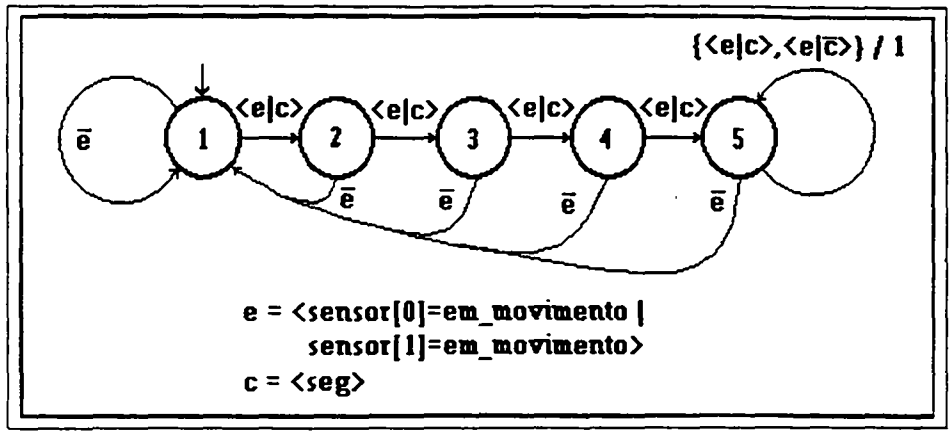

Figura 4.5: A condição temporal presente em R-C2 como transdutor

sensor[0]	sensor[1]	valvula	estado-sensor[1]
aberto	fechado	aberto	falha
aberto	em-movimento	aberto	falha
fechado	aberto	fechado	falha
fechado	em-movimento	fechado	falha
outras	combinações	possíveis	OK

Figura 4.6: Modelos para um subconjunto dos atributos no módulo carga-de-produto

- a regra `seguranca` descreve uma propriedade desejada de funcionamento, incorporada diretamente à descrição do problema;
- o próximo passo do processo de compilação é a geração dos modelos da teoria lógica obtida a partir deste conjunto de regras; a figura 4.6 apresenta os modelos obtidos para o subconjunto de atributos `sensor[0]`, `sensor[1]`, `valvula` e `estado-sensor[1]`;
- finalmente a estrutura de evolução dos modelos (SFE) é obtida, criando o autômato que é utilizado como código de execução para o módulo.

Composição dos subsistemas

O sistema global é obtido pela composição dos três módulos descritos anteriormente, da seguinte forma:

```
(MODULE  sistema-global
         DIMENSION teste[3], solucao[2],
                   sensor[2], estado-sensor[2];
         INPUTS concentracao, derivada-concentracao,
                teste[0..2],
```

```
            caminhao-no-lugar, caminhao-cheio,
            sensor[0..1], apos-17-horas, clock;
        OUTPUTS teta, solucao[0..1],
            valvula, OK-para-sair, pare;

(mistura ||
 anti-corrosao ||
 carga-de-produto
(LINK clock = seg)  )  )
```

De forma geral foram empregados os mesmos nomes para os atributos no sistema global e nos subsistemas, fazendo com que as conexões sejam efetuadas diretamente; uma exceção é o atributo SEG, que fica conectado no módulo `sistema-global` ao atributo `clock`, que serve como sinal de temporização.

No exemplo não ocorre ciclo de causalidade, e os módulos podem ser executados em paralelo, como indicado por || entre os módulos, pois não há conexões entre suas entradas e saídas.

A partir da descrição apresentada do problema e por meio do processo de compilação é possível a obtenção, de acordo com o formalismo exposto no capítulo anterior, de autômatos que descrevem o comportamento de cada módulo. O sistema global pode então ser considerado como a sucessão das aplicações destes autômatos; a estrutura final obtida (que também forma um autômato [LEWIS 81]) deve ser empregada para a verificação do atendimento da hipótese do sincronismo pelo sistema, e, finalmente, para a operação direta do sistema frente ao seu ambiente.

4.6 Conclusão

Apresentou-se neste capítulo uma proposta para a construção de um núcleo gerador de SBCTR, que se fundamenta diretamente no formalismo descrito neste trabalho.

A metodologia proposta preve uma fase de desenvolvimento, na qual o problema é modelado segundo uma linguagem de descrição; e uma fase de operação, na qual o autômato resultante da etapa anterior será colocado em funcionamento frente ao ambiente para o qual o sistema foi projetado.

Uma proposta preliminar da linguagem de descrição foi apresentada neste capítulo; sua principal característica é a decomposição do sistema em módulos, cada um dos quais segue a forma declarativa de representação fundamentada no paradigma Sistema Baseado em Regras. Propriedades tempo-real podem ser diretamente codificadas na linguagem, garantindo seu atendimento no produto resultante de sua compilação.

O processo de compilação realiza a transformação desta linguagem para o seu formato lógico equivalente, conforme o formalismo descrito anteriormente; os modelos correspondentes à descrição de cada módulo podem então ser calculados. A estrutura final obtida é um autômato, que pode ser empregado diretamente frente ao ambiente - etapa de operação, o que assegura o atendimento aos requisitos de determinismo, reatividade e satisfação das condições temporais impostas.

A aplicabilidade do enfoque proposto foi ilustrada através de um exemplo na área de Controle de Processos e Automação Industrial.

Capítulo 5

Conclusões e perspectivas

Uma visão geral do trabalho

Apresentou-se neste trabalho um formalismo e uma proposta de metodologia para o desenvolvimento de Sistemas Baseados no Conhecimento para aplicações Tempo-Real. A elaboração de uma proposta para a construção de SBCTR deve levar em conta dois aspectos conflitantes: o não-determinismo que caracteriza os sistemas de Inteligência Artificial, advindo da aplicação associativa do conhecimento que os compõem, e o desejado determinismo dos Sistemas Tempo-Real.

O formalismo se fundamenta no enfoque síncrono para o tempo, de forma a atender aos requisitos temporais impostos e a exigência de reação instantânea do sistema a eventos externos. Sua construção se baseia na definição de um conjunto de cálculos proposicionais. O mais complexo destes cálculos, denominado PTT_A é temporal - com uma estrutura linear para o passado e arborescente para o futuro - e paraconsistente, i.e. admite teorias inconsistentes e não triviais. A expressividade deste cálculo é ainda ampliada pela inclusão de condições temporais expressas por meio de autômatos.

Em cada um destes cálculos um conjunto de fórmulas - uma teoria - descreve os relacionamentos lógicos sobre uma série de atributos - que correspondem aos elementos atômicos de cada cálculo.

A evolução temporal do sistema pode ser considerada a partir da caracterização dos atributos como sendo de entrada, de estado, e de saída. O cálculo explícito de todos os modelos da teoria permite que, para uma dada situação dos atributos de entrada e de estado, seja possível obter os atributos de saída correspondentes.

Esta abordagem se assemelha à evolução de um Sistema Dinâmico sujeito à Restrições, que forma a base para a implementação do sistema, e representa um compromisso conveniente entre a utilização associativa do conhecimento disponível e o determinismo desejado da estrutura de execução.

Outros elementos também podem ser incorporados diretamente no formalismo, como funções e sistemas dinâmicos qualitativos, que representam o conhecimento profundo do sistema e fórmulas temporais sobre o futuro, que se referem às suas possíveis evoluções, facilitando a codificação direta de propriedades tempo-real.

Uma vez definido o quadro formal a ser utilizado, foram desenvolvidos algoritmos e estruturas de dados que implementam a estratégia proposta, tanto para o cálculo dos modelos como para a verificação de satisfação de fórmulas temporais.

A proposta feita neste trabalho é de construir SBCTR que tem por base formal o

cálculo citado (temporal e paraconsistente), e que utiliza o paradigma "Sistema Baseado em Regras". Além de ser compatível com o formalismo adotado, esta é a forma de representação do conhecimento mais utilizada nas áreas de aplicação consideradas - Controle de Processos, Automação Industrial e Robótica.

A metodologia de geração de um SBCTR prevê uma primeira etapa de **Desenvolvimento do Sistema**, na qual se supõe que o conhecimento necessário à solução do problema é descrito pelo especialista na forma de um conjunto de módulos, cada um dos quais contendo um conjunto de regras que descreve o comportamento do módulo.

O conjunto de regras é então convertido, através de um processo de compilação, para a forma lógica equivalente. O formalismo pode então ser aplicado, obtendo-se os modelos da teoria em questão. A estrutura obtida para a representação dos modelos dos módulos é finalmente convertida para um autômato, que pode ser diretamente apliçado na segunda etapa, dita de **Operação do Sistema**. O uso de um autômato para a execução assegura o determinismo e reatividade ao ambiente desejados para o SBCTR, além de garantir a satisfação das condições temporais, a partir da verificação da hipótese do sincronismo.

Os vários requisitos necessários à construção de um SBCTR foram assim convenientemente considerados como indicado à figura 2.5:

O enfoque síncrono e a verificação da satisfação de sua hipótese de base garantem o atendimento aos requisitos de correção temporal, reatividade e determinismo desejados dos STR.

Quanto à representação do conhecimento e aos mecanismos de inferência, o tratamento paraconsistente dado ao formalismo permite a expressão formalizada de elementos imprecisos e incertos, e o tratamento de problemas de inconsistência e incompletude da Base de Conhecimentos dos sistema; o uso de autômatos aumenta o poder de expressão da lógica adotada, sendo conveniente para representar condições temporais; finalmente, a relação estabelecida entre o formalismo e os Sistemas Dinâmicos Qualitativos permite a fácil inclusão deste elementos como forma de representação do conhecimento profundo no sistema.

Avaliação da proposta de SBCTR

Principais características

Os pontos de destaque do trabalho são os seguintes:

- a utilização de um formalismo matemático bem definido, fundamentado em extensões da lógica temporal;
- construção de um SBCTR que segue o enfoque síncrono, o que simplifica o tratamento da questão temporal;
- o manuseio pelo sistema de construções temporais complexas construídas com o auxílio de autômatos; estas construções vem na direção do aumento de expressividade da lógica adotada, e permitem a aplicação das primitivas temporais a qualquer sinal, como se exige na abordagem síncrona;
- o tratamento uniforme dado às regras que compõem o sistema e funções e sistemas dinâmicos qualitativos; este tratamento responde a várias questões relativas à atuação integrada destas duas formas de representação, comum nas aplicações de Controle e Supervisão de sistemas que constituem uma das motivações principais deste trabalho;

- a adoção de um tratamento paraconsistente indica um caminho na direção do tratamento automático de questões relativas à consistência e à completude de Bases de Conhecimentos, garantindo a correta aplicação do conhecimento em cada caso;

- o mesmo tratamento paraconsistente permite ainda que se incorporem ao sistema de modo formalizado elementos incertos e imprecisos, fundamentais para a representação de heurísticas;

- a incorporação de propriedades tempo-real a verificar diretamente no formalismo, unificando as etapas de desenvolvimento e verificação de propriedades no sistema; e

- a análise da correspondência entre condições temporais, Sistemas Dinâmicos sujeitos a Restrições e autômatos; esta correspondência permite a geração de um código de execução determinista, de funcionamento reativo e que garante o atendimento às restrições temporais impostas; o uso desta estrutura do tipo autômato leva a uma execução extremamente eficiente.

Limitações da proposta

Entretanto a proposta apresentada neste trabalho possui uma série de limitações, em especial as provenientes da adoção do enfoque síncrono e da forma de implementação correspondente:

- a impossibilidade de modificações incrementais: qualquer modificação no sistema implica em uma nova compilação e conseqüente geração de outra estrutura de execução;

- o uso do sistema em aplicações desenvolvidas sobre um ambiente distribuído é problemática, pois só são possíveis procedimentos de compilação separada para cada componente do sistema; e

- a utilização de mecanismos complementares normalmente empregados em Sistemas Baseados no Conhecimento é difícil: em particular os mecanismos de aprendizagem e de explanação, que exigem o interfaceamento interativo com o usuário do sistema.

Do exposto acima, constata-se que ocorre uma perda na flexibilidade de aplicação do conhecimento, que é uma das características principais dos Sistemas Baseados no Conhecimento. Entretanto esta perda teve que ser admitida e tolerada quando se considera o determinismo que deve caracterizar os Sistemas Tempo-Real, como um requisito essencial a ser obedecido nesta proposta.

No que diz respeito a eficiência do código gerado, as seguintes considerações podem ser feitas:

- como o enfoque síncrono não previlegia nenhum evento em particular, em certos casos um tratamento mais eficiente poderia ser obtido para as condições temporais, a partir da focalização do sistema em uma "variável tempo" especial, e.g. um sinal de relógio proveniente diretamente do *hardware*; e

- a conversão dos valores dos atributos para a sua forma codificada, de acordo com a proposta, é sempre feita numa etapa inicial, antes da execução do autômato; a utilização de uma rede otimizada, onde os testes só são efetuados à medida que necessários, como é o caso no sistema KHEOPS, leva a tempos de reação menores aos obtidos de acordo com esta proposta.

Finalmente outro fator limitante da aplicação deste formalismo é o bem conhecido fenômeno da explosão combinatória originária de um cálculo exaustivo como o proposto. Embora esta tenha sido uma preocupação sempre presente ao longo do desenvolvimento do trabalho, se constitui indubitavelmente em uma limitação de sua aplicabilidade.

Perspectivas futuras

Frente ao estado atual do trabalho, considera-se, além da implementação de um protótipo de gerador SBCTR, as seguintes possibilidades para a sua extensão:

- A inclusão de condições temporais generalizadas por meio do uso de autômatos sobre reticulados [DOUGHERTY 88]; este tratamento é coerente com a lógica paraconsistente empregada no formalismo; esta extensão permitirá um aumento no poder de expressão das condições temporais.

- Um estudo mais profundo sobre o uso das técnicas de abstração propostas e sua relação com fórmulas temporais; em particular o uso da teoria paraconsistente deve possibilitar a generalização do conceito de abstração, permitindo sua exploração de forma mais efetiva, em outros pontos da etapa de desenvolvimento.

- A introdução no formalismo de um esquema de tipagem de dados, o que permitirá uma maior concisão na expressão de condições temporais e a sua utilização em sistemas mais complexos.

- O funcionamente cooperativo de agentes reativos; além de poder ser utilizado de forma isolada e individual, é possível se imaginar um conjunto de agentes reativos inteligentes atuando em conjunto de forma assíncrona para realização de uma tarefa comum; assim o uso de vários destes módulos se comunicando, seja via *blackboard* [RICH 83] ou de por troca de mensagens, pode ser uma alternativa atraente para sistemas em ambiente distribuído ou para sistemas demasiadamente complexos.

