Síntese de Supervisores Através de Verificação de Modelo

Wellington de Araújo Bastos

Dissertação de Mestrado submetida à Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Paraíba – Campus II como parte necessária para obter o grau de Mestre em Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Angelo Perkusich, D.Sc.
Orientador

Campina Grande, Paraíba, Brasil

B237s Bastos, Wellington de Araujo
Sintese de supervisores atraves de verificacao de modelo / Wellington de Araujo Bastos. - Campina Grande, 1999.
98 f.

Dissertaca (Mestrado em Engenharia Eletrica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Redes de Petri 2. Sistemas Eventos 3. Dissertacao - Engenharia Eletrica I. Perkusich, Angelo II. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 681.3.01(043)

SÍNTESE DE SUPERVISORES ATRAVÉS DE VERIFICAÇÃO DE MODELO

WELLINGTON DE ARAÚJO BASTOS

Dissertação Aprovada em 10.05.1999

PROF. ANGELO PERKUSICH, D.Sc., UFPB
Orientador

PROF. ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFPB
Componente da Banca

PROF. GIOVANI CORDEIRO BARROSO, D.Sc., UFC
Componente da Banca

CAMPINA GRANDE - PB
Maio - 1999

Agradecimentos

Este trabalho não teria sido possível sem a cooperação de algumas pessoas. Primeiramente, expresso especiais agradecimentos ao Prof. Angelo Perkusich pela supervisão, orientação, leitura e correção desta dissertação. Ao amigo Marcos Ricardo de A. Morais pelo suporte técnico fornecido na instalação das ferramentas PEP e SMV. Ao Prof. Tomaz de Carvalho Barros pela colaboração nos exemplos contidos neste trabalho. A todos os professores e amigos das disciplinas do curso de mestrado da UFPb, que contribuíram de forma direta ou indireta na minha formação e experiência para a conclusão deste trabalho.

Por último, um agradecimento especial a minha família pelo incentivo e força transmitidos durante todo o período de minha formação acadêmica e profissional.

Resumo

As redes de Petri têm sido amplamente empregadas para modelagem, análise e controle de sistemas a eventos discretos. Atualmente muitos autores têm empregado a teoria de controle supervisório em conjunto com as redes de Petri, com o objetivo de buscar uma solução para a síntese de supervisores para sistemas a eventos discretos. Algumas destas abordagens solucionam o problema da síntese de supervisores, no entanto, enfrentam o problema da explosão de estados. Este trabalho introduz um algoritmo de síntese de supervisor para sistemas a eventos discretos, que baseia-se na verificação simbólica de modelos. O Algoritmo é validado através da utilização de duas ferramentas computacionais o PEP (*Pragramming Enviroment based Petri Nets*) e SMV (*Simbolic Model Verifier*). A verificação de modelo é realizada com base na redução da representação do espaço de estado através de *Diagrama de Decisão Binário Ordenado* (OBDD), que possibilita uma representação muito compacta de espaços de estados definidas por expressões booleanas. Neste trabalho os casos estudados são sistemas de produção.

Abstract

Petri nets have been extensively used to the modeling, analysis and control of discrete event systems. Nowadays different researchers use supervisory control theory together with aiming at the synthesis of supervisors for discrete event systems. Many approaches have been proposed in order to solve the synthesis problem, but all them face the problem of state explosion. This work introduces an algorithm to the synthesis of the supervisor for discrete event systems based on symbolic model checking. The introduced algorithm is validated using two different computational tools, namely PEP (*Pragramming Enviroment based Petri Nets*) and SMV (*Simbolic Model Verifier*). Model checking is carried on based on a Ordered Binary Decision Diagrams, that allow a compact representation of the state space based on boolean expressions. Also, in this work we apply the algorithm to two different production systems examples.

