

Contribuição ao Uso da Lógica Temporal na Especificação de Comportamento de Sistemas a Eventos Discretos

Eduard Montgomery Meira Costa

Dissertação de Mestrado submetida à Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Mestre em Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Antônio Marcus Nogueira Lima, Dr.
Orientador

Campina Grande, Paraíba, Brasil
©Eduard Montgomery Meira Costa, Setembro de 1997

Contribuição ao Uso da Lógica Temporal na Especificação de Comportamento de Sistemas a Eventos Discretos

Eduard Montgomery Meira Costa

Dissertação de Mestrado apresentada em Setembro de 1997

Antônio Marcus Nogueira Lima, Dr.
Orientador

Angelo Perkusich, DSc.
Componente da Banca
Rafael Santos Mendes, Dr.
Componente da Banca

Campina Grande, Paraíba, Brasil, Setembro de 1997

Dedicatória

Dedico este trabalho aos amantes da filosofia e das ciências, que buscam nos pensamentos abstratos, concretos e na pesquisa sua realização pessoal. Aos autodidatas, que através de seus sentidos e dedicação aos estudos, ampliam seus conhecimentos, engrandecendo-se. Aos críticos, que não se deixam levar facilmente pelas afirmações do cotidiano. Aos criativos, que em tudo, encontram uma nova forma de resolver os problemas do dia a dia. Aos ficcionistas e sonhadores, que idealizam concepções científicas na realidade do infinito universo quadridimensional.

Também dedico, a todas aquelas pessoas que direta ou indiretamente colaboraram com esta, e a todos os meus amigos que conviveram comigo durante esta realização, em horas de dificuldades e de alegrias. Aos meus familiares e, em especial, aos meus filhos Ariadne Messalina e Ícaro Gilgamesh.

* * * * *

"Tudo é simbólico e transcendental nesta gigantesca epopéia dos destinos humanos."

Eliphas Levi

* * * * *

"A coisa mais bela que o homem pode experimentar é o misterioso. É esta a emoção fundamental que está na raiz de toda ciência e arte."

Albert Einstein

* * * * *

Agradecimentos

Agradeço ao professor Antônio Marcus Nogueira Lima pela sua orientação e colaboração, sem as quais não seria possível a realização deste trabalho. A Giovanni Cordeiro Barroso, pela sua colaboração e co-orientação inicial, que me iniciou neste projeto. A todos os meus amigos e amigas que me ajudaram direta ou indiretamente.

Ao CNPq, que proporcionou o suporte financeiro para viabilizar a realização deste trabalho.

Resumo

As redes de Petri têm se tornado uma ferramenta de grande utilidade para modelagem e análise de Sistemas a Eventos Discretos. O uso da lógica temporal para a especificação de comportamentos de modelos de SEDs é, aqui, introduzido, desde que esta é uma ferramenta de alta abrangência, substituindo a utilização das linguagens formais, na busca da solução da síntese do supervisor. A principal classe de redes de Petri utilizada para a modelagem neste trabalho, são as Redes de Petri com Função de Habilitação de Transições, que apresentam funções lógicas em suas transições, que geram uma seqüência desejada, para servir como supervisor, o qual deve possuir a mesma estrutura do modelo do sistema a ser controlado.

Abstract

Petri nets are a tool of great utility for modeling and analysis of Discrete Event Systems (DES). The use of the temporal logic to the behavior's specification of DES's models is, here, introduced, since that is a high cover tool, substituting the utilization of the formal languages, in the search of the supervisor's synthesis. The main class of Petri nets used to the supervisor's modeling in this work, are the Petri nets with Transitions Enabling Function, that presents logic functions in its transitions, that generate a desired sequence, to serve as supervisor, that must have the same structure of system's model to be controlled.

Sumário

1	Introdução	1
1.1	Objetivos e Apresentação do Trabalho	10
2	Conceitos Básicos	12
2.1	Sistemas Dinâmicos a Variáveis Contínuas	12
2.2	Sistema Dinâmicos a Variáveis Discretas	13
2.3	Sistemas Dinâmicos a Eventos Discretos	14
2.4	Linguagens Formais	15
2.5	Autômatos	18
2.6	Geradores	20
2.7	Redes de Petri	23
2.8	Redes de Petri com Função de Habilitação de Transições	26
2.9	Linguagens de Redes de Petri	28
2.10	Considerações Gerais	29
3	Teoria de Controle Supervisório	31
3.1	Supervisores e Condições de Existência	32
3.2	Abordagem para Síntese de Supervisores por Redes de Petri	37
4	Redes de Petri e Lógica Temporal	41
4.1	Introdução	41
4.2	Lógica Temporal no Contexto das Redes de Petri	43
5	Apresentação dos Algoritmos e Exemplos	55
5.1	Algoritmo Modificado da Árvore de Alcançabilidade	55
5.2	Algoritmo para a Construção do Gerador da SupC(L)	56
5.3	Algoritmo do Parser para Lógica Temporal	62
5.4	Exemplos da Utilização dos Algoritmos	65

6	Considerações Finais	73
6.1	Discussões e Conclusões	73
6.2	Trabalhos Futuros	74

Lista de Figuras

1.1	Trajectoria de um SED	4
1.2	Estrutura de Controle de um SED	6
2.1	Trajectoria típica de um SDVC	13
2.2	Trajectoria típica de um SDVD, comparado a um SDVC	14
2.3	Exemplo de um Autômato	20
2.4	Exemplo de uma Rede de Petri antes e após o disparo de sua transição	25
2.5	Exemplo de uma RPFHT	28
2.6	Exemplo de uma rede de Petri que gera a linguagem $(\alpha\beta)^*$	30
3.1	Supervisão de um SED	33
3.2	Utilização de redes de Petri e teoria de controle Supervisório para a concepção, análise e controle de um SED	39
3.3	Diagrama em blocos do procedimento de síntese do supervisor	39
4.1	Rede de Petri utilizada no exemplo 3.1	49
4.2	Árvore de alcançabilidade da RP, do exemplo 3.1.	50
4.3	Sistema de Transmissão/Recepção modelado por Rede de Petri	52
4.4	Árvore de alcançabilidade da RP, para o sistema Transmissão/Recepção	52
5.1	Modelo do sistema produtor/consumidor via RP	57
5.2	Árvore de alcançabilidade do sistema produtor/consumidor modelado por RP, com utilização do AMArA	58
5.3	Exemplo da construção da supC(L) pelo ACGS	61
5.4	Simple rede de Petri usada no exemplo do uso do parser	64
5.5	Árvore de Alcançabilidade da rede de Petri usada no exemplo do uso do parser	64
5.6	Modelo em Rede de Petri para um Sistema de Manufatura com Recursos Compartilhados	66

5.7	Modelo em RPFHT para um Sistema de Manufatura com Recursos Compartilhados, para realizar o processamento de uma única peça por vez .	67
5.8	Modelo em RPFHT para um Sistema de Manufatura com Recursos Compartilhados, para realizar o processamento de duas peças em paralelo .	68
5.9	Sistema do exemplo, modelado por rede de Petri	70
5.10	Sistema do exemplo, modelado por rede de Petri, eliminando o problema do acionamento do motor A, por intermédio da modificação da rede de Petri	71
5.11	Sistema do exemplo, modelado por RPFHT, eliminando o problema do acionamento do motor A.	72

Capítulo 1

Introdução

A teoria de Controle estuda os sistemas físicos dinâmicos, os quais são descritos por sistemas de equações diferenciais ordinárias e parciais, havendo vários estudos acerca destes, abrangendo os mais variados tipos de controladores desenvolvidos até o momento atual: proporcional, proporcional integral (pi), proporcional diferencial (pd), proporcional integral derivativo (pid), robusto e adaptativo. O grande problema encontrado hoje em dia, está no estudo dos sistemas dinâmicos feitos pelo homem (man-made dynamic systems) os quais têm por denominação Sistemas Dinâmicos a Eventos Discretos, ou simplesmente, Sistemas a Eventos Discretos (SED). Estes sistemas estão presentes em muitas aplicações do cotidiano, como: redes de computadores, sistemas de manufatura, supervisão de tráfego aéreo e ferroviário, sistemas operacionais, [Lib96][Bar96][RW89][KH96][SBS92]. Estes sistemas não são descritos pelas equações usuais da teoria de controle (equações diferenciais ordinárias e parciais) facilmente [Ho89], tendo vários paradigmas que podem ser usados na sua modelagem [Ho89][Bar96]. Embora nenhum tenha se tornado universal, são utilizados para aplicações específicas. São eles:

- Cadeias de Markov, as quais se adequam a modelagem de sistemas onde interessam as probabilidades de ocorrência dos eventos. Este paradigma é utilizado para modelar um sistema que se caracteriza por um conjunto enumerável de estados em que seu comportamento é observável em vários instantes de tempo, num dado momento qualquer. O comportamento futuro do sistema é influenciado apenas pelo estado atual. Logo, em um dado sistema, podemos representá-lo por uma Cadeia de Markov se forem conhecidas as probabilidades de transição de estado, como podemos ver em Cao e Ho [CH90];
- Teoria das filas, que são utilizadas particularmente em sistemas de fluxo, onde nestes, objetos (ou informações) são transportados por um ou mais canais, os

quais têm capacidade finita. Devido a esta limitação de capacidade, são formadas filas de espera pela sua liberação. Torna-se necessário o conhecimento processual de chegada de solicitação de transporte, além da forma como são atendidas. Isto gera vários tipos de fila, de acordo com a representação do sistema. Este tipo de abordagem aplica-se a análise de desempenho de SEDs. Esta aplicação é encontrada em Cao e Ho [CH90];

- Processos semi-Markovianos generalizados, que formaliza programas e linguagens de simulação de SEDs, em que as partes discretas de um sistema são representados por estados, por exemplo, a quantidade de recursos disponíveis, e as partes contínuas são chamados de tempo de vida dos eventos. Estas especificam o tempo decorrido desde a mudança para um estado, até a próxima ocorrência de um outro evento que modifique o estado atual do sistema. O sistema evolui no tempo por um processo repetitivo dado pelas ocorrências de eventos, formando uma seqüência [CH90];
- Álgebra de processos, a qual, a partir de um conjunto de eventos seqüenciais de um SED, denominado de processo, e de um conjunto de equações sobre os processos que permitam uma representação de sua composição, podemos escrever expressões algébricas envolvendo os processos, além de demonstrar identidades entre si, utilizando-nos do emprego destas operações. A álgebra de processos é formulada por um conjunto de operadores, axiomas e identidades entre tais expressões como desenvolvida por Milner [Mil80];
- Álgebra Max-Plus, que foi proposta por Cohen *et al* [GCV85] está baseada em uma estrutura algébrica constituída de um conjunto e duas operações definidas sobre o mesmo. Neste tipo de estrutura, as propriedades algébricas são úteis para descrever os SEDs. Particularmente, os SEDs cujo comportamento envolve um conjunto de atividades repetitivas, pode ser caracterizado a partir da solução de uma determinada equação matricial escrita nesta álgebra;
- Teoria de linguagens formais e autômatos, cuja aplicação é bastante ampla na modelagem dos SEDs [RW89]. Apresentam o problema de que a dimensão do espaço de estado cresce exponencialmente com o número de componentes do SED (explosão de estados), tornando sua representação gráfica complexa e reduzindo sua visualização e compreensão;
- Redes de Petri (RP) [Mur89] [Pet81a] [Rei85] [Rei92], as quais têm tido um rápido desenvolvimento teórico e na utilização na modelagem de SEDs. Estas são uma

ferramenta de alto poder matemático e gráfico, que facilita a visualização de todo o modelo do sistema e seu estado em qualquer instante de tempo. A diferença destas para as linguagens formais e os autômatos, dá-se em que, o aumento de componentes em um SED está diretamente ligado ao aumento linear de seus lugares, transições e arcos.

Os SEDs têm uma diferença dada pela comunhão da forma com que os mesmos percebem as ocorrências do ambiente, as quais denominam-se eventos, em que tais acontecimentos, causam mudanças na configuração interna, ou estado do sistema. Tais eventos ocorrem instantaneamente, o que lhes dão caráter discreto no tempo. Logo, o estado do sistema só é mudado após uma nova ocorrência de um determinado evento. Exemplo disto, é a transferência de dados em um sistema de comunicação, onde esperamos uma ordem para enviar uma mensagem e, após seu envio esperamos uma outra mensagem de retorno devolvida pela outra máquina avisando de seu recebimento. Observemos que não há, neste caso, tempo determinado para haver o envio da mensagem, e o tempo decorrido na transferência dos dados entre as máquinas, recepção e retorno, dependem de fatores externos. Sendo assim, as máquinas permanecem no mesmo estado, até que ocorra algum destes eventos, os quais irão mudar o estado do sistema.

Levamos em consideração que estes sistemas interagem com o mundo externo, sendo limitados, através de suas fronteiras. Os mesmos têm como características fundamentais:

- Ciclo de funcionamento descrito através do encadeamento de eventos;
- Ocorrência de eventos em paralelo;
- Necessidade de sincronização.

Os SEDs têm sua evolução no tempo dada pela ocorrência dos eventos, os quais podem ser internos ou externos, além do que, a ocorrência de eventos pode depender de fatores alheios ao sistema definidos como eventos indesejáveis (a falta de energia elétrica, a quebra de uma peça ou um erro de transferência de dados são típicos exemplos disto).

Definimos formalmente um SED, como em Ramadge e Wonham [RW89]:

Definição 1.1 *Um Sistema a Eventos Discretos (SED) é um sistema que evolui com a ocorrência abrupta de eventos físicos, a intervalos de tempo, em geral irregulares e desconhecidos.*

Visto que um SED mantém seu estado constante entre dois eventos consecutivos, podemos ver sua evolução dinâmica no tempo pelo gráfico apresentado na figura 1.1.

Nela representamos os eventos pelas letras gregas α , β , γ , δ e ε , e os estados do suposto sistema por s_0 , s_1 , s_2 , s_3 e s_4 , onde s_0 é o estado inicial do SED, ou mais conhecido por *home-state*.

Um evento pode ocorrer antes ou depois de qualquer outro evento, sem importar a ordem. Também levamos em consideração que a quantidade de diferentes eventos existentes em um SED, é finita, imposto como um caso particular de importância básica, como geralmente são os sistemas físicos reais.

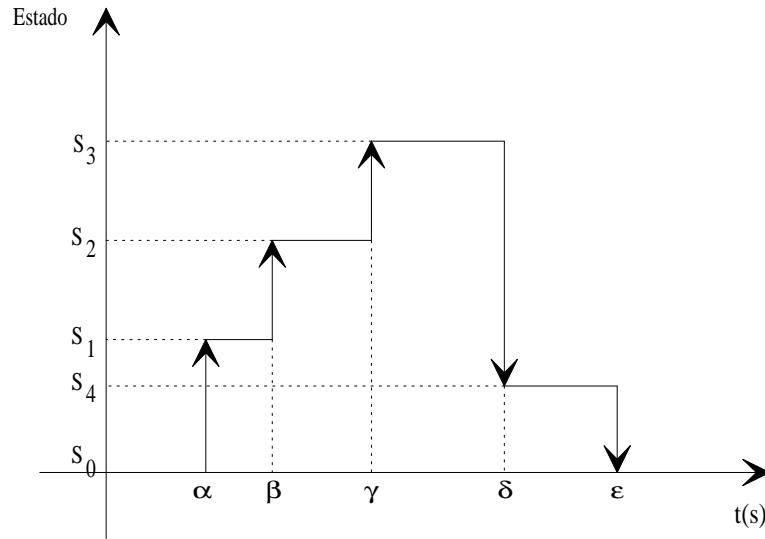


Figura 1.1: Trajetória de um SED

Anteriormente, destacamos que a modelagem de SEDs, pode ser realizada com base em vários paradigmas. Embora nenhum modelo tenha se tornado universal, há estudos utilizando cada um dos modelos apresentados, todavia direcionados a aplicações específicas [Bar96]. As RPs têm atualmente, aumentado seu domínio na modelagem de SEDs. Estas já foram utilizadas na teoria de controle clássica para desenvolvimento de software para controle de sistemas em tempo real, como vemos em Cofrancesco *et al* [PCS91]. As vantagens que as RP apresentam sobre os outros métodos de análise, são as seguintes:

- Facilidade de modelagem relacionadas com as características de um SED (concorrência, sincronismo e assincronismo, conflito, exclusão mútua, relações de precedência, não determinismo e bloqueio);
- Ótima visualização de dependência de sistemas;

- Possibilidade de geração de códigos de controle supervisorio diretamente da representação gráfica;
- Transmissão visual de informações locais;
- Testes de propriedades indesejáveis (bloqueio e reinicialização)
- A análise de desempenho sem simulação é sempre possível para muitos sistemas;
- Simulação dos eventos discretos a partir do modelo;
- Visualização do estado atual do sistema para monitoração em tempo real;
- Abordagens de modelagem do tipo refinamento e do tipo composição modular.

Devido a estas vantagens, têm-se utilizado muito das RPs, tanto lugar/transição como as de alto nível (coloridas, controladas, etiquetadas, com temporização nebulosa) que têm se tornado mais usuais para a compactação e compreensão dos modelos dos SEDs [HK90][SK92][Sre93][GD94][KH96], levando-nos a buscar a concepção de controladores para os mesmos e, conseqüentemente, cedendo mais algumas contribuições à Teoria de Controle Supervisorio (TCS) [RW89].

A representação dos SEDs por intermédio das RPs, tem uma grande utilidade no mundo real, devido ao avanço tecnológico computacional, em que encontramos softwares com interfaces homem-máquina de alto nível, onde podemos criar um modelo de um determinado sistema via RP e executá-lo, ou seja, fazer simulações através desta interface e de hardwares adicionais que liguem um computador que simule um supervisor, a um outro que simule uma planta controlada, ou mesmo a um sistema real. A questão da simulação é de alta prioridade, desde que podemos observar na execução do modelo de um sistema todo seu comportamento antes de pô-lo em prática, evitando problemas de ordem superior como acidentes ou gastos desnecessários. Como o hardware do computador pode se comunicar com o mundo exterior, então, a partir de conversores e transdutores, podemos controlar um sistema real, o qual esteja modelado no software, e conceber sua atuação imediata, apenas olhando para o monitor de vídeo. Isto é a condição intrínseca da aplicabilidade industrial da TCS.

De acordo com o que foi explanado, a concepção de controladores para os SEDs, tem se tornado, cada vez mais, vinculada às RPs. Estas estão facilitando o trabalho de pesquisa desejado na TCS, que é a modelagem, controle, simulação e análise de desempenho. As mesmas também têm a possibilidade de representar componentes de hardware, software, humano e interações entre todos. Temos então, uma ferramenta de alta capacidade, para os estudos necessários à área.

Estruturamos o controle de um SED real em dois níveis:

- Controle Supervisório ou relacionado a eventos;
- Controle de processos tradicional.

Esta estrutura de controle está representada na figura 1.2. Nela, observamos que a implementação do controle hierárquico, em um ambiente automatizado, é dado pelos seguintes métodos [Bar96]:

1. Controle centralizado, onde se usa um computador para a realização e sincronismo das atividades que têm relações com os eventos e, para controlar os processos de baixo nível;
2. Controle distribuído, onde os componentes de um determinado sistema podem ser vistos como sub-sistemas independentes que mantêm comunicação entre si para solucionar o problema de controle. Um supervisor irá coordenar as tarefas assíncronas entre estes componentes, de modo a satisfazer o que o sistema impõe.

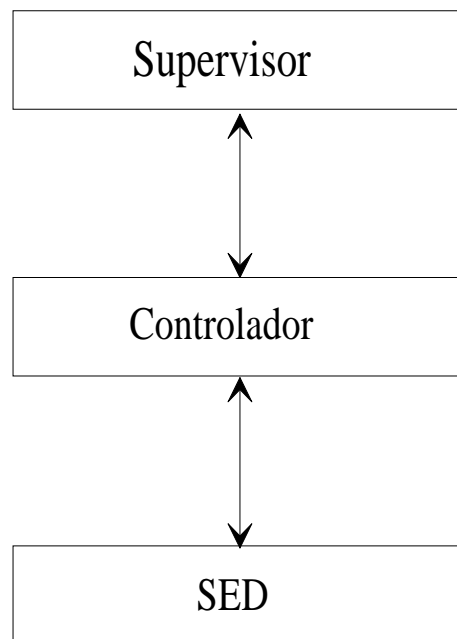


Figura 1.2: Estrutura de Controle de um SED

Neste ponto, deparamo-nos com a idéia do supervisor. E é onde entra a teoria desenvolvida recentemente por Ramadge e Wonham [RW89] que se denomina Teoria de Controle Supervisório (TCS). Esta foi inicialmente desenvolvida baseada em linguagens

formais e autômatos, mas que é totalmente independente das ferramentas utilizadas para a modelagem do sistema.

A vantagem encontrada na TCS sobre outras teorias e modelos de análise para os SEDs, é que esta pode separar explicitamente o sistema a ser controlado, do controlador em si. Dessa maneira, podemos apresentar as condições necessárias e suficientes para a existência do supervisor. A formulação genérica do problema de controle no contexto da TCS visa a determinação do autômato supervisor a partir do modelo do sistema controlador, além da especificação de comportamento requerida, ou desejada para o SED. A função do supervisor é assegurar que a linguagem gerada pela sua associação com o SED que desejamos controlar, represente a especificação de comportamento desejada, ou seja, fazer com que o SED realize uma tarefa específica. A solução deste problema é conhecida como síntese do supervisor.

Nesta teoria, um SED tem seu modelo baseado nos autômatos, e seu comportamento é descrito pela linguagem gerada por este autômato, onde o comportamento requerido é especificado através de linguagens formais.

A especificação do comportamento de um dado SED representa a tarefa que desejamos realizar com este sistema. Desde que definimos a tarefa que desejamos realizar, é necessário traduzi-la numa forma que seja compatível com a metodologia de projeto adotada para a concepção do supervisor. De modo geral a definição da tarefa, ou especificação de comportamento, é feita utilizando-se linguagem natural, a qual é entendida no contexto deste trabalho como sendo aquela na qual expressamos as idéias da nossa mente. A existência de proposições ambíguas é uma característica inerente da linguagem natural, o que dificulta sobremaneira sua formalização matemática. Assim, considerando a necessidade de uma formalização matemática, é necessário avaliar o emprego de outras ferramentas para a definição da especificação do comportamento do SED que desejamos controlar. Neste contexto, a lógica temporal, uma ferramenta que já demonstrou sua utilidade na solução de outros problemas [McM92], surge como uma das alternativas viáveis para a especificação do comportamento dos SEDs, substituindo as linguagens formais utilizadas por Ramadge e Wonham [RW89], desde que as linguagens formais apresentam o problema de não poderem representar ocorrências de eventos em paralelo.

A formulação matemática de uma determinada tarefa especificada para um dado SED, pode ser exemplificada como a seguir: se desejarmos que um braço robótico *levante* e, logo após, *baixe*. Esta mensagem teria de ser transformada matematicamente para que o sistema (braço robótico) realizasse a ordem. Caso os eventos associados a este sistema fossem representados por: *levantar* = α e *baixar* = β , a ordem matemática

na forma de linguagem formal seria $\alpha\beta$, e na forma de lógica temporal, seria $\alpha U \beta$. Estas linguagens serão explicitadas nos capítulos 2 e 4, respectivamente.