- O uso do formalismo apresentado para a verificação da satisfação de propriedades temporais em outros sistemas, particularmente no sistema KHEOPS, que apresenta características similares; neste sistema cada ramo da árvore de decisão gerada para estrutura de execução pode ser encarado como modelo no sentido do formalismo; a definição das equações de estado a partir das evoluções restritas permitirão a aplicação direta dos algoritmos desenvolvidos com este objetivo.

- A aplicação do sistema como uma alternativa para a construção de supervisores para os Sistemas a Eventos Discretos, com base em heurísticas e regras de conduta descritas por operadores; neste caso passam a ser importantes noções relativas à diferenciação entre eventos controláveis e não-controláveis [RAMADGE 87].

- A construção direta em *hardware* de Sistemas Baseados no Conhecimento; esta alternativa tem sido cada vez mais considerada [TOGAI 86], [BERRY 91] e pode ser facilmente implementada a partir da proposta, devido as peculiaridades da estrutura de execução obtida.

- As linguagens síncronas para a programação de Sistemas Tempo-Real, atuam segundo paradigmas diversos: paradigma linguagem imperativa (e.g. ESTEREL), paradigma fluxo de dados (e.g. SIGNAL) e sistema baseado em regras (KHEOPS e esta proposta). A utilização do mesmo princípio de base (hipótese do sincronismo), a geração de um mesmo tipo de código objeto (autômatos para execução), e a complementaridade dos propósitos e das representações colocam como perspectiva promissora o uso de diversos paradigmas de forma conjugada em aplicações reais.

Considerações finais

A proposta deste trabalho se apresenta como uma alternativa para a aplicação de técnicas de Inteligência Artificial em Sistemas Tempo-Real. A adoção da abordagem síncrona para o tempo e o cálculo explícito de modelos permite que se obtenham estruturas altamente eficientes para a etapa de execução, garantindo o aplicabilidade do enfoque nos casos práticos.

O sistema que implementará esta proposta encontra-se na etapa de especificação, sendo que os algoritmos descritos foram testados sobre procedimentos escritos em Lisp; o sistema deve ser em seguida completamente implementado no LCMI-EEL-UFSC [1], na forma de um protótipo de gerador de SBCTR e em seguida aplicado e avaliado num ambiente industrial real.

[1] Laboratório de Controle e Microinformática, Departamento de Engenharia Elétrica, Universidade Federal de Santa Catarina.

Apêndice A

Apresentação suscinta das linguagens síncronas ESTEREL e SIGNAL

A.1 ESTEREL

ESTEREL [BERRY 83], [BERRY 89], [BOUSSINOT 91] é definida por seus autores como "uma linguagem imperativa paralela que possui uma semântica matemática rigorosa e uma implementação completa" [BERRY 87].

Um programa ESTEREL é composto por módulos; cada módulo contém uma parte declarativa, que descreve os dados utilizados e seus tipos, as constantes e funções empregadas, e uma parte procedural, composta pelas instruções propriamente ditas.

As instruções são muito similares àquelas de uma linguagem imperativa clássica: a seqüência (;), as atribuições de valor (:=), os condicionais ($if \ldots then \ldots else \ldots$), bem como comandos para paralelismo explícito (||) e para a emissão de sinais ($emit(S)$); além disto a linguagem inclui instruções temporais tais como disparadores ($await(S)$), cães de guarda ($watchdog\ S$) e laços temporais ($loop \ldots each\ S$).

O principal elemento manipulado é o *evento*, definido como a emissão ou recepção simultânea de vários sinais; cada sinal representa uma interação atômica entre o sistema e o ambiente. Sinais simultâneos são combinados e difundidos a todos os componentes do sistema segundo o princípio da difusão instantânea (*instantaneous broadcasting*), e utilizando uma função pré-definida de agregação. O tempo é considerado multiforme: cada sinal define uma marcação temporal, sobre a qual podem ser aplicados os comandos temporais da linguagem.

A partir da semântica formal definida para a linguagem o compilador ESTEREL constrói um código de execução eficiente: as regras operacionais permitem a definição do novo estado do sistema a partir do estado atual e do evento recebido na entrada. Um autômato é construído simplesmente por agregação (*gathering*) das transições, identificando estados com os resíduos sucessivos obtidos da aplicação das regras operacionais da linguagem [BERRY 86].

A tradução para um autômato feita pelo compilador ESTEREL traz inúmeras vantagens:

- o programa paralelo inicial é transformado num código seqüencial equivalente: não há processos a gerenciar durante a execução, e problemas de sincronização e comunicação

```
module TREINAMENTO:
input  METRO, VOLTA,
       PASSADA, SEGUNDO;
output CORRER_CADENCIADAMENTE, EXPIRAR, SALTAR,
       CORRER_O_MAIS_RAPIDO_POSSIVEL;
do                % DURANTE 2 VOLTAS
  loop            % CADA VOLTA
     do
       emit CORRER_CADENCIADAMENTE
     upto 100 METRO;

     do
     every PASSADA do
            emit SALTAR
          ||
            emit EXPIRAR
     end
     upto 20 SEGUNDO;
     emit CORRER_O_MAIS_RAPIDO_POSSIVEL
  each VOLTA;
upto 2 VOLTA.
```

Figura A.1: Programa temporal de treinamento escrito em ESTEREL

são minimizados;

- o determinismo das aplicações é conservado;
- a hipótese do sincronismo pode ser facilmente verificada: o tempo da transição maximal do autômato é calculável e imperativamente limitado.

Para a verificação de propriedades tempo-real o código gerado por ESTEREL pode ser submetido de forma integrada a um sistema verificador de especificações (o sistema AUTO [VERGAMINI 86]). O compilador ESTEREL gera código "C", o que possibilita a fácil integração do mesmo ao nível das aplicações. Um exemplo típico de utilização é apresentado a seguir.

Exemplo A.1.1 Este exemplo extraído de [BERRY 87] visa mostrar a versatilidade do uso de ESTEREL no manuseio de instruções temporais.

Considere-se um corredor que deve executar o seguinte programa de treinamento: fazer duas voltas na pista, correndo cadenciadamente 100 metros, depois por 20 segundos saltar expirando a cada passada; terminar a volta correndo o mais rápido possível. A seqüência deve ser repetida na segunda volta.

O código ESTEREL deste procedimento é apresentado à figura A.1:

Saliente-se que este programa não pode ser facilmente transcrito nas linguagens clássicas. O autômato gerado pelo compilador ESTEREL para este exemplo pode ser visualizado na figura A.2.

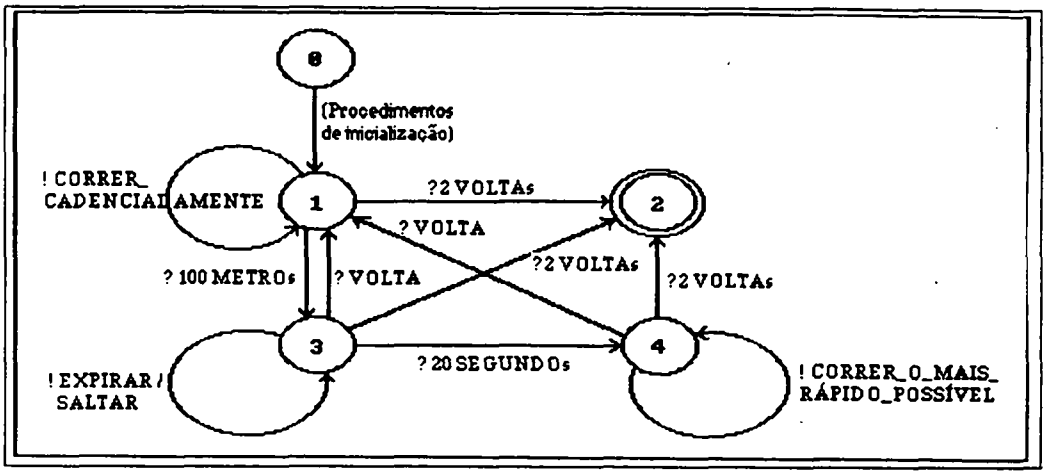

Figura A.2: Autômato ESTEREL para o exemplo

A.2 SIGNAL

SIGNAL [BENVENISTE 90], [GAUTIER 87], [LeGUERNIC 91] é uma linguagem do tipo "fluxo de dados", cuja programação tem um estilo declarativo: um programa é constituído por um conjunto de equações que devem ser verificadas e mantidas consistentes durante a execução. Este enfoque busca resgatar os mesmos princípios que são empregados na Teoria de Controle de Sistemas Contínuos, onde ferramentas matemáticas - tais como equações diferenciais, etc - constituem verdades imutáveis ao longo da evolução do processo.

As equações são implicitamente executadas em paralelo. O elemento primitivo da linguagem se denomina *sinal* (*signal*, na linguagem), e é constituído por uma seqüência de valores ordenados temporalmente.

O tempo é multiforme: cada sinal produz uma temporização, formada pelas reações em que o mesmo está presente. O sinal booleano $H(S)$ definido a partir do sinal S da seguinte forma: $H(S) = 1$ $(TRUE)$ se e somente se S está presente na reação corrente e $H(S) = 0$ $(FALSE)$ no outro caso, determina o *relógio* do sinal S. A noção de sincronismo entre sinais é imediata: dois sinais são síncronos se tem o mesmo relógio, i.e. $H(S_1) = H(S_2)$. Cada sinal booleano B pode também servir de suporte temporal para a existência de novos sinais: basta considerar que os mesmos estão definidos apenas nos instantes em que B é verdadeiro.

As operações elementares da linguagem são de dois tipos:

- extensões da aplicação de funções "instantâneas" aos sinais, considerando a aplicação das mesmas pontualmente ao valor corrente de cada sinal e definidas apenas para sinais simultâneos; e

p	x0		x1	x2		x3	x4	x5	...
q	y0	y1	y2		y3		y4		...
b	*0*		*1*	*0*	*1*	*1*		*0*	...
$p\$1$			x0	x1		x2	x3	x4	...
p *when* b			x1			x3			...
p *default* q	x0	y1	x1	x2	y3	x3	x4	x5	...
p *cell* q	x0	x0	x1	x2	x2	x3	x4	x5	...

Figura A.3: A semântica dos operadores temporais SIGNAL

- operadores que consideram o aspecto temporal dos sinais, relacionando valores que não estão presentes numa mesma reação.

No primeiro caso se colocam tanto as extensões de todas as funções atemporais usuais - e.g. se $X = (x0, x1, x2, \ldots)$ e $Y = (y0, y1, y2, \ldots)$ são dois sinais simultâneos, $X + Y$ representa o sinal adição $X + Y = (x0 + y0, x1 + y1, x2 + y2, \ldots)$; bem como condicionais do tipo $if\ X > Y\ then\ X - Y\ else\ Y - X$, cujo resultado denota o sinal que tem para n-ésimo termo a diferença absoluta entre os n-ésimos termos de X e Y.

Os principais operadores temporais são: o atraso ($1), a amostragem (*when*), a fusão determinista (*default*), e a obtenção do último valor presente de um sinal (*cell*). A figura A.2 apresenta de forma resumida a semântica destes operadores.

SIGNAL possui também um cálculo de sincronização, chamado *cálculo de relógios*, formalmente definido sobre polinômios de n variáveis que assumem valores no corpo dos inteiros módulo 3 ($Z/3Z$) [BORGNE 89]. Este cálculo analisa a sincronização entre os sinais e verifica a existência de laços de causalidade, reportando ao usuário os problemas existentes. Os relógios são ordenados segundo uma hierarquia de freqüências, formando um reticulado. O valor maximal deste reticulado constituirá o passo do autômato que é gerado pelo compilador SIGNAL para a execução do programa. Um código totalmente seqüencial é produzido: os estados do autômato não são produzidos de modo explícito, mas ficam mascarados nos valores particulares dos sinais; as transições se traduzem por uma série de condicionais portando sobre os valores dos estados.

As mesmas observações feitas para a linguagem ESTEREL em relação à geração de um autômato para execução valem neste caso. Em particular a hipótese do sincronismo pode ser facilmente verificada a partir do tempo de transição maximal do autômato.

A linguagem SIGNAL permite a construção de um sistema através de sua decomposição em módulos, de forma similar aos diagramas de blocos empregados em Controle. As equações determinam o comportamento individual do bloco, e são previstos mecanismos que designam os sinais de entrada e saída e a forma de interconexão dos blocos. Esta tarefa é facilitada pela existência, no ambiente de desenvolvimento SIGNAL, de um editor gráfico especializado. O editor também permite a construção hierarquizada dos sistemas, pela inclusão de blocos pré-definidos no interior de um bloco em construção.

Quanto a verificação de propriedades tempo-real, além de se utilizarem instrumentos de análise de autômatos, uma outra abordagem é possível: a codificação direta em SIGNAL das propriedades a verificar. Neste caso o processo de análise de sincronismo e causalidade automaticamente permitirá o atendimento aos requisitos impostos.

O compilador SIGNAL permite a geração de código FORTRAN ou "C". Apresenta-se a seguir um exemplo típico da sua aplicação.

Exemplo A.2.1 A linguagem SIGNAL tem como principal caraterística o tratamento do problema de sincronização de sinais. Este exemplo extraído de [DUTERTRE 91] ilustra o problema.

Seja um dispositivo que pode estar ATIVO ou INATIVO, de acordo com os comandos ATIVAR e DESATIVAR. Seu comportamento pode ser descrito pelo autômato apresentado na figura A.4.