Conteúdo

Lista de Figuras

Capítulo 1

1.1 Introdução

Sistemas a Eventos Discretos (SED) são sistemas cuja dinâmica é determinada pela ocorrência de eventos e mudanças de estados em instantes discretos no tempo. A evolução da ocorrência das mudanças de estados de um SED forma um conjunto discreto e quase sempre finito, ao qual denomina-se *espaço de estados*. A natureza discreta do espaço de estado torna possível capturar a lógica e a dinâmica da evolução de sistemas práticos complexos. Os SEDs estão presentes em aplicações tais como sistemas de manufatura, plantas para controle de processos, redes de comunicações, sistemas operacionais, redes de computadores e protocolos de comunicação, entre outros. Este trabalho está particularmente voltado para os Sistemas de Manufatura.

Os SEDs podem ser assíncronos e/ou sequênciais, apresentando características como: paralelismo, concorrência, conflito, exclusão mútua e não determinismo. No sentido de capturar todas estas características vários métodos foram desenvolvidos para modelagem, analise e verificação destes sistemas. Uma breve descrição destes métodos abordando aplicabilidade, vantagens e desvantagens podem ser encontrados em [Bar96], são eles:

- Cadeias de Markov;
- Teoria de Filas;

- Processos Semi-Markovianos Generalizados (*GSM*);
- Álgebra de Processos;
- Álgebra Max-Plus;
- Teoria de Linguagens Formais e Autômatos;
- Redes de Petri.

Dentre estes, os autômatos finitos têm sido amplamente utilizados para modelar e analisar SEDs [Car89], [RW87b] e [RW89]. No entanto, sua utilização é dificultada pela representação gráfica, que aumenta exponencialmente o número de estados do SED à medida em que o mesmo cresce em sua estrutura.

Uma alternativa para este problema é a modelagem de SEDs com redes de Petri (PNs). As redes de Petri [Pet81, Bra83, Rei85, Mur89], são uma ferramenta matemática e gráfica utilizadas na modelagem, análise e verificação de sistemas. A representação gráfica intuitiva compacta e a poderosa formulação matemática das redes de Petri têm possibilitado a sua grande utilização nos SEDs. Comparada com outros métodos [ZD93] as redes de Petri possuem as seguintes vantagens:

- Facilidade para modelar as características de um SED, ou seja: modelar concorrência, sincronismo e assincronismo, conflito, exclusão mútua, relações de precedência, não determinismo e bloqueio;
- Excelente visualização de dependência de sistemas;
- Foco na informação local;
- Disponibilidade de modelagem do tipo refinamento (*top-down*) e do tipo composição modular (*botton-up*);
- Possibilidade de gerar o código de controle supervisório diretamente de sua representação gráfica;
- Possibilidade de verificar propriedades indesejáveis para o sistema, tais como bloqueio e reinicialização;

- Simulação de eventos discretos diretamente a partir do modelo;
- Informações do estado atual do sistema que permite monitoração em tempo real.

Devido as vantagens apresentadas pelas redes de Petri, sua utilização tem crescido amplamente nas aplicações voltadas a SEDs, consequentemente surge a necessidade de ferramentas que viabilizem sua utilização no contexto de sistemas complexos.

Neste trabalho optamos pelo uso do PEP (**P**rogramming **E**nviroment based on **P**etri nets) [BG]. O PEP será apresentado com maiores detalhes no Capítulo 3. Os motivos para sua utilização tornam-se claros no decorrer da apresentação deste trabalho. O PEP é um exemplo de ferramenta que fornece um ambiente de programação baseado em redes de Petri, ela disponibiliza as funcionalidades necessárias para todas as fases de projeto e a análise de sistemas usando modelos de redes de Petri, como mostrado na Figura 1.1 [ZD93]. Uma das principais fases deste processo é a análise estrutural e de comportamento do sistema modelado. Dentre as formas de análise que podem ser empregadas na validação de um modelo encontram-se a simulação e a verificação de modelo. A utilização de simulação com dados de teste, nunca poderá assegurar que um determinado erro não existe (por exemplo, devido a um estado de erro ser pouco provável de acontecer). Este não é o caso da verificação de modelo, que é incorporada ao PEP, em que todos os estados alcançáveis são considerados. Verificadores de modelo já são empregados com sucesso na indústria para encontrar erros sutis em protocolos de comunicação ou sistemas de hardware [McM93].