Há estudos sendo feitos nesta área, como encontramos em Knight e Passino [KP90], baseando-se na inclusão da lógica temporal à TCS, desde que aquela pode ser vinculada às redes de Petri [RB97], e também por meio dos autômatos seqüenciais de Büchi e da LPTL (*Linear Propositional Temporal Logic*), como apresentado por Uchihira e Honiden [UH90], que se aplicam à redes de Petri que tenham em um dado instante, apenas uma única transição habilitada (condição de evento singular). Veremos em seguida, que para uma rede de Petri mais generalizada, ou seja, que admita a representação de concorrências e/ou conflitos, a CTL (*Computation Tree Logic*) é mais indicada pois trabalha com a condição de em um dado momento, haver mais de uma transição habilitada. A lógica temporal já foi estudada desde muito tempo. Encontramos boas referências em McMillan [McM92] que trabalhou com o problema da explosão de estados, utilizando a verificação simbólica do modelo, dando uma visão geral da teoria da classe de Lógica Temporal conhecida por CTL (*Computation Tree Logic*). Também em Byrnes [Byr97] encontramos a CTL aplicada ao desenvolvimento de um programa chamado *Feature Integrator*, o qual aparenta expandir a funcionalidade de um dado sistema, integrando um conjunto de características dentro deste sistema, para criar um novo sistema expandido para ser verificado pelo modelo. Em Sakalauskaitė [Sak96] vemos toda a descrição da BPTL (*Branching Propositional Temporal Logic*) e seu uso na resolução do sistema não-preceitual. Costa [Cos92] nos dá todo o formalismo da Lógica Clássica Proposicional e suas aplicações, assim como Galton [Gal87] e Ostroff [Ost89] com a Lógica Temporal, onde este último aplica esta a sistemas em tempo real.

Podemos citar ainda, Grumberg e Long [GL94], usando a CTL para verificação do modelo e verificação modular de processos de estados finitos; Nicola e Vaandrager [NV95], com a CTL* que é uma extensão da CTL, para trabalhar com dois sistemas concorrentes (*bissimulação*); Emerson e Halpern [EH86] apresentando e fazendo a comparação da BTL (*Branching Temporal Logic*) e da LTTL (*Linear Time Temporal Logic*) em programação concorrente; Abadi e Manna [AM87], descrevendo um outro tipo de linguagem de programação de lógica temporal denominada de *TEMPLOG*, para incluir programas com construções temporais. Já Finger e Gabbay [FG93], introduziram uma metodologia em que um sistema lógico pode ser enriquecido com características temporais para criar um novo sistema temporizado sem modificar as propriedades do sistema lógico original. Em Fix e Grumberg [FG96], encontramos o estudo da verificação de propriedades temporais na CTL e em Galton [Gal96] vemos um estudo baseado no princípio do "*Non-intermingling*", que é a análise de que uma dada proposição não

permita mudar seu valor verdadeiro, infinitas vezes em um período de tempo finito.

Como sendo uma ferramenta de alto poder de descrição matemática, a Lógica Temporal pode nos levar a determinar o comportamento desejado do sistema. A mesma trabalha com proposições atômicas e conectivos booleanos, gerando assim uma fórmula. Esta fórmula é mais específica e abrangente para esta definição de comportamento do que as linguagens formais. Devemos observar que, neste caso, um usuário deve ter uma boa concepção da matemática e da lógica, de forma a transformar o comportamento procurado em uma fórmula de lógica temporal para a resolução do problema de controle.

Dado o modelo e a especificação do comportamento desejado, podemos analisar o funcionamento do sistema em estudo, utilizando-nos da simulação, a qual é produzida a partir de computadores, utilizando de softwares interativos, com interfaces homem-máquina amigável. Temos estudos na área de simulação, como demonstra Banks *et al* [JBN96], com todo um estudo analítico e gráfico; Chen [Che96], com a proposta de limitação inferior para a quantificação probabilística de confiança usada para otimização de SEDs, utilizando testes numéricos, e Evans [Eva93] com o desenvolvimento da ferramenta *DEVNET*, que apresenta facilidades para incorporar a representação de sistemas de alta complexidade.

A TCS teve grande avanço desde seu início, já que há muitos pesquisadores distribuídos em todo o planeta, desenvolvendo estudos sobre ela, nos mais variados aspectos. Barroso [Bar96] estudou a TCS, colocando uma nova abordagem para a síntese de supervisores a eventos discretos utilizando-se das Redes de Petri com Função de Habilitação de Transições (RPFHT) [PC92]. Este trabalhou com a modelagem de sistemas por meio de redes de Petri de alto nível, tais como: redes de Petri com Temporização Nebulosa [Fig94], redes de Petri Coloridas [Jen92], entre outras, além de modelagem de sistemas complexos por meio de sistemas de G-Nets [Per94], e introduziu os algoritmos AMArA (Algoritmo Modificado da Árvore de Alcançabilidade) que cria toda a árvore de alcançabilidade do sistema modelado pela rede de Petri, a qual tem capacidade finita, com a diferença do algoritmo original de que apresenta um passo a mais na direção da determinação dos estados não permitidos, de forma a termos a segurança e garantia das seqüências de eventos não aceitáveis, e o ACGS (Algoritmo da Construção do Gerador da Suprema Liguagem Controlável - *SupC(L)*), onde este nos devolve, a partir de uma determinada especificação de entrada, a máxima linguagem possível (seqüência de eventos) em que o sistema pode seguir, sem que haja problemas.

O estudo dos anéis [Kuc91], já utilizado na teoria clássica de controle [Emr80] [EK82], é utilizado no estudo da TCS em Libeaut [Lib96]. Aqui, encontramos a análise

dos anéis (*rings*) e semi-anéis (*semi-rings*), que são formas algébricas polinomiais de matrizes, e desenvolvimento, a partir destes, de uma forma de aproximação algébrica para os sistemas do tipo apresentados por Kucera [Kuc91] e Vidyasagar [Vid85], por sistemas lineares clássicos, tendo como objetivo, dar uma descrição algébrica simples e completa das redes de Petri temporizadas para redução dos problemas de comando em termos de resolução de equações e inequações. Também aqui, as redes de Petri são utilizadas como método de modelagem. Já Cohen *et al.* [GCV89] mostra como as redes de Petri conhecidas por *grafos de eventos*, podem ser representadas como sistemas lineares de dimensões finitas invariantes no tempo com o uso dos semi-anéis, para avaliar a performance de SEDs.

Outras referências de pesquisas desenvolvidas em TCS são: Sampath [MST95], que descreve uma aproximação para o problema de diagnóstico de falhas em SEDs, e apresenta um procedimento sistemático para a detecção e isolamento de falhas de eventos, fazendo observações *on-line* do comportamento do sistema; Glasserman [GY95] que trabalhou com a análise da estabilidade de SEDs modelados como processos semi-Markovianos generalizados e Cofer [CG95], que utilizou-se da teoria *Lattice* para ordenar as seqüências de eventos, onde analisou o comportamento de SEDs em controle supervisorio com tempo real. Também temos uma outra citação importante, a qual pode ser encontrada em Kumar [KH96], que em seus estudos, estendeu os resultados da TCS pela introdução de um algoritmo para computação do controle mínimo restritivo, onde o comportamento desejado é dado na forma de linguagens regulares, utilizando-se de linguagens de redes de Petri Determinísticas, para realizar a modelagem do comportamento da planta. As pesquisas de SEDs, encontram-se também nos chamados sistemas híbridos, como podemos ver em Cury *et al* [JECN96], dando uma metodologia para projetos de controladores destes sistemas, os quais são sistemas que trabalham conjuntamente com SEDs e sistemas contínuos.

1.1 Objetivos e Apresentação do Trabalho

O objetivo principal deste trabalho é avaliar o emprego da lógica temporal (CTL) como ferramenta de especificação de comportamentos de um dado SED, que é a tarefa que queremos que o sistema realize. Precisamos, então, traduzir a especificação desejada numa forma que seja compatível com a metodologia de projeto adotada para a concepção do supervisor. Aqui, utilizamo-nos da metodologia de projeto de controladores para SEDs, proposta por Barroso [Bar96]. De forma resumida, nesta metodologia, o modelo do sistema a ser controlado é descrito através de uma rede de Petri e a síntese

do supervisor, que tem a mesma estrutura do sistema, é obtida executando-se, numa primeira fase, o Algoritmo Modificado da Árvore de Alcançabilidade (AMArA) e posteriormente o Algoritmo para Criação da Suprema Linguagem Controlável (ACGS), para determinação da máxima sub-linguagem controlável. Este utiliza os dados gerados pelo AMArA e a especificação do comportamento desejada. Para o mesmo sistema, caso desejemos realizar nova tarefa, precisamos apenas redefinir a especificação do comportamento e executar novamente o ACGS. Nosso trabalho trata, justamente, do problema da especificação de comportamentos de SED no contexto da metodologia de projeto delineada acima.

Definido o objetivo deste trabalho, organizamo-lo da seguinte maneira: o capítulo 2 formaliza todas as definições necessárias à compreensão do trabalho, identificando as diferenças características entre os sistemas contínuos e os SEDs, e dando as formas de modelagem para os SEDs, tanto na formalização clássica (Autômatos e Linguagens Formais), quanto na realização deste trabalho (Redes de Petri).

O capítulo 3 apresenta conceitos básicos sobre a Teoria de Controle Supervisório como desenvolvido por Ramadge e Wonham [RW89], e com respeito à abordagem feita por Barroso [Bar96].

No capítulo 4 apresentamos a CTL e como utilizá-la juntamente com as Redes de Petri, para determinação de seu comportamento, dando alguns exemplos para isto.

O capítulo 5 contém os algoritmos de síntese do supervisor e exemplos demonstrando como é formalizada a resolução do problema de controle.

Nas considerações finais, analisamos o conteúdo do trabalho com relação aos objetivos propostos, além de apresentarmos algumas sugestões de trabalhos futuros.

Capítulo 2

Conceitos Básicos

Apresentamos neste capítulo os conceitos e esclarecimentos gerais, relativos ao estudo desenvolvido, onde definiremos e exemplificaremos os sistemas dinâmicos clássicos, os SEDs, as redes de Petri Lugar/Transição e redes de Petri com Funções de Habilitação de Transições, as Linguagens Formais e os Autômatos e Geradores, explicitando a diferença entre estes dois últimos.

2.1 Sistemas Dinâmicos a Variáveis Contínuas

Os Sistemas Dinâmicos a Variáveis Contínuas (SDVC) são sistemas que têm todas as suas variáveis em função do tempo, ou seja, o espaço de estados destes, varia a cada instante. Estes são descritos matematicamente por um conjunto de equações diferenciais que representam a dinâmica do sistema. Então seu comportamento é descrito por uma função que relaciona o estado (variável dependente) com o tempo (variável independente) [Oga93].

Podemos exemplificar um destes sistemas por um braço robótico, em que um determinado movimento seu, é uma característica de um SDVC, pois o mesmo varia continuamente seu estado no tempo. Logo, vê-se que esta teoria se preocupa com a realização física das atividades do sistema.

O gráfico relacionado à trajetória de um SDVC, está representado na figura 2.1, onde pode-se observar a continuidade temporal da mesma, e a mudança de estado a cada instante de tempo.

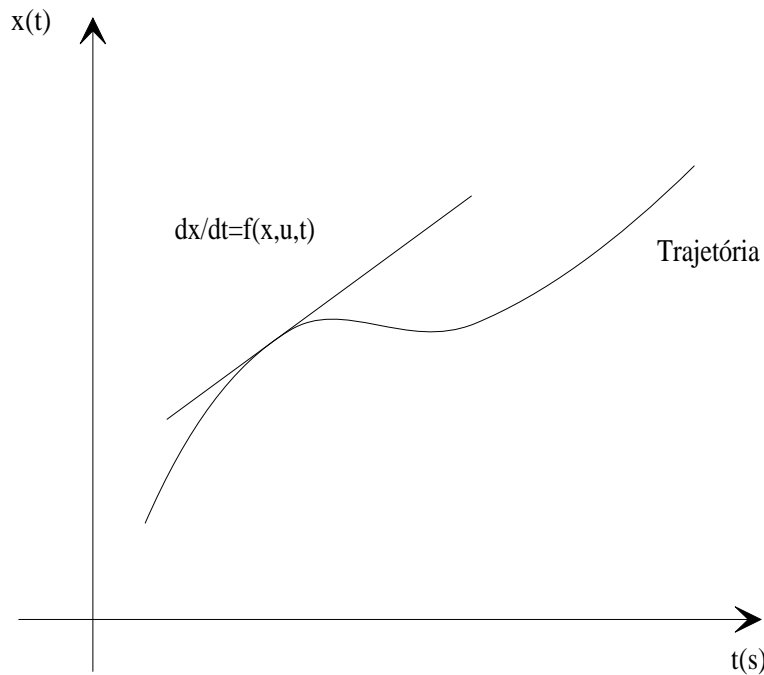


Figura 2.1: Trajetória típica de um SDVC

2.2 Sistema Dinâmicos a Variáveis Discretas

Com o advento da tecnologia digital, os SDVCs passaram a ser estudados discretamente, isto é, o tempo e os estados do sistema são discretizados, de tal forma que se possa fazer um controle de processos e simulações de sistemas por intermédio de computadores.

Neste caso, os Sistema Dinâmicos a Variáveis Discretas (SDVD) [GFFW92] diferem dos SDVCs, apenas pela amostragem do estado de acordo com uma constante de tempo, ou seja, embora o sistema tenha a mesma curva característica contínua, seu estado só é observado em cada instante de tempo Δt . Logo, para o tempo t_1 , temos a amostra da curva de estado do sistema no estado e_1 . Para o tempo $t_2 = t_1 + \Delta t$, temos a amostra e_2 , e assim sucessivamente. Observamos então, que o tempo está discretizado e a observação dos estados é feita a cada variação temporal Δt , e, conseqüentemente, perdemos alguma informação do sistema durante este intervalo.

As equações matemáticas que regem estes sistemas, são aproximações numéricas das equações diferenciais que regem os sistemas contínuos. Vemos na figura 2.2, como se processa sua trajetória, em comparação com a trajetória dos SDVCs.

Para o caso destes sistemas, quanto menor for o valor de Δt , mais próximo da função real do comportamento do sistema, ficará o SDVD.

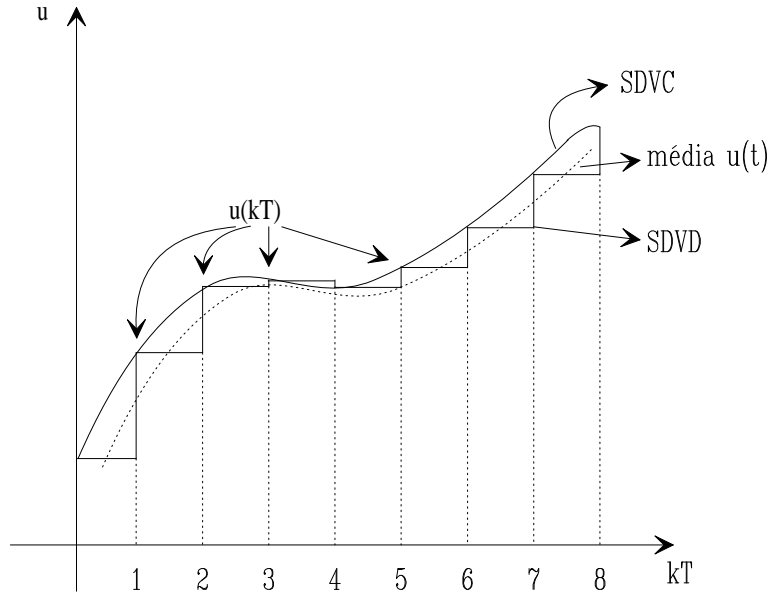


Figura 2.2: Trajetória típica de um SDVD, comparado a um SDVC

2.3 Sistemas Dinâmicos a Eventos Discretos

Como mencionado anteriormente, os SEDs são sistemas que variam sua trajetória independentemente do tempo, isto é, a diferença dos SEDs para os SDVDs, é que a variação temporal não é uma constante Δt , mas geralmente irregular e desconhecida, já que não temos o controle absoluto das variáveis. Por exemplo, a soldagem de uma peça feita por um braço robótico que segura um certo maçarico, varia de peça para peça. Se este tempo fosse constante, poderia haver problemas por uma peça não ser bem soldada. Também se estas peças estivessem sendo conduzidas por uma esteira, as distâncias entre elas teriam de ser extremamente bem calculadas, de forma a se pôr, exatamente na posição de soldagem. Dessa forma, vê-se que no caso dos SEDs, teriam de haver sensores que indicassem a posição da peça para que o braço robótico a soldasse. Após isto, rolaria a esteira para levar nova peça para soldar, enquanto a peça já soldada iria passar por uma certa análise. Isto nos leva a ver que a evolução dinâmica dos SEDs é dada pela geração de eventos, isto é, um SED é um gerador de eventos físicos.

É importante considerar que o funcionamento de um SED é caracterizado pela ocorrência de eventos pertencentes a duas classes distintas: eventos controláveis e eventos não-controláveis. Um evento dito controlável é aquele para o qual há, explicitamente, uma maneira pela qual podemos permitir ou impedir sua ocorrência. A ligação de uma máquina e o envio de uma mensagem podem ser classificados como eventos controláveis, pois sempre é possível não ligar a máquina ou não enviar uma dada mensa-

gem. Por outro lado, um evento é dito não-controlável quando sua ocorrência não pode ser permitida ou inibida, isto é, tal evento ocorre espontaneamente. A quebra de uma esteira, o término do processamento de uma máquina de comando numérico, a chegada no receptor de uma mensagem enviada são exemplos de eventos não-controláveis. Em outros termos, um SED depende de fatores alheios ao sistema.

Quando ocorre um evento não-controlável que leva o sistema a um estado indesejável, é necessário a tomada de decisões para desviar o processo, eliminando a ociosidade parcial do sistema. Neste caso, dado como exemplo um sistema integrado de manufatura, em que tenhamos vários braços robóticos realizando um processo, a quebra de um deles impõe que outro deva absorver o trabalho antes desenvolvido por aquele, enquanto o mesmo está a ser consertado.

Na figura 1.1, vemos a evolução dinâmica de um SED, onde podemos observar que a ocorrência de um evento é instantânea, e causa uma transição ou mudança de estado no sistema.

Quando o SED se encontra antes da ocorrência do primeiro evento, dizemos que o mesmo está no estado inicial. O processo denominado *reinicialização*, é quando o sistema, após a realização de uma determinada tarefa, retorna ao seu estado inicial.

O espaço de estados de um SED é, geralmente limitado a um conjunto enumerável, não tendo uma função geral que relacione seu estado ao tempo. Logo, para o caso dos modelos não temporizados, sua trajetória é representada por uma seqüência de eventos, explicitados na forma $\{\sigma_1, \sigma_2, \sigma_3, \dots\}$.

No caso dos SED, a teoria trata do controle de atividades executáveis no sistema, sem dar importância a realização física das mesmas (o movimento do braço do robô não é uma característica de um SED, mas a ação mover, é).

Foram utilizados variados modelos para os SEDs, como visto na introdução desta dissertação, com utilizações específicas. Contudo, as redes de Petri, as quais trataremos delas mais adiante, têm se tornado mais utilizadas, devido as várias vantagens que elas oferecem, inclusive quando levamos em consideração as análises de sistemas utilizando simulação, com a utilização de computadores, pois podemos conceber toda a evolução do SED em seu monitor de vídeo. Também a realização de um processo, no caso de junção destes com sistemas eletrônicos, para ligá-los a um sistema físico real.

2.4 Linguagens Formais

As linguagens formais são representadas por alfabetos de símbolos (letras ou dígitos), onde um determinado símbolo é uma entidade não definida formalmente. Contudo, um

conjunto não vazio de símbolos, define um alfabeto, o qual é representado por uma letra grega maiúscula. Geralmente, utiliza-se a letra grega Σ para designar um alfabeto, o qual pode ser exemplificado por $\Sigma = \{\alpha, \beta, \gamma, \delta\}$, onde temos os símbolos $\alpha, \beta, \gamma, \delta$ que são os elementos do alfabeto Σ .

Quando temos uma justaposição finita de símbolos, analogamente à gramática, temos uma palavra. De acordo com o alfabeto dado anteriormente, exemplificamos algumas palavras da forma: $\alpha\alpha\beta, \alpha\gamma\beta, \alpha\delta\alpha$, etc. Uma palavra qualquer, é designada por s .

O comprimento de uma palavra é definido como:

Definição 2.1 *O comprimento de uma palavra s , representado pela cardinalidade $|s|$, é igual ao número de símbolos que a compõe.*

Entretanto, temos a palavra nula, a qual é representada por ϵ e é a única palavra de comprimento nulo. Sendo assim, $\epsilon \notin \Sigma$ pois é uma palavra, e não um símbolo.

Temos na Teoria de Linguagens os conjuntos de palavras, os quais são definidos como segue:

Definição 2.2 *Dado $k \in \mathbb{N}$, denota-se por Σ^k o conjunto de todas as palavras sobre Σ cujo comprimento é igual a k .*

Logo, de acordo com a definição, temos que, dado $\Sigma = \{\alpha, \gamma\}$:

$$\Sigma^0 = \{\epsilon\};$$

$$\Sigma^1 = \{\alpha, \gamma\};$$

$$\Sigma^2 = \{\alpha\alpha, \alpha\gamma, \gamma\alpha, \gamma\gamma\};$$

$$\Sigma^3 = \{\alpha\alpha\alpha, \alpha\alpha\gamma, \alpha\gamma\alpha, \alpha\gamma\gamma, \gamma\alpha\gamma, \gamma\gamma\alpha, \gamma\gamma\gamma, \gamma\alpha\alpha\}.$$

Também definimos dois conjuntos especiais, que são Σ^+ e Σ^* , onde

$$\begin{aligned}\Sigma^+ &= \bigcup_{k=1}^{\infty} \Sigma^k = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots \\ \Sigma^* &= \bigcup_{k=0}^{\infty} \Sigma^k = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots = \Sigma^0 \cup \Sigma^+.\end{aligned}$$

Comparativamente, Σ^+ seria o dicionário do alfabeto Σ . Aqui, definimos que um conjunto de palavras formado com os símbolos de um alfabeto Σ , é uma linguagem, ou seja:

Definição 2.3 *Dado um alfabeto Σ , L é uma linguagem sobre Σ se e só se, $L \subseteq \Sigma^*$.*

Isto implica que uma linguagem tanto pode ser Σ^* como $\emptyset = \{ \}$ (linguagem vazia, a qual é diferente da linguagem Σ^0).

Compomos palavras por concatenação de duas ou mais palavras. Por definição:

Definição 2.4 Dadas duas palavras u e v sobre um alfabeto Σ , com $u = \sigma_1 \dots \sigma_k$ e $v = \sigma_{k+1} \dots \sigma_n$, sua concatenação é

$$s = uv = \sigma_1 \dots \sigma_k \sigma_{k+1} \dots \sigma_n. \quad (2.1)$$

Na Teoria de Linguagens, é necessário definir a noção de prefixo, que é uma parte inicial de comprimento arbitrário de uma palavra:

Definição 2.5 O prefixo de uma palavra s sobre um alfabeto Σ é qualquer palavra $u \in \Sigma^*$ que possa ser completada com outra palavra $v \in \Sigma^*$ para formar a palavra s .

Por exemplo, da definição 2.4, u é um prefixo de s visto $\exists (v = \sigma_{k+1} \dots \sigma_n) \in \Sigma^*$ tal que, $uv = s$.

Dessa forma, encontramos um conjunto de prefixos de uma dada palavra s . Logo:

Definição 2.6 Dada uma palavra s , denota-se por $\text{Pre}(s)$ o conjunto de todos os prefixos de s , incluindo a palavra vazia ϵ .

Além do mais, o conjunto total, o qual inclui todas as palavras de uma linguagem $L \subseteq \Sigma^*$ e todos os seus prefixos é:

Definição 2.7 O prefixo-fechamento, ou fechamento de L , é dado por:

$$\overline{L} = \{u : \exists v \in \Sigma^* \wedge uv \in L\}, \quad (2.2)$$

ou seja, desde que tenhamos uma linguagem $L \subseteq \Sigma^*$, existe uma linguagem associada a L , denotada por \overline{L} , a qual é formada pelas palavras de L e por todos os seus prefixos. Logo, concluímos que $L \subseteq \overline{L}$.

Utilizamo-nos de algumas expressões para estudos baseados em linguagens formais, como apresentado por Hopcroft e Ullman [HU79]. São elas:

- σ^n - representa a repetição do símbolo σ , por n vezes;
- s^n - representa a repetição da palavra s , por n vezes;
- σ^* - representa a repetição do símbolo σ , por um número arbitrário de vezes;
- s^* - representa a repetição da palavra s , por um número arbitrário de vezes;
- $+$ - símbolo empregado como o operador lógico *ou*, indicando uma opção entre duas ou mais possibilidades.