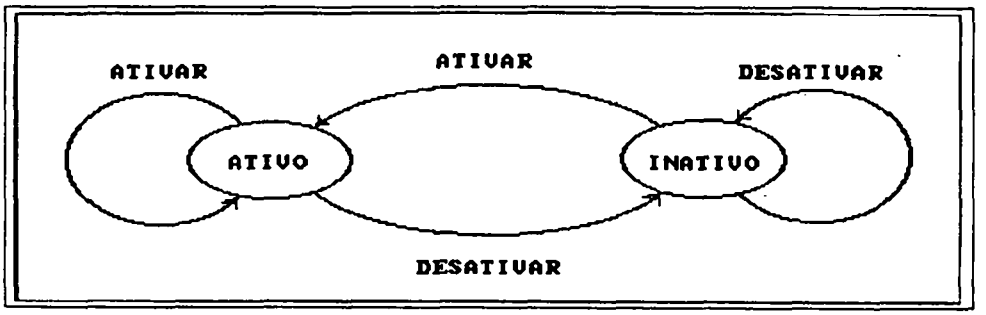

Figura A.4: Autômato que representa um dispositico de dois estados

O sinal booleano SATIVO definido pela expressão

```
SATIVO := ATIVAR default (not DESATIVAR)
```

produz um valor VERDADEIRO (respectivamente FALSO) se ATIVAR (respectivamente DESATIVAR) está presente, com prioridade para a ativação. Porém com esta definição SATIVO só está disponível nos instantes em que ATIVAR ou DESATIVAR existem. Para efetuar uma amostragem em qualquer instante podem ser utilizadas as expressões:

```
ATIVO  := SATIVO default (not ZATIVO) e
ZATIVO := ATIVO $ 1 .
```

fazendo com que ATIVO seja igual ao valor mais recente do sinal SATIVO, e um sinal de H que designa os instantes de consulta.

O processo ESTADO completo pode ser visto à figura A.5:

O cronograma a seguir A.6 descreve uma possível evolução de ESTADO, com o sinal ZATIVO inicialmente falso:

Um exemplo de utilização do processo ESTADO é dado no modelo INTERRUPTOR A.7 abaixo, e cuja representação gráfica feita através do editor dedicado SIGNAL pode ser vista à figura A.8.

Este modelo controla a transmissão do sinal ENTRADA para SAIDA pelos comandos ABRIR e FECHAR e foi aplicado no problema de descrição e simulação de um sistema de controle de passagem ferroviária em nível [DUTERTRE 91].

```
process ESTADO =
       { ? event   ATIVAR, DESATIVAR, H
         ! logical ATIVO, ZATIVO        }

       ( | SATIVO := ATIVAR default (not DESATIVAR)
         | ATIVO  := SATIVO default ZATIVO
         | ZATIVO := ATIVO $ 1
         | HATIVO := (event SATIVO) default H
         | synchro {ATIVO, HATIVO}
         | )
       where
          event HATIVO
end
```

Figura A.5: O processo ESTADO em SIGNAL

ATIVAR	-				-	-		-	...
DESATIVAR			-				-	-	...
SATIVO	*V*		*F*		*V*	*V*	*F*	*V*	...
H		-		-	-				...
ATIVO	*V*	*V*	*F*	*F*	*V*	*V*	*F*	*V*	...
ZATIVO	*F*	*V*	*V*	*F*	*F*	*V*	*V*	*F*	...

Figura A.6: Evolução dos sinais no processo ESTADO

```
process INTERRUPTOR =
       (logical INICIAL)
       { ? event   ABRIR, FECHAR, ENTRADA
         ! event   SAIDA                  }

       ( | { ATIVO, ZATIVO } :=
                        ESTADO() {ABRIR, FECHAR, ENTRADA}
         | SAIDA := ENTRADA when ZATIVO
         | )
       where
         logical ATIVO, ZATIVO init INICIAL
end
```

Figura A.7: O processo INTERRUPTOR em SIGNAL

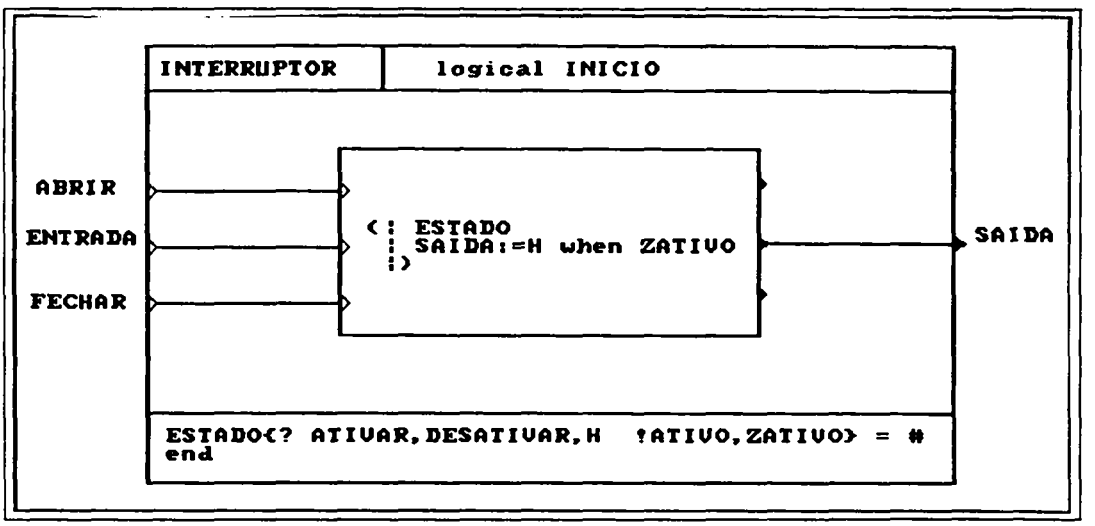

Figura A.8: O processo interruptor na visão gráfica (Editor SIGNAL)

Apêndice B

Alguns Sistemas Baseados no Conhecimento Tempo-Real existentes

PICON

O sistema PICON [LEINWEBER 87] é o primeiro gerador de Sistemas Especialistas (SE) Tempo-Real disponível comercialmente para o desenvolvimento de aplicações em Controle de Processos. A introdução do conhecimento no sistema é facilitada pelo uso de dois níveis de representação: inicialmente é estabelecido um vocabulário de *frames*, que utiliza uma representação icônica para os componentes do sistema e que deixa explícita a estrutura de base da aplicação; em seguida um editor especial de regras utiliza estes frames para criar as regras, sem variáveis, que são utilizadas pelo sistema para a inferência. O sistema PICON foi desenvolvido em resposta aos seguintes requisitos: (1) utilização da variável tempo, pois a sintaxe das regras permite a referência a intervalos de tempo; (2) ativação de regras altamente eficiente por meio de uma hierarquia de contextos, o que proporciona ao sistema, em tempo de execução, um instrumento de foco de atenção poderoso; (3) reciclagem constante da memória de trabalho, com a eliminação des elementos não mais necessários com a passagem do tempo; (4) aceitação de comandos interativos, o que permite a condução do sistema com base nas indicações dos operadores; (5) indicação para os sensores dos tempos de validade das leituras e dos períodos de amostragem; os sensores também estão sujeitos a uma função de metadiagnose que realiza a análise dos dados históricos e sua tendência, e permite testes de consistência para os valores medidos; e (6) possibilidade de comunicação entre diversos SE, seja para redundância, cooperação entre elementos especializados ou, numa organização hierárquica, para a supervisão. O sistema foi implementado sobre uma máquina Lisp, e suporta algum processamento periférico através de processadores dedicados; esta parte do sistema é implementada em "C". O PICON tem sido empregado principalmente em aplicações aero-espaciais, para planejamento e otimização, escalonamento de atividades e análise a longo prazo do comportamento de estações espaciais, monitoração de satélites, etc. Além disto também foi empregado na indústria petroquímica, e em outras aplicações de controle de processos complexos.

HEXSCON

O sistema HEXSCON [WRIGHT 86] - uma sigla para *Hybrid Expert System Controler* - é um SE experimental dedicado às tarefas de controle em Tempo-Real encontradas em aplicações militares e industriais avançadas. O sistema foi desenvolvido considerando uma utilização com as seguintes características: (1) capacidade de 5000 regras em um sistema microcomputador de 512K de memória; (2) tempo de resposta de 10 a 100 ms; (3) habilidade de manuseio de múltiplos objetos; e (4) habilidade de funcionamento contínuo, a despeito da presença de incertezas. O termo "híbrido" empregado no nome deste sistema surge da integração proposta entre o conhecimento convencional (procedural) e o obtido com as técnicas de manuseio do conhecimento. O uso das técnicas convencionais baseia-se no reconhecimento que muitas operações podem ser manuseadas satisfatoriamente pela lógica booleana. O sistema incorpora um "tradutor de representações", que permite a troca de informações entre estas duas partes componentes do sistema, sem a necessidade de uma compatibilidade de representação completa. A execução do sistema é feita sobre um suporte operacional tempo-real, sendo que a parte baseada no conhecimento constitui uma única tarefa para este suporte. A forma de representação de conhecimento utilizada é a de regras, que são compiladas produzindo um código compacto. São admitidos vários objetos no sistema, mas a hipótese subjacente é que todas as regras se aplicam sobre o objeto que se está raciocinando no momento, de modo que as ligações entre objetos são limitadas. O tempo é incorporado através de *frames* presente, passado e futuro, estas duas últimas armazenadas em disco devido a necessidade de preservar a capacidade de memória do microcomputador. O mecanismo de inferência utiliza o denominado "raciocínio progressivo", de forma a permitir a obtenção da melhor resposta possível dentro do tempo disponível. O sistema possui quatro níveis de raciocínio, o primeiro dos quais é implementado em lógica convencional (componente reflexa do sistema), e os demais são obtidos a partir da base de conhecimentos do sistema, com uso das regras. Cada nível utiliza mais dados e mais tempo de processamento que o nível anterior. É possível o uso de encadeamento para frente e para trás, e os problemas são manuseados de acordo com prioridades associadas, garantindo que problemas mais urgentes sejam atendidos em primeiro lugar. São também utilizados parâmetros de crença ou confiança que indicam a importância relativa dos fatos nas regras, que também servem para o controle do sistema: as regras de maior confiança associada são disparadas prioritariamente. Várias linhas de raciocínio são possíveis simultaneamente, e a combinação das evidências é feita usando-se a técnica de Dempster-Shafer [KUENG-CHI 90]. A implementação do sistema foi feita em Pascal, de modo a possibilitar uma posterior migração para Ada.

G2

O sistema G2 [MOORE 87], [MOORE 90] foi desenvolvido para aplicações onde centenas ou milhares de variáveis são monitoradas concorrentemente, como em aplicações de controle de processos, aplicações industriais CIM, gerência de redes, aplicações financeiras on-line, etc. O G2 permite simulações antes da efetiva utilização, através da criação de módulos que geram dados como se estes estivessem sendo enviados pelo ambiente, em uma aplicação real. Assim problemas específicos e modos de falha podem ser simulados de forma a testar a qualidade dos conhecimentos empregados e validar o raciocínio sobre vários cenários. O G2 foi desenvolvido de modo a atender os seguintes critérios gerais: (1) representação de conhecimento: os componentes do sistema são descritos através de *frames*, sobre os quais

operam duas formas de representação: regras para o conhecimento heurístico e equações dinâmicas para o conhecimento profundo; as regras possuem variáveis de forma a possibilitar generalizações; (2) dimensão temporal: o sistema é dotado de mecanismos capazes de expressar o conhecimento sobre o tempo tanto nas equações dinâmicas como por meio das regras; (3) performance: a capacidade de execução do sistema é elevada; (4) interface para entrada do conhecimento: é realizada por um conjunto de ícones, facilitando bastante a tarefa do especialista e clarificando as relações entre os componentes do sistema; (5) manutenção da verdade: o sistema é provido com um mecanismo sofisticado que permite a modificação dinâmica da cadeia de inferência para frente ou para trás, quando de alterações tempo-real sobre os dados, de modo a manter sempre a validade das conclusões; (6) foco de atenção: como uso de metaconhecimento, o sistema proporciona a invocação da parte de seu conhecimento global necessária à resolução de um conjunto particular de circunstâncias, enquanto apenas monitora o restante do sistema a fim de detetar eventuais situações de emergência; e (7) manuseio de incerteza: o sistema permite o emprego de medidas incertas associadas aos dados. Além disto o G2 prove uma interface gráfica sofisticada para os operadores, mecanismos de recepção dos dados on-line e possibilidade de ligação com outros sistemas G2 por meio de redes. O sistema é implementado em CommonLisp sobre estações de trabalho.

EUREKA-II

O sistema Eureka-II [TANO 88] é uma ferramenta de Engenharia do Conhecimento que visa suportar a construção de Sistemas Baseados no Conhecimento para aplicações Tempo-Real em Controle de Processos. Os requisitos considerados para sua construção foram derivados da análise e modelagem do comportamento de operadores especialistas. Estes requisitos são: (1) a forma de representação de conhecimento e o método de inferência devem corresponder à linha de raciocínio seguida pelo operador; (2) o sistema deve permitir uma fácil conexão com o *software* convencional; e (3) o sistema deve ter uma eficiente capacidade de processamento. Para atender estes objetivos o sistema utiliza cinco tipos de conhecimento: conhecimento heurístico na forma de regras; conhecimento factual; conhecimento procedural; conhecimento hipotético; e metaconhecimento, obtido por metaregras e com uso de prioridades. Estas diferentes formas de conhecimento são processadas por um mecanismo de inferência eficiente, constituído de um supervisor, um processador de rede de regras, um gerenciador de *frames* e um gerenciador de hipóteses. As regras são compiladas de acordo com uma extensão do algoritmo RETE [FORGY 82], que provê a melhoria deste algoritmo através de partições na rede, otimização da ordem com que são efetuadas as junções e eficiente manuseio das memórias. É possível o encadeamento misto e hierárquico, com o uso extensivo da divisão em sub-problemas. A arquitetura permite o processamento de alto nível do conhecimento e a sua integração com programas de aplicação existentes. O sistema foi utilizado para aplicações em controle de aeração em fornalhas e em outras aplicações industriais.

PAMELA

PAMELA [BARACHINI 88] é uma linguagem para o desenvolvimento de Sistemas Especialistas Tempo-Real. O objetivo essencial do sistema é manter sua operação de acordo com a escala de tempo de processo a controlar. O sistema baseia-se no paradigma sistemas de produção, e os esforços se concentram na utilização do algoritmo de unificação RETE

[FORGY 82] e suas possíveis extensões, visando uma performance tempo-real. Um sistema de produção puro não é diretamente aplicável a STR, pois eventos esporádicos podem influenciar o ciclo de base durante a operação. As extensões propostas permitem o manuseio de interrupções e o atendimento destas interrupções mesmo quando da execução das regras. De acordo com a metodologia proposta, os nós da rede RETE são implementados através de procedimentos recursivos, após uma etapa de redução da rede. Assim estes procedimentos podem ser interrompidos como qualquer procedimento ordinário. Há funções especiais que permitem a recuperação, criação, deleção e modificação de objetos fora do contexto das regras, a partir de rotinas de interrupção. Em todos os casos as ações de criação, remoção e modificação são sempre consideradas como atômicas, mas não são executadas diretamente: seu atendimento é feito por um escalonador, que as executa quando conveniente de acordo com a estratégia de escalonamento vigente e as prioridades das demais tarefas no sistema. Em contraposição são também utilizados *demons*, executados imediatamente após sua instanciação, em qualquer ponto do ciclo de inferência. A execução dos *demons* exige a utilização de funções especiais para determinar quais os elementos da memória de trabalho foram modificados e assegurar a coerência da rede. Várias medidas de performance do sistema foram realizadas sobre microcomputadores, embora a concepção do sistema vise seu uso em *hardware* dedicado.