Um dos métodos de verificação para redes de Petri existentes na ferramenta PEP é baseado em BDD (diagrama de decisão binário) [Bry86]. BDD é um método eficiente para representar espaço de estados muito grandes (maiores que 10^{20}), evitando o problema da explosão de estado. Entre os verificadores de modelos que utilizam o BDD incluem-se o sistema SMV (Simbolic Model Verifier) [McM92] e o sistema VIS (Verification Interacting with Sintesis) [VSS], para o qual cada um toma como entrada um modelo de

estado finito. A ferramenta PEP já incorpora um tradutor de redes de Petri de baixo nível para a linguagem de entrada do SMV. Desta forma, a ferramenta agrega todo o potencial de verificação de modelo disponibilizado pelo SMV.

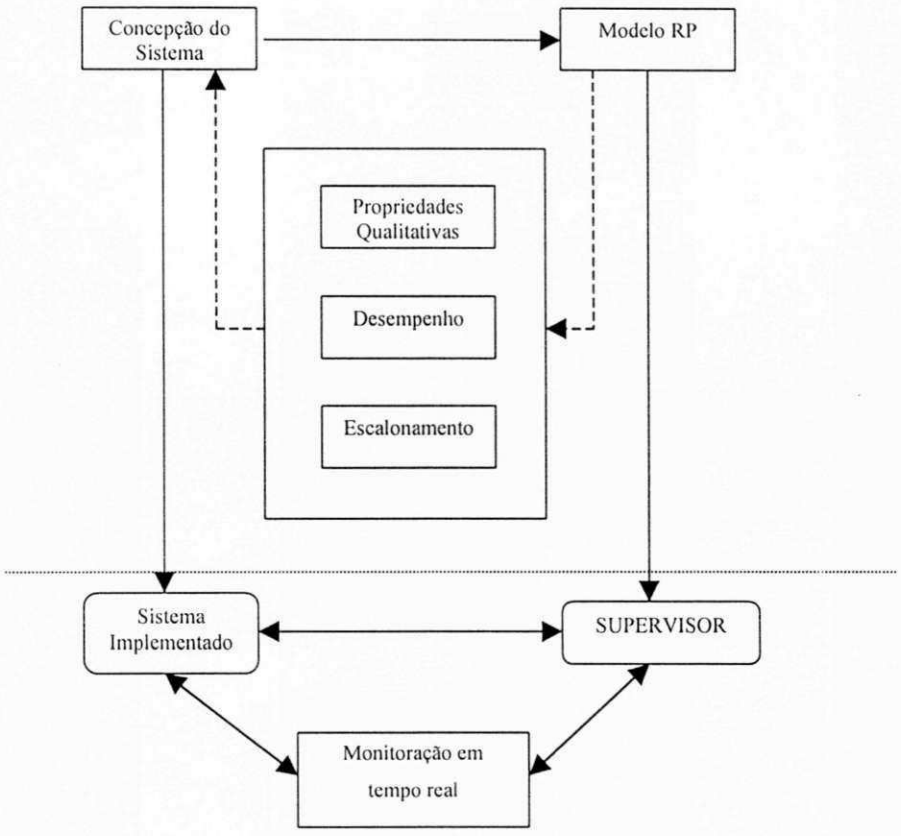


Figura 1.1: Esquema para modelagem, análise e controle de um SED

1.1.1 Controladores para SEDs

Normalmente é necessário regular ou supervisionar o comportamento de um sistema para que este funcione adequadamente de acordo com alguns critérios preestabelecidos, por exemplo, prevenir que veículos automaticamente guiados (AGVs) colidam no chão de fábrica, pela restrição de seus acessos a certas zonas de tráfego mútuo.