Estas notações facilitam na representação finita para linguagens formadas por um número infinito de palavras, isto é, compactam em uma única fórmula. Por exemplo, no alfabeto $\Sigma = \{\alpha, \beta\}$, com uma linguagem prefixo-fechada L e com os símbolos ocorrendo alternadamente, com α ocorrendo primeiro, temos:

$$L = \{\epsilon, \alpha, \alpha\beta, \alpha\beta\alpha, \alpha\beta\alpha\beta, \dots\} = (\alpha\beta)^*(\alpha + \epsilon) \quad (2.3)$$

de onde podemos ver que $\alpha\beta$ pode não ocorrer, ou ocorrer um número arbitrário de vezes e, logo após, α ocorre, ou ϵ , isto é, nada. Também devemos observar que L é subconjunto próprio da linguagem $\Sigma^* = \{\alpha^*\beta^*\}^*$, que contém as palavras de L e as palavras $\alpha\alpha\beta$, $\alpha\beta\beta$, $\beta\alpha$, $\beta\beta$, etc, que não são palavras de L , devido à sua definição.

2.5 Autômatos

Os autômatos são modelos matemáticos de máquinas que têm entradas e saídas discretas e que reconhecem um conjunto de palavras sobre um dado alfabeto. Logo, um autômato pode ser visto como uma entidade de controle que lê, de forma seqüencial, uma lista de símbolos em um alfabeto, aceitando uma determinada palavra, desde que a mesma pertença ao conjunto de palavras reconhecidas.

Os autômatos podem ser finitos ou infinitos, além de determinísticos ou não determinísticos e, seu estado atual (configuração interna do modelo em um dado instante) é uma síntese das informações de estados anteriores, os quais são necessários na determinação dos estados futuros.

No tocante ao autômato finito, este consiste de um conjunto finito de estados, isto é, a quantidade de estados é enumerável, e um conjunto de transições. Estes estados são designados por Q , e o conjunto de transições de estados são designados por δ . Este último, ocorre a partir de símbolos de entrada escolhidos dentro de um alfabeto Σ .

O estado inicial de um autômato é designado por q_0 , e alguns estados finais, ou estados marcados, por Q_m . Esta seqüência de palavras lidas é reconhecida por um autômato de forma que, quando processadas, levam-no a um estado marcado.

Um autômato é dito determinístico se, a cada símbolo de entrada, há unicamente uma transição de saída de cada estado. Se houver mais de uma, o autômato é dito não determinístico.

A definição formal de um autômato é a seguinte:

Definição 2.8 *Um autômato determinístico finito, ou simplesmente um autômato é uma quintupla*

$$A_s = (Q, \Sigma, \delta, q_0, Q_m)$$

onde:

- Q é um conjunto finito de estados q ;
- Σ é o alfabeto ou conjunto de símbolos σ ;
- $\delta : \Sigma \times Q \rightarrow Q$ é a função de transição de estados, onde

$$\begin{aligned} \delta(\epsilon, q) &= q & e \\ \delta(\sigma, q) &= q', \quad \text{para } q, q' \in Q \text{ e } \sigma \in \Sigma; \end{aligned} \quad (2.4)$$

- $q_0 \in Q$ é o estado inicial;
- $Q_m \subseteq Q$ é o conjunto de estados marcados.

Observando a função de transição de estados, vemos que q' só será um estado do autômato A_s , no caso de σ ser uma entrada aceita por A_s .

Os autômatos são representados graficamente por um grafo direcionado, onde seus vértices são os estados e os arcos as funções de transição. Os estados marcados são representados por vértices em linha dupla (círculos concêntricos) e o estado inicial indicado por uma seta que não é saída de nenhum vértice. Podemos ver um exemplo deste na figura 2.3, onde:

- $\Sigma = (\alpha, \beta, \gamma)$;
- $Q = (0, 1, 2)$;
- $\delta(\alpha, 0) = 1, \delta(\alpha, 1) = 2, \delta(\alpha, 2) = 2, \delta(\beta, 0) = 2, \delta(\beta, 1) = 0, \delta(\beta, 2) = 1, \delta(\gamma, 1) = 1, \delta(\gamma, 2) = 0$;
- $q_0 = 0$ e
- $Q_m = \{1, 2\}$.

Observamos que sua construção é dada pelas funções de transição, onde $\delta(x, y)$ é a posição de término, ou seja, para onde o arco direcionado aponta, x é uma transição e y é o lugar de origem do arco. Em outras palavras, será o arco que indica uma transição que vai do estado y para o estado $\delta(x, y)$.

O processamento de palavras em um autômato decorre da definição a seguir, que estende a idéia de transição.

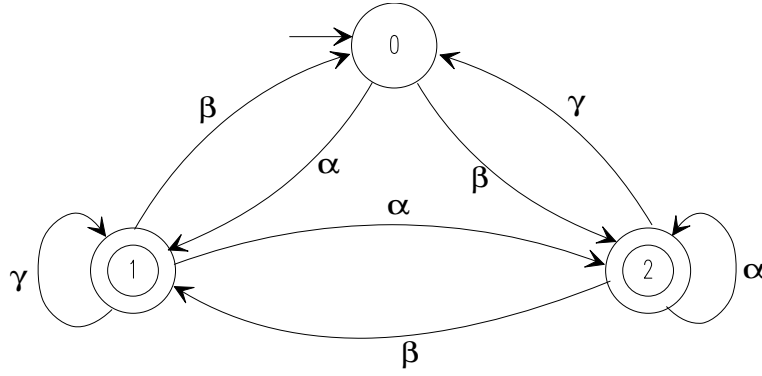


Figura 2.3: Exemplo de um Autômato

Definição 2.9 *Seja um autômato $A_s = (Q, \Sigma, \delta, q_0, Q_m)$ e seja uma função de transição estendida δ^* , que é a função*

$$\delta^* : \Sigma^* \rightarrow Q,$$

de tal forma que:

$$\begin{aligned} \delta^*(\epsilon, q) &= q & e \\ \delta^*(s\sigma, q) &= \delta(\sigma, \delta^*(s, q)) \text{ para } q \in Q \text{ e } s \in \Sigma^*. \end{aligned} \quad (2.5)$$

Desde que $\delta^*(\sigma, q) = \delta(\sigma, \delta^*(s, q)) = \delta(\sigma, q)$ para o caso em que $s = \epsilon$, podemos utilizar, então, δ ao invés de δ^* , por uma questão de conveniência.

Conhecidas a função de transição e o estado atual do autômato, é possível determinar seu estado após processarmos um símbolo dado. Assim, processando uma palavra a partir de um dado estado, poderemos confirmar se o estado alcançado pertence ou não ao conjunto de estados marcados Q_m . Logo:

Definição 2.10 *Dado um autômato $A_s = (Q, \Sigma, \delta, q_0, Q_m)$, a linguagem marcada ou reconhecida de A_s , é:*

$$L_m(A_s) = \{s : s \in \Sigma^* \wedge \delta(s, q_0) \in Q_m\}. \quad (2.6)$$

A linguagem reconhecida por um autômato pode ser encontrada, seguindo os arcos no grafo do mesmo.

2.6 Geradores

Um gerador é um autômato, que, não necessariamente, segue todas as seqüências de transições de estado, definidas como na função de transição dos autômatos. Nos geradores a função de transição permite uma representação de um subconjunto próprio de

Σ^* . Então, precisamos definir um gerador, para determinar sua compatibilidade com a representação dos SEDs. Formalmente, temos:

Definição 2.11 *Um gerador é uma quintupla*

$$G = (Q, \Sigma, \delta, q_0, Q_m)$$

onde os elementos Q , Σ , q_0 , Q_m , e δ têm a mesma definição do autômato 2.8, com δ definido como a função, geralmente parcial, de transição de estados.

Como podemos ver, a diferença existente entre os autômatos e os geradores é que, no caso dos autômatos, a função de transição não pode ser parcial (definida apenas para um subconjunto de eventos para cada estado do gerador).

Igualmente aos autômatos, temos o conjunto de eventos, a função de transição estendida, sua linguagem e sua linguagem marcada, como mostra as seguintes definições:

Definição 2.12 *Dado um gerador $G = (Q, \Sigma, \delta, q_0, Q_m)$, associa-se a cada estado $q \in Q$ o conjunto de eventos definidos $\Sigma(q)$, dado por:*

$$\Sigma(q) = \{\sigma : \sigma \in \Sigma \wedge \delta(\sigma, q)!\} \quad (2.7)$$

com $\delta(\sigma, q)!$ identificando que δ é definido para o par (σ, q) .

Definição 2.13 *Seja um gerador $G = (Q, \Sigma, \delta, q_0, Q_m)$. Sua função de transição estendida, denotada por δ^* , é uma função*

$$\delta^* : \Sigma^* \times Q \rightarrow Q$$

tal que:

$$\begin{aligned} \delta^*(\epsilon, q) &= q \\ \delta^*(s\sigma, q) &= \delta(\sigma, \delta^*(s, q)), \quad \text{para } q \in Q \text{ e } s \in \Sigma^* \text{ sempre que } q' = \delta^*(s, q) \\ &\quad \text{e } \delta^*(\sigma, q') \text{ estiverem ambos definidos} \end{aligned} \quad (2.8)$$

Definição 2.14 *Seja um gerador $G = (Q, \Sigma, \delta, q_0, Q_m)$. Sua linguagem gerada $L(G)$ é:*

$$L(G) = \{s : s \in \Sigma^* \wedge \delta(s, q_0)!\} . \quad (2.9)$$

Denotamos a função de transição estendida por δ , ao invés de δ^* pelas mesmas razões dos autômatos, ou seja, por conveniência.

Observamos que a linguagem gerada $L(G)$ pelo gerador é *prefixo-fechada*, isto é,

$$\begin{aligned} \forall G = (Q, \Sigma, \delta, q_0, Q_m), \\ L(G) &= \overline{L(G)} \end{aligned} \quad (2.10)$$

Com relação a linguagem marcada do gerador, analogamente aos autômatos, temos:

Definição 2.15 Dado um gerador $G = (Q, \Sigma, \delta, q_0, Q_m)$, a linguagem marcada de G , denotada por $L_m(G)$, é:

$$L_m(G) = \{s : s \in \Sigma^* \wedge \delta(s, q_0) \in Q_m\}. \quad (2.11)$$

Tendo esses conceitos, precisamos definir a posição dos SEDs relativa às palavras que o mesmo gera, para entendermos como os mesmos podem ser representados pelos geradores.

Desde que temos que o comportamento de um SED é caracterizado pela ocorrência de eventos, isto é, um SED gera palavras de comprimento crescente, à medida que evolui e, de acordo com o alfabeto gerado pelo SED, podemos encontrar seqüências de símbolos, ou palavras que não representam seqüências de eventos fisicamente possíveis. Logo, a linguagem gerada pelo sistema é um subconjunto próprio de Σ^* . Desta forma, esta linguagem gerada inclui, para cada palavra, todos os seus prefixos.

A linguagem *prefixo-fechada* representa o comportamento lógico de um SED, em que não ocorrem eventos simultâneos. Definimo-la formalmente, da seguinte maneira:

Definição 2.16 A linguagem *prefixo-fechada* que representa o comportamento lógico de um SED é denominada de linguagem gerada do sistema;

e,

Definição 2.17 A linguagem que representa o conjunto de tarefas que um SED é capaz de executar, é denominada linguagem marcada do sistema.

Considerando que L seja a linguagem gerada por um sistema e L_m , sua linguagem marcada, temos, de acordo com estas definições, que a linguagem marcada L_m contém as palavras geradas pelo SED que também gera todos os seus prefixos, ou seja, este SED produz as palavras contidas em $\overline{L_m}$. Desta forma, a linguagem marcada do SED não é necessariamente *prefixo-fechada*. Assim, temos que

$$L_m \subseteq \overline{L_m} \subseteq L = \overline{L}.$$

Logo, é possível representar um SED de linguagem marcada L_m por um autômato A_s tal que

$$L_m(A_s) = L_m.$$

Mas, desde que os autômatos não têm a capacidade de representar a linguagem gerada pelo sistema, restringimos a função de transição, definindo-a apenas para alguns pares (*evento, estado*) do conjunto $\Sigma \times Q$. É aqui que encontramos a posição explícita que

formaliza a representação dos SEDs pelos geradores, desde que os geradores podem representar ambas as linguagens associadas aos SEDs (não possível nos autômatos), isto é, um SED com linguagem gerada L e linguagem marcada L_m , é representado por um gerador, de tal forma que $L(G) = L$ e $L_m(G) = L_m$.

Os geradores, por sua própria definição, não têm restrições à sua estrutura. Logo, as definições a seguir, são de importância fundamental para a definição de estrutura, devido à imposição da análise das componentes de acessibilidade e de coacessibilidade dos mesmos.

Definição 2.18 *A componente acessível de um gerador $G = (Q, \Sigma, \delta, q_0, Q_m)$, denotada por $Ac(G)$ é:*

$$\begin{aligned} Ac(G) &= (Q_{ac}, \Sigma, \delta_{ac}, q_0, Q_{ac,m}), \text{ onde} \\ Q_{ac} &= \{q : \exists s \in \Sigma^* \wedge \delta(s, q_0) = q\}; \\ Q_{ac,m} &= Q_{ac} \cap Q_m; \\ \delta_{ac} &= \delta \mid (\Sigma \times Q_{ac}). \end{aligned}$$

Aqui, δ_{ac} é a função δ restrita ao domínio $\Sigma \times Q_{ac}$. Assim, um gerador G é dito acessível, na condição única de $G = Ac(G)$.

Definição 2.19 *Dado um gerador $G = (Q, \Sigma, \delta, q_0, Q_m)$, este será coacessível se e só se, toda palavra em $L(G)$ for um prefixo de uma palavra em $L_m(G)$, isto é:*

$$L(G) \subseteq \overline{L_m(G)}. \quad (2.12)$$

Desta definição vemos que em um gerador coacessível, existe pelo menos uma seqüência de eventos que leva-o a um estado marcado.

Os geradores que são, ao mesmo tempo, acessíveis e coacessíveis, são denominados ajustados ou *trim*.

2.7 Redes de Petri

Rede de Petri (RP) é uma ferramenta utilizada para modelar sistemas dos mais variados tipos. Estas são grafos direcionados bi-partidos e ponderados com uma marcação inicial, ou estado inicial. A condição bi-partido define que uma RP tem dois tipos de nós, que são denominados lugares e transições. Os arcos são direcionados sempre de um lugar para uma transição e de uma transição sempre para um lugar. No primeiro

caso, denominamos lugar de entrada da transição e, transição de saída do lugar, e vice-versa, no segundo caso.

A representação gráfica dos lugares em uma RP são círculos, e as transições são barras ou retângulos. Um arco com peso k representa k arcos paralelos ligando dois nós. A marcação de uma RP atribui a cada lugar, um número inteiro não negativo n , de onde se diz que o lugar com marcação n tem n fichas, as quais são representadas por pequenos círculos pretos.

A marcação é designada por um vetor $M = (n_1, n_2, n_3, \dots, n_k)$ onde k é o número de lugares da rede, e n_i o número de fichas no lugar i .

As RP podem ser de capacidade infinita (cada lugar pode conter um número ilimitado de fichas) ou de capacidade finita (número de fichas limitado para cada lugar).

Estas redes aqui tratadas, são também definidas por RP Lugar/Transição (RP L/Tr).

As RP são definidas matematicamente como:

Definição 2.20 *Uma rede de Petri é uma sextupla*

$$RP = (P, T, A, K, W, M_0)$$

onde:

- $P = \{p_1, p_2, \dots, p_m\}$ é um conjunto finito de lugares;
- $T = \{t_1, t_2, \dots, t_n\}$ é um conjunto finito de transições;
- $A \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos;
- $K : P \rightarrow N \cup \{\infty\}$ é a função de capacidade;
- $W : A \rightarrow N^+$ é a função de ponderação;
- $M_0 : P \rightarrow N$ é a função de marcação inicial, que satisfaz $\forall p \in P : M_0(p) \leq K(p)$.

Denominamos estrutura de RP, uma rede sem marcação inicial e denotamo-la por $E_{rp} = (P, T, A, K, W)$.

Definição 2.21 *A variação da marcação da rede de Petri segue as seguintes regras de disparo das transições:*

1. *Uma transição t é dita estar habilitada (pronta para disparar) em uma marcação M se e só se*

$$\begin{aligned} &\forall p \in P \text{ que é entrada de } t : W(p, t) \leq M(p) \\ &\forall p \in P \text{ que é saída de } t : M(p) \leq K(p) - W(t, p); \end{aligned}$$

2. Uma transição habilitada pode ou não disparar;
3. O disparo de uma transição $t \in T$, habilitada na marcação M , é instantânea e resulta em uma nova marcação M' da rede dada pela equação:

$$M'(p) = M(p) - W(p, t) + W(t, p), \forall p \in P; \quad (2.13)$$

4. A ocorrência do disparo de t , que modifica a marcação M da rede para uma nova marcação M' , é denotada por $M[t > M']$, ou $M' = \delta(t, M)$, como analogia à função de próximo estado dos autômatos.

Damos abaixo um exemplo de uma RP com marcação inicial $M_0 = (2, 0, 0)$, e sua nova marcação, após o disparo da transição t :

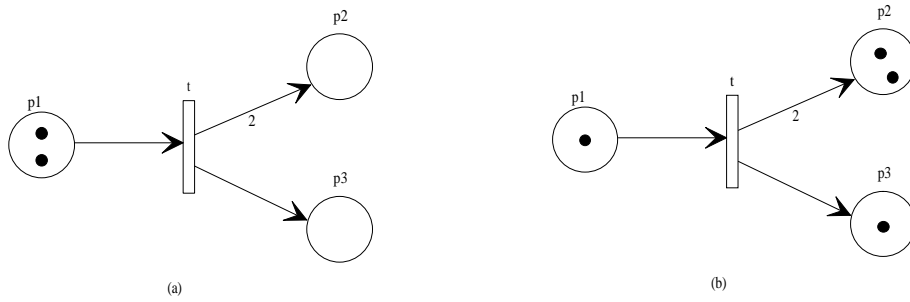


Figura 2.4: Exemplo de uma Rede de Petri antes e após o disparo de sua transição

Observamos que após o disparo de t , a nova marcação é $M_1 = (1, 2, 1)$ e, que esta marcação está de acordo com a regra 3 de disparo das transições.

O processo de execução de disparos de transições em uma RP pode ser contínuo, desde que haja, pelo menos uma transição habilitada em cada marcação atingida.

Definição 2.22 *Seja $R(E_{rp}, M_0)$, o conjunto de marcações alcançáveis de uma rede de Petri a partir de M_0 , tal que:*

- $M \in R(E_{rp}, M_0)$ e,
- se $M_1 \in R(E_{rp}, M_0)$ é para uma dada transição $t \in T$, $M_1[t > M_2]$, então

$$M_2 \in R(E_{rp}, M_0).$$

Esta definição nos indica que toda marcação alcançada a partir do disparo de uma transição t , também pertence ao conjunto de marcações alcançáveis da RP.

Quando fazemos uma seqüência de disparos de transições, encontramos uma seqüência de marcações possíveis, o que torna possível mapear todas as marcações possíveis, o qual é denominado de grafo de alcançabilidade.

Vemos, então que esta ferramenta ajuda na visualização de sistemas modelados além de facilitar na compreensão de seu comportamento dinâmico. Também, por ser uma ferramenta matemática, podemos usar equações de estado, equações algébricas, ou mesmo de linguagens formais ou lógica temporal, para descrever o comportamento dinâmico do sistema.

No próximo tópico, tratamos das redes de Petri com Funções de Habilitação de Transições, desde que utilizamos estas para a modelagem do supervisor neste trabalho.

2.8 Redes de Petri com Função de Habilitação de Transições

Uma das formas de controlar uma RP, é pondo restrições às habilitações de algumas transições nela, isto é, uma determinada transição só estará habilitada, se todos os parâmetros de uma função imposta a ela, satisfaçam a condição verdadeiro, ou seja, o valor 1 (um). Dessa maneira, embora nas condições normais, isto é, um lugar de entrada da transição tenha o número suficiente de fichas exigido pelo arco que os liga, ela não estará habilitada se este número não satisfizer a função de habilitação.

Definição 2.23 *Uma rede de Petri com Função de Habilitação de Transição, é uma quártupla*

$$RPFHT = (E_{rp}, K, l, M_0, \Phi)$$

em que

- $E_{rp} = (P, T, A, W)$ é uma estrutura de RP;
- $K : P \rightarrow N \cup \{\infty\}$ é a função de capacidade;
- $l : T \rightarrow \Sigma$, é a função que etiqueta as transições, onde Σ é um alfabeto;
- M_0 é a marcação inicial, como definido para as RPs;
- $\Phi = \{\phi_1, \dots, \phi_m\} : R(E_{rp}, M_0) \rightarrow \{0, 1\}$ é a função de habilitação das transições, que mapeia o conjunto de marcações alcançáveis em 0 ou 1;

Sua regra de disparo é definida assim:

Definição 2.24 *O estado, ou marcação, de uma RPFHT varia de acordo com a seguinte regra de disparo das transições:*

1. *Uma transição t_j é dita habilitada (para disparar) em M se e só se*

$$\begin{aligned} &\forall p \in P \text{ que é entrada de } t_j : W(p, t_j) \leq M(p); \\ &\forall p \in P \text{ que é saída de } t_j : M(p) \leq K(p) - W(p, t_j); \\ &\phi_j = 1; \end{aligned}$$

2. *Uma transição habilitada pode ou não disparar;*

3. *O disparo de uma transição $t_j \in T$, habilitada na marcação M , é instantânea e resulta em uma nova marcação M' dada pela equação:*

$$M'(p) = M(p) - W(p, t_j) + W(t_j, p), \forall p \in P; \quad (2.14)$$

4. *A ocorrência do disparo de t_j , que modifica a marcação M da rede para uma nova marcação M' , é denotado por $M[t > M']$, ou (em analogia à função de próximo estado δ dos autômatos) $M' = \delta(t_j, M)$.*

As RPFHTs têm uma função análoga às redes de Petri Controladas (RPCtl)[Kro87], pois eliminam o crescimento ou modificação da rede, em que tentamos evitar bloqueios, ou quando desejamos que a mesma siga uma dada seqüência de disparos de transições. Sua diferença básica encontra-se na forma de controle, ou seja, enquanto as RPCtls utilizam-se de lugares de controle para este fim, onde temos as fichas nestes, para controlar sua evolução, as RPFHTs usam funções definidas em suas transições, as quais criam condições específicas, de acordo com as fichas em determinados lugares da rede, para sua execução. Em outros termos, produzem o mesmo efeito por maneiras diferentes. Disto resulta em uma relação de equivalência em que, enquanto as RPCtls têm uma descrição mais algébrica, as RPFHTs são mais lógicas, o que se coloca como condição de melhor alternativa para os nossos propósitos.

Um exemplo de RPFHT está mostrado na figura 2.5, onde na transição t , está definida a função de habilitação dada por

$$\phi = [M(p_1) \geq 2 \wedge M(p_2) > 0 \wedge M(p_3) = 0].$$

Neste exemplo, vemos que pela função de habilitação ϕ , no caso da figura 2.5 (a), a transição t está habilitada podendo disparar, visto todos os parâmetros serem verdadeiros, tornando a função $\phi = 1$. Após o disparo, figura 2.5 (b), os dois primeiro parâmetros da função são verdadeiros, contudo o terceiro $M(p_3) \neq 0$, o que torna a função $\phi = 0$, desabilitando a transição t .