RUM

O sistema RUM [BONISSONE 90] apresenta uma nova abordagem para os SBCTR, baseada na clara distinção entre as duas etapas presentes na utilização destes sistemas: desenvolvimento e execução. O sistema é primariamente dirigido a problemas de classificação dinâmica, caracterizados pela seguinte estrutura: colecionamento de dados provenientes de diversas fontes de forma a gerar um padrão; mapeamento do padrão em um conjunto de soluções possíveis; e escolha da solução mais adequada para o caso. Em especial o sistema é adequado a situações onde os dados e a base de conhecimentos são permeados de incerteza. Os requisitos considerados para o desenvolvimento do sistema foram: (1) propagação eficiente e correta da informação incerta; (2) representação de *defaults*; (3) retração de conclusões inconsistentes com novas e mais confiáveis evidências; (4) reconhecimento e resolução de conflitos evidenciais; (5) modificação incremental da base de conhecimentos de forma a eliminar computações desnecessárias; (6) possibilidade de interrupção do sistema por eventos esporádicos; e (7) manutenção das necessidades de tempo e recursos necessários à resposta tempo-real. O sistema é composto de quatro partes: um ambiente de simulação (Lotta), um ambiente de desenvolvimento (Rum), um sistema rápido e pequeno para execução (Rum Runner) e um *software* para tradução entre os sistemas de desenvolvimento e execução (Rum e Rum Runner). São utilizadas qualificações para os fatos em termos de graus de confirmação e refutação, e graus de suficiência e necessidade para as regras; estes elementos são utilizados para o raciocínio aproximado, de forma a compensar uma maior rapidez na inferência pela perda de qualidade da resposta em termos da certeza associada a esta resposta. O sistema de desenvolvimento é construído sobre frames de um sistema convencional (sistema KEE) e regras de primeira ordem. Permite também a integração com conhecimento procedural, a utilização de contextos e revisão de crenças, além de mecanismos de edição, checagem de erros, depuração, etc. Após o desenvolvimento o sistema gerado é compilado, de forma que em tempo de execução nenhum novo conhecimento é criado. As regras com variáveis são explodidas em suas instâncias proposicionais,

de forma a eliminar a necessidade de complexos mecanismos de unificação. O tradutor analisa a topologia das regras e deriva um grafo direto acíclico (DAG), que representa as dependências entre as regras sob a forma de uma rede compilada, e sobre o qual são feitas as inferências. O DAG suporta motores de inferência em encadeamento para frente e para trás e também mantem e propaga as incertezas entre fatos e regras. Os esforços maiores no desenvolvimento deste sistema se concentraram na construção de um "metacontrolador" utilizado na execução. Este metacontrolador é capaz de manusear interrupções e entradas externas, organizar e escalonar tarefas com o uso de prioridades, e controlar os diversos componentes de forma a atender as restrições temporais com a solução de melhor qualidade possível. O metacontrolador é constituído por: (1) um escalonador baseado em agenda, que manuseia as entradas normais e esporádicas, associando a cada solicitação uma prioridade e sua deadline; o escalonamento é feito então por *shortest deadline*; e (2) por um planejador, que determina como atender a restrição temporal imposta; esta determinação exige a geração de planos (compostos por caminhos no DAG), avaliação de sua utilidade e do seu custo, e seleção do plano para atender a deadline; como há incerteza associada, quanto maior o número de caminhos utilizados, melhor será a qualidade da solução. Os custos associados aos caminhos no DAG são obtidos a partir de intensivas simulações, através do outro componente do sistema (Lotta), que gera vários cenários e obtém os tempos de execução dos planos gerados. Quando da execução o melhor plano que atende a restrição temporal pode ser realizado. Como já citado, neste sistema a melhoria da qualidade de uma solução é dada essencialmente pelo aumento da certeza associada a esta solução. O sistema é implementado em Lisp, e foi utilizado em diversas aplicações, e.g. o componente de situation assessment do sistema *Pilot's Associate* [POMEROY 90], de auxílio ao piloto em caças táticos estratégicos.

DVMT

O sistema Distributed Vehicle Monitoring Testbed (DVMT) [DECKER 90], [DURFEE 88] tem como domínio a monitoração de veículos em movimento detetados por sensores acústicos. O sistema deve analisar os dados provenientes dos sensores acústicos de forma a identificar, localizar e seguir os padrões de veículos que se movem em um espaço bidimensional. O sistema utiliza uma arquitetura tipo quadro negro (*blackboard*) [RICH 83] com quatro níveis principais: sinais, para análise de baixo nível; grupo, para coleções de sons que são produzidos por alguma parte de um veículo; veículo, para sons gerados por cada veículo individualmente; e padrão, composto por vários veículos que atuam de forma coordenada. As fontes de conhecimento executam tarefas de resolução de problema estendendo ou refinando soluções parciais ou hipóteses. Uma das principais características do sistema é que o mesmo faz uso extensivo de raciocínio aproximado: as hipóteses possuem graus de certeza e refutação associadas, e as regras são associadas a graus de certeza, plausibilidade, refutação e dúvida. Estes elementos fornecem as ferramentas para a definição de três tipos básicos de aproximação: busca aproximada, onde uma região menor do espaço de busca é explorada; aproximações sobre os dados, que provém uma visão mais abstrata e mais tratável dos dados; e aproximações no conhecimento empregado, com o uso de apenas parte do conhecimento disponível. O usuário determina critérios para a aceitação de uma solução, incluindo *deadlines* e preferências nas características desta solução (e.g., maior completude e menor certeza). O DVMT aplica as fontes de conhecimento para estender e refinar as soluções parciais e formar a solução que satisfaz as características solicitadas, decidindo

sobre as formas de aproximação a empregar. Vários resultados de simulações do sistema são apresentados, e refletem o compromisso entre a qualidade da solução e a estratégia de aproximação escolhida em função do tempo de execução disponível.

RT-1

O sistema RT-1 [DODHIAWALA 89] representa uma arquitetura de SBCTR construída sobre os requisitos de velocidade de execução, reatividade, correção temporal e "adaptação suave" ou habilidade de reavaliação das prioridades das tarefas de acordo com a carga do sistema e a disponibilidade de recursos. O RT-1 é um sistema de pequena escala, distribuído, baseado no paradigma *blackboard*. Suas principais características são: (1) é formado por uma coleção de módulos de raciocínio que compartilham um espaço comum e se comunicam por meio de eventos sinalizados; (2) estes módulos operam assincronamente e podem estar em dois estados: comunicação e computação; (3) cada módulo de raciocínio é dirigido pelos eventos recebidos; (4) um diretório de eventos especifica o destino dos eventos externos ou sinais enviados; e (5) o *blackboard* é utilizado como depósito dos dados por todos os módulos de raciocínio, podendo ser ele próprio um elemento distribuído. Cada um dos módulos de raciocínio é composto por três processos: (1) um processo entrada / saída, responsável pela recepção e envio dos sinais; (2) um processo *blackboard*, que é compartilhado por todos os módulos e inclui os objetos que podem ser criados, modificados ou deletados em tempo de execução por outros módulos; e (3) um processo raciocínio, feito de acordo com as seguintes etapas: acionamento, responsável pela recepção aos eventos externos e estabelecimento das respostas relevantes a estas mudanças; teste de pré-condição, que assegura que um contexto de informação existe para a execução da ação correspondente; escalonamento, que determina a execução das ações de acordo com heurísticas de controle; e execução, que realiza, num mesmo ciclo, todas as ações indicadas. A reatividade do sistema é "regulável" via uma margem de execução, que permite o contrabalanceamento entre as atividades de atendimento aos eventos - resolução de problema - e atendimento às restrições temporais. Assim a execução do sistema pode ser completamente reativa, completamente dirigida pelos objetivos (*goal-directed*) ou algum estágio intermediário entre estas situações. O sistema RT-1 é escrito em Lisp utilizando *Flavors* sobre uma máquina Lisp, e é instrumentado com um conjunto de métricas para a avaliação do atendimento aos requisitos exigidos (performance, reatividade, correção temporal, etc).

O Atendimento aos Requisitos Tempo-Real nestes Sistemas

A tabela comparativa apresentada à figura B.1 resume as características destes sistemas, sob o ponto de vista do atendimento aos requisitos de um SBCTR apresentados no texto.

A análise da tabela apresentada permite verificar os requisitos para os sistemas citados. Todos os sistemas apresentados (a excessão do KHEOPS) utilizam a abordagem assíncrona para um STR; na maioria dos casos a execução é feita sob a forma de uma tarefa dentro de um ambiente ou executivo dedicado ao sistema de tempo real.

Requisito	Sistema								
	PICON	HEXSCON	G2	Eurekall	Pamela	Kheops	Rum	DVMT	RT-1
Quanto a Representação do Conhecimento (RC)									
Forma da RC	frames + regras proposic.	regras proposic.	frames + regras 1a.ordem	frames + regras 1a.ordem	regras 1a.ordem	regras proposic.	regras: 1a.ordem proposic.	black board	black board
Represen. Temporal	referenc. a intervalo	frames pres.pass e futuro	dimensão temporal nos dados	-	-	valor passado imediato	-	-	-
Impreciso e Incerto	-	parâmetro de crença	existe, não especificado	fatores de certeza	-	sobrepos. de intervalo	graus de crença e refutação	várias medidas certeza	-
Conhecim. Profundo	na entrada (icones)	-	equações dinâmicas	-	-	-	-	-	-
Conhecim. Procedur.	?	priori-zado	sim	?	?	funcional	?	?	?
Metaconh.	seleção de contextos	?	metaregra	metaregra	-	-	seleção de contextos	-	-
Quanto aos Mecanismos de Inferência (MI)									
Direção da Inferênc.	para frente	para frente e para trás	para frente e para trás	para frente e para trás	para frente	para frente	para frente e para trás	-	-
Manutenç. da Verdade	-	-	TMS	-	-	não precisa monotônic	-	-	-
Foco de Atenção	por meio de contextos	?	mecanismo seletivo	seleção metaregra	-	-	por meio de contextos	agenda	agenda
Hipóteses Prioridad	?	?	?	agenda	-	-	?	agenda+ metas	agenda+ metas

Figura B.1: Tabela comparativa do atendimento aos requisitos Tempo-Real em diversos sistemas

Apêndice C

Exemplo de utilização do sistema KHEOPS

Exemplo C..2 Apresenta-se a seguir um exemplo de programação do sistema KHEOPS. No exemplo (ver figura C.1) uma função exterior (`random 2`) fornece um valor (ação `RESTRICT`) 0 ou 1 aleatório ao atributo `zero-or-one`. O conjunto de regras permite a obtenção na saída do atributo `out` que está associado a estabilidade do valor aleatório entre dois ciclos consecutivos. Se o valor fornecido a `zero-or-one` é diferente de 0 ou 1 nenhuma regra é aplicada e o sistema assinala um caso de incompletude (os atributos de saída não são determinados). Apesar disto o valor `nil` é atribuído a `stability` (devido a exceção representada pelo comando `ELSE`). As regras (`r4 t`) e (`r4 nil`) ilustram uma facilidade sintática de agrupamento de regras que está disponível. A rede decisional construída pelo compilador KHEOPS é apresentada à figura C.2.

```
(INPUTS nil)

(OUTPUTS out stability zero-or-one)

(INITIALIZATIONS (RESTRICT %out t))

(RULE r1    IF
            THEN
                 (RESTRICT zero-or-one (random 2)))

(RULE r2    IF    (= zero-or-one 0)
            THEN
                 (RESTRICT out nil))

(RULE r3    IF    (= zero-or-one 1)
            THEN
                 (RESTRICT out t))

{*i '(t nil)
    (RULE (r4 *i)  IF    (= %out *i)
                         (= out  *i)
                   THEN
                         (RESTRICT stability t))
    }

(ELSE e1 (RESTRICT stability nil))
```

Figura C.1: Regras para testar a estabilidade de uma variável aleatória

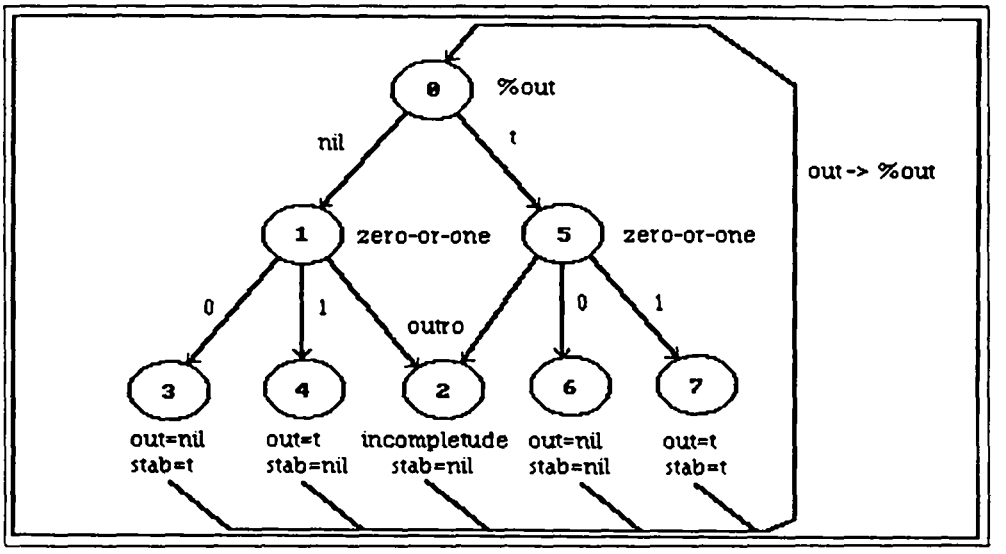

Figura C.2: Rede construída pelo compilador KHEOPS

Apêndice D

Algoritmos para operar sobre hipercubos e conjuntos de hipercubos

Utilizar-se-á a notação $h_1, h_2, \ldots$ para os **hcubo** (hipercubos) e $H_1, H_2, \ldots$ para os **Hcuboset** (conjuntos não redundantes de hipercubos). Todos estes elementos são considerados sobre um mesmo conjunto de índices J. A i-ésima coordenada do **hcubo** h será denotada $h[i]$ e os índices serão denotados por $i, j, \ldots$.