Supervisores de SEDs são utilizados para assegurar que o comportamento de uma planta (sistema a ser controlado) não viole um conjunto de restrições sob uma variedade de condições de operação (comportamento desejável). As ações reguladoras do supervisor são baseadas na observação dos eventos e estados da planta, resultando na realimentação de controle.

Ramadge e Wonham [RW87b, RW87c, RW89] desenvolveram uma teoria de controle para SEDs baseada em linguagens formais e autômatos, denominada de *Teoria de Controle Supervisório* (TCS). A teoria estabelece as condições para a existência de um supervisor baseada na linguagem gerada pelo autômato que modela o sistema, mais a linguagem formal que especifica o comportamento. Desta maneira, a síntese de supervisor consiste em encontrar o autômato controlador formado por supervisor/sistema que gere uma linguagem satisfazendo a especificação.

Um supervisor controla o SED através de ações que habilitam ou inibem a ocorrência de um evento controlável. Desta forma, a TCS separa explicitamente o sistema a ser controlado (malha aberta) do controlador (realimentação de controle), portanto, apresentando as condições necessárias e suficientes para a existência do supervisor.

Vários autores [RW88, RW87a], [LW88a, BW94], [BHG+93] e [LW88b] apresentaram modelos de supervisores baseados em autômatos e linguagens formais, no entanto, os problemas de controle tratados por estas abordagens são para exemplos simples. Isto pois a análise de sistemas reais modelados por autômatos se torna difícil, devido principalmente a problemas de explosão de estados e outros fatores peculiares a cada abordagem. Uma comparação entre estas várias abordagens pode ser encontrada em [Bar96].

Por outro lado, vários outros autores, Inan e Varayia [IV88], Guia e DiCesare [GD91, GD92, GD94a], Sreenivas e Krogh [SK92], Sreenivas [Sre93], Holloway e Krogh [HK90], Boel et. Al. [BLNB95], Barroso et. Al. [BLP95, BLP96a], Moody e Antsaklis [MA98], investigam a utilização de redes de Petri em conjunto com a TCS para modelar o controle de SEDs. Uma descrição destes vários métodos pode ser encontrado em [Bar96].

A utilização das redes de Petri na TCS em lugar de modelos mais simples tais como máquinas de estados apresenta várias vantagens, dentre elas [Bar96]:

- a estrutura de uma rede não requer uma enumeração explícita de todos os estados do sistema e em muitos casos podem ser utilizadas técnicas de análise, baseadas somente na estrutura da rede, que podem permitir uma boa solução para o problema de controle;
- o modelo de rede de Petri cresce linearmente com o número de componentes do sistema modelado;
- uma rede de Petri supervisora contém uma ação de controle implícita na própria estrutura. Nesse caso o modelo em malha fechada do sistema controlado pode ser construído e analizado usando-se as mesmas técnicas de análise de redes de Petri que são usadas para se analizar o sistema.

Barroso [Bar96] apresenta uma solução para síntese de supervisores baseado no *Algoritmo da Árvore de Alcançabilidade* [Pet81, Mur89] e o nos algoritmos apresentados por Ramadge e Wonham [RW87b] e Ziller [ZC94]. A síntese do supervisor é realizada executando-se os algoritmos tendo como dados o modelo do sistema e seu comportamento desejável.

A utilização do *Algoritmo da Árvore de Alcançabilidade* para a análise de sistemas a eventos discretos é impraticável para a grande maioria dos sistemas reais, devido ao grande número de estados que os mesmos podem alcançar, inviabilizando sua análise. Este problema é melhor gerenciado através de uma abordagem modular baseada em *G-Nets* [PdFC94, Per94] e *redes de Petri Coloridas* [Jen92].