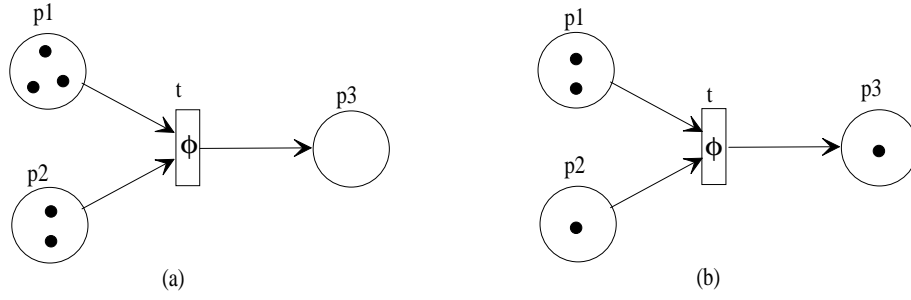


Figura 2.5: Exemplo de uma RPFHT

2.9 Linguagens de Redes de Petri

A linguagem gerada por uma rede de Petri é relativa ao disparo de transições que são etiquetadas com símbolos de um dado alfabeto. A etiquetagem das transições da rede com os símbolos do alfabeto é feita através da função de etiquetagem h definida como

$$h : T \rightarrow \Sigma.$$

Desta forma, uma rede de Petri que tenha esta característica é chamada de rede de Petri Etiquetada [Pet81b] [Jan87]. As seguintes definições apresentadas, são necessárias para podermos definir formalmente uma linguagem de RP:

Definição 2.25 *Seja T um conjunto finito de transições de uma RP. Denota-se por T^* o conjunto de todas as seqüências finitas de transições disparáveis, incluindo a seqüência vazia T^0 , sobre T e por T^∞ o conjunto de todas as seqüências infinitas de transições disparáveis.*

Definição 2.26 *Uma seqüência de transições $\theta \in T^*$ é chamada uma seqüência de disparos para uma rede de Petri, se a seqüência de disparos de transições é permitida pela regra de disparos das transições na rede de Petri RP e uma seqüência infinita de transições $\theta \in T^\infty$ é uma seqüência infinita de disparos de transições, se todo prefixo é uma seqüência de disparos.*

Definição 2.27 *O conjunto de todas as seqüências finitas de disparos da rede de Petri RP é denotado por $F(RP)$ e o conjunto de todas as seqüências infinitas de disparos da rede de Petri RP é denotado por $F_\infty(RP)$.*

Dessa forma, conhecendo-se estas definições, formalizamos uma linguagem de RP, do seguinte modo:

Definição 2.28 $L(RP)$ é uma linguagem de uma rede de Petri se

$$L(RP) \equiv \{h(\theta) \mid \theta \in F(RP)\}$$

e $L_\infty(RP)$ é uma linguagem infinita de rede de Petri, gerada da rede de Petri RP , se

$$L_\infty(RP) \equiv \{h(\theta) \mid \theta \in F_\infty(RP)\},$$

onde $h(\theta)$ é a função de etiquetagem das seqüências de transições que transformam-nas em uma linguagem.

Complementamo-las com a definição de *Linguagem Regular de Rede de Petri*.

Definição 2.29 Linguagem regular de rede de Petri $L_\infty^{fair}(RP) \subset L_\infty(RP)$ é uma linguagem infinita regular de rede de Petri que supõe a condição de regularidade: sempre que uma transição t é infinitamente habilitada, então a transição t eventualmente disparará.

A definição 2.29 é a que nos garante uma conexão entre as redes de Petri e a classe de lógica temporal denominada LPTL, estudada por Uchihira e Honiden [UH90].

Como exemplo, seja a RP na figura 2.6. As transições estão etiquetadas por $h(t_1) = \alpha$ e $h(t_2) = \beta$. Logo, como podemos ver, a seqüência de disparos das transições é

$$\theta = t_1 t_2 t_1 t_2 \dots$$

que através da função de etiquetagem $h(\theta)$, transforma-a na linguagem

$$h(\theta) = \alpha\beta\alpha\beta\dots$$

que podemos abreviar por $(\alpha\beta)^*$, que é a linguagem gerada pela RP.

2.10 Considerações Gerais

A formalismo apresentado neste capítulo, é necessário à compreensão do estudo desenvolvido, o qual se refere à abordagem da síntese de supervisores de SEDs, na teoria de controle supervisório (TCS), que trataremos no próximo capítulo.

Dentro das definições dadas, analisamos brevemente os sistemas, e também estudamos algumas redes de Petri. Consideramos de alta relevância a compreensão histórica da evolução dos sistemas e suas formas de modelagem, para entendermos o porquê da introdução das redes de Petri no contexto da TCS, desde que as mesmas tem se tornado um paradigma muito aceito na modelagem dos SEDs, para a síntese do supervisor.

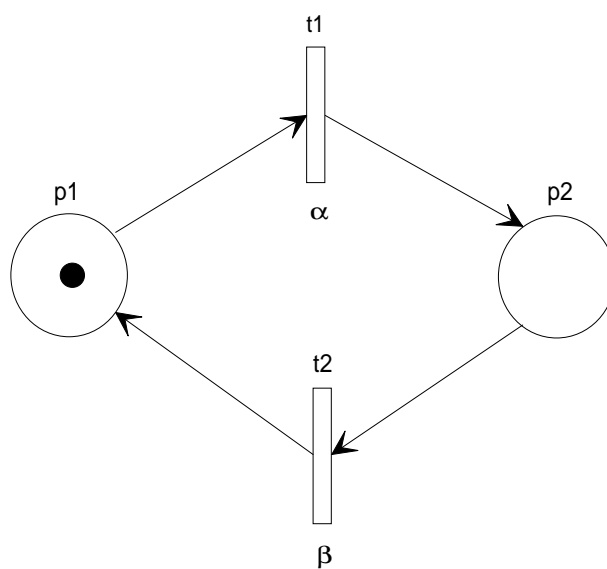


Figura 2.6: Exemplo de uma rede de Petri que gera a linguagem $(\alpha\beta)^*$

Capítulo 3

Teoria de Controle Supervisório

A Teoria de Controle Supervisório (TCS) foi desenvolvida por Ramadge e Wonham [RW89] baseada na teoria de linguagens formais e nos autômatos. Nesta, encontramos os SEDs representados por geradores, onde sua linguagem gerada é $L(G) = L$ e a linguagem marcada é $L_m(G) = L_m$.

O controle supervisório dá-se nas condições de colocarmos um controle externo ao gerador, de forma a assegurar que o gerador não seja um gerador espontâneo de cadeias de eventos. Desta forma, impomos condições para que o SED tenha um comportamento tal, que realize uma tarefa específica, inibindo a ocorrência de algumas seqüências de eventos fisicamente possíveis, todavia, indesejáveis. Logo, precisamos desabilitar tais eventos do sistema quando for desejado, a partir desse controle. Sendo assim, dividimos o alfabeto Σ gerado pelo gerador em duas categorias. Estes são os eventos controláveis e os eventos não controláveis, representados por Σ_c e Σ_u respectivamente. Assim, temos que o alfabeto Σ é a união entre Σ_c e Σ_u , ou seja

$$\Sigma = \Sigma_c \cup \Sigma_u.$$

Estes dois alfabetos definidos dentro do alfabeto Σ , não têm eventos comuns, isto é, sua interseção é o conjunto vazio, ou

$$\Sigma_c \cap \Sigma_u = \emptyset.$$

Os eventos controláveis em Σ_c são aqueles que podem ser habilitados ou desabilitados em qualquer momento, enquanto que os eventos não controláveis em Σ_u não sofrem ação de controle, como é o caso da quebra de uma máquina, ou falta de energia elétrica. O processo controlável pode ser exemplificado pelo início da operação da máquina. Logo, esta união entre eventos refletem características de sistemas reais.

Os eventos controláveis são habilitados ou desabilitados pela entrada de controle que é definida formalmente por:

Definição 3.1 Dado um gerador $G = (\Sigma, Q, \delta, q_0, Q_m)$, cujo alfabeto é particionado em $\Sigma = \Sigma_c \cup \Sigma_u$, o conjunto de entradas de controle associado a G é dado por:

$$\Gamma = \{\gamma : \Sigma_u \subseteq \gamma \subseteq \Sigma\}.$$

Os eventos de Σ_u são não controláveis, isto é, estão sempre habilitados, pois não sofrem influência da ação de controle.

Como estamos interessados no gerador controlado, definimo-lo do seguinte modo:

Definição 3.2 Dado $\Gamma \subseteq 2^\Sigma$ como sendo o conjunto de entradas de controle, define-se um gerador controlado G_c como um par (G, Γ) onde G é um gerador com alfabeto Σ , particionado em eventos controláveis Σ_c e eventos não controláveis Σ_u , equipado com um conjunto de entradas de controle Γ .

Denominamos *planta*, o modelo do sistema a ser controlado, semelhantemente à teoria clássica de controle. O comportamento do sistema na ausência de qualquer ação de controle é definida como *linguagem da planta*, a qual representa o comportamento do sistema.

Desde que apliquemos uma entrada de controle γ a uma planta, esta irá se comportar como se os eventos inibidos fossem eliminados de sua estrutura de transição. Dessa forma, em um gerador cujo alfabeto é particionado em eventos controláveis e não controláveis, a função de transição deste gerador não está definida para os eventos inibidos por uma dada entrada de controle que seja aplicada ao gerador em um determinado instante. Logo, como este é o mecanismo de controle adotado pela TCS, necessitamos, então, chavear as entradas de controle para especificar, a partir da linguagem gerada, determinadas tarefas a serem realizadas. Este mecanismo de controle nos leva às definições de supervisor e suas condições de existência.

3.1 Supervisores e Condições de Existência

Ao fundamentarmos a idéia de geradores controlados, estamos interessados em chavear a entrada de controle em resposta à cadeia de eventos previamente gerada pelo gerador G . Este chaveamento é feito pelo supervisor, que é o agente externo que determina a ação de controle.

Um supervisor é definido formalmente como:

Definição 3.3 Um supervisor para um gerador controlado $G_c = (G, \Gamma)$ é um par $S = (S, \Theta)$, composto de um gerador $S = (\Sigma, X, \xi, x_0, X_m)$ e de um mapa de controle Θ , em que:

- Σ é o mesmo alfabeto de G ;
- X é um conjunto de estados;
- $\xi : \Sigma^* \times X \rightarrow X$ é uma função de transição parcial estendida;
- $x_0 \in X$ é o estado inicial;
- $X_m \subseteq X$ é o conjunto de estados marcados;
- $\Theta : X \rightarrow \Gamma$ é uma função que associa a cada $x \in X$ uma entrada de controle $\gamma \in \Gamma$.

Vemos na figura 3.1, como é representado o sistema composto pelo gerador controlado G_c supervisionado pelo supervisor \mathbf{S} , em malha fechada. Nesta figura podemos ver que o gerador controlado recebe a ação de controle e realimenta o supervisor com os seus eventos gerados. Sendo assim, o gerador segue a especificação dada pelo supervisor.

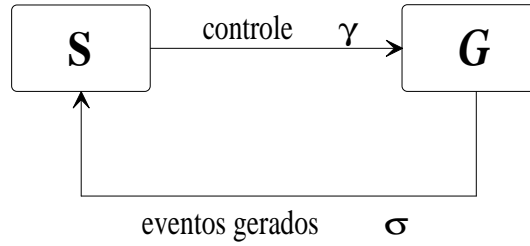


Figura 3.1: Supervisão de um SED

De acordo com a definição de supervisor, vemos que a ação de controle modifica as linguagens associadas ao gerador, pois pode inibir seqüências de eventos que antes podiam ocorrer. Logo, sua linguagem, pode ser assim definida:

Definição 3.4 *Dados um gerador G_c e um supervisor \mathbf{S} , a linguagem gerada pelo sistema supervisionado, denotada por $L(\mathbf{S}/G)$, é tal que*

$$\begin{aligned} e &\in L(\mathbf{S}/G) \quad e \\ s\sigma &\in L(\mathbf{S}/G) \text{ se e só se } s \in L(\mathbf{S}/G) \wedge s\sigma \in L(G) \wedge \sigma \in \Theta(\xi(s, x_0)), \end{aligned}$$

onde Θ é o mapa de controle que associa a cada estado $x \in X$ uma entrada de controle $\gamma \in \Gamma$, que é aplicada a G .

Observamos, então, que, pela definição 3.4, somente os eventos que estão habilitados em cada estado, é que ocorrem sob supervisão.

Devemos ver que, dado uma palavra s que pertença a $L(\mathbf{S}/G)$, também pertence a $L(G)$, o que nos dá

$$L(\mathbf{S}/G) \subseteq L(G).$$

Então, temos que $L(\mathbf{S}/G)$ é uma linguagem prefixo-fechada, visto que uma palavra $s\sigma \in L(\mathbf{S}/G)$ somente se $s \in L(\mathbf{S}/G)$. Desta forma, a linguagem controlada do sistema supervisionado é definida por

$$L_c(\mathbf{S}/G) = L(\mathbf{S}/G) \cap L_m(G), \quad (3.1)$$

ou seja, $L_c(\mathbf{S}/G)$ é a parte da linguagem marcada original sob ação de controle que representa as tarefas que são completadas sob supervisão. Logo, isto implica nas condições:

$$L_c(\mathbf{S}/G) \subseteq L(\mathbf{S}/G)$$

e

$$L_c(\mathbf{S}/G) \subseteq L_m(G)$$

Disso decorre que a linguagem marcada do sistema é

$$L_m(\mathbf{S}/G) = L_c(\mathbf{S}/G) \cap L_m(S) \quad (3.2)$$

Tendo em vista estas particularidades, concluímos que

$$L_m(\mathbf{S}/G) \subseteq L_c(\mathbf{S}/G) \subseteq L(\mathbf{S}/G) \subseteq L(G),$$

ou seja, a linguagem $L(\mathbf{S}/G)$, que é gerada pelo sistema composto pelo supervisor e o gerador controlado \mathbf{S}/G_c , pode ser interpretada como o conjunto de todas as possíveis seqüências finitas de eventos que têm possibilidade de ocorrer no sistema.

Precisamos garantir que os eventos no supervisor \mathbf{S} só devam ocorrer, quando eles também ocorrerem em G_c e estiverem habilitados por Θ . Para tanto, é necessário que, para todo $s \in \Sigma^*$, e $\sigma \in \Sigma$, as seguintes condições sejam verdadeiras:

$$\begin{aligned} s &\in L(\mathbf{S}/G), \\ s\sigma &\in L(G) \quad \text{e} \\ \sigma &\in \Theta(\xi(s, x_0)), \end{aligned}$$

que juntas implicam na condição de que

$$\xi(s, x_0)!, \text{ isto é, } \xi(s, x_0) \text{ é definido.}$$

Esta relação feita é a condição do supervisor \mathbf{S} ser dito completo em relação a um gerador controlado G_c . Isto nos diz que, se s é uma palavra que pode ocorrer

no sistema supervisionado e o evento σ é uma continuação fisicamente possível desta palavra, e este evento está habilitado, então a palavra $s\sigma$ deve estar definida na função de transição do supervisor.

Impomos mais duas restrições a serem satisfeitas pelas linguagens $L(\mathbf{S}/G)$, $L_c(\mathbf{S}/G)$ e $L_m(\mathbf{S}/G)$, que são:

Definição 3.5 *Um supervisor \mathbf{S} é dito não bloqueável se e só se*

$$\overline{L_c(\mathbf{S}/G)} = L(\mathbf{S}/G) \quad (3.3)$$

e,

Definição 3.6 *Um supervisor \mathbf{S} é dito não rejeitável se e só se*

$$\overline{L_m(\mathbf{S}/G)} = \overline{L_c(\mathbf{S}/G)}. \quad (3.4)$$

Destas duas definições, podemos ver que um supervisor é bloqueável se existir pelo menos uma palavra fisicamente possível em $L(\mathbf{S}/G)$ que não é prefixo de qualquer palavra em $L_c(\mathbf{S}/G)$, isto é, o sistema nunca pode completar uma tarefa especificada. Por outro lado, um supervisor é rejeitável caso exista ao menos uma palavra em $\overline{L_c(\mathbf{S}/G)}$ representando uma palavra completada, contudo que não pertence a linguagem marcada $L_m(\mathbf{S}/G)$. Neste último caso, é possível atingir um estado em que nenhuma tarefa seja reconhecida. Para a síntese do supervisor ser completa, impomos que ele não tenha nenhum desses problemas, o que nos leva a definição de supervisor próprio, que é a seguinte:

Definição 3.7 *Um supervisor completo, não bloqueável e não rejeitável, é dito ser um supervisor próprio. Em consequência, um supervisor próprio é um supervisor completo tal que*

$$\overline{L_m(\mathbf{S}/G)} = \overline{L_c(\mathbf{S}/G)} = L(\mathbf{S}/G). \quad (3.5)$$

Visto isto, chegamos ao problema principal da TCS, o qual encontra-se em modificar o comportamento de um SED G . Para tanto, definiremos dois conceitos básicos necessários:

Definição 3.8 *Sejam duas linguagens $K, L \subseteq \Sigma^*$, K é dita fechada em relação a L , ou L — fechada se e só se*

$$K = \overline{K} \cap L$$

e,

Definição 3.9 Dadas duas linguagens $K, L \subseteq \Sigma^*$ e um alfabeto $\Sigma = \Sigma_c \cup \Sigma_u$, diz-se que K é L – controlável se e só se

$$\overline{K}\Sigma_u \cap L \subseteq \overline{K}.$$

Estas definições são os conceitos conhecidos por *fechamento* e *controlabilidade*, respectivamente.

Resta-nos agora, determinar as condições de existência do supervisor para a realização de uma dada tarefa. É dada, então, a seguinte proposição:

Proposição 3.1 Seja \mathbf{S} um supervisor completo para G_c . Então $L(\mathbf{S}/G)$ é prefixo-fechada e $L(G)$ – controlável.

Também é necessário estabelecer as condições de existência de supervisores para os problemas formulados em termos de linguagens geradas e marcadas. Damos para isto os teoremas a seguir:

Teorema 3.1 Dados um gerador G tal que $L(G)$ represente seu comportamento fisicamente possível e uma linguagem especificada $K \subseteq L(G)$, existe um supervisor completo \mathbf{S} tal que $L(\mathbf{S}/G) = K$ se e só se K for prefixo-fechada e $L(G)$ -controlável.

Teorema 3.2 Dados um gerador G tal que $L_m(G)$ represente as tarefas que podem ser completadas pelo sistema na ausência de qualquer ação de controle e uma linguagem especificada $K \subseteq L_m(G)$ então

1. Existe um supervisor não bloqueável \mathbf{S} tal que $L_c(\mathbf{S}/G) = K$ se e só se K for $L(G)$ -fechada e $L(G)$ -controlável;
2. O supervisor \mathbf{S} será próprio se e só se o gerador $S = (\Sigma, X, \xi, x_0, X_m)$ for tal que $X_m = X$.

Devemos resolver as equações

$$L(S) = K \quad \text{e} \quad (3.6)$$

$$\Theta(x) = \gamma : \gamma \in \Gamma \wedge \gamma \supseteq \Sigma_x^1 \wedge \gamma \cap \Sigma_x^0 = \emptyset \quad (3.7)$$

para encontrar o gerador \mathbf{S} e o mapa de controle Θ , desde que o supervisor é assegurado pelas condições do Teorema 3.1.

Na equação 3.7, temos Σ_x^0 como sendo o conjunto de eventos σ , possíveis de ocorrerem fisicamente após $s \in K$ e $s\sigma \in K$, a serem desabilitados quando o sistema se

encontrar no estado x , ou seja, não devem pertencer ao mapa $\Theta(x)$, enquanto que Σ_x^1 é o conjunto de eventos σ , que são continuções das palavras s tais que $s\sigma \in K$, e que devem estar habilitados quando s se encontrar no estado x .

Quando a linguagem K não satisfaz as condições exigidas à existência do supervisor, necessitamos encontrar uma sublinguagem $K^\dagger \subseteq K$, a qual é sempre possível e satisfaz todas as condições restritivamente. Esta sublinguagem é conhecida por *Suprema Linguagem Controlável* K^\dagger , ou $SupC(L)$, que soluciona o problema do supervisor, contudo restringe seus resultados ao mínimo, de forma a evitar problemas na execução de uma tarefa devido a eventos alheios e indesejáveis ao sistema.

3.2 Abordagem para Síntese de Supervisores por Redes de Petri

Barroso [Bar96] descreve uma outra forma de abordagem para a síntese de supervisores utilizando-se das redes de Petri para a modelagem do gerador G e do supervisor S , devido às vantagens apresentadas por estas, pois permitem uma descrição mais detalhada do sistema além da síntese modular de sistemas compostos por subsistemas interagentes. Devemos observar que, nesta abordagem, Barroso [Bar96] utiliza também de uma sistemática modular para a determinação do supervisor, baseado em sistemas de *G-nets* [Per94] juntamente com *Redes de Petri com Temporização Nebulosa* (RPTN) [Fig94] para as propriedades de tolerâncias a falhas dependentes do tempo.

Podemos ver na figura 3.2, o diagrama em blocos que nos mostra a utilização das redes de Petri na modelagem, análise e controle de um SED. Nela, vemos que, a partir do modelo do sistema, sintetiza-se o controlador baseado em uma ou mais especificações de tarefas a serem realizadas pelo sistema sob supervisão. Dessa forma, esta abordagem se torna mais flexível, visto que, dado o modelo de redes de Petri do sistema, várias tarefas podem ser especificadas em diferentes tempos, provocando mudanças apenas no supervisor. Então, o problema encontra-se na etapa de síntese do supervisor, a qual está apresentada na figura 3.3, onde vemos que ela é realizada tendo como dados o modelo do sistema e seu comportamento desejável.

A especificação funcional, tem por base, explicitar uma tarefa física por meio de uma linguagem, como já dito anteriormente. Devemos entrar com o espaço de estados, que é a árvore de alcançabilidade da rede de Petri que modela o sistema, e a especificação de comportamento requerida para o algoritmo que irá determinar se tal especificação é factível ou qual sua aproximação menos restritiva, isto é, a especificação possível, ou suprema linguagem controlável, como podemos ver na figura 3.3. Disto, procedemos

à determinação das funções de transição que irão compor a rede de Petri supervisora (RPFHT).

Neste trabalho adotamos a mesma abordagem proposta por Barroso [Bar96] para a síntese do supervisor. Entretanto, de forma mais específica nosso trabalho objetiva formalizar a especificação do comportamento desejado através do emprego da Lógica Temporal (LT). A formalização desta especificação através da LT proporcionará uma maior consistência do supervisor já que as ambigüidades inerentes da linguagem natural são eliminadas. Por outro lado, caso desejemos redefinir a especificação, apenas entramos com a nova especificação do comportamento desejado em LT, e executamos novamente o algoritmo que nos dará as novas funções das transições da RPFHT supervisora, sem modificar o modelo do SED.

Assim, como colocado anteriormente, utilizaremos da Lógica Temporal (LT) como forma de especificar comportamentos de um dado SED, através da CTL. A idéia de utilizar a CTL, é que esta pode definir um determinado comportamento com eventos que ocorram em paralelo (concorrência), o que não é possível com as linguagens formais.

Devemos considerar que, embora a LT seja muito mais abrangente, ela exige um conhecimento muito específico e um tratamento mais lógico e matemático por parte do usuário, o que não acontece com as linguagens formais, que são mais simples em sua concepção.

Definido a especificação funcional, ou de comportamento desejado, precisamos proceder às condições de existência do supervisor para um dado SED. Como já visto, a condição necessária e suficiente para a existência de um supervisor de um SED é o conceito de controlabilidade, em que um gerador controlado G_c gera uma linguagem, a qual é prefixo da linguagem do gerador não controlado G . Como temos que esses geradores estão modelados por RPs, o gerador controlado G_c tem um conjunto próprio de todas as seqüências de transições que podem disparar e que sempre será um subconjunto do conjunto de seqüências de transições disparáveis do gerador não controlado G . Isto dá-nos a possibilidade de execução síncrona da RP que modela o sistema juntamente com o mesmo, de acordo com as restrições impostas ao seu comportamento dinâmico. Neste caso, essas restrições são dadas pelas funções de habilitação de transições impostas às transições na RPFHT supervisora, de forma a evitar um comportamento não desejado. Assim, a rede supervisora segue uma seqüência de transições requerida, impedindo bloqueios ou comportamentos não permitidos ao sistema, fazendo com que o mesmo realize uma tarefa desejada.