As operações definidas e os algoritmos correspondentes são:

- $h_1 \cap h_2$:

```
procedimento hcubo-INTERSEÇÃO (h1, h2):
variáveis:   h3 := [ ];
para todo    i ∈ I fazer
                 h3[i] := h1[i] ∩ h2[i]
             fim-fazer;
retornar (h3);
fim-procedimento.
```

- $h_1 \setminus h_2$:

```
procedimento hcubo-DIFERENÇA (h1, h2):
variáveis:   haux1 := h1;
             haux2 := h1;
             H := { };
bloco (sair)
             para todo i ∈ I fazer
                          haux2[i] := haux1[i] \ h2[i];
                          haux1[i] := haux2[i] ∩ h2[i];
                          H :=hcubo-Hcuboset-ADIÇÃO (haux2, H);
                          se haux1[i] = { } então
                                       sair-do-bloco (sair);
                                       fim-se;
                          fim-fazer;
             fim-bloco;
retornar (H);
fim-procedimento.
```

- $h_1 \cup h_2$:

```
procedimento hcubo-UNIÃO (h1, h2):
retornar (hcubo-Hcuboset-ADIÇÃO (h1, (hcubo-DIFERENÇA (h1, h2))));
fim-procedimento.
```

- $h_1 \cap H_2$:

```
procedimento hcubo-Hcuboset-INTERSEÇÃO (h1, H2):
variáveis:   H := { };
para todo    h2 ∈ H2 fazer
             H := hcubo-Hcuboset-ADIÇÃO (hcubo-INTERSEÇÃO (h1, h2), H);
             fim-fazer;
retornar (H);
fim-procedimento.
```

- $H_1 \cap H_2$:

```
procedimento Hcuboset-Hcuboset-INTERSEÇÃO (H1, H2):
variáveis:   H := { };
             Haux := { };
para todo    h1 ∈ H1 fazer
             Haux := hcubo-Hcuboset-INTERSEÇÃO (h1, H2);
             para todo h ∈ Haux fazer
                       H := hcubo-Hcuboset-ADIÇÃO (h, H);
                       fim-fazer;
             fim-fazer;
retornar (H);
fim-procedimento.
```

- $h_1 \cup H_2$:

```
procedimento hcubo-Hcuboset-UNIÃO (h1, H2)
variáveis:   H := {h1};
para todo    h2 ∈ H2 fazer
             H := hcubo-Hcuboset-ADIÇÃO (hcubo-DIFERENÇA (h2, h1), H);
             fim-fazer;
retornar (H);
fim-procedimento.
```

- $H_1 \cup H_2$:

```
procedimento Hcuboset-UNIÃO (H1, H2)
variáveis:   H := H2;
para todo    h1 ∈ H1 fazer
             H := hcubo-Hcuboset-UNIÃO (h1, H);
             fim-fazer;
retornar (H);
fim-procedimento.
```

- $\bar{h}$ (complemento de um **hcubo**):

```
procedimento hcubo-COMPLEMENTO (h)
variáveis:   haux := ∏i∈I Dai;
retornar (hcubo-DIFERENÇA (haux, h));
fim-procedimento.
```

- $\bar{H}$:

```
procedimento Hcuboset-COMPLEMENTO (H)
variáveis:   Haux := h-cube-COMPLEMENTO (primeiro-elemento (H));
para todo    h ∈ (outros-elementos (H)) fazer
             Haux := hcubo-Hcuboset-INTERSEÇÃO (hcubo-COMPLEMENTO (h), H);
             fim-fazer;
retornar (Haux);
fim-procedimento.
```

- $H_1 \setminus h_2$ (diferença de conjuntos):

```
procedimento Hcuboset-hcubo-DIFERENÇA (H1, h2)
variáveis:   Haux := { };
para todo    h1 ∈ H1 fazer
             Haux := Hcuboset-UNIÃO (hcubo-DIFERENÇA (h1, h2), Haux);
             fim-fazer;
retornar (Haux);
fim-procedimento.
```

- $H_1 \setminus H_2$:

```
procedimento Hcuboset-DIFERENÇA (H1, H2)
variáveis:   Haux := Hcuboset-hcubo-DIFERENÇA (H1, primeiro-elemento (H2));
para todo    h2 ∈ outros-elementos (H2) fazer
             Haux := Hcuboset-hcubo-DIFERENÇA (Haux, h2);
             fim-fazer;
retornar (Haux);
fim-procedimento.
```

- $h_1 \oplus h_2$:

```
procedimento hcubo-ADIÇÃO (h1, h2):
variáveis:   haux := h1;
se existe    um único i ∈ I com h1[i] ≠ h2[i]
             então haux[i] := h1[i] ∪ h2[i];
                   retornar (haux);
             senão
                   retornar ({ });
             fim-se;
fim-procedimento.
```

- $h_1 \oplus H_2$:

```
procedimento hcubo-Hcuboset-ADIÇÃO (h1, H2)
variáveis:   haux := [ ];
para todo    h2 ∈ H2 fazer
             haux := hcubo-ADIÇÃO (h1, h2);
             se haux ≠ { }
                     então retornar (hcubo-Hcuboset-UNIÃO (haux,
                                     Hcuboset-hcubo-DIFERENÇA (H2, h2)));
                     fim-se;
             fim-fazer;
retornar (hcubo-Hcuboset-UNIÃO (h1, H2));
fim-procedimento.
```

- $H|_{i=v_i}$ (restrição de um **Hcuboset** para um índice i):

```
procedimento Hcuboset-RESTRIÇÃO-SOB-INDICE (H, i, vi)
variáveis:   haux := ∏i∈I Dai;
fazer        haux[i] := vi;
             fim-fazer;
retornar (hcubo-Hcuboset-INTERSEÇÃO (haux, H));
fim-procedimento.
```

- $H|_{\{i=v_i \mid \forall i \in I\}}$ (restrição para um subconjunto de índices):

```
procedimento Hcuboset-RESTRIÇÃO-SOB-CJTO-INDICES (H, I, vI)
variáveis:   Haux1 := H;
             Haux2 := { };
             haux := [ ];
para todo    i ∈ I fazer
             haux := ∏i∈I Dai;
             haux[i] := vi;
             Haux1 := hcubo-Hcuboset-INTERSEÇÃO (haux, Haux1);
             Haux2 := Hcuboset-UNIÃO (Haux1, Haux2);
             fim-fazer;
retornar (Haux2);
fim-procedimento.
```

- $H_1|_{H_2}$ (restrição de um **Hcuboset** em relação a outro **Hcuboset**):

```
procedimento Hcuboset-RESTRIÇÃO-SOB-SUB-Hcuboset(H1, H2)
variáveis:   Haux1 := { };
             Haux2 := { };
             haux := [ ];
para todo    h2 ∈ H2 fazer
             Haux1 := H1;
             haux := ∏i∈I Da_i;
             para todo i ∈ I fazer
                        haux[i] := h2[i];
                        Haux1 := hcubo-Hcuboset-INTERSEÇÃO (haux, Haux1);
                        Haux2 := Hcuboset-UNIÃO (Haux1, Haux2);
                        fim-fazer;
             fim-fazer;
retornar (Haux2);
fim-procedimento.
```

- $proj_i\ H$ (projeção de um **Hcuboset** em relação a um eixo coordenado):

```
procedimento Hcuboset-PROJEÇÃO-SOB-INDICE (H, i)
variáveis:   v := (primeiro-elemento (H))[i];
para todo    h ∈ outros-elementos (H) fazer
             v := v ∪ h[i];
             fim-fazer;
retornar (v);
fim-procedimento.
```

- $proj_{\{i \in I\}}\ H$ (projeção em relação a um conjunto de eixos coordenados):

```
procedimento Hcuboset-PROJEÇÃO-SOB-CJTO-INDICES (H, I)
variáveis:   v := [ ];
             V := { };
para todo    i ∈ I fazer
             v[i] := (primeiro-elemento (H))[i];
             fim-fazer;
V:= {v};
para todo    h ∈ outros-elementos (H) fazer
             para todo i ∈ I fazer
                       v[i] := h[i];
                       fim-fazer;
             V := hcubo-Hcuboset-ADIÇÃO (v, V);
             fim-fazer;
retornar (V);
fim-procedimento.
```

A complexidade $\mathcal{O}$ destes algoritmos pode ser analisada em função da cardinalidade do conjunto de índices J ($n = \|J\|$), sobre o qual os **hcubo** estão definidos, ou, de forma mais ampla, em relação ao número de atributos ($\|A\|$) da teoria correspondente.
O procedimento $h_1 \cap h_2$ é de complexidade linear em relação à n. Da mesma forma $h_1 \oplus h_2$, que realiza a operação de adição entre dois **hcubo**, e que também tem por objetivo reduzir o número de **hcubo** que compõem um **Hcuboset**, tem complexidade $\mathcal{O}(n)$.
Os demais procedimentos são interdependentes; a tabela D.1 mostra a relação entre cada procedimento e as chamadas de outros procedimentos que o mesmo realiza. O cálculo final dos modelos é um problema $\mathcal{NP}$-completo, portanto de complexidade exponencial [LEWIS 81].

procedimento	operações utilizadas na sua definição
$h_1 \cap h_2$	-
$h_1 \setminus h_2$	$h_1 \oplus H_2$
$h_1 \cup h_2$	$h_1 \oplus H_2$, $h_1 \setminus h_2$
$h_1 \cap H_2$	$h_1 \oplus H_2$, $h_1 \cap h_2$
$H_1 \cap H_2$	$h_1 \oplus H_2$, $h_1 \cap H_2$
$h_1 \cup H_2$	$h_1 \oplus H_2$, $h_1 \setminus h_2$
$H_1 \cup H_2$	$h_1 \cup H_2$
h	$h_1 \setminus h_2$
H	$h_1 \cap H_2$, h
$H_1 \setminus h_2$	$H_1 \cup H_2$, $h_1 \setminus h_2$
$H_1 \setminus H_2$	$H_1 \setminus h_2$
$h_1 \oplus h_2$	-
$h_1 \oplus H_2$	$h_1 \oplus h_2$, $h_1 \cup H_2$, $H_1 \setminus h_2$
$H\|_{i=v_i}$	$h_1 \cap H_2$
$H\|_{\{i=v_i \mid \forall i \in I\}}$	$h_1 \cap H_2$, $H_1 \cup H_2$
$H_1\|_{H_2}$	$h_1 \cap H_2$, $H_1 \cup H_2$
$proj_i\ H$	-
$proj_{\{i \in I\}}\ H$	$h_1 \oplus H_2$

Figura D.1: Interdependência entre as operações sobre hipercubos e conjuntos de hipercubos

Apêndice E

Algoritmos para o cálculo de modelos em uma teoria temporal

Na apresentação destes algoritmos considera-se a teoria Γ particionada nas teorias Γ_P (parte atemporal e temporal sobre o passado) e Γ_F (parte temporal sobre o futuro).

Os algoritmos são os seguintes:

- Algoritmo para a propagação de polaridades e tipos em Γ_P:

```
procedimento PROPAGAÇÃO-DE-POLARIDADE-E-TIPO (Γ_P):
para cada axioma A ∈ Γ_P fazer
     para cada sub-fórmula F ∈ A fazer
          colocar tipo(F) e polaridade(F)
                  de acordo com a tabela 3.25
                  fim-fazer;
          fim-fazer;
retornar(1);
fim-procedimento.
```

- Algoritmo para a propagação de polaridades e tipos em Γ_F:
  Este algoritmo é idêntico ao anterior, utilizando como conectores temporais (sobre o futuro) as definições apresentadas em 3.12.2, sobre tipos e polaridades.

- Algoritmo para a construção ascendente dos **Hcuboset** associados a cada sub-fórmula de Γ_P:

```
procedimento Hcuboset-PROPAGAÇÃO (Γ_P):
para cada    sub-fórmula F ∈ A_{Γ_P} (árvore sintática de Γ_P) fazer
             se tipo (F) é
                atômica (sem tipo):
                        o cálculo do Hcuboset associado é direto;
                α
                        o Hcuboset associado é
                        a interseção dos Hcuboset das fórmulas filhas;
                β
                        o Hcuboset associado é
                        a união dos Hcuboset das fórmulas filhas;
fim-procedimento.
```

- Algoritmo para a geração do grafo $\mathcal{G}_\Gamma$ (SFE associado a Γ):

```
procedimento GERA-GRAFO (Γ):
variáveis:   x⁺[i];
             x[j];
             (equação de estado:  x⁺[i] = x[j])
fazer        nós (G_Γ) = M(Γ);
             (obtidos por PROPAGAÇÃO-DE-POLARIDADE-E-TIPO e
             Hcuboset-PROPAGAÇÃO de Γ_P )
             fim-fazer;
para todo    m ∈ nós (G_Γ) fazer
             para todo m' ∈ nós (G_Γ) fazer
                       estabelecer um ramo entre m e m'
                       se e somente se
                       x⁺[i](m) = x[j](m')∀i,j
                       fim-fazer;
             fim-fazer;
fim-procedimento.
```

- Algoritmo para a geração ascendente dos mundos que satisfazem cada sub-fórmula de Γ_F:

```
procedimento PROPAGAÇÃO-DE-MUNDOS (Γ_F):
para cada    sub-fórmula F ∈ A_Γ_F (árvore sintática de Γ_F) fazer
             se tipo (F) é
                atômica (sem tipo):
                       o cálculo dos mundos w que satisfazem F é direto;
                α
                       o conjunto de mundos associado é
                       a interseção dos mundos associados
                       às fórmulas filhas;
                β
                       o conjunto de mundos associado é
                       a união dos mundos associados
                       às fórmulas filhas;
                γ
                       o conjunto de mundos associado é
                       função do conector temporal aplicado às fórmulas filhas
                       (ver definição dos algoritmos a seguir)
fim-procedimento.
```

- Algoritmos utilizados no cálculo de satisfação para as fórmulas temporais (futuro arborescente):

- sXF:

```
procedimento ALGUM-PRÓXIMO (W):
variáveis:   Waux := { };
para cada    mundo w ∈ W fazer
             Waux := Waux ∪ Pre(w);
             fim-fazer;
retornar (Waux);
fim-procedimento.
```