1.1.2 Objetivo e Organização deste Trabalho

Conforme introduzido na seção anterior várias abordagens para síntese de supervisores utilizando redes de Petri em conjunto com a Teoria de Controle Supervisório foram definidas. Uma das dificuldades apresentadas pelas abordagens cuja síntese baseia-se na árvore de alcançabilidade é a explosão do número de estados determinados por sistemas reais, tornando a análise destes sistemas bastante difícil.

Este trabalho apresenta uma abordagem para a síntese de supervisores, usando a Teoria de Controle Supervisório e redes de Petri Lugar/Transição limitadas, modelando sistemas de manufatura reais. A síntese é baseada em um algoritmo de *Síntese de Supervisor com SMV* que utiliza a redução da representação do espaço de estados gerado pelo sistema não controlado, através da representação implícita, usando OBDD (Diagrama de Decisão Binário Ordenado). O algoritmo é executado sobre a especificação desejada para o sistema, através da ferramenta de verificação de modelo SMV, expressos em fórmula de lógica temporal (CTL). A especificação desejada é composta por um conjunto de transições de estado que representam a evolução de estados do SED. O espaço de estados representado pelo OBDD é gerado pela rede de Petri que modela o SED, utilizando a ferramenta PEP para modelagem e geração do programa de entrada do SMV.

Esta dissertação está estruturada da seguinte forma: No Capítulo 2 são apresentados os fundamentos teóricos de redes de Petri, teoria de controle supervisório, verificação de modelo, lógica temporal CTL, o algoritmo de verificação de modelo, diagrama de decisão binário, e exemplo de CTL aplicado a sistemas a eventos discretos. No Capítulo 3 são apresentadas as ferramentas de modelagem e análise (PEP) e SMV. No Capítulo 4 é introduzido o algoritmo de Síntese de Supervisor com SMV (SSS). No Capítulo 5 apresentamos dois exemplos de aplicação do algoritmo SSS a SED, o primeiro exemplo é um Sistema de Produção em Lote (SPL) e o segundo um Sistema Flexível de Manufatura (SFM). Finalizando são apresentadas as conclusões no Capítulo 6. No Anexo A são apresentados os *traces* da aplicação do algoritmo SSS no exemplo *Um Sistema Flexível de Manufatura (SFM)* apresentado no Capítulo 5.

Capítulo 2

Fundamentos Teóricos

O propósito deste Capítulo é apresentar os conceitos básicos de redes de Petri, Teoria de Controle Supervisório e verificação de modelo baseado em Diagrama de Decisão Binário Ordenado (*Ordered Binary Decision Diagram - OBDD*). Estes conceitos são apresentados com o objetivo de disponibilizar ao leitor o embasamento teórico fundamental para o entendimento deste trabalho. A apresentação dos conceitos é baseada em Wimmel [Win97], Bryant[Bry86], McMillan [McM93] e Barroso [Bar96].

2.1 Redes de Petri

Uma rede de Petri é um grafo direcionado bipartido e ponderado que possui um estado inicial. Uma rede de Petri é descrita por uma quintupla $N = (P, T, F, W, m_0)$, em que P e T são conjuntos disjuntos que representam os vértices de um grafo, conhecidos como lugares e transições respectivamente. Estes vértices são interligados por um conjunto de arcos, representados por F. Desta forma uma rede de Petri N é definida formalmente por:

$P = \{p_1, p_2, ..., p_m\}$ é um conjunto finito de *lugares*,

$T = \{t_1, t_2, ... t_n\}$ é um conjunto finito de *transições* $\{P \cap T = \varnothing$ e $P \cup T \neq \varnothing\}$,

$F \subseteq (P \times T) \cup (T \times P)$ é o conjunto de *arcos*,

$W: F \rightarrow IN^*$ é uma função peso, e

$m_0 : P \rightarrow IN$ é a marcação inicial.

$\forall x \in P \cup T$, $\bullet x = \{y \mid (y,x) \in F\}$ é o *preset* de x e $x\bullet = \{y \mid (x,y) \in F\}$ o *postset* de x.