Os estudos baseados nessa abordagem inclui o *Algoritmo Modificado da Árvore de Alcançabilidade* (AMArA) e o *Algoritmo para a Contrução da Suprema Lingua-*

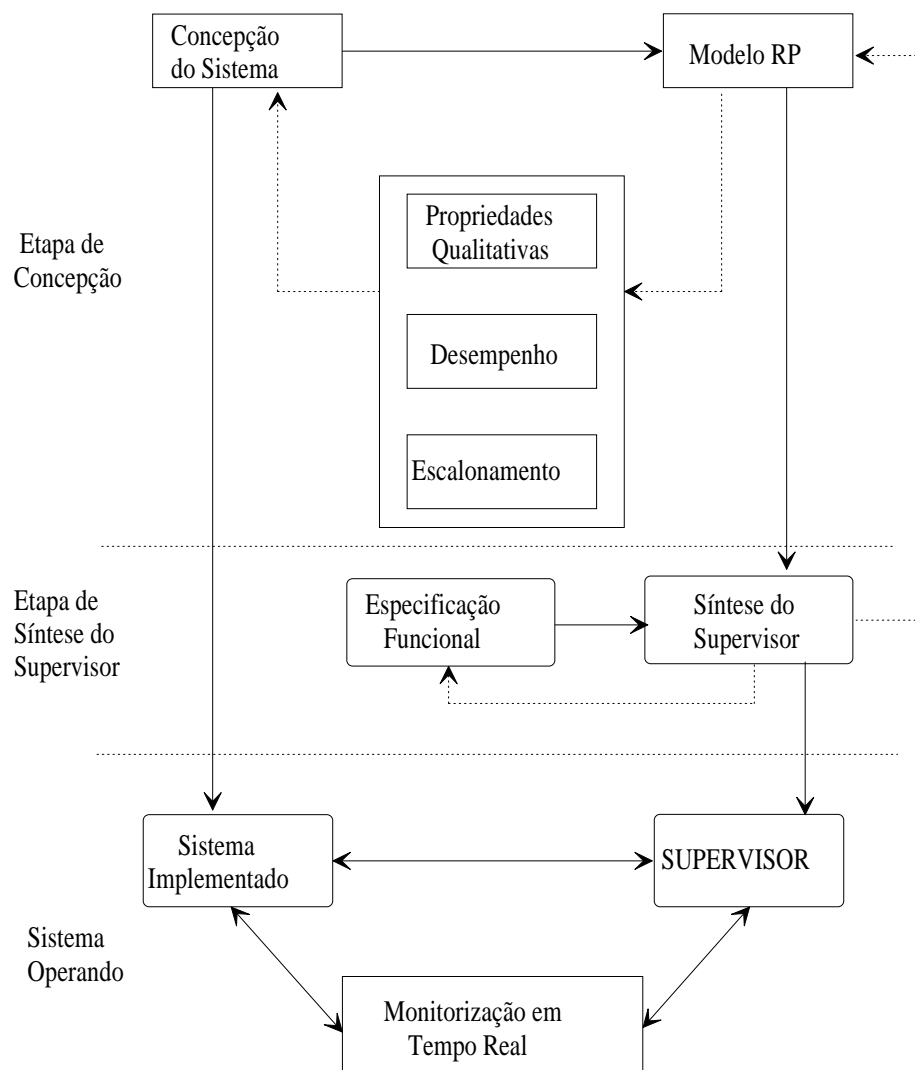


Figura 3.2: Utilização de redes de Petri e teoria de controle Supervisório para a concepção, análise e controle de um SED

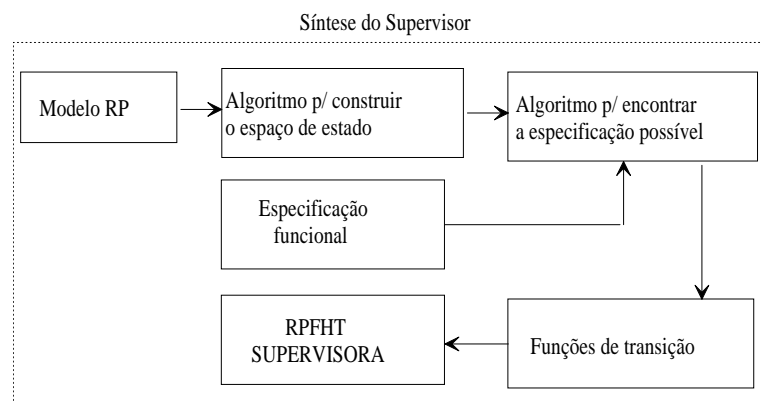


Figura 3.3: Diagrama em blocos do procedimento de síntese do supervisor

gem Controlável (ACGS), que são os algoritmos responsáveis, respectivamente, pela construção da árvore de alcançabilidade associada à RP que modela o sistema e pela construção da Suprema Linguagem Controlável $SupC(L)$ que será aplicado a RPFHT supervisora por intermédio das funções de habilitação de transições. Estes algoritmos serão melhor explicitados no capítulo 5 desta dissertação.

Devemos fazer alusão a que, uma abordagem deste tipo apresenta grandes vantagens para aplicações industriais, visto que nas RPs, devido a sua atual interface computacional, podemos ver todos os estados imediatos do sistema, dando melhor visualização e acompanhamento total da execução de uma tarefa, o que não é possível no caso dos autômatos.

Capítulo 4

Redes de Petri e Lógica Temporal

4.1 Introdução

Na filosofia, temos uma ramificação conhecida por lógica, que tem por objetivo formalizar o raciocínio humano, permitindo tratar de problemas do cotidiano científico, utilizando métodos sistemáticos para expressar idéias com precisão e sem ambigüidades, melhorando o conhecimento que se tem do mundo ao nosso redor.

A lógica é tratada por dois enfoques diferentes: a lógica matemática e a lógica dialética. A lógica dialética se ocupa da representação do conhecimento por meio da linguagem natural, usando técnicas de argumentação, e apelando fortemente para o conhecimento heurístico do objeto de estudo em questão. Não trataremos dela aqui. Ocupar-nos-emos apenas com a lógica matemática, mais útil no tratamento dos problemas que se seguirão.

Em Copi [Cop68], lógica é definida como o estudo dos métodos e princípios usados para distinguir o raciocínio correto do incorreto.

Para expressar sentenças declarativas que encerrem em si um sentido completo e as quais se possam atribuir um valor lógico por vez (verdadeiro ou falso), mas não ambos (princípio do terceiro excluído), usamos proposições. Por exemplo, *a neve é branca*, é uma proposição, dita também proposição atômica, fórmula atômica, fórmula ou simplesmente, átomos.

A partir de proposições, podemos criar proposições compostas usando conectivos lógicos. Exemplos de proposições compostas são: *a neve é branca e o céu está encoberto*. Temos aqui, o conectivo *e*. Na lógica proposicional, usamos os seguintes conectivos lógicos: \neg (NOT), \wedge (AND), \vee (OR), \rightarrow (implica) e \leftrightarrow (se e só se). Estes cinco conectivos lógicos podem ser usados para compor proposições a partir de proposições, de maneira recursiva, conforme abaixo:

Definição 4.1 *Fórmulas em lógica proposicional, são definidos recursivamente como segue:*

1. Uma proposição atômica é uma fórmula;
2. Se g é uma fórmula, então $\neg g$ é uma fórmula;
3. Se g e h são fórmulas, então $g \wedge h$, $g \vee h$, $g \rightarrow h$ e $g \leftrightarrow h$, são fórmulas
4. Todas as fórmulas são geradas aplicando-se as regras acima.

A ordem crescente de prioridade na efetuação das operações é:

$$\leftrightarrow, \rightarrow, \wedge, \vee, \neg$$

e requer que o conectivo de mais alta ordem seja avaliado primeiro. Assim, a fórmula

$$p \rightarrow q \wedge r$$

fica

$$(p \rightarrow (q \wedge r))$$

e

$$p \rightarrow q \wedge \neg r \vee s$$

será

$$(p \rightarrow (q \wedge ((\neg r) \vee s))) .$$

Definição 4.2 *Uma interpretação de uma fórmula é a avaliação de seu valor lógica, uma vez estabelecidos os valores lógicos das fórmulas componentes, a qual pode ter valor verdadeiro ou falso, mas não ambos.*

Definição 4.3 *Uma fórmula é dita ser consistente, quando ela assume valor verdadeiro em pelo menos uma interpretação de sua tabela verdade.*

Definição 4.4 *Uma fórmula é dita inconsistente quando ela assumir valor falso em todas as interpretações de sua tabela verdade.*

Definição 4.5 *Uma fórmula é dita ser válida, ou uma tautologia, quando assume valor lógico verdadeiro em todas as interpretações de sua tabela verdade.*

Se houver n proposições atômicas distintas na fórmula, teremos 2^n interpretações distintas da fórmula.

No caso da linguagem natural, encontramos o problema da ambigüidade, necessitando da sua resolução antes de sua formalização, como no exemplo "todos os marinheiros têm uma madrinha", que pode significar que cada um tem a sua madrinha, mas também que todos têm uma única madrinha. Este caso não se aplica às proposições matemáticas, desde que elas já se apresentam semiformalizadas.

4.2 Lógica Temporal no Contexto das Redes de Petri

A Lógica Temporal (LT), é uma poderosa ferramenta já conhecida desde muito tempo. Pnueli [Pnu77], propôs sua utilização em questões de programas concorrentes. A mesma proporciona a base formal para a descrição e raciocínio relativos ao comportamento correto de sistemas, especialmente de sistemas não seqüenciais, expressando propriedades relativas à ocorrência de eventos no tempo.

Existem basicamente duas versões de lógica temporal que são a lógica temporal linear e a lógica temporal de tempo ramificado. Na lógica temporal linear considera-se que as execuções do sistema, as quais representam seu comportamento e são representadas tanto por seqüências de eventos ordenados no tempo ou por seqüências de eventos em ordem causal, não são relacionadas. Por outro lado, na lógica temporal de tempo ramificado as execuções do sistema são representadas por seqüências de eventos ordenados no tempo ou por seqüências de eventos em ordem causal e estão relacionadas numa estrutura em árvore com ramos que indicam alternativas e conflitos.

A lógica temporal provê uma técnica para especificar sistemas, especialmente sistemas não seqüenciais, e para formular e provar propriedades destes sistemas. Isto se refere à semântica operacional para a qual considera-se que cada execução do sistema é representada por seqüências de estados e transições. As propriedades de segurança (*safety*) e vivacidade (*liveness*) podem ser caracterizadas através de fórmulas CTL. Na propriedade de segurança objetiva-se separar os estados desejáveis dos indesejáveis enquanto que na propriedade de vivacidade procura-se determinar se alguns estados desejados serão alcançados.

Assim, a partir da LT, podemos descrever o comportamento dinâmico de uma rede de Petri. Nosso trabalho, então, consiste em descrever as formas de criação e análise de fórmulas em LT, de maneira que possamos determinar o comportamento de um determinado sistema que esteja modelado por uma rede de Petri.

Na LT, as modalidades temporais podem ser combinadas para expressar regularmente afirmações sobre presente, passado e futuro de uma proposição dada, a partir de um determinado estado em que um sistema se encontre. No contexto da Teoria de Controle Supervisório, este estado é o estado inicial q_0 , ou a marcação inicial da rede de Petri que modela o sistema, M_0 , visto que estamos interessados em modelos de sistemas de automação industrial.

A classe LT que aqui utilizamos, é a lógica de tempo ramificado conhecido como *Computation Tree Logic* (CTL), na qual os operadores temporais ocorrem apenas aos

pares consistindo de A_l (necessariamente) ou E_l (possivelmente), seguido por F_l (eventualmente), G_l (sempre), U (até que) ou X (próximo). Aqui, operadores de tempo passado não são permitidos e, além do mais, os operadores formais não podem ser combinados diretamente com os conectivos proposicionais. Enfatizamos que o operador temporal F_l traduz a informação de que uma fórmula, tem pelo menos um caminho possível de ser seguido.

Precisamos determinar a sintaxe e a semântica da CTL, para podermos trabalhar com a mesma.

A sintaxe de fórmulas CTL é definida como se segue:

Definição 4.6 *Uma fórmula em CTL, é construída de*

1. *Um conjunto de proposições atômicas:*

$$\text{Prop} = \{p_1, p_2, \dots\};$$

2. *Conectivos lógicos: \wedge (AND), \vee (OR), \neg (NOT);*

3. *Operadores temporais: X (next), U (until).*

Desse modo, as regras de formação de fórmulas são, de acordo com a definição 4.1 de fórmula em lógica proposicional:

Definição 4.7 *Seja Prop um conjunto finito. O conjunto $L(\text{Prop})$ é o menor conjunto de fórmulas proposicionais temporais definidos sobre Prop tais que*

- i) $\text{Prop} \subseteq L(\text{Prop})$;
- ii) *Se $f \in L(\text{Prop})$ então $\neg f, G_l f, A_l X f, \dots, X f \in L(\text{Prop})$;*
- iii) *Se $f, g \in L(\text{Prop})$ então $f \wedge g, f \vee g, E_l (f U g), A_l (f U g), \dots \in L(\text{Prop})$.*

Então, de acordo com a definição 4.7, são fórmulas CTL todas e quaisquer junções possíveis entre os conectivos lógicos, operadores temporais e fórmulas CTL.

Vemos então, que para definir uma fórmula CTL, necessitamos de um conjunto de proposições atômicas (ou fórmulas) $\text{Prop} = (p_1, p_2, \dots)$, conectivos lógicos ($\wedge = \text{AND}$, $\vee = \text{OR}$, $\neg = \text{NOT}$) e os operadores temporais X (next) e U (until).

Os operadores F_l e G_l são abreviações para

$$\begin{aligned} & \text{true } U f \text{ e} \\ & \neg F \neg f \end{aligned}$$

respectivamente, nas fórmulas $F_l f$ e $G_l f$, sendo f uma fórmula CTL. Também temos que o conectivo lógico \vee (ou) na fórmula

$$f \vee g$$

abrevia

$$\neg(\neg f \wedge \neg g),$$

e as fórmulas

$$f \supset g \text{ e}$$

$$f \Leftrightarrow g$$

abreviam respectivamente

$$\neg f \vee g \text{ e} \\ (f \supset g) \wedge (g \supset f),$$

com f e g sendo proposições atômicas ou fórmulas CTL. Logo, podemos trabalhar apenas com o conjunto $Prop$, os conectivos lógicos \wedge e \neg e os operadores U , X , A_l e E_l , pois os outros podem ser deduzidos a partir destes.

A veracidade ou falsidade de fórmulas é definido com respeito ao modelo de Kripke, porém é não-padrão, onde, para a CTL, o modelo é uma tripla (Q, R, L) , onde Q é o conjunto de estados, R é a relação de transição e L é a avaliação. A relação de transição é o conjunto de todos os pares (q, q') , tal que q' é um sucessor imediato de q .

A semântica para CTL é determinada a partir do modelo de Kripke. Um caminho de um modelo de Kripke $K=(Q, R, L)$ é uma seqüência infinita de estados $(q_0, q_1, \dots) \in Q^\infty$ tal que cada par sucessivo de estados (q_i, q_{i+1}) é um elemento de R , e Q^∞ é definido como o conjunto de todos as seqüências de estados possíveis.

Na semântica da CTL, usamos a notação

$$K, q \models f,$$

significando que f é verdadeiro no estado q do modelo de Kripke K . A interpretação de uma fórmula CTL f com relação ao de modelo de Kripke, é dado sobre a estrutura das fórmulas [McM92].

No nosso trabalho, não utilizamos da semântica definida com respeito ao modelo de Kripke, pois apenas necessitamos de verificar a validade da fórmula sobre a árvore de alcançabilidade da rede de Petri que modela o sistema.

De acordo com a sintaxe da CTL, podemos exemplificar algumas fórmulas, de acordo com a semântica da CTL:

1. $E_l q$ — significa que a proposição q é possivelmente verdadeira;

2. $A_l q$ — implica que a proposição q é necessariamente verdadeira;
3. $f U g$ — quer dizer que a proposição f é verdadeira até que g seja;
4. Xp — impõe que a proposição p é verdadeira para o próximo estado;
5. $A_l X(p \wedge \neg q)$ — necessariamente, no próximo estado, p e a negação de q , são verdadeiros;
6. $G_l q$ — a proposição q é sempre verdadeira;
7. $E_l G_l f$ — possivelmente, f é sempre verdadeira (pelo menos em um caminho percorrido);
8. $A_l G_l t$ — necessariamente t é sempre verdadeira;
9. $E_l F_l p$ — nos diz que, possivelmente, p pode ser verdadeira;
10. $E_l X(q U (f \wedge p))$ — expressa que, possivelmente o próximo estado de q é verdadeiro, até que f e p , ao mesmo tempo, sejam também;
11. $F_l(f U \neg Xg)$ — eventualmente, f é verdadeira até que o próximo estado de g seja falso, ou seja, f é verdadeira, até o próximo estado de g . Em outros termos, existe ao menos um caminho que torna a fórmula verdadeira.

Observemos que, a diferença entre a análise direta e a definição do modelo de Kripke, é mínima, pois no caso do item 1, teríamos apenas $K, s \models E_l q$, que significa que q é possivelmente verdadeira no estado s . Logo, na especificação de comportamentos, que é o nosso objetivo, utilizaremos a análise direta da fórmula, desde que o estado considerado para os nossos propósitos é sempre o estado inicial q_0 .

Vemos, então, que podemos fazer uma avaliação de toda a mudança no tempo, de proposições dadas, a partir das fórmulas CTL.

Para fazermos uma conexão da LT com as RPs, utilizamo-nos desta classe de LT, pois é ela que dá-nos as possibilidades de decisão entre os vários ramos do grafo de alcançabilidade da RP, além de poder definir propriedades concorrentes e conflitantes na mesma, onde podemos habilitar mais de uma transição ao mesmo tempo, ou fazer uma decisão em um conflito, ou concorrência da rede.

No caso da LPTL, sua utilização é basicamente para estruturas que não apresentem concorrências entre as transições, isto é, só haja uma única transição habilitada em um determinado instante (*condição de evento singular*) [UH90]. Para estas, os autômatos seqüenciais de Büchi são utilizados como interligação. Como nosso problema é mais

abrangente, passaremos a estudar a CTL que envolve toda uma gama de problemas mais reais à TCS.

Definimos a linguagem gerada de uma fórmula CTL da seguinte forma:

Definição 4.8 *Seja f uma fórmula CTL. $L_s(f)$ é uma linguagem gerada da fórmula CTL f que representa seqüências de proposições pertencentes ao conjunto Prop .*

Então, podemos proceder à ligação das RPs com a CTL. Dessa forma, definimos formalmente a ligação das RPs com a CTL, pela condição das seqüências de transições de uma RP com transições etiquetadas, da seguinte forma:

Definição 4.9 *Seja $RP = (P, T, A, W, M_0)$ uma rede de Petri e $\text{Prop} \subset T$ um conjunto de transições que está contido em uma fórmula CTL f . Então*

$$L(RP, f) \equiv L_\infty(RP) \cap L_s(f)$$

onde $L_\infty(RP)$ é a linguagem gerada da rede de Petri etiquetada

$$RP = (P, T, A, W, \Omega, M_0)$$

com $\Omega : T \rightarrow \text{Prop}$, e $L_s(f)$ é a linguagem gerada da fórmula CTL f .

Então, a linguagem da RP etiquetada [Jan87][Pet81b] com símbolos do conjunto Prop , terá sua linguagem relacionada com a linguagem da fórmula CTL.

Devemos observar que esta nova linguagem $L(RP, f)$ está definida por que, nem sempre, necessariamente, todas as transições da rede de Petri estão ligadas a uma proposição atômica do conjunto Prop , isto é, algumas podem ser invisíveis, para uma dada fórmula CTL f .

O seguinte teorema nos garante a decidibilidade do problema de nulidade da linguagem gerada da fórmula CTL f .

Teorema 4.1 *Para uma dada rede de Petri $RP = (P, T, A, W, M_0)$ e uma fórmula CTL f composta de um conjunto de proposições atômicas Prop , o problema de nulidade de $L(RP, f)$ é decidível.*

O problema de *nulidade* é definido como a avaliação da especificação dada com a fórmula CTL sobre uma RP. Em outros termos, se existe uma seqüência válida de disparos de transições na RP que satisfaça a especificação em CTL f .

Tendo estas definições formalizadas, podemos analisar um exemplo, onde utilizamos a CTL, como verificação de comportamento do modelo, para melhor situar seu uso:

Exemplo 4.1 *Considere a RP mostrada na figura 4.1. Esta rede, tem seu grafo de alcançabilidade, mostrado na figura 4.2. Se definirmos as transições t_1, t_2 e t_3 como sendo as proposições atômicas, podemos verificar se uma fórmula CTL é verdadeira, de acordo com o que se segue:*

- *Se desejarmos que a rede siga sempre os disparos de transições t_1 e t_2 , nesta ordem, a fórmula CTL que devemos colocar, deve ser $G_1(t_1 U t_2)$, desde que a linguagem gerada por esta fórmula é $(t_1 t_2)^*$ e também, a mesma pertence a linguagem gerada da rede de Petri;*
- *Se quisermos que a rede tenha o comportamento especificado, como $t_1 t_2 t_3$, poríamos a fórmula como $A_1(t_1 U t_2) \wedge X(t_1 \supset X t_3)$. Contudo, observamos que a fórmula não é possível para a marcação inicial dada, pois esta seqüência não é possível diretamente, como podemos ver na árvore de alcançabilidade da mesma. Observemos que, a partir do estado inicial (marcação inicial da rede) a seqüência $t_1 t_2 t_3$, não existe. Logo, a linguagem gerada da fórmula não está contida na linguagem da RP;*
- *Uma seqüência da forma $(t_1 t_2 t_1 t_2 t_3)^*$, pode ser especificada com o uso da fórmula $G_1((t_1 U X t_2) U X t_3)$. Observemos aqui, que, a primeira parte da equação nos dá uma imposição de que, t_1 deve disparar, seguida do disparo de t_2 . Todavia, o disparo de t_3 será obrigatório, após novo disparo da seqüência $t_1 t_2$, que torna t_3 habilitada;*
- *A especificação de uma seqüência em que, tenhamos mais de uma transição habilitada de uma única vez, é feita com a fórmula $A_1(t_1 U t_2) \vee A_1(t_2 \wedge t_3)$. Nesta especificação, habilitamos t_1 até que t_2 seja habilitada. Depois de duas seqüências desta, a transição t_3 está habilitada e, com o disparo de t_1 , novamente, t_2 e t_3 , podem disparar conjuntamente.*

Desse forma, concluímos que uma fórmula é uma sentença que será verdadeira se houver um caminho, na árvore de alcançabilidade da RP, possível de ser seguido.

Uma das formas de ligação entre as Redes de Petri e a CTL, é dada pelos conceitos de Estruturas de Predicados (EP), as quais são usadas como representação simbólica do conjunto de marcações de uma RP, junto a um conjunto de operações associadas, como demonstra Racloz e Buchs [RB97], ou McMillan [McM92], com a verificação simbólica do modelo, que é restrito à sistemas de estados finitos. Esta é usada para gerar e provar propriedades de sistemas baseados em transições, que é o caso dos SEDs. Nesta, sendo dada uma RP com seu conjunto de marcações, podemos analisar se uma

determinada fórmula CTL é verdadeira. Contudo, no contexto da TCS, podemos proceder à verificação da veracidade de uma fórmula CTL, apenas com relação à árvore de alcançabilidade do sistema modelado por rede de Petri, desde que temos já todo o conjunto de marcações possíveis, e também, todas as seqüências que podem ser seguidas na rede. Dessa forma, as proposições atômicas utilizadas em uma fórmula, estão mapeadas diretamente às transições do modelo, isto é, cada proposição atômica CTL, representa uma transição no conjunto de transições da RP, ou seja,

$$\text{Prop} = \Omega(T),$$

onde Ω é um operador que mapeia cada transição da rede com uma determinada proposição atômica do conjunto Prop, isto é

$$\Omega : T \rightarrow \text{Prop}$$

onde podemos defini-la como a função de etiquetação das transições no conjunto Prop, como já mostrado anteriormente na definição 4.9.