- aXF:

```
procedimento TODO-PRÓXIMO (W):
variáveis:   Waux := { };
para cada    mundo w ∈ ALGUM-PRÓXIMO(W) fazer
             se Suc(w) ⊂ W então
                Waux := Waux ∪ w;
                fim-se;
             fim-fazer;
retornar (Waux);
fim-procedimento.
```

- sFF:

```
procedimento ALGUM-FUTURO (W):
variáveis:   Waux1 := { };
             Waux2 := { };
             Wresp := { };
para cada    mundo w ∈ W fazer
             Waux1_1 := {w}
             enquanto existe w' ∈ Waux1 fazer
                      para todo w'' ∈ Waux1 fazer
                      Wresp := Wresp ∪ {w''};
                      Waux2 := Waux2 ∪ Pre(w'');
                      fim-fazer;
                      Waux1 := Waux2 \ Wresp
             fim-fazer;
retornar (Wresp);
fim-procedimento.
```

- aFF:

```
procedimento TODO-FUTURO (W):
variáveis:   Waux := { };
para cada    mundo w ∈ ALGUM-FUTURO (W) fazer
             se Suc(w) ⊂ W então
                Waux := Waux ∪ w;
                fim-se;
             fim-fazer;
retornar (Waux);
fim-procedimento.
```

- sGF:

```
procedimento ALGUM-INDO (W):
variáveis:   Waux1 := { };
             Waux2 := { };
             Wold := { };
             Wresp := { };
para cada    mundo w ∈ W fazer
             Waux1_1 := {w}
             enquanto existe w' ∈ Waux1 fazer
                      para todo w'' ∈ Waux1 fazer
                      se Suc(w'') ∩ W ≠ ∅ então
                      Wresp := Wresp ∪ {w''};
                      fim-se
                      Wold := Wold ∪ {w''};
                      Waux2 := Waux2 ∪ Pre(w'');
                      fim-fazer;
                      Waux1 := Waux2 \ Wold
             fim-fazer;
retornar (Wresp);
fim-procedimento.
```

- aGF:

```
procedimento TODO-INDO (W):
variáveis:   Waux := { };
para cada    mundo w ∈ ALGUM-INDO (W) fazer
             se Suc(w) ⊂ W então
                Waux := Waux ∪ w;
                fim-se;
             fim-fazer;
retornar (Waux);
fim-procedimento.
```

A complexidade de alguns destes algoritmos pode ser analisada em função do número de estados w: `GERA-GRAFO` tem complexidade $\mathcal{O}(w^2)$; `ALGUM-PROXIMO` e `TODO-PROXIMO` tem complexidade $\mathcal{O}(w)$, assim como `ALGUM-FUTURO` e `ALGUM-INDO`, que tem complexidade $\mathcal{O}(w + r)$, onde r é a cardinalidade da relação de acessibilidade entre os estados [AUDUREAU 89], [XUONG 92]; finalmente `TODO-PROXIMO` e `TODO-INDO` calculam o fechamento sobre o cálculo efetuado nos conectivos anteriores, podendo ser efetuados em $\mathcal{O}((n + r)^2)$ [AHO 74].

Apêndice F

Descrição sintática preliminar da linguagem de entrada

Apresenta-se a seguir uma descrição sintática preliminar da linguagem de entrada proposta neste trabalho para o núcleo gerador de SBCTR, num formato BNF. Os símbolos não terminais serão prefixados por < e sufixados por >. São utilizados os meta-símbolos : := (é definido por), * (repetição zero ou mais vezes da expressão quantificada), + (repetição uma ou mais vezes da expressão quantificada), | (alternância "ou"), e { e } (que cercam elementos opcionais e servem para o agrupamento de ítens).

Alguns outros elementos são considerados como terminais, tais como `numero`, `numero-inteiro` `cadeia-de-caracteres`, etc. Literais são delimitados por aspas.

```
<programa> ::= <modulo>+

<modulo> ::= (MODULE <nome-do-modulo>
                     <declaracoes>

                     INPUTS  { <atributo> |
                               <atributo>[<faixa>] }*
                     OUTPUTS { <atributo> |
                               <atributo>[<faixa>] }*

                     <corpo-do-modulo> )

<nome-do-modulo> ::= cadeia-de-caracteres

<declaracoes> ::= { <dimensionamento> }
                  { <inicializacao> }
                  { <dominio>+ }

<dimensionamento> ::= DIMENSION {<atributo>[<indice>]}+

<atributo> ::= cadeia-de-caracteres

<indice> ::= numero-inteiro
```

```
<inicializacao> ::= (INITIAL
                    { { <atributo> = <valor> |
                        <atributo>[<indice>] } = <valor> }+ )

<valor> ::= numero | cadeia-de-caracteres

<dominio> ::= (DOMAIN  <atributo> [<valor>+]) |
              (LATTICE <atributo> [{<valor> "<" <valor>}+] |
              (LATTICE <atributo> [<reticulado-predefinido>])

<reticulado-predefinido> ::= "2" | "4" | "0,1" | "CD"

<faixa> ::= numero-inteiro | numero-inteiro..numero-inteiro

<corpo-do-modulo> ::= { <regra> |
                        <acao> |
                        <equacao-qualitativa> |
                        <restricao-de-evolucao> |
                        <modulo> |
                       (<atributo>+) = <nome-do-modulo>(<atributo>*)
                        <conexao> }+

<regra> ::= (RULE   <nome-da-regra> <formula> )

<nome-da-regra> ::= cadeia-de-caracteres

<formula> ::= (IF      <formula> THEN    <formula>) |
              (IF      <formula> ONLY-IF <formula>) |
              (<formula> AND     <formula>) |
              (<formula> OR      <formula>) |
              (NOT     <formula>)   |
              (LAST    <formula>)   |
              (PAST    <formula>)   |
              (HAS-BEEN  <formula>) |
              (ALL-NEXT  <formula>) |
              (ALL-FUTUR <formula>) |
              (ALL-GOING <formula>) |
              (SOME-NEXT <formula>) |
              (SOME-FUTUR <formula>)|
              (SOME-GOING <formula>)|
              <formula-atomica>
```

```
<formula-atomica> ::= <atomo> |
                      <atomo> <predicado> <valor> |
                      <atomo> <predicado> <atomo> |
                      <atomo> IN <conjunto-de-valores> |
                      <condicao-temporal>

<atomo> ::= <atributo> | <atributo> $1

<predicado> ::= "=" | "<" | "<=" | ">" | ">=" | "<>"

<conjunto-de-valores> ::= "{" <valor>+ "}"

<condicao-temporal> ::= <operador-temporal> (<atomo>+) |
                        <contador-temporal> |
                        <nome-do-modulo> (<atomo>+)

<operador-temporal> ::= { BEFORE          | AFTER         |
                          STARTS-AT       | STARTS-AFTER  |
                          ONCE-AFTER      | ONCE-SINCE    |
                          ALWAYS-AFTER    | ALWAYS-SINCE  |
                          ALWAYS-FROM-TO  | ONCE-FROM-TO  }

<contador-temporal> ::= (# <formula-atomica>
                             { DURING | AFTER }
                           <formula-atomica>
                             { EQ | LT | LE | GT | GE }
                           numero-inteiro )


<acao> ::=  (ACTION <nome-da-acao>
             WHEN <formula> DO <procedimento-externo> )

<nome-da-acao> ::= cadeia-de-caracteres

<procedimento-externo> ::= chamada-de-funcao

<equacao-qualitativa> ::= (QUALITATIVE-EQUATION
                             <atributo> "="
                             <atributo>
                           <operacao-qualitativa> <atomo> ) |
                           (QUALITATIVE-EQUATION
                             <atributo> "="
                             <funcao-qualitativa> (<atomo>+) )
```

```
<operacao-qualitativa> ::= "+" | "-" | "*" | "/"

<funcao-qualitativa> ::= <nome-do-modulo>

<restricao-de-evolucao> ::= (DOMAIN-CONTINUITY-RESTRICTION
                                  <atomo>)+ |
                            (DERIVATIVE-RESTRICTION
                                  <atributo> "=" D <atributo>)+

<conexao> ::= ({LINK} <atributo> "=" <atributo>)
```

Bibliografia

[ANDSLEY 90] N. Andsley et al. *Timeliness: Summary and Conclusions.* First-Year Report Task B PDCS, vol.2, cap.2, 1990, pp. 1-17.

[AHO 74] A.V. Aho; J.E. Hopcroft; J.D. Ulmann. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.

[AHO 88] A.V. Aho; R. Sethi; J.D. Ulmann. *Compilers, Principles, Techniques and Tools.* Addison-Wesley, Reading, Mass., 1988.

[AIKINS 83] J. Aikins. *Prototypical knowledge for expert systems.* Artificial Intelligence 20, 1983, pp. 163-210.

[ALBERT 91] L. Albert. *Présentation et évaluation de la complexité en moyenne des algorithmes d'unification.* Revue d'intelligence artificielle, vol.5, n.2, Hermes, 1991, pp.21-60.

[ALUR 91a] R. Alur; D. Dill. *The Theory of Timed Automata.* Lecturer Notes in Computer Science 600, pp.45-73, 1991.

[ALUR 91b] R. Alur; T.A. Herzinger. *Logics and Models of Real Time: A Survey* Lecturer Notes in Computer Science 600, pp.74-106, 1991.

[ARNOLD 90] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Techniques et Sciences Informatiques, vol.9, n.3, 1990, pp.193-216.

[AUDUREAU 89] E. Audureau; P. Enjalbert; L. Farinas del Cerro. *Logique Temporelle.* Masson, Paris, 1989, 244p.

[AUDUREAU 87] E. Audureau; L. Farinas del Cerro; P. Enjalbert. *Théorie de la programmation et logique temporelle: 1 ère Partie: Validation d'algorithmes séquentiels.* Techniques et Sciences Informatiques, vol.6, n.6, 1987, pp.527-540.

[AUDUREAU 88] E. Audureau; L. Farinas del Cerro; P. Enjalbert. *Théorie de la programmation et logique temporelle: 2 nde. Partie: Validation d'algorithmes parallèles.* Techniques et Sciences Informatiques, vol.7, n.2, 1988, pp.181-200.

[BARACHINI 88] F. Barachini; N. Theuretzbacher. *The Challange of Real-Time Process Control for Production Systems.* Proceedings of the AAAI 88, pp.705-709, 1988.

[BARR 86] A. Barr; E. Feigenbaun (eds). *The handbook of artificial intelligence.* vol.1, Addison-Wesley, 1986.

[BARR 86b] A. Barr; E. Feigenbaun (eds). *The handbook of artificial intelligence.* vol. 2, Addison-Wesley, 1986.

[BARR 8] A. Barr; E. Feigenbaun (eds). *The handbook of artificial intelligence.* vol.3, Addison-Wesley, 1986.

[BECK 90] T. Beck; R.J. Lauber. *Integration of an Expert System into a Real-Time Software System.* Proceedins of the 11 th World Congress on Automatic Control, IFAC, vol.7, pp.158-161, 1990.

[BENVENISTE 87] A. Benveniste; P. LeGuernic; C. Jacquemot. *The SIGNAL Software Environment for Real-Time System Specification, Design, and Implementation.* Rapport de Recherche INRIA/IRISA n.761, 1987.

[BENVENISTE 90] A. Benveniste; P. Le Guernic. *Hybrid Dynamical Systems Theory and the SIGNAL Language.* IEEE Transactions on Automatic Control, vol.35, n.5, pp.535-546, 1990.

[BENVENISTE 91a] A. Benveniste; G. Berry. Prolog to *The Special Section on Another Look at Real-Time Programming.* Proceedings of the IEEE, vol.79, n.9, Setembro 1991.

[BENVENISTE 91b] A. Benveniste; G. Berry. *The Synchronous Approach to Reactive and Real-Time Systems.* Proceedings of the IEEE, vol.79, n.9, pp.1270-1282, 1991.

[BENNETT 84] S. Bennett; D. Linkers (eds). *Real-Time Computer Control.* Peter Peregrinus, 1984.

[BENNETT 88] S. Bennett. *Real-Time Computer Control.* Prentice-Hall, 1988.

[BERNARD 89] R. Bernard et al. *A Quick Survey of the ESTEREL V3 System.* Rapport Technique INRIA, 1989.

[BERRY 83] G. Berry; S. Moisan; J.P. Rigault. *ESTEREL: Towards a Synchronous and Semantically Sound High-Level Language for Real-Time Applications.* Proceedings of the IEEE Real-Time Systems Simposium, pp.30-37, 1983.

[BERRY 86] G. Berry. *Finite automata and Regular Expressions.* Rapport Technique INRIA, 1986.

[BERRY 87] G. Berry; P. Couronne; G. Gonthier. *Programmation Synchrone des systèmes réactifs: le langage ESTEREL.* Techniques et Sciences Informatiques, vol.6, n.4, pp. 305-316, 1987.

[BERRY 89] G. Berry; G. Gonthier. *The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation.* Rapport Technique INRIA, 1989.

[BERRY 91] G. Berry. *A Hardware Implementation of Pure ESTEREL.* Paris Research Laboratory, Digital Equipment Corporation, 1991.

[BESTOUGEFF 89] H. Bestougeff; G. Ligozat. *Outils logiques pour le traitement du temps.* Masson, Paris, 282 p., 1989.

[BHATTACHARRYA 90] A. Bhattacharyya. *On Time and Real-Time Systems.* Private Repport PDCS York Group, 1990.

[BIBEL 82] W. Bibel. *Automated Theorem Proving.* Vieweg, Weisbaden, 292 p, 1982.

[BLAIR 88] H. Blair; V. Subrahmanian. *Paraconsistent logic programming.* 7. Int. Conf. on Foundations of Software Tech. & Theoret. Computer Science, 1988.

[BONISSONE 90] P.P. Bonissone; P.C. Halverson. *Time-Constrained Reasoning Under Incertainty.* Real-Time Systems, vol.2, n.1/2, pp.25-46, Maio 1990.

[BORGNE 89] M. Le Borgne; A. Benveniste; P. Le Guernic. *Polynomial Ideal Theory Methods in Discrete Event, and Hybrid Dynamical Systems.* Proceedings of the 28 th Conference on Decision and Control, Tampa, Flórida, Dezembro 1989.