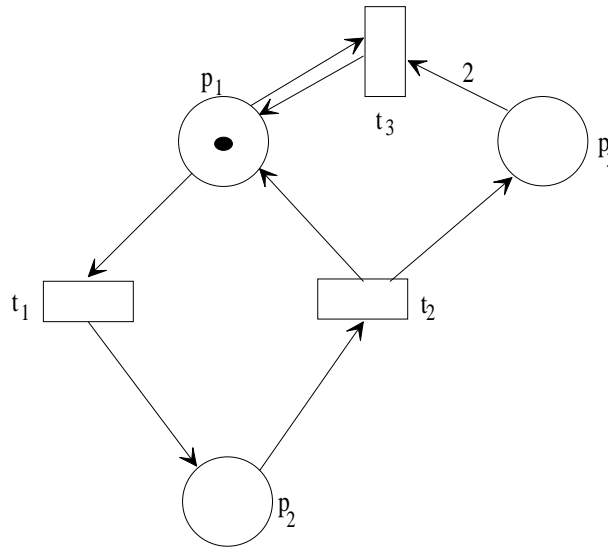


Figura 4.1: Rede de Petri utilizada no exemplo 3.1

Uma proposição atômica em CTL, tem uma relação com uma RP, da seguinte forma:

- Uma proposição atômica $\Omega(t)$ é verdadeira se e só se a transição t está habilitada ou dispara.

Então, nosso trabalho está em definir a especificação de comportamentos de SEDs. Assim, para nossos propósitos, que é o de estarmos interessados apenas nas seqüências

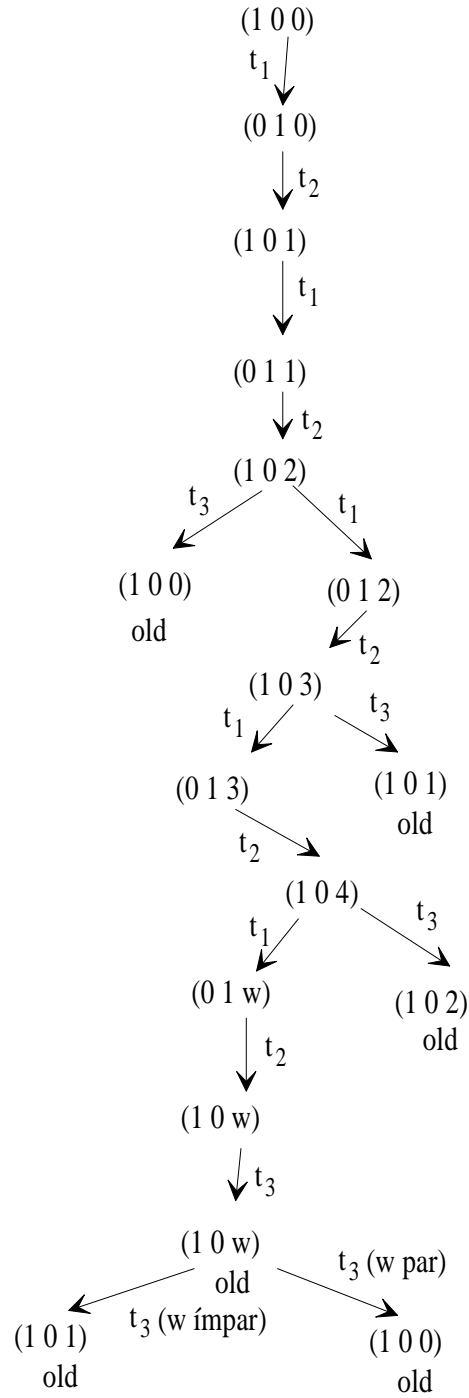


Figura 4.2: Árvore de alcançabilidade da RP, do exemplo 3.1.

de disparos de transições que implica na execução de uma tarefa específica, isto é, a especificação do comportamento, nosso trabalho irá se compor da consideração de que o conjunto de proposições atômicas seja determinado apenas pelo conjunto de transições da RP, que modela o sistema em estudo, como mostrado no exemplo 4.1. Logo, após entrarmos com a fórmula CTL, para especificar o comportamento do sistema, procedemos à verificação e validação da fórmula, como veremos adiante, nos exemplos do próximo capítulo.

A partir de um modelo em RP, e dada sua árvore de alcançabilidade, uma fórmula CTL pode ser avaliada para determinar se é possível se seguir as seqüências de transições que a mesma explicita.

Dado um SED modelado por uma RP, precisamos tornar a especificação de comportamento, a qual é feita em uma linguagem natural definindo uma tarefa a ser realizada, em uma fórmula CTL. Para tanto, torna-se necessário a compreensão de como o sistema irá operar, ou seja, como se dá a passagem de estado para estado em cada ocorrência de um evento, de acordo com o que se deseja. Daí, é que iniciamos o processo de transformação da linguagem, em fórmula CTL. Esta, é feita por intermédio das ligações na frase que objetiva o comportamento requerido, isto é, cada verbo é uma ação; cada ação é um evento e, cada evento é uma transição. Após isto, feita a formulação da seqüência de eventos, deve-se colocar os advérbios ligados com seus respectivos operadores temporais, ou seja, se temos que mandar que um determinado programa sempre execute, ligamos o verbo executar como um evento, digamos que α seja o evento associado, e o advérbio sempre, com o operador temporal G_I . Logo, a fórmula que descreve o que requeremos, deve ficar: $G_I\alpha$, isto é, sempre execute, como desejávamos.

Assim, podemos explicitar como exemplo, um suposto sistema de transmissão e recepção de dados. Este está apresentado na figura 4.3, modelado por uma RP. Neste sistema, o transmissor está sempre pronto para enviar uma mensagem que deve passar por um *buffer* antes de ser transmitida pelo canal, e o receptor deve enviar uma mensagem de retorno, avisando de sua recepção. Contudo, nova mensagem só pode ser enviada, se houver o aviso de recepção. Desejamos formular uma especificação de comportamento em uma linguagem natural, e formalizá-la em uma fórmula CTL. Precisamos, então, da árvore de alcançabilidade do sistema, a qual está mostrada na figura 4.4, onde temos todos os caminhos possíveis, seguindo os disparos das transições. A partir deste grafo, validamos a fórmula CTL, quando encontramos pelo menos uma seqüência que a satisfaça.

Assim, queremos que, este sistema, esteja sempre preparando novas mensagens para ser enviadas ao receptor, indiferentemente de sua transmissão. Então, devemos

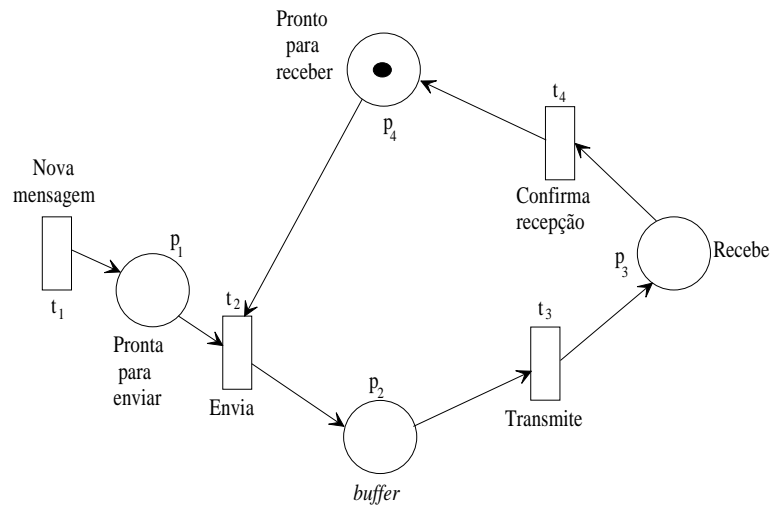


Figura 4.3: Sistema de Transmissão/Recepção modelado por Rede de Petri

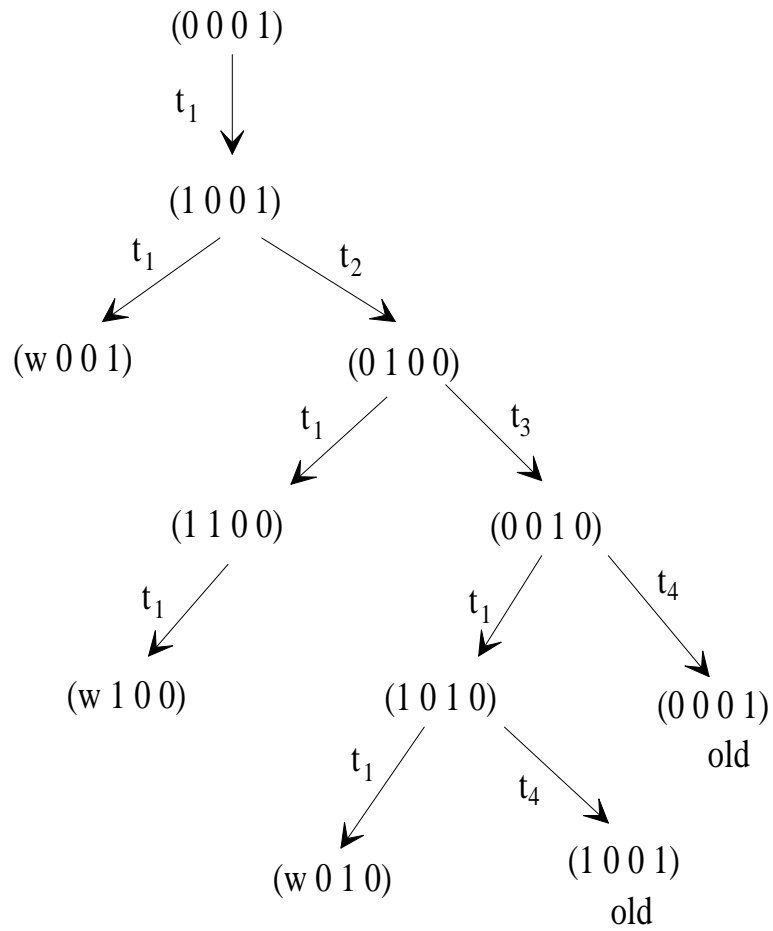


Figura 4.4: Árvore de alcançabilidade da RP, para o sistema Transmissão/Recepção

observar que para isto, devemos habilitar e disparar, sempre t_1 , isto é, devemos definir com a especificação

$$G_I t_1.$$

Se o nosso desejo for de que uma mensagem seja preparada de vez em quando, isto é, eventualmente, para ser enviada, entramos com a fórmula

$$F_I t_1;$$

mas se quisermos impor que esta mensagem só seja preparada se o sistema receber uma mensagem de retorno, já impomos que t_4 , seja disparada. Logo, a fórmula CTL, fica da forma

$$A_I t_1 \wedge X(\neg t_1 U A_I t_4),$$

desde que t_1 deve ser bloqueada até que t_4 (mensagem de recepção enviada) dispare.

Uma outra forma de darmos uma especificação, seria

$$F_I(t_1 U t_2) \wedge A_I G_I(X((t_1 \wedge t_3) \vee (t_1 \wedge t_4))).$$

Esta fórmula, indica que, eventualmente, uma mensagem seja preparada, até que a mesma seja enviada, e após este último evento (envio da mensagem = disparo de t_2), necessariamente, sempre o próximo estado tem de ser o preparo de novas mensagens, enquanto a mensagem é transferida do *buffer* para o receptor, ou quando o receptor manda o aviso de mensagem recebida.

Também, podemos definir como especificação de comportamento a situação em que, para cada mensagem preparada, esta deva ser enviada imediatamente, e esperar até a chegada de uma resposta do receptor. Em outros termos, queremos impor que uma outra mensagem só será preparada se houver uma mensagem de retorno do receptor, isto é, desejamos que o sistema se mantenha sempre em estado de espera até ser avisado de que sua mensagem foi recebida. Então, devemos assegurar que t_1 tem que disparar e, logo após, t_2 , t_3 e t_4 , devem disparar, obrigatoriamente, nesta seqüência. Dessa forma, entramos com a fórmula CTL definida por

$$G_I(F_I t_1 \wedge A_I X((t_2 U t_3) U t_4)).$$

Dados estes exemplos, podemos ver como se faz a transcrição do comportamento desejado da rede de Petri, para uma fórmula CTL, utilizando de seus operadores e conectivos.

Aqui finalizamos a idéia de utilização da CTL como forma de especificação de comportamentos para uma RP, através de sua árvore de alcançabilidade, que é no que

estamos interessados. Daqui, com o uso das RPs para a modelagem de SEDs, a LT, como mostrada, é de muito mais vantagens neste contexto, pois, como já explicitado, as linguagens formais têm suas limitações na descrição de comportamentos destes sistemas. Observemos que com a CTL, podemos definir caminhos no modelo do SED, que podem ter até mais de uma transição habilitada ou disparando conjuntamente. Esta situação não é possível nas linguagens formais, pois a mesma é seqüencial, não conseguindo representar esta condição, o que nos leva a introduzir a LT no estudo da Teoria de Controle Supervisório para melhor trabalharmos com simulações e aproximarmos de um sistema real, isto é, de uma aplicação industrial.

Capítulo 5

Apresentação dos Algoritmos e Exemplos

A síntese de supervisores de SEDs utilizando RPs para modelagem dos sistemas, necessita dos algoritmos apresentados por Barroso [Bar96], os quais em sua execução criam a árvore de alcançabilidade da RP (AMArA) e determina, a partir deste grafo, juntamente com a especificação requerida, todas as funções que devem ser aplicadas às transições do supervisor modelado, para que o sistema não entre em *bloqueio* (ACGS).

Entretanto, no uso da lógica temporal para a determinação da especificação do comportamento desejado, torna-se necessário a análise da sintaxe de fórmulas CTL, de forma a que, uma determinada fórmula que especifique este comportamento, seja analisada corretamente, apresentando quaisquer possíveis erros.

Apresentamos neste capítulo, estes algoritmos e comentários a respeito dos mesmos, para maiores esclarecimentos, utilizando para isto, de alguns exemplos.

5.1 Algoritmo Modificado da Árvore de Alcançabilidade

Este algoritmo - AMArA - tem uma modificação simples com respeito ao algoritmo original, que é um passo a mais na determinação de estados não permitidos (ramos da árvore com crescimento infinito). Esta mudança no algoritmo é determinante para modelagem de SEDs, visto que estes, para serem fisicamente realizáveis, têm de ter capacidade finita. É neste ponto onde o AMArA lista todos os estados possíveis de serem alcançados pelo sistema, juntamente com as seqüências de transições disponíveis a partir de cada estado. Observamos, então, que o mesmo é válido para redes com

capacidade limitada ou finita.

Algoritmo 1 AMArA

- *Início*

1. Rotule a marcação inicial M_0 como raiz e etiqüete-a como *nova*;
2. Enquanto existirem marcações *nova* faça:
 - a) Selecione uma marcação *nova* M ;
 - b) Se M for idêntica a uma outra marcação já existente no caminho da raiz até M , etiqüete-a como *antiga*;
 - c) Se nenhuma transição está habilitada em M , etiqüete M como *bloqueada*;
 - d) Enquanto existirem transições habilitadas em M , faça o seguinte para cada transição habilitada:
 - i. Obtenha a marcação M' que resulta do disparo de t em M ;
 - ii. Se a capacidade de algum lugar p é excedida na marcação M' , então substitua $M'(p)$ por ω ;
 - iii. Introduza M' como um nó da árvore, ligue um arco, com rótulo t , de M para M' e etiqüete M' como *não-permitida* se a capacidade de algum lugar foi excedida, de outra forma, etiqüete-a como *nova*.

- *Fim.*

Um exemplo do uso deste algoritmo está ilustrado para o caso do sistema representado na figura 5.1, cujo modelo em RP é de um sistema produtor/consumidor. Neste caso, a utilização deste algoritmo, resulta na árvore de alcançabilidade apresentada na figura 5.2, onde os estados mostrados nos vetores, são relativos ao vetor de marcação $M = (p_1 \ p_2 \ p_3 \ p_4 \ p_5)$. Observamos que, devido à seqüência $(\alpha_1\beta_1)^*$, o sistema atinge uma marcação *não-permitida*, pois o lugar p_5 irá ter um aumento descontrolado no número de fichas. Esta seqüência deve ser evitada pela ação de controle externo ao sistema.

5.2 Algoritmo para a Construção do Gerador da SupC(L)

Aqui temos um algoritmo que trabalha com a utilização dos dados referentes à árvore de alcançabilidade gerada pelo AMArA e da especificação do comportamento que se

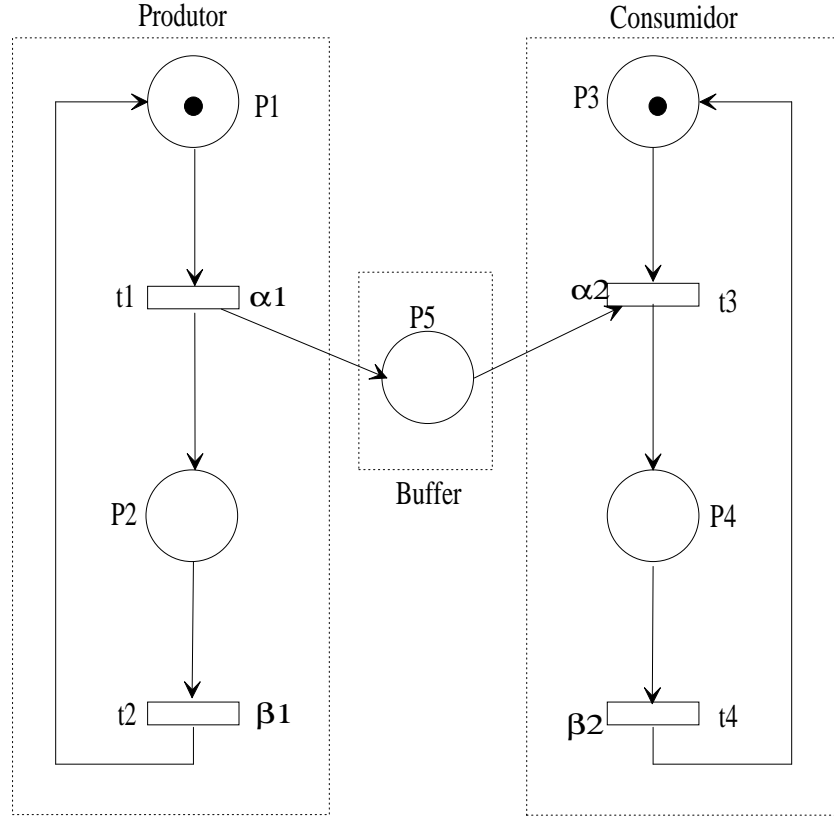


Figura 5.1: Modelo do sistema produtor/consumidor via RP

deseja para o sistema. Logo, este algoritmo só pode ser executado após a execução do AMArA e da entrada desta especificação, a qual é feita através do *parser* para determinar se a fórmula está devidamente correta e coerente com os dados do sistema modelado. De posse destes dados, a execução do ACGS dá-nos uma lista de eventos que devem ser desabilitados nos devidos estados, determinando a $\text{SupC}(L)$, e com estas saídas, implementam-se as funções de habilitação que necessitam ser impostas às respectivas transições para impedir que o sistema entre em bloqueio ou siga a especificação desejada, caso esta seja possível.

Algoritmo 2 ACGS

- *Início*

1. Criar uma lista dinâmica, *lista-bloc*, e incluir na mesma os estados ou marcações bloqueadas, incluindo as marcações do tipo *não-permitida*, onde:

$M : M[t_j] > 0$, é uma marcação bloqueada e
 $M : M(p_i) = w$ é uma marcação não-permitida;

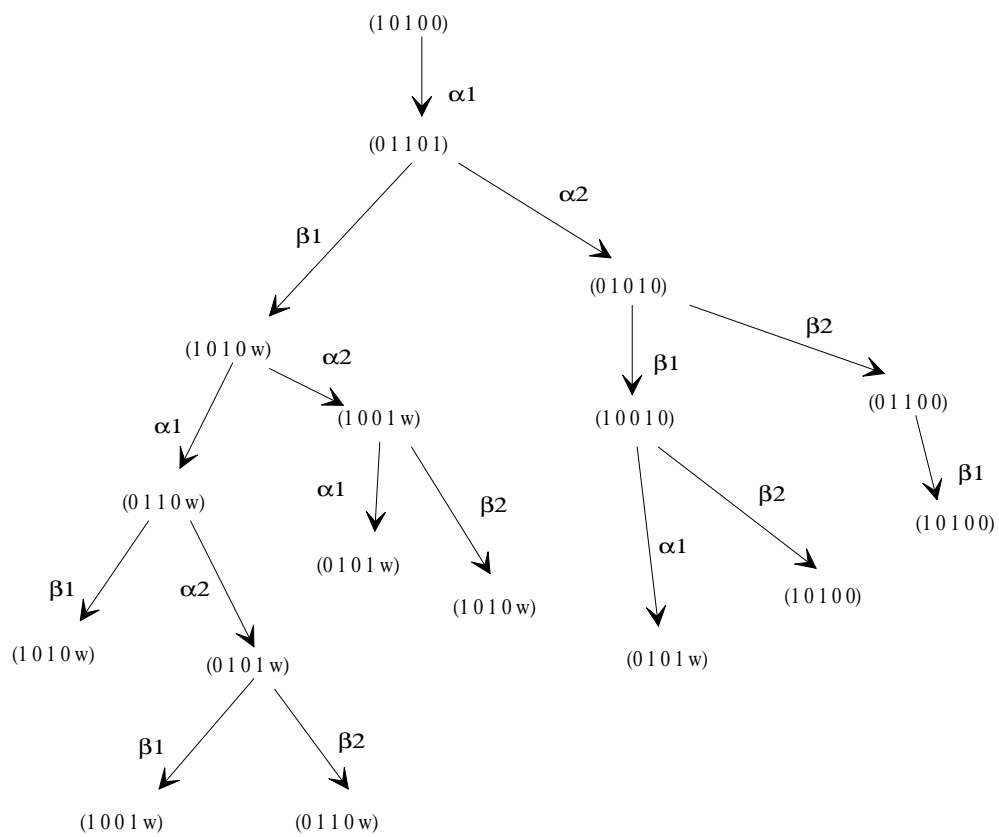


Figura 5.2: Árvore de alcançabilidade do sistema produtor/consumidor modelado por RP, com utilização do AMArA

2. Adicionar à *lista-bloc* os estados, não marcados, cuja única transição habilitada, se disparada, leva o sistema a um estado bloqueado, ou seja:

$$\begin{aligned} M : M[t_j > M', t_j \text{ é única transição habilitada em } M, \\ M \notin Q_m \text{ e } M' \in \textit{lista-bloc}; \end{aligned}$$

3. Criar uma lista, *lista-perigo*, com os estados antecessores dos elementos (estados) da *lista-bloc*, juntamente com o evento que os liga, desde que o antecessor não esteja na *lista-bloc*. Esses eventos deverão estar sempre desabilitados quando o sistema se encontrar nesses estados, ou seja:

$$\exists \beta \in \Sigma_c | l(t_j) = \beta \text{ e } M[t_j > M', \text{ e } M \notin \textit{lista-bloc} \text{ e } M' \in \textit{lista-bloc};$$

4. Adicionar à *lista-perigo* os estados nos quais exista pelo menos uma transição habilitada, etiquetada por um evento não controlável, cujo disparo da transição leve o sistema para uma marcação na *lista-bloc*, ou seja

$$\exists \alpha \in \Sigma_u, l(t_j) = \alpha \text{ e } M[t_j > M', M' \in \textit{lista-bloc};$$

5. Dada a especificação desejada para o sistema, encontre a suprema linguagem controlável - SupC(L);
6. Adicionar à *lista-perigo* os estados e seus respectivos eventos de saída a serem desabilitados para que a linguagem executada, desde que estes estados não estejam ainda na *lista-perigo*, ou seja:

$$\exists \beta \in \Sigma_c | l(t_j) = \beta \text{ e } M[t_j > M', \text{ e } M \notin \textit{lista-perigo} \text{ e } M' \notin G(\text{Sup } C(L)).$$

• *Fim.*

Neste algoritmo, o passo 5 se processa da seguinte forma:

Algoritmo 3 Passo 5 do ACGS

• Dados o gerador trim e o gerador **H** (especificação), faça:

1. Adicione à *lista-bloc* os estados que não satisfazem

$$\Sigma(\mathbf{H}(x)) \cap \Sigma_u \subseteq \Sigma(x);$$

2. Para cada estado x_i , na *lista-bloc*, adicione à *lista-bloc*:

(a) os estados

$$x_j : (\exists \sigma_u \in \Sigma_u) x_i = \xi(\sigma_u, x_j);$$

(b) os estados

$$x_k : (\exists \sigma_c \in \Sigma_c) x_i = \xi(\sigma_c, x_k) \wedge x_k \notin X_m \wedge |\Sigma(x_k)| = 1;$$

3. Encontre a componente acessível do gerador resultante.

Aqui, podemos ver que, a especificação desejada é uma entrada feita pelo *parser*. Também, vemos que no passo 1, adiciona-se à *lista-bloc* todos os estados em que, um evento não controlável, que é fisicamente possível, não é definido na especificação; no passo 2 processa-se e incrementa-se a lista e, por fim, no passo 3, remove-se qualquer estado inacessível deixado pelos passos anteriores. Além do mais, a segunda parte do passo 2 preserva a coacessibilidade. No final destas operações, temos o gerador da suprema linguagem controlável para a especificação dada.

Na rede mostrada na figura 5.1, a utilização do ACGS, dar-nos-ia a saída, que são as funções de habilitação de transições que impedem que o sistema se ponha em estados não permitidos ou bloqueados. De acordo com uma especificação do tipo $\alpha_1\beta_1\alpha_2\beta_2$, estas funções seriam $\varphi_1 = \overline{p_5}$, $\varphi_2 = \varphi_3 = \varphi_4 = 1$. Nesta condição, a função de habilitação da transição p_1 significa que esta só pode disparar se não houverem fichas no lugar p_5 , ou seja, $\varphi_1 = 1$ se $M(p_5) = 0$.

Na figura 5.3, encontramos um exemplo prático da criação da suprema linguagem controlável a partir da especificação dada, o que facilita a visualização do processo. Podemos ver que, dado o Gerador e a especificação, e os eventos α_1 , α_2 e β , onde os dois primeiros são controláveis e o último é não-controlável, os passos de processamento do ACGS para a determinação da SupC(L), seguem-se a partir da situação em que, ao atingir o estado 2, de acordo com a especificação, o sistema deveria parar. Contudo, devido à condição de que o evento β é não-controlável, o sistema não seguirá corretamente a especificação. Dessa forma, devemos eliminar o estado 2. Da mesma forma, no segundo passo da execução do ACGS, o estado 1, é atingido pela ocorrência do evento α_1 , o qual é controlável, mas leva o sistema a um estado em que pode ocorrer o evento β , levando novamente ao erro da especificação. Logo, como α_1 é um evento controlável, podemos impor que o estado 1 não seja atingido colocando uma ação de controle externo. Assim, a SupC(L), torna-se $\alpha_2\beta^*$.

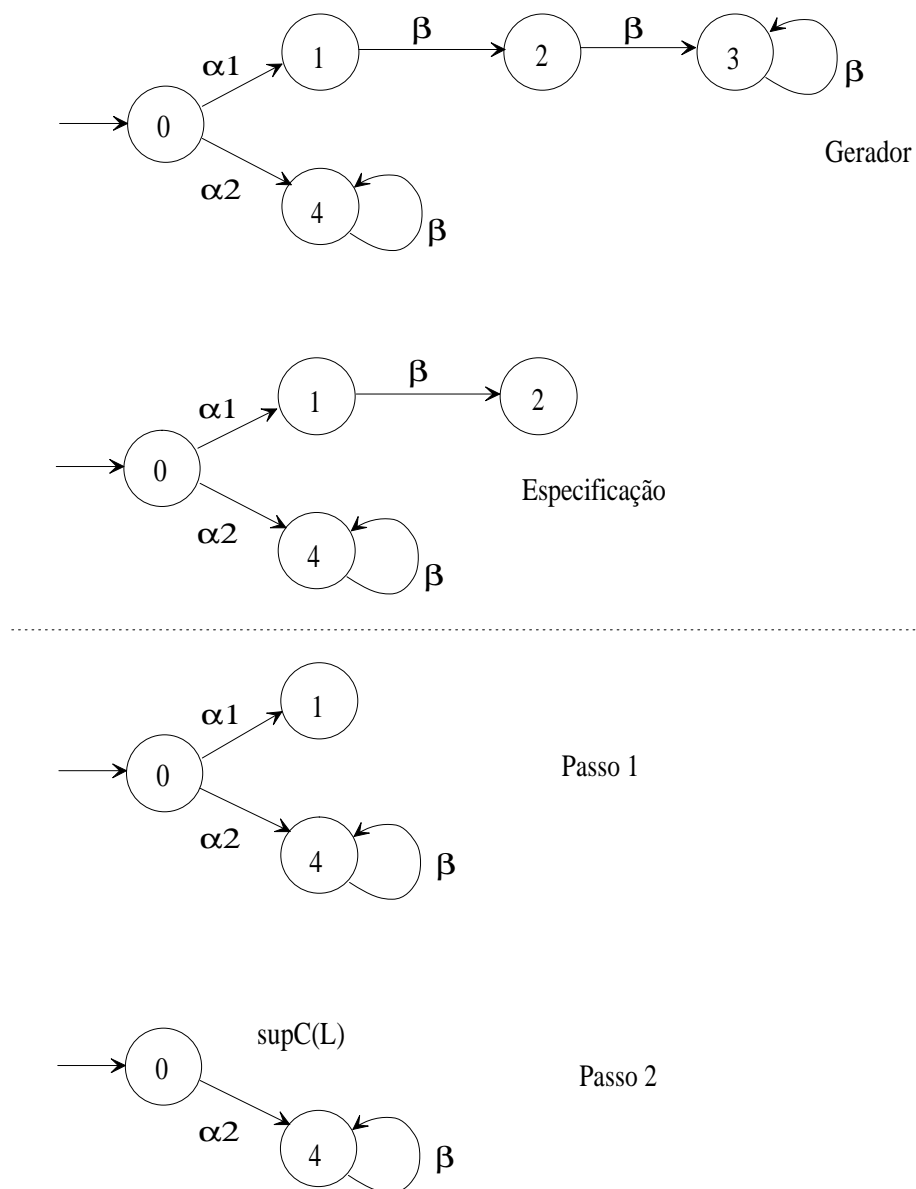


Figura 5.3: Exemplo da construção da $\text{supC}(L)$ pelo ACGS

5.3 Algoritmo do Parser para Lógica Temporal

Este é o algoritmo que, a partir da entrada de uma fórmula CTL, analisa-a buscando erros relativos a mal estruturação sintática da fórmula CTL, bem como sua conexão dentro do sistema modelado. O mesmo só depende da execução do AMArA, pois apenas estuda a coerência sintática da fórmula CTL e compara cada proposição atômica com as transições da RP que modela o sistema, além de analisar se a seqüência que a fórmula exprime existe na árvore de alcançabilidade.

Algoritmo 4 APLT

- *Início*

1. Entre com a fórmula CTL;
2. Faça uma leitura caractere a caractere da fórmula CTL;
3. Conte o número de parênteses abrindo - "(" - (npa) e parênteses fechando - ")" - (npf). Indique erro caso $npa \neq npf$;
4. A cada *Until* (U), retorne com erro se existir, antes dele, parêntese abrindo, operador lógico ou conectivo lógico, depois dele parêntese fechando e de conectivo lógico só pode ter $NOT(\neg)$;
5. A cada *Next* (X), aponte erro se houver, antes dele, *true*, *false* ou proposição atômica, e após ele, só pode ter uma proposição atômica, conectivo lógico $NOT(\neg)$, ou parêntese abrindo - "(";
6. Para cada proposição atômica, declare erro se ela não existir no conjunto de transições T da rede de Petri que modela o sistema;
7. Crie uma lista, *lista-seq*, que irá guardar cada seqüência de transições possível na dada fórmula;
8. Confirme se na árvore de alcançabilidade existe alguma seqüência de transições que satisfaça a fórmula CTL;
9. Para cada seqüência existente, coloque-a na *lista-seq*, como uma das seqüências que a fórmula exprime;
10. Caso não haja nenhuma seqüência de transições que seja expressa pela fórmula na árvore de alcançabilidade da rede, declare que a fórmula é inválida para a marcação M_0 .

- *Fim.*

Neste parser, após a entrada de uma fórmula CTL, podemos ver se a mesma apresenta erros ou não, e se a seqüência, ou seqüências de transições que a fórmula expressa, existe na árvore de alcançabilidade da rede de Petri que modela o SED. Por exemplo, seja a RP da figura 5.4. Seu conjunto de transições é $T = \{a, b, c\}$. Ao se entrar com uma fórmula CTL, o parser irá analisá-la devolvendo mensagens de erro, ou que a fórmula está correta, da seguinte maneira:

- $(aUb) \wedge ((cUb) \wedge \neg(cUa)) \rightarrow$ erro no número de parênteses;
- $(aUc) \wedge \neg(cUd) \rightarrow$ erro de proposição atômica inexistente no conjunto *Prop*, o qual é equivalente ao conjunto de transições T da rede de Petri, ou seja, há uma proposição atômica que não se liga a nenhuma transição;
- $aX\neg b \rightarrow$ erro com o uso do operador *Next*;
- $a \wedge XU b \rightarrow$ erro com a utilização do operador *Next*;
- $(U(b \wedge \neg c)) \rightarrow$ erro com o uso do operador *Until*;
- $(aUXb) \wedge Xc \rightarrow$ aqui temos um erro de seqüência de transições, pois a fórmula indica que deve ser seguida a seqüência abc , desde que temos que a transição a tem de ser verdadeira até que a transição b no próximo estado, seja (o que é verdade) e no próximo estado, a transição c tem de disparar (o que é impossível nesta marcação). Logo esta seqüência de disparos não é possível para a rede, ou seja, ela é inexistente na árvore de alcançabilidade;
- $G_l(aUb) \wedge \neg c \rightarrow$ fórmula correta, pois exprime a seqüência dada por $(ab)^*$ que existe no grafo de alcançabilidade da RP.

Para as duas últimas fórmulas, podemos comparar as seqüências de transições possíveis de serem seguidas na rede, a qual está mostrada na figura 5.5, onde temos a marcação inicial $M_0 = (1 \ 0 \ 0)$, sendo o conjunto de marcações $M = (p_1 \ p_2 \ p_3)$.

Como podemos ver nestes exemplos, o *Parser* analisa cada operador e cada proposição atômica, só deixando passar a fórmula CTL, caso ela esteja corretamente escrita, com relação ao grafo de alcançabilidade.

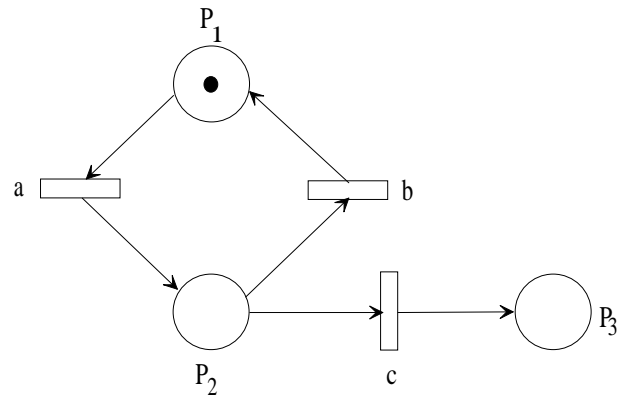


Figura 5.4: Simplex rede de Petri usada no exemplo do uso do parser

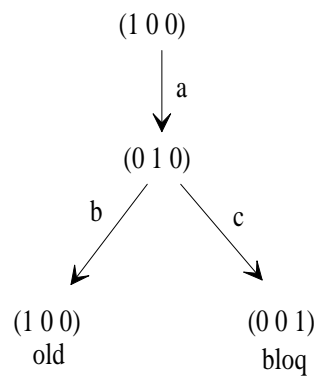


Figura 5.5: Árvore de Alcançabilidade da rede de Petri usada no exemplo do uso do parser

5.4 Exemplos da Utilização dos Algoritmos

Considere o sistema de manufatura com recursos compartilhados modelado por rede de Petri, apresentado na figura 5.6, em que sua marcação inicial seja $M_0 = (6 \ 0 \ 0 \ 0 \ 0 \ 0 \ 8)$. Este sistema representa a produção de itens utilizando recursos, os quais são representados pela marcação no lugar p_7 . Desejamos nele, determinar inicialmente sua árvore de alcançabilidade, para em seguida definirmos uma especificação de comportamento e determinarmos as funções de habilitação das transições que execute a mesma, impedindo o sistema de cair num estado indesejável, isto é, bloqueado, como é o caso da seqüência de disparos das transições dada por

$$t_1 t_2 t_1 t_2 t_1 t_1 t_3 t_3 t_2 t_3 t_1,$$

que leva a rede a atingir a marcação $M' = (1 \ 2 \ 0 \ 3 \ 0 \ 0 \ 0)$ em que nenhuma transição está habilitada.

A utilização do AMArA gera a árvore de alcançabilidade deste sistema, a qual guarda todas as seqüências de transições possíveis e todos os estados alcançados pelo mesmo, dizendo para cada seqüência, qual leva a um estado bloqueado ou não.

Como desejamos especificar o comportamento deste sistema, digamos que queremos que uma única peça seja processada por vez, utilizamos o APTL, para a entrada da especificação requerida em Lógica Temporal. Aqui, podemos declarar a especificação como

$$G_l(A_l t_1 U t_2) \wedge G_l(\neg(t_1 \wedge t_2) U X t_3) \wedge G_l(\neg(t_1 \wedge t_2 \wedge t_3) U X t_4) \wedge A_l t_5.$$

Nesta, podemos ler o que se segue: após o disparo de t_1 e t_2 , as transições t_1 e t_2 estarão bloqueadas, de forma a impor que só t_3 seja habilitada, podendo disparar. Após seu disparo, isto é, t_3 , tornamos apenas t_4 habilitada, enquanto t_1 , t_2 e t_3 não estão. Em outros termos, impedimos que as transições anteriores a que desejamos habilitar, estejam não-habilitadas. Por fim, impomos que t_5 seja disparada. Assim, o sistema irá trabalhar uma única peça por vez. Observamos aí, que de uma forma ou de outra, as transições que desabilitamos, não necessitam ser desabilitadas, desde que, como só temos uma única peça sendo processada, só precisamos desabilitar t_1 , pois as outras transições estão naturalmente desabilitadas. Disto resulta que, podemos definir como especificação de comportamento do sistema, apenas a função

$$A_l t_1 \wedge G_l(X(\neg t_1 U A_l t_5)),$$

que torna t_1 desabilitada, até que t_5 seja disparada.

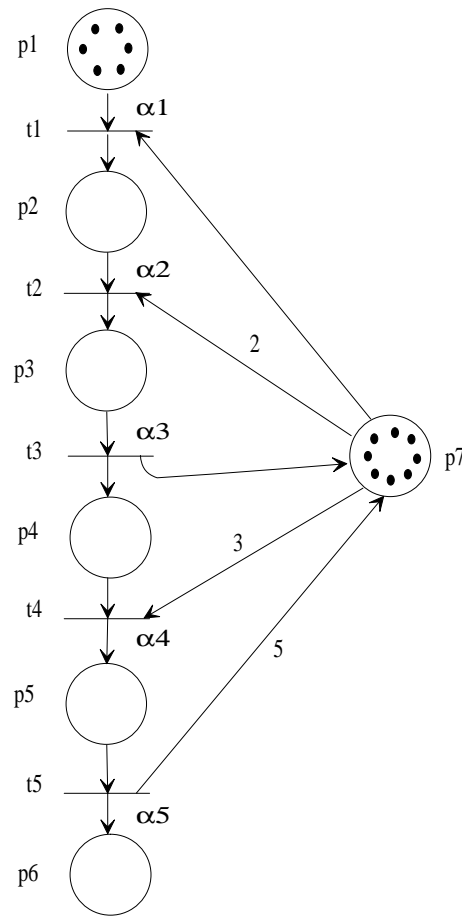


Figura 5.6: Modelo em Rede de Petri para um Sistema de Manufatura com Recursos Compartilhados

Devemos levar em consideração, que para estas duas especificações, partes do sistema estará parado em algumas posições, isto é, não haverá processamento de nenhum outro item, enquanto o que estiver sendo processado não chegar ao final, ou seja, no lugar p_6 . Em outros termos, isto é uma condição indesejável para o sistema, devido à sua capacidade ociosa, pois pela definição de redes de Petri, uma transição habilitada pode ou não disparar. Logo, se o item habilitar uma transição, digamos t_4 , não garantimos seu disparo e, conseqüentemente, o sistema irá ficar em estado de espera. Dessa forma, irá ficar parado. Para esta especificação, o ACGS nos dá uma função, a qual estará ligada à transição t_1 . Esta função é dada por

$$\varphi_1 = \{M(p_7) = 8\},$$

a qual impede que seja iniciado novo processamento, enquanto o item não estiver completamente terminado. Neste caso, o sistema modelado por uma RPFHT, fica como mostrado na figura 5.7.

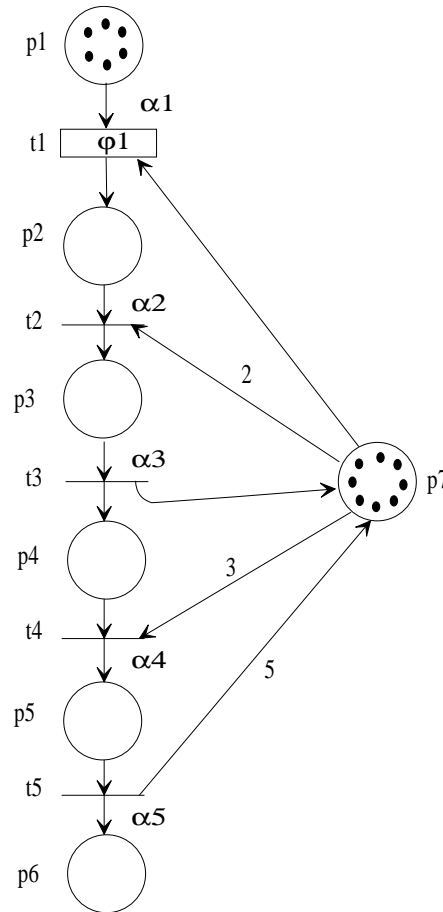


Figura 5.7: Modelo em RPFHT para um Sistema de Manufatura com Recursos Compartilhados, para realizar o processamento de uma única peça por vez

Para este mesmo sistema, poderíamos definir uma outra especificação de comportamento, de forma a fazermos o processamento paralelo de mais de uma peça. A partir da especificação

$$A_I t_1 \wedge G_I(t_1 U t_2) \wedge X(\neg t_1 U A_I t_5)$$

determinamos que, a transição t_1 pode disparar duas vezes seguidas, deixando o sistema trabalhando em paralelo. Isto nos leva a gerar as funções de habilitação de transições

$$\varphi_1 = \{M(p_7) > 6\}$$

$$\varphi_4 = \{M(p_7) \geq 3\},$$

que controlarão o sistema para impedir seus possíveis bloqueios e seguir a especificação dada. Este novo modelo, encontra-se na figura 5.8.

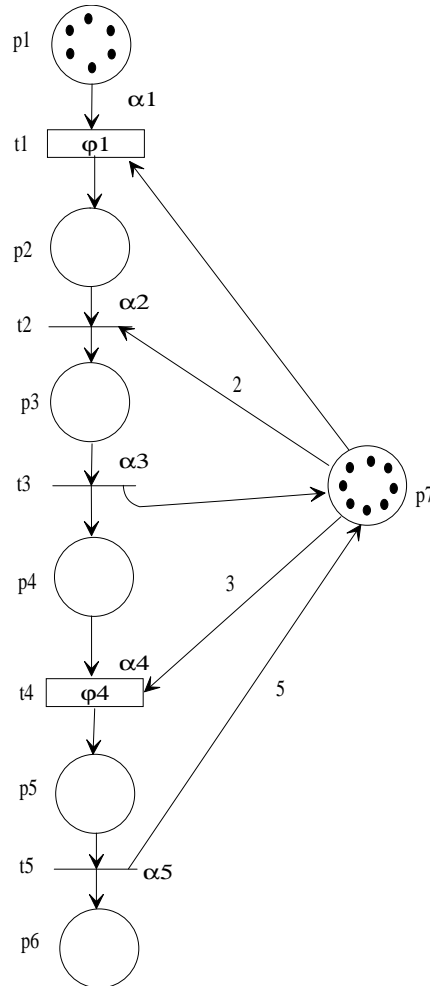


Figura 5.8: Modelo em RPFHT para um Sistema de Manufatura com Recursos Compartilhados, para realizar o processamento de duas peças em paralelo

Devemos observar que, na especificação de comportamento dada no primeiro caso (uma única peça por vez), em lógica temporal, é muito mais abrangente do que se

fosse dada em linguagens formais, pois nestas, temos de descrever toda a seqüência de disparos de transições a ser seguida, para termos tal comportamento, isto é,

$$(t_1 t_2 t_3 t_4 t_5)^*.$$

No segundo caso, ou seja, das duas peças em paralelo, teríamos que determinar por qual das maneiras isto teria de ser feito, isto é, de acordo com a fórmula CTL, estas duas seqüências de transições podem ser obtidas:

$$(t_1 t_1 t_2 t_2 t_3 t_3 t_4 t_5 t_4 t_5)^* \text{ ou,}$$

$$(t_1 t_1 t_2 t_2 t_3 t_4 t_5 t_3 t_4 t_5)^*,$$

enquanto em linguagens formais, só uma delas pode ser considerada por vez. Esta é uma condição intrínseca do uso da LT na TCS, visto que um sistema grande modelado por RP, de acordo com a especificação, tende a se tornar muito mais complexa e difícil de ser visualizada e analisada em linguagens formais, como visto no segundo caso da especificação do problema anterior. No caso da LT, além da fórmula ser bem menor, devido a que, necessitamos determinar apenas alguns pontos de habilitação e desabilitação da rede, ainda temos a vantagem de que a fórmula pode nos indicar mais de um caminho para executar a tarefa especificada, dando-nos maior flexibilidade no processamento, isto é, no caso de uma transição habilitada não disparar, há uma outra condição para continuar o funcionamento do sistema, sem que o mesmo fique em estado estacionário, ou seja, ocioso. Neste caso, isto é considerado e abre a possibilidade de otimização da escolha.

Um outro exemplo que podemos citar, é de um sistema de dois motores: A e B , e um programa computacional que controla o motor A . A situação é descrita pela condição em que, o motor A deve estar trabalhando enquanto o motor B estiver operando. Caso o motor B pare de operar, um computador, através de um programa controlador, deve verificar este estado e parar o motor A . Em outros termos, o motor A deve ser ativado ou desativado seguindo o estado do motor B . Devemos observar que o motor B é independente. Este sistema está modelado por uma rede de Petri na figura 5.9. Observemos que, de acordo com este modelo, o motor A apenas é desativado quando o motor B está inativo. Contudo, o mesmo pode ser ativado independentemente do motor B entrar em funcionamento. O que desejamos é evitar este problema. Há duas maneiras de fazer isto:

1. Modificar o modelo do sistema, incluindo novos arcos que impeçam este problema;
2. Definir uma especificação de comportamento, de forma a evitar isto.