[BOUSSINOT 91] F. Boussinot; R. de Simone. *The ESTEREL Language.* Proceedings of the IEEE, vol.79, n.9, pp.1293-1304, 1991.

[BOURNAI 91] P. Bournai; P. Le Guernic; V. Kerscaven. *Un environnement graphique pour le langage SIGNAL.* Rapport interne INRIA, 1991.

[BRONWSTON 86] L. Bronwston et all. *Programming expert systems in OPS5.* Addison-Wesley, 1986.

[BROWNE 86] M.C. Browne; E.M. Clarke; D.L. Dill; B. Mishra. *Automatic Verification of Sequential Circuits using Temporal Logic.* IEEE Transactions on Computers, vol.C-35, n.12, pp.1035-1044, 1986.

[BRUNESSAUX 92] S. Brunessaux; J.M. Valad. *Rapport Projet Prometheus.* Matra MS2i, 1992.

[BURRIS 81] S. Burris; H.P. Sankappanavar. *A Course in Universal Algebra.* Springer-Verlag, 275p., 1981.

[CASANOVA 87] M. Casanova; F. Giorgio; A. Furtado. *Programação em lógica e a linguagem PROLOG.* Edgard Blücher, 1987.

[CASPI 87] P. Caspi; N. Halbwachs; D. Pilaud; J. Plaice. *Le langage LUSTRE et sa sémantique operationnnelle.* Actes du 2 ème Colloque C3 d'Angoulême, pp. 81-98, 1987.

[CATACH 90] L. Catach. *Logiques non classiques. Présentation et applications à l'intelligence artificielle.* em *Modèles logiques et systèmes d'intelligence artificielle (coord. L. Iturrioz et A. Duchaussoy).* Hermes, Paris, 1990.

[CHARNIAK 86] E. Charniak; D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, 1986.

[CLARKE 91] E.M. Clarke Jr.; D.E. Long; K.L. McMillan. *A Language for Compositional Specification and Verification of Finite State Hardware Controllers.* Proceedings of the IEEE, vol.79, n.9, pp.1283-1292, 1991.

[COSTA 86] N.C.A. da Costa; W.A. Carnielli. *On Paraconsistent Deontic Logic.* Philosophia, Philosophical quarterly of Israel, vol.16, n.3-4, pp.293-305, 1986.

[COSTA 89] N.C.A. da Costa; V.S. Subrahmanian. *Paraconsistent Logics as a Formalism for Reasoning About Inconsistent Knowledge Bases.* Artificial Intelligence in Medicine vol.1, pp.167-174, 1989.

[COSTA 90] N.C.A. da Costa; L. Henschen; J. Lu; V.S. Subrahmanian. *Automatic Theorem Proving in Paraconsistent Logics: Theory and Implementation.* Coleção Documentos: Série Lógica e Teoria da Ciência, USP, Instituto de Estudos Avançados, maio 1990.

[CRUBILLE 88] P. Crubillé. *MEC: un outil de calcul sur les systèmes de transitions.* CFIP'88. in *Ingénierie des protocoles.* R. Castanet e O. Rafiq (eds), 1988.

[D'AMBROSIO 87] B. D'Ambrosio et all. *Real-Time Process Management for Materials Composition in Chemical Manufacturing* IEEE Expert, pp. 80-93, verão 1987.

[DASARATHY 85] B. Dasarathy. *Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them.* IEEE Transactions on Sotware Engineering, vol.SE 11, n. 1, pp. 80-86, 1985.

[DAVIS 77] R. Davis; J. King. *An Overview of Productions Systems.* Machine Intelligence 8, pp.300-332, 1977.

[DECKER 90] K.S. Decker; V.R. Lesser; R.C. Whitehair. *Extending a Blackboard Architecture for Approximate Processing.* Real-Time Systems, vol.2, n.1/2, pp.47-80, maio 1990.

[DeKLEER 79] J. De Kleer; J. Doyle; G.L. Steele; G.J. Sussman. *Explicit Control of Reasoning.* Artificial Intelligence - A MIT Perspective - Winston & Brown (eds), pp.93-118, 1979.

[DeKLEER 86a] J. De Kleer. *An Assuption-based TMS.* Artificial Intelligence 28, pp.127-162, 1986.

[DELPECH 87] P. Delpech; A. Potet; C. Sayettat. *Démonstration automatique et logiques temporelles.* Techniques et Sciences Informatiques, vol.6 , n.6, pp. 541-557, 1987.

[DEUTSCH 88] M.S. Deutsch. *Focusing Real-Time Systems Analysis on User Operations.* IEEE Software, pp.39-50, setembro 1988.

[DODHIAWALA 89] R. Dodhiawala; N.S. Sridharan; P. Raulefs; C. Pickering. *Real-Time AI Systems: A Definition and an Architecture.* Proceedings of the IJCAI 89, pp. 256-261, 1989.

[DOYLE 79] J. Doyle. *A Truth Maintenance System.* Artificial Intelligence 12, pp. 231-272, 1979.

[DOUGHERTY 88] E.R. Dougherty; C.R. Giardina. *Mathematical Methods for Artificial Intelligence and Autonomous Systems.* Prentice-Hall, Cambridge, 446p, 1988.

[DUBOIS 80] D. Dubois; H. Prade. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, Orlando, Flórida, 1980.

[DUBOIS 88] D. Dubois; H. Prade. *An Introduction to Possibilistic and Fuzzy Logics.* P.Smets Editor, Non-Standard Logics for Automated Reasoning, Academic Press, 1988.

[DUFRESNE 84] P. Dufresne. *PSC: an Efficient Production Rule Interpreter.* ECAI 84, pp.525-528, setembro 1984.

[DURFEE 88] E.H. Durfee; V.R. Lesser. *Incremental Planning to Control Time-Constrained Blackboard-Based Problem Solver.* IEEE Transactions on Aerospace and Electronic Systems, vol.24, n.5, pp. 647-662, 1988.

[DUTERTRE 91] B. Dutertre; P. LeGuernic. *Description et simulation d'un système de contrôle de passage à niveau en SIGNAL.* Publication interne n.580, IRISA, INRIA, 1991.

[DZIERGOWSKI 90] D. Dziergowski. *Quatre exemples de langages ou environnements pour le développement de programmes où le temps intervient.* Techniques et Sciences Informatiques, vol.9, n.4 , pp. 289-312, 1990.

[ERMAN 81] L. Erman; P. London; S. Fickas. *The design and an example use of the Hearsay-III.* Proceedings of the 7. IJCAI, pp. 409-415, Vancouver, 1981.

[ESCALADA 91] G. Escalada-Imaz. *Optimisation d'Algorithmes d'Inférence Monotone en Logique des Propositions et du Premier Ordre.* Tese de Doutorado da Universidade Paul Sabatier de Toulouse, Rapport LAAS n.89204, julho 1989.

[ETHERINGTON 87] D.W. Etherington. *Formalizing Nonmonotonic Reasoning Systems.* Artificial Intelligence 31, pp. 41-85, 1987.

[FARRENY 87] H. Farreny; M. Ghallab. *Eléménts d'intelligence artificielle.* Hermes, Paris, 1987.

[FILMAN 88] R.E. Filman. *Reasoning with Worlds and Truth Maintenance in a Knowlegde-based Programming Environment.* Communications of the ACM, vol.31, pp.382-401, 1988.

[FIKES 85] R. Fikes; T. Kehler. *The Role of Frame-Based Representation in Reasoning.* Communications of the ACM, vol.28, pp.904-920, 1985.

[FORGY 82] C.L. Forgy. *Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern-Match Problem.* Artificial Intelligence 19, pp.17-37, 1982.

[GARNOUSSET 88] H.E. Garnousset; C.A.A. Kaestner. *SP1 : motor de inferência para sistemas de regras de produção.* Anais do 5 o. Simpósio Brasileiro de Inteligência Artificial, Natal, 1988.

[GAUTIER 87] T. Gautier; P. Le Guernic; L. Bernard. *SIGNAL: a Declarative Language for Synchronous Programming of Real-Time Systems.* Rapport de Recherche INRIA/IRISA n.761, 1987.

[GEORGEFF 82] M.P. Georgeff. *Procedural control in production systems.* Artificial Intelligence 18, pp.175-201, 1982.

[GHALLAB 81] M. Ghallab. *Decision trees for optimizing pattern-matching algorithms in production systems.* Proc. 7o. IJCAI, 310-312, Vancouver, 1981.

[GHALLAB 82] M. Ghallab. *Optimisation de Processus Décisionnels pour la Robotique.* Tese de Doutorado de Estado da Universidade Paul Sabatier de Toulouse, 1982.

[GHALLAB 84] M. Ghallab; P. Dufresne. *Moteurs d'Inférence pour Systèmes de Règles de Production: Techniques de Compilations et d'Interprétation.* Colóquio Internacional de Inteligência Artificial, Marseille, pp.89-103, 1984.

[GHALLAB 86] M. Ghallab. *Environnement de Developpement d'un Système de Diagnostic Embarqué: Projet SIAD.* Rapport LAAS n.86312, outubro 1986.

[GHALLAB 88] M. Ghallab. *Compilation de bases de connaissances.* Actes des Journees Nationales, PRC - Greco, pp.231-253, Toulouse, 1988.

[GHALLAB 88b] M. Ghallab; H. Phillipe. *A Compiler for Real-Time Knowledge-Based Systems.* International Workshop on Artificial Intelligence for Industrial Applications, Hitachi, pp.387-393, maio 1988.

[GHALLAB 93] M. Ghallab; F.F. Ingrand; C.A.A. Kaestner. *Programming Paradigms for Real-Time AI Systems.* Em preparação.

[GIUNCHIGLIA 89] F. Giunchiglia; T. Walsh. *Abstract Theorem Proving.* Proceedings of the IJCAI 89, pp.372-377, 1989.

[GOCHET 90] P. Gochet; P. Gribomont. *Logique - vol. 1: methodes pour l'informatique fondamentalle.* Hermes, Paris, 466p., 1990.

[GUIDA 84] G. Guida; C. Tasso. *A New Approach to the Expert System Architectures.* Artificial Intelligence and Information Control Systems of Robots, Elsevier Science Publishers B.V. (North-Holland), 1984.

[GUPTA 89] A. Gupta; C. Forgy; A. Newell. *High-Speed Implementations of Rule-Based Systems.* ACM Transactions on Computer Systems, vol.7, n.2, pp. 119-146, maio 1989.

[HALBWACHS 89] N. Halbwachs; D. Pilaud; F. Ouabdesselam. *Specifying, Programming and Verifying Real-Time Systems Using a Synchronous Declarative Language.* Lecture Notes in Computer Science 407, pp. 213-231, 1989.

[HALBWACHS 91] N. Halbwachs; P. Caspi; P. Raymond; D. Pilaud. *The Synchronous Data Flow Programming Language LUSTRE.* Proceedings of the IEEE, vol.79, n.9, pp.1305-1320, 1991.

[HAMILTON 78] A.G. Hamilton. *Logic for Mathematicians.* Cambridge University Press, 1978.

[HAREL 85] D. Harel; A. Pnueli. *On the Development of Reactive Systems: Logic and Models of Concurrent Systems.* Proceedings NATO Advanced Study Institute on Logics and Models for Verification and Specification on Concurrent Systems, NATO ASI Series F, vol.13, Springer Verlag, pp. 477-498, 1985.

[HAYES-ROTH 75] F. Hayes-Roth; D. Mostow. *An automatically compilable recognition network for structured patterns.* Proceedings of the 4. IJCAI, pp.246-251, Tbilissi, 1975.

[HAYES-ROTH 83] F. Hayes-Roth; D. Waterman; D. Lenat (eds). *Building expert systems.* Addison-Wesley, 1983.

[HAYES-ROTH 90] B. Hayes-Roth. *Architectural Foundations for Real-Time Performance in Intelligent Agents.* Real-Time Systems, vol.2, n.1/2, pp. 99-126, 1990.

[HERZINGER 91] T. Herzinger; Z. Manna; A. Pnueli. *Timed Transition Systems.* Lecturer Notes in Computer Science 600, pp.226-251, 1991.

[HOARE 78] C.A.R. Hoare. *Communicating Sequential Processes.* Communications of the ACM, vol.21, n.8, pp.666-677, agosto 1978.

[HOPCROFT 69] J.E. Hopcroft; J.D. Ulmann. *Formal Languages and their Relation to Automata.* Addison-Wesley, Reading, Massachussets, 242p., 1969.

[HOPCROFT 79] J.E. Hopcroft; J.D. Ulmann. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979.

[HORWITZ 89] E.J. Horwitz; G.F. Cooper; D.E. Heckerman. *Reflection and Action Under Scarce Resources: Theoretical Principles and Empirical Study.* Proceedings of the IJCAI 89, pp.1121-1127, 1989.

[HOWE 90] A.E. Howe; D.M. Hart; P.R. Cohen. *Addressing Real-Time Constraints in Design of Autonomous Agents.* Real-Time Systems, vol.2, n.1/2, pp.81-98, maio 1990.

[HSU 87] C.C. Hsu; S.M. WU; J.J. WU. *A Distributed Approach for Infering Production Systems.* Proceedings of the IJCAI 87, pp.62-67, 1987.

[HUGHES 68] G. Hughes; M. Cresswell. *An Introduction to Modal Logic.* Methuen, London, 1968.

[IPPOLITO 90] S. Ippolito; H. Phillipe. *Aide à la Conduite d'un Processus Industriel au Moyen d'un Systèmes Expert.* RAIRO - APII, vol.24, n.3, pp.189-214, 1990.

[ITURRIOZ 90] L. Iturrioz. *Logiques multivaluées: un outil souple et pluriel.* in *Modèles Logiques et systèmes d'intelligence artificielle (coord. L. Iturrioz et A. Duchaussoy).* Hermes, Paris, 1990.

[IWASAKI 89] Y. Iwasaki. *Qualitative Physics.* in A. Barr; E. Feigenbaun (eds). *The handbook of artificial intelligence.* vol.4, Addison-Wesley, 1986.

[JAHANIAN 86] F. Jahanian; A. Mok. *Safety Analysis of Timing Properties in Real-Time Systems.* IEEE Transactions on Software Engineering, vol.SE 12, n.9, pp. 890-904, 1986.

[JAKOB 90] F. Jakob; P. Sulenschi. *Situation Assessment for Process Control.* IEEE Expert, pp. 49-59, 1990.

[KAELBLING 88] L.P. Kaelbling; N.J. Wilson. *Rex Programmer's Manual.* SRI International, 1988.