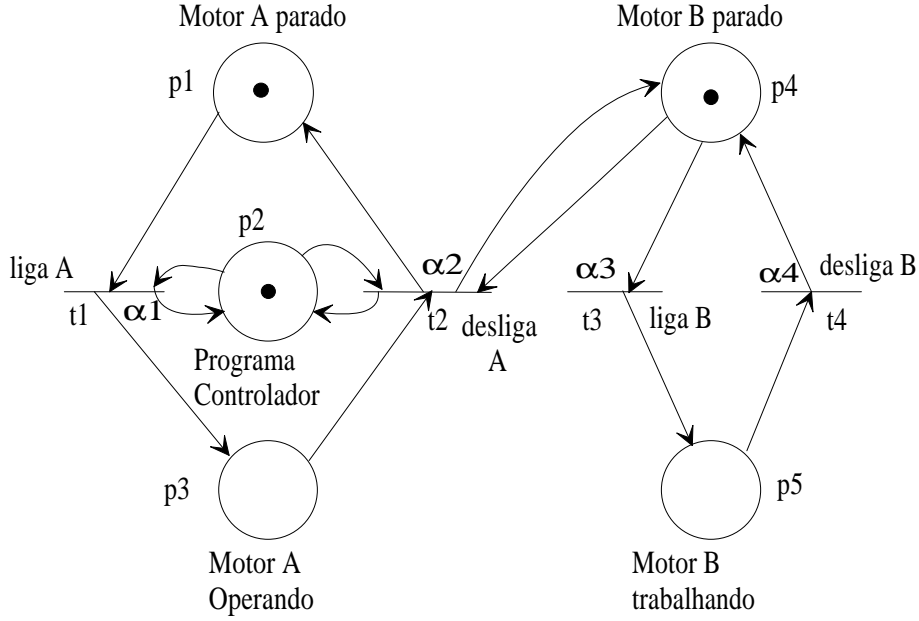


Figura 5.9: Sistema do exemplo, modelado por rede de Petri

Para o primeiro caso, podemos ver o resultado da nova modelagem do sistema na figura 5.10. Nesta, vemos que a transição t_1 , que liga o motor A , só dispara se o motor B estiver ligado. Contudo, dado o modelo, queremos que o mesmo funcione sem termos de modificá-lo. Dessa forma, como estamos interessados em termos uma ferramenta que a partir do modelo, realize qualquer especificação dada, sem termos de modificá-lo, trabalhamos com o segundo caso, para determinarmos as funções de habilitação das transições. Este comportamento é definido pela fórmula

$$G_l(X(\neg t_1 U t_3)) \wedge X t_1 \wedge G_l(t_2 \supset (X(t_4 U t_2))).$$

Esta fórmula nos diz que, sempre deve-se disparar t_3 , depois t_1 e, após isto, disparar t_4 e t_2 . Em termos da marcação inicial $M_0 = [1 \ 1 \ 0 \ 1 \ 0]$, dizemos: ligue o motor B e, após a ocorrência deste evento, o programa computacional liga o motor A , e só desligue o motor A se o motor B for desligado.

Outra forma de fazermos a especificação para este sistema, é através da fórmula

$$G_l(\neg t_1 U X t_3)$$

pois, de acordo com esta, podemos ver que, a transição t_1 só estará habilitada após t_3 disparar. Isto quer dizer, que não necessitamos fazer alusão explícita nem a t_2 nem a t_4 .

Devido a qualquer uma destas especificações, o uso do ACGS irá nos dar a função para a transição t_1 como

$$\varphi_1 = \{M(p_4) = 1\}.$$

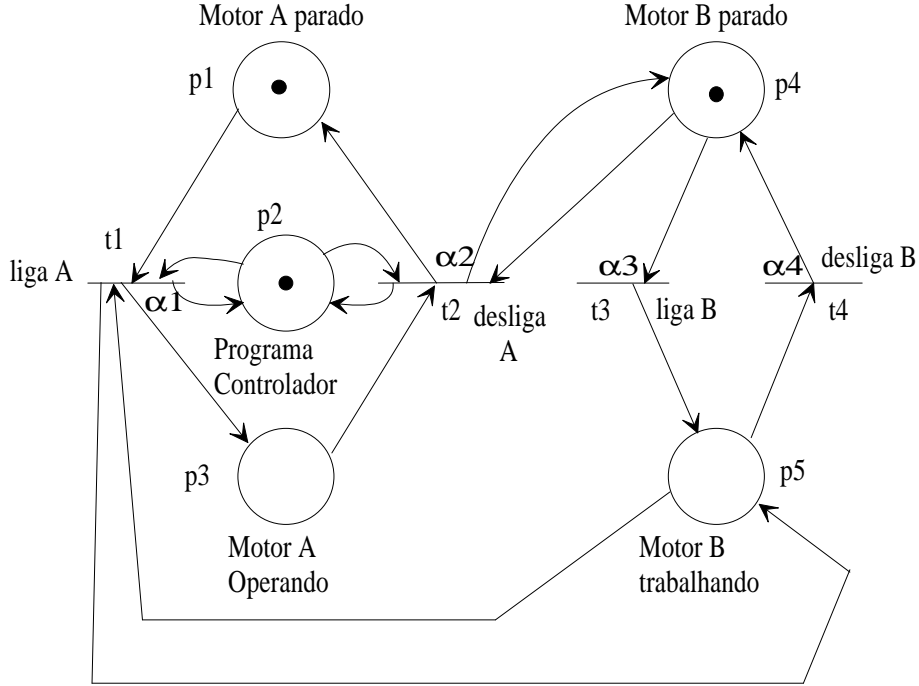


Figura 5.10: Sistema do exemplo, modelado por rede de Petri, eliminando o problema do acionamento do motor A, por intermédio da modificação da rede de Petri

Com esta função de habilitação de transição, não modificamos a rede de Petri modeladora do sistema, e impomos que a especificação seja seguida. A RPFHT está mostrada na figura 5.11.

Para este caso, se fôssemos dar a especificação do comportamento desejado para este sistema por linguagens formais, deveríamos pô-la da seguinte maneira:

$$(t_3 t_1 t_4 t_2)^*.$$

Podemos ver neste exemplo que esta é menos compacta que a segunda alternativa (fórmula CTL), visto termos de fazer toda a seqüência que desejamos que o sistema siga. Aqui também vemos que o uso da lógica temporal é uma melhor alternativa na especificação do comportamento dos SEDs.

Embora a fórmula em linguagem formal aparente ser mais simples que a lógica temporal, não é abrangente o bastante, além do que, como já foi dito anteriormente, se o sistema fosse muito maior, a seqüência de transições em linguagens formais, tornar-se-ia grande demais e, conseqüentemente, difícil de ser visualizada e compreendida, além de se ter mais de uma seqüência que dá uma mesma especificação. Isto não acontece com a lógica temporal, pois em uma única fórmula, podemos ter mais de uma seqüência de transições que podem ser seguidas, além de necessitarmos colocar, apenas as transições principais de controle que devemos situar na especificação.

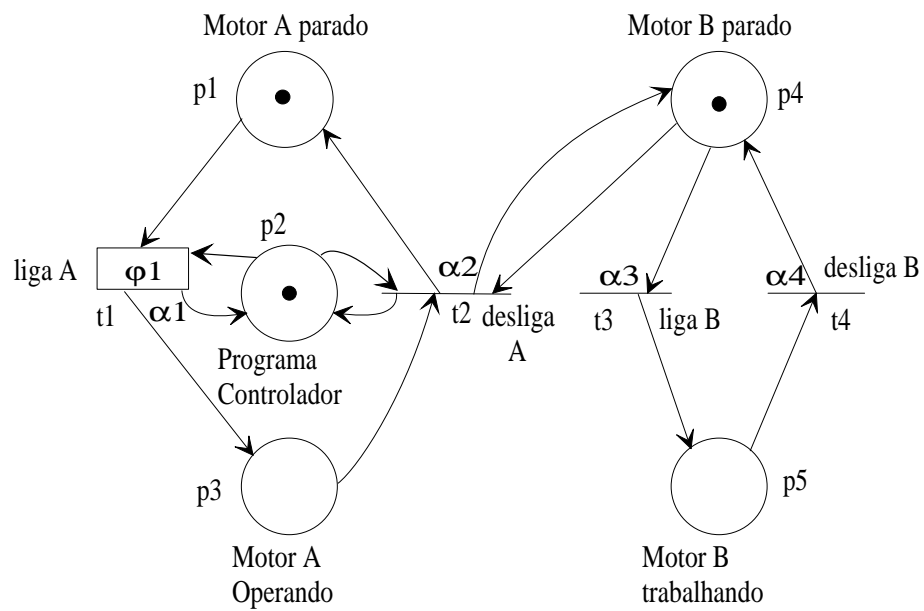


Figura 5.11: Sistema do exemplo, modelado por RPFHT, eliminando o problema do acionamento do motor A.

Capítulo 6

Considerações Finais

6.1 Discussões e Conclusões

Dentro do presente trabalho apresentado, estudamos a classe de lógica temporal, *Computation Tree Logic* - CTL, utilizando-a para especificação de comportamentos de Sistemas a Eventos Discretos - SED, para a determinação das funções em uma *Rede de Petri com Funções de Habilitação de Transições* - RPFHT, para a síntese do supervisor que gera a *Suprema Linguagem Controlável* - SupC(L). Aqui, as fórmulas CTL são analisadas sintaticamente e transformadas em seqüências de transições pelo *Algoritmo do Parser para Lógica Temporal* - APLT, as quais são comparadas com as seqüências dentro da árvore de alcançabilidade, de forma a confirmar sua validação com a rede que modela o sistema em estudo. Estas fórmulas são mais abrangentes que as linguagens formais, visto que, com elas podemos definir especificações de comportamentos para os SEDs, incluindo concorrências.

Fazemos alusão no uso das RPFHT para modelagem do supervisor como uma forma simples de resolução do problema de controle, desde que não precisamos modificar o modelo do sistema para que o mesmo realize nossa especificação de comportamento.

Os exemplos apresentados, demonstram o uso da CTL para a especificação do comportamento de alguns sistemas, além de explicitar como se devemos proceder para transformar a linguagem natural que define o que é desejado em uma fórmula CTL, a partir do modelo projetado em RP.

Observamos que esta introdução da CTL como uma nova forma de abordar o problema da especificação de comportamentos de SED para a síntese do supervisor é mais uma contribuição para a teoria de controle supervisiório.

6.2 Trabalhos Futuros

Uma outra forma de realizar a síntese do supervisor, seria feito pelo estudo das *Estruturas de Predicados* - EP - dando a validação de fórmulas pela verificação simbólica do modelo, sem a utilização da árvore de alcançabilidade. Também citamos que, a implementação definitiva dos algoritmos deve ser complementada, dando a estes uma performance em termos de interface homem-máquina e uma inter-conexão, para sua inclusão direta, ou na forma bibliotecas, em softwares já existentes, como o *Design CPN*, desde que estes podem ser utilizados experimentalmente para análises científicas, ou em simulação. O desenvolvimento de hardwares, deve ser levado em consideração, para a criação de protótipos de SEDs, visando sua ampliação para a aplicação à sistemas reais, como processos de automação industrial.

Os trabalhos relativos à síntese do supervisor, também podem ser implementados pelo estudo das *Redes de Petri Controladas* - RPctl, utilizando os lugares de controle para gerar condições que levem a uma sequência desejada. Dessa forma, um estudo poderia ser feito para a implementação de um algoritmo que avaliasse quais as posições estratégicas e em que condições seriam postos os lugares de controle para a geração da SupC(L).

Um trabalho de grande valia no estudo tanto da teoria de controle supervísório, como das rede de Petri, seria o desenvolvimento de uma nova classe de RP que tenha um conjunto de transições, em que cada transição é um multiconjunto de transições, onde estas, se ligam aos arcos, os quais sempre impeçam, de acordo com o estado geral do sistema, seu bloqueio. Em outros termos, cada transição apresenta uma função que determina qual arco que dela sai, coloca (ou retira) fichas no seu lugar específico.

Lista de Símbolos e Abreviaturas

ACGS - Algoritmo para Construção da Suprema Linguagem Controlável

AMArA - Algoritmo Modificado da Árvore de Alcançabilidade

APLT - Algoritmo do Parser para Lógica Temporal

BPTL - Branching Propositional Temporal Logic

BTL - Branching Temporal Logic

CTL - Computation Tree Logic

LPTL - Linear Propositional Temporal logic

LT - Lógica Temporal

LTTL - Linear Time temporal Logic

RP - rede de Petri

RPCtl - rede de Petri Controlada

RPFHT - rede de Petri com Função de Habilitação de Transição

RP L/Tr - rede de Petri Lugar/Transição

SDVC - Sistemas Dinâmicos a Variáveis Contínuas

SDVD - Sistemas Dinâmicos a Variáveis Discretas

SED - Sistemas a Eventos Discretos

SupC(L) - Suprema Linguagem Controlável

TCS - Teoria de Controle Supervisório

A_s - autômato

Ac - componente acessível do gerador

Σ - alfabeto de símbolos

Σ^* - conjunto de todas as palavras construídas dos símbolos de Σ

$\alpha, \beta, \gamma, \dots$ - símbolos para eventos

s - palavra construída pela justaposição de símbolos

ϵ - palavra de comprimento nulo

$\Sigma^+ - \Sigma^* - \epsilon$

L - linguagem sobre o alfabeto Σ

\overline{L} - linguagem prefixo-fechada sobre o alfabeto Σ

G - gerador

G_c - gerador controlável

$L(G)$ - linguagem gerada por um gerador

$\overline{L(G)}$ - linguagem prefixo-fechada gerada por um gerador

$L_m(G)$ - linguagem marcada gerada por um gerador

Q - conjunto de estados q

δ - função de transição de estados

δ^* - função de transição estendida

$\delta(\sigma, q)!$ - função de transição definida para o par (σ, q)

Q_m - conjunto de estados marcados

L_m - linguagem marcada ou reconhecida por um autômato

Q_{ac} - conjunto de estados da componente acessível de um gerador

E_{rp} - estrutura de uma RP

P - conjunto de lugares de uma RP

A - conjunto de arcos de uma RP

T - conjunto de transições de uma RP

K - função de capacidade de uma RP

W - função de ponderação de uma RP

M - marcação de uma RP

l - função que etiqueta as transições em uma RPFHT

Φ - função de habilitação das transições em uma RPFHT

A_l - significa *necessariamente* em CTL

E_l - significa *possivelmente* em CTL

G_l - significa *sempre* em CTL

F_l - significa *eventualmente* em CTL

U - *Until*, ou até que, em CTL

X - *neXt*, ou verdadeiro no próximo estado, em CTL

\vee - OR

\wedge - AND

\neg - NOT

Prop - conjunto de proposições atômicas, em CTL

Σ_c - conjunto de eventos controláveis na TCS

Σ_u - conjunto de eventos não-controláveis na TCS

Γ - conjunto associado, ou entrada de controle na TCS

S - supervisor

Θ - função de entrada de controle ou mapa de controle

$|s|$ - cardinalidade ou comprimento da palavra s

$+$ - operador lógico *ou* indicando opções em linguagens formais

T^∞ - conjunto de todas as seqüências infinitas de transições disparáveis em uma rede de Petri

$L(RP)$ - linguagem de rede de Petri

θ - seqüência de disparos e transições de uma rede de petri

$L_\infty(RP)$ - linguagem infinita de rede de Petri

$h(\theta)$ - função de etiquetagem de uma rede de Petri

$F(RP)$ - conjunto de todas as seqüências finitas de disparos de transições de uma rede de Petri

$F_\infty(RP)$ - conjunto de todas as seqüências infinitas de disparos de transições de uma rede de Petri

\mathbf{S}/G - sistema supervisionado

X - conjunto de estados para um gerador controlado

X_m - conjunto de estados marcados para um gerador controlado

ξ - função de transição parcial estendida para um gerador controlado

K^\uparrow - $\text{SupC}(L)$

Q_{ac} - conjunto de estados da componente acessível de um gerador

$Q_{ac,m}$ - conjunto de estados marcados da componente acessível de um gerador

δ_{ac} - função δ restrita ao domínio $\Sigma \times Q_{ac}$

T^* - conjunto de todas as seqüências finitas de transições disparáveis em uma rede de Petri

$L_\infty^{fair}(RP)$ - linguagem infinita regular de rede de Petri

L_s - linguagem gerada da fórmula CTL

Ω - função de mapeamento (ou etiquetação) de transições em uma rede de Petri sobre o conjunto Prop

$L(RP, f)$ - linguagem definida da rede de Petri na fórmula CTL

Referências Bibliográficas

- [AM87] M. Abadi and Z. Manna. Temporal logic programming. *Proceedings 1987 Symposium on Logic Programming*, pages 4–16, 1987.
- [Bar96] G. C. Barroso. *Uma Nova Abordagem para a Síntese de Supervisores de Sistemas a Eventos Discretos*. PhD thesis, Universidade Federal da Paraíba - Campus II, Campina Grande, PB - BR, 1996.
- [Byr97] N. Byrnes. Investigating features in software systems. Master's thesis, School of Computer Science, The University of Birmingham, 1997.
- [CG95] D. D. Cofer and V. K. Garg. Supervisory control of real time discrete-event systems sing lattice theory. *IEEE Transactions on Automatic Control*, 41(2):199–209, 1995.
- [CH90] X-R Cao and Y-C Ho. Models of discrete event dynamic systems. *IEEE Control System Magazine*, 10(4):69–76, 1990.
- [Che96] C. H. Chen. A lower bound for the correct subset-selection probability and its application to discrete-event systems simulations. *IEEE Transactions on Automatic Control*, 41(8):1227–1231, 1996.
- [Cop68] I. M. Copi. *Introdução à Lógica*. Mestre Jou, 1968.
- [Cos92] M. M. C. Costa. *Introdução à Lógica Modal Aplicada a Computação*. Thaisy Silva Weber, 1992.
- [EH86] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.
- [EK82] E. Emre and P. P. Khargonekar. Regulation of split linear systems over rings: Coeficient assignment and observers. *IEEE Transactions on Automatic Control*, AC-27(1):104–113, 1982.

- [Emr80] E. Emre. On a natural realization of matrix fractions descriptions. *IEEE Transactions on Automatic Control*, AC-25:288–289, 1980.
- [Eva93] J. B. Evans. *The Devnet: a Petri Net for Discrete Event Simulation*. Springer-Verlag, 1993.
- [FG93] M. Finger and D. M. Gabbay. *Adding a Temporal Dimension to a Logic System*. Springer-Verlag, 1993.
- [FG96] L. Fix and O. Grumberg. Verification of temporal properties. *Journal of Logic and Computation*, 6(3):343–361, 1996.
- [Fig94] J. C. A. Figueiredo. *Redes de Petri com Temporização Nebulosa*. PhD thesis, Universidade Federal da Paraíba - Campus II, Campina Grande, PB - BR, 1994.
- [Gal87] A. Galton. *Temporal Logics and their Applications*. Academic Press, 1987.
- [Gal96] A. Galton. An investigation of "non-intermingling" principles in temporal logic. *Journal of Logic and Computation*, 6(2):271–294, 1996.
- [GCV85] J. P. Quadrat G. Cohen, D. Dubois and M. Viot. A linear system theoretic view of discrete event process and its use for performance evaluation in manufacturing. *IEEE Transactions on Automatic Control*, 30(3):210–220, 1985.
- [GCV89] J. P. Quadrat G. Cohen, P. Moller and M. Viot. Algebraic tools for the performance evaluation of discrete event systems. *Proceedings of the IEEE*, 77(1):39–56, january 1989.
- [GD94] A. Giua and F. DiCesare. Blocking and controllability of petri nets in supervisory control. *IEEE Transactions on Automatic Control*, 39(4):818–823, 1994.
- [GFFW92] J. D. Powell G. F. Franklin and M. L. Workman. *Digital Control of Dynamics Systems*. ADDISON-WESLEY Publishing Company, 2 edition, 1992.
- [GL94] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, 1994.

- [GY95] P. Glasserman and D. D. Yao. Subadditivity and stability of a class of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1514–1527, 1995.
- [HK90] L. E. Holloway and B. H. Krogh. Synthesis of feedback control logic for a class of controlled petri nets. *IEEE Transactions on Automatic Control*, 35(5):514–523, 1990.
- [Ho89] Y-C Ho. Dynamics of discrete event systems. *Proceedings of the IEEE*, 77(1):3–6, 1989.
- [HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, USA, 1979.
- [Jan87] M. Jantzen. *Language Theory of Petri Nets*, pages 254–I:397–412. Central Models and their Properties, Advances in Petri Nets, Lectures Notes in Computer Sciences, 1987.
- [JBN96] J. S. Carson II J. Banks and B. L. Nelson. *Discrete-Event System Simulation*. Prentice Hall, 1996.
- [JECN96] B. H. Krogh J. E. Cury and T. Niinomi. A methodology for the design of supervisory controllers for a class of hybrid systems. *Anais do Congresso Brasileiro de Automática*, pages 347–352, 1996.
- [Jen92] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*, volume 1 : Basic Concepts. Springer-Verlag, 1992.
- [KH96] R. Kumar and L. E. Holloway. Supervisory control of deterministic petri nets with regular specification languages. *IEEE Transactions on Automatic Control*, 41(2):245–249, 1996.
- [KP90] J. F. Knight and K. M. Passino. Decidability for a temporal logic used in discrete-event system analysis. *International Journal of Control*, 52(6):1489–1506, 1990.
- [Kro87] B. H. Krogh. Controlled petri nets and maximally permissive feedback logic. In *25th Annual Allerton Conference*, pages 317–326. University of Illinois Urbana, USA, September 1987.
- [Kuc91] V. Kucera. *Analysis and Design of Discrete Linear Control Systems*. Academia, 1991.

- [Lib96] L. Libeaut. *Sur l'utilisation des Diodes pour la Commande des Systèmes a Événements Discrets*. PhD thesis, Ecole Doctorale Sciences pour L'Ingenieur de Nantes, 1996.
- [McM92] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, 1992.
- [Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer Verlag, New York, USA, 1980.
- [MST95] S. Lafortune K. Sinnamohideen M. Sampath, R. Sengupta and D. Teneketzi. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.
- [Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
- [NV95] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of the ACM*, 42(2):458–487, 1995.
- [Oga93] K. Ogata. *Engenharia de Controle Moderno*. Prentice/Hall do Brasil Ltda, 2 edition, 1993.
- [Ost89] J. S. Ostroff. *Temporal Logic for Real-Time Systems*. Research Studies Press Ltd., 1989.
- [PC92] A. Papelis and T. L. Casavant. Specification and analysis of parallel/distributed software and systems by petri nets with transition enabling functions. *IEEE Transactions on Software Engineering*, 18(3):252–261, 1992.
- [PCS91] A. Cristoforetti P. Cofrancesco and R. Scattolini. Petri nets based approach to software development for real-time control. *IEE Proceedings-D Control Theory and Applications*, 138(5):474–478, 1991.
- [Per94] A. Perkusich. *Análise de Sistemas Complexos Baseada na Decomposição de Sistemas de G-Nets*. PhD thesis, Universidade Federal da Paraíba - Campus II, Campina Grande, PB - BR, 1994.
- [Pet81a] J. L. Peterson. *Petri Net Theory and Modeling of Systems*. Prentice Hall, 1981.

- [Pet81b] J. L. Peterson. *Petri Nets Theory and the Modeling of Systems*. Prentice-Hall, N. J., USA, 1981.
- [Pnu77] A. Pnueli. The temporal semantics of concurrent programs. In *18th Symposium on Foundations of Computer Science*, 1977.
- [RB97] P. Racloz and D. Buchs. Symbolic proof of ctl formulae over petri nets. Technical report, C.U.I. University of Geneva, 1997.
- [Rei85] W. Reisig. *Petri Nets An Introduction*. Springer-Verlag Berlin Heidelberg, 1985.
- [Rei92] W. Reisig. *A Primer in Petri Net Design*. Springer-Verlag, 1992.
- [RW89] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.
- [Sak96] J. Sakalauskaitė. Nonclasual resolution system for branching temporal logic. Technical report, Institute of Matematics and Informatic, Lithuania, 1996.
- [SBS92] P. Kozak S. Balemi and R. Smedinga, editors. *Discrete Event Systems: Modeling and Control*. Birkhauser, 1992.
- [SK92] S. R. Sreenivas and B. H. Krogh. On petri net models of infinite state supervisors. *IEEE Transactions on Automatic Control*, 37(2):818–823, 1992.
- [Sre93] R. S. Sreenivas. A note on deciding the controllability of a language k with respect to a language l. *IEEE Transactions on Automatic Control*, 38(4):658–662, 1993.
- [UH90] N. Uchihira and S. Honiden. Verification and synthesis of concurrent programs using petri nets and temporal logic. Technical report, Institute for New Generation Computer Technology, 1990.
- [Vid85] M. Vidyasagar. *Control Systems Syntesis - a Factorization Approach*. MIT Pres, 1985.