[KAESTNER 89] C.A.A. Kaestner. *Contribuição ao Estudo e Desenvolvimento de um Sistema de Regras de Produção.* Dissertação de Mestrado, Departamento de Engenharia Elétrica, Universidade Federal de Santa Catarina, 1989.

[KAESTNER 90a] C.A.A. Kaestner. *Fuz-0+: Sistema de Produção de Ordem 0+ Usando Técnicas de Raciocínio Aproximado.* Anais do 7o. Simpósio Brasileiro de Inteligência Artificial, Campina Grande - PB, novembro 1990.

[KAESTNER 90b] C.A.A. Kaestner; H.E. Garnousset. *Um Algoritmo de Unificação Eficiente para Sistemas de Regras de Produção* Anais do 1. Simpósio de Automação Integrada CEFET-PR, pp.286-290, 1990.

[KAESTNER 91] C.A.A. Kaestner. *Uma Proposta de Sistema Baseado no Conhecimento para Aplicações Tempo-Real.* Monografia submetida ao exame de qualificação para o Doutorado, Departamento de Engenharia Elétrica, Universidade Federal de Santa Catarina, 102 p, 1991.

[KAESTNER 92a] C.A.A. Kaestner. *Systèmes IA Temps-Réel: l'Apport de l'Approche Synchrone aux Systèmes à Base de Règles.* Rapport de Recherche LAAS n. 92221, 59 p., junho 1992.

[KAESTNER 92b] C.A.A. Kaestner. *Sistemas IA Tempo-real: O Enfoque Síncrono em Sistemas à Base de Regras.* Anais do IX Simpósio Brasileiro de IA, pp 94-106, 1992.

[KAESTNER 93] C.A.A. Kaestner. *Formalismo baseado em Lógica Paraconsistente para aplicação em sistemas de regras.* Em preparação.

[KING 86] R. King; F. Karonis. *Rule-based systems in the Process Industry.* Proc. of the 25. Conference on Decision and Control, Atenas, pp. 622-626, 1986.

[KOHAVI 78] Z. Kohavi. *Switching and Finite Automata Theory.* Tata McGraw-Hill Publishing Co. Ltd., New Delhi, 1978.

[KONOLIGE 79] K. Konolige. *An inference net compiler for the Prospector rule-based consultation system.* Proceedings of the 6. IJCAI, pp.487-489, Tokyo, 1979.

[KOPETZ 90] H. Kopetz. *Fault Tolerance in Real-Time Systems.* Proceedings of the 11 th. World Congress on Automatic Control - IFAC, vol.7, pp.111-118, 1990.

[KOPETZ 92] H. Kopetz. *An advanced course on distributed systems.* Estoril, Portugal, -1992.

[KUENG-CHI 90] N. Kueng-Chi; B. Abramson. *Uncertainty Management in Expert Systems.* IEEE Expert, pp.29-48, abril 1990.

[KUIPERS 86] B. Kuipers. *Qualitative Simulation.* Artificial Inteligence 29 (3), pp.289-338, 1986.

[KUO 85] B.C. Kuo. *Sistemas de Controle Automático.* Prentice-Hall do Brasil, 646p., 1985.

[LAFFEY 88] T.J. Laffey et all. *Real-Time Knowledge Based Systems.* AI Magazine, vol.9, n.1, pp.27-45, 1988.

[LALEMENT 90] R. Lalement. *Logique, Reduction, Resolution.* Masson, Paris, 384p, 1990.

[LARK 90] J.S. Lark et all. *Concepts, Methods, and Languages for Building Timely Intelligent Systems.* Real-Time Systems, vol.2, n.1/2, pp.127-148, maio 1990.

[LAURENT 84] J.P. Laurent. *La structure de contrôle dans les systemes experts.* Techniques et Sciences Informatiques, vol.3, n.3, 1984.

[LeGUERNIC 90] P. Le Guernic; T. Gautier. *Data-flow to von Neumann: The SIGNAL Approach.* Rapport de Recherche INRIA n.1229, maio 1990.

[LeGUERNIC 91] P. Le Guernic; T. Gautier; M. Le Borgne; C. Le Maire. *Programming Real-Time Applications with SIGNAL.* Proceedings of the IEEE, vol.79, n.9, pp.1321-1336, 1991.

[LEINWEBER 87] D. Leinweber. *Experts Systems in Space.* IEEE Expert 2 (1), pp. 26-36, 1987.

[LESSER 88] V.R. Lesser; J. Pavlin; E. Durfee. *Approximate Processing in Real-Time Problem Solving.* AI Magazine, pp. 49-61, 1988.

[LEVI 90] S.T. Levi; A. Agrawala. *Real-Time System Design.* McGraw-Hill, 1990.

[LEWIS 81] H.R. Lewis; C.H. Papadimitriou. *Elements of the theory of computation.* Prentice-Hall, 466p., 1981.

[MALER 91] O. Maler; Z. Manna; A. Pnueli. *From Timed to Hybrid Systems.* Lecturer Notes in Computer Science 600, pp.447-484, 1991.

[McDERMOTT 78] D. McDermott; A. Newell; J. Moore. *The Efficiency of Certain Production Inference Systems.* Pattern-Directed Inference Systems (D. Waterman e F. Hayes-Roth eds), Academic Press, 1978.

[MENDELSON 79] E. Mendelson. *Introduction to mathematical logic.* Litton E.P., 2 ed., 1979.

[MICHEL 84] M. Michel. *Algebre de Machines et Logique Temporelle.* STACS 1984, Lecturer Notes in Computer Science vol.166, 1984.

[MILNER 83] R. Milner. *Calculi for Synchrony and Assynchrony.* Theoretical Computer Science 25, pp.267-310, 1983.

[MILNER 89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MIRANKER 87] D.P. Miranker. *TREAT: A Better Algorithm for AI Production Systems.* Proceedings of the AAAI 87, pp.42-47, 1987.

[MOK 84] A. Mok. *The Design of Real-Time Programming Systems Based on Process Models.* Proceedings of the IEEE Symposium on Real-Time Programming, pp.5-17, 1984.

[MOK 90] A. Mok. *Real-Time Logic, Programming and Scheduling.* Proceedings of the Joint University of Newcastle Upon Tyne / International Computers Limited Seminar, pp. V1-V27, 1989.

[MOORE 87] R.L. Moore et all. *Experts Systems Methodology for Real-Time Process Control.* Proceedings of the 10. World Congress on Automatic Control, IFAC, pp. 274-281, 1987.

[MOORE 90] R.L. Moore; H. Rosenof; G. Stanley. *Process Control Using a Real-Time Expert System.* Proceedings of the 11. World Congress on Automatic Contro, IFAC, pp.234-239, 1990.

[NEF 90] F. Nef. *Quelques systèmes de logique temporelle propositionnelle.* in *Modèles Logiques et systèmes d'intelligence artificielle (coord. L. Iturrioz et A. Duchaussoy).* Hermes, Paris, 1990.

[NICOLLIN 91] X. Nicollin; J. Sifakis; S. Yovine. *From ATP to Times Graphs and Hybrid Systems.* Lecturer Notes in Computer Science 600, pp.549-572, 1991.

[NILSSON 80] N. Nilsson. *Principles of artificial intelligence.* Tioga Publishing, 1980.

[PERKINS 90] W.A. Perkins; A. Austin. *Adding Temporal Reasoning to Expert-Systems Building Environments.* IEEE Expert, pp.23-30, fevereiro 1990.

[OGATA 82] K. Ogata. *Engenharia de Controle Moderno.* Prentice-Hall do Brasil, 929p., 1982.

[OSTROFF 89] J. Ostroff. *Temporal Logic for Real-Time Systems.* John Wiley & Sons, 1989.

[OSTROFF 91] J. Ostroff. *Verification of Safety Critical Systems Using TTM/RTTL.* Lecturer Notes in Computer Science 600, pp.573-602, 1991.

[PHILLIPE 89] H. Phillipe. *Algorithmes pour la Compilation de Bases de Connaissances en Logique Propositionnelle et du Premier Ordre: les Systèmes KHEOPS et CLOPS.* Tese de Doutorado da Universidade Paul Sabatier de Toulouse, Rapport LAAS n.89122, maio de 1989.

[PHILLIPE 91] H. Phillipe; H. Rogary. *Système Expert dedié à la Validation des Résultats d'Analyses Médicales.* Congrès Européen de l'Informatique Hospitalière, CNIT, Paris, junho 1991.

[PILAUD 88] D. Pilaud; N. Halbwachs. *From a Synchronous Declarative Language to a Temporal Logic Dealing with Multiform Time.* Proceedings of the Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, Warwich, pp.99-110, setembro 1988.

[PLOTKIN 81] G.D. Plotkin. *A Structural Approach to Operational Semantics.* Lectures Notes, Aarhus University, 1981.

[POMEROY 90] B. Pomeroy; R. Irving. *A Blackboard Approach for Diagnosis in Pilot's Associate* IEEE Expert, pp. 39-46, agosto 1990.

[RAMADGE 87] P.J. Ramadge; W.M. Wonham. *Supervisory Control of a Class of Discret Events Processes.* SIAM Journal of Control and Optimization, vol.25, n.1, pp.206-230, 1987.

[REITER 80] R. Reiter. *A Logic for Default Reasoning.* Artificial Intelligence 13, pp. 81-132, 1980.

[RICH 83] E. Rich. *Artificial intelligence*, McGraw-Hill, 1983.

[SCHAEFER 86] P.R. Schaefer; I.H. Bozma; R.D. Beer. *Knowlegde-based Validity Maintenance for Production Systems.* Proceedings of AAAI-86, pp. 918-922, 1986.

[SHORTLIFFE 76] E.H. Shortliffe. *Computer-based medical consultations Mycin.* American Elsevier, 1976.

[SILVA 92] B.I. da Silva Jr. *Lógica Temporal de Tempo-Real Generalizada Aplicada ao Controle e Simulação de Sistemas Dinâmicos a Eventos Discretos.* Dissertação de Mestrado, Faculdade de Engenharia Elétrica, Universidade Estadual de Campinas, julho de 1992.

[SOUZA 92] L.F.P. de Souza; S.R.M. Veloso. *Uma automação de lógica temporal.* Anais do IX Simpósio Brasileiro de Inteligência Artificial, pp.257-269, 1992.

[STANKOVIC 88] J.A. Stankovic. *A Serious Problem for Next-Generation Systems.* IEEE Computer, pp.10-19, 1988.

[STEELE 90] G.L. Steele. *CommonLisp: The Language.* Digital Press, 2 ed., 1990.

[STEYER 91] J.P. Steyer. *Sur une Approche Qualitative des Systèmes Physiques: Aide en Temps-Réel à la Conduite des Procedés Fermentaires.* Tese de Doutorado da Universidade Paul Sabatier de Toulouse, Rapport LAAS n.91445, dezembro 1991.

[SZASZ 71] G. Szász. *Théorie des treillis.* Dunod, 1971.

[TALA 90] J.E. Tala; M.R. Souza. *Um Estudo Algébrico da Lógica Temporal Linear de Programas.* Anais do 7 o. Simpósio Brasileiro de Inteligência Artificial, pp.255-264, 1990.

[TANO 88] S. Tano; S. Masui; T. Nakano; K. Mori. *Eureka-II: A Programming Tool for Knowledge-Based Real Time Control Systems.* Proceedings of the IEEE International Workshop on AI for Industrial Applications, pp. 370-375, 1988.

[TAUB 84] H. Taub. *Circuitos Digitais e Microprocessadores.* McGraw-Hill, 510p., 1984.

[TAYLOR 87] J. Taylor; R. Gerhardt; E. Luce. *Rule-Based Real-Time Control Systems.* Proceedings of the 26. Conference on Decision and Control, pp.1923-1928, Los Angeles, 1987.

[TEPANDI 90] J. Tepandi. *Verification, Testing and Validation of Rule-Based Expert Systems.* 11. World Congress on Automatic Control, IFAC, vol.7, pp.162-167, 1990.

[THAYSE 89] A. Thayse et all. *Approche logique de l'intelligence artificielle - 2. De la logique modale a la logique des bases de donnes.* Dunod-Informatique, Paris, 427 p., 1989

[THAYSE 90] A. Thayse et all. *Approche logique de l'intelligence artificielle - 1. De la logique classique a la programmation logique.* Dunod-Informatique, Paris, 386 p, 1990.

[TOGAI 86] M. Togai; H. Watanabe. *Expert System on a Chip: An Engine for Real-Time Approximate Reasoning.* IEEE Expert, pp.55-62, 1986.

[TRAVE-MASSUYES 90] L. Travé-Massuyès; A. Missler; N. Piera. *Qualitative Models for Automatic Control Process Supervision.* 11. World Congress on Automatic Control, IFAC, vol.7, pp. 198-203, 1990.

[TURNER 86] R. Turner. *Logiques pour l'Intelligence Artificielle.* Masson, Paris, 1986.

[VERGAMINI 86] D. Vergamini. *Verification by means of Observational Equivalence on Automata.* INRIA Repport n.501, 1986.

[VERNADAT 86] F. Vernadat. *Verification formelle d'applications reparties caracterisation logique d'une equivalence de comportement.* Tese de doutorado da Universidade Paul Sabatier de Toulouse, 1989.

[WALLEN 87] L.A. Wallen. *Matrix Proof Methods for Modal Logics.* Proceedings of the IJCAI 87, Milan, pp.917-923, 1987.

[WASHINGTON 89] R. Washington; B. Hayes-Roth. *Input Data Management in Real-time AI Systems.* Proceedins of the IJCAI 89, pp.250-255, 1989.

[WATERMAN 86] D.A. Waterman. *A Guide to Expert Systems.* Addison-Wesley, 1986.

[WRIGHT 86] M.L. Wright; M.W. Green; G. Fiegl; P.F. Cross. *An Expert System for Real-Time Control.* IEEE Software, pp.16-24, 1986.

[XUONG 92] N.H. Xuong. *Mathematiques Discretes et Informatique.* Masson, Paris, 432 p., 1992.

[YOUNG 82] S.J. Young. *Real-Time Languages.* Ellis Horwood Publishers, 1982.

[ZADEH 78] L.A. Zadeh. *Fuzzy Sets as a Basis for a Theory of Possibility.* Fuzzy Sets and Systems 1, pp.3-28, 1978.

[ZIMMERMANN 85] H.J. Zimmermann. *Fuzzy Set Theory and Its Applications.* Kluwer-Nijhoff Publishing, 1985.