Uma rede de Petri pode ser visualizada como um grafo direto e bipartido que consiste de dois tipos de nós (vértices): lugares e transições, representados pelos símbolos círculos e retângulos respectivamente. Os arcos de uma rede de Petri são desenhados com uma seta direcionada de uma transição para um lugar ou vice-versa. A Figura 2.1 mostra uma rede de Petri simples.

Uma marcação de uma rede de Petri é uma função $m : P \rightarrow IN$ que atribui um inteiro não negativo a cada lugar. No grafo correspondente, se $m(p) = k$, k pontos (fichas) são colocadas dentro do círculo que representa um lugar. No exemplo da Figura 2.1, os lugares p_1 , p_2 e p_5 estão marcados pois possuem cada um uma ficha, e os outros lugares não estão marcados.

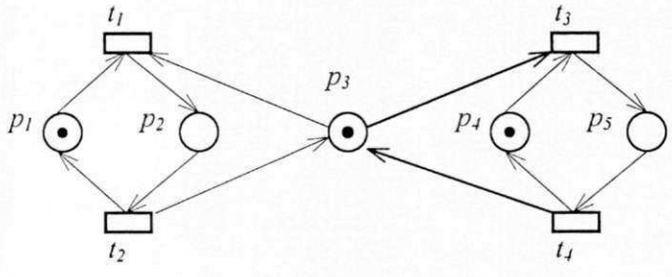


Figura 2.1 - Rede de Petri (Exclusão Mútua)

O comportamento de uma rede de Petri é determinado como segue. Uma transição t está habilitada quando cada lugar precedente possui pelo menos uma ficha. Observando o exemplo da Figura 2.1, t_1 está habilitada (porque p_1 e p_3 estão marcados) e t_3 também está habilitada. Uma rede de Petri modifica sua marcação através da ocorrência não deterministica de uma transição. Quando uma transição ocorre para uma marcação m, fichas são removidas dos lugares de entrada e fichas são depositadas nos lugares de saída,

em que o número de fichas removidas e adicionadas são determinadas pelo peso do arco (w) respectivo ao lugar de estrada e saída, tal que uma nova marcação m' é alcançada. A mudança de marcação pode ser escrita como $m[T\rangle m'$, com

$$m'(p) = \begin{cases} m(p) - w & se\ p \in \bullet T \setminus T \bullet \\ m(p) + w & se\ p \in T \bullet \setminus \bullet T \\ m(p) & qualquer\ outro \end{cases}$$

Suponha que a transição t_1 ocorra para a rede de Petri da Figura 2.1. Então fichas são removidas dos lugares p_1 e p_3, desta forma $m'(p_1) = 0$ e $m'(p_3) = 0$, e uma ficha é adicionada a p_2, tal que $m'(p_2) = 1$. Quando m é escrito na forma de um vetor ($m(p_1)$, ..., $m(p_5)$), a transformação de uma marcação em outra, devido a ocorrência de uma transição, pode ser denotada como $[10101]^T \xrightarrow{t_1} [01001]^T$.

Para concluir a seção introdutória de redes de Petri, algumas definições importantes que serão utilizadas no decorrer desta dissertação são apresentadas.

Uma marcação m' é *alcançável* a partir de uma marcação m, se existe uma seqüência de transições $t_1, t_2,...,t_n$ tal que $m[t_1 t_2 \Lambda\ t_n\rangle m'$. Uma marcação m' é *alcançável*, se esta é alcançada a partir do estado inicial $m_0.t_1 t_2...t_n$ é chamada de *seqüência de disparo*. Observe que o comportamento de uma rede de Petri pode ser determinado pela construção do estado de espaço e a observação de estados alcançáveis pela rede através das seqüências de disparo.

Uma rede de Petri é dita limitada se para cada lugar p da rede existe um inteiro k tal que toda marcação alcançável $m(p) \leq k$.

Uma rede de Petri é dita *segura* se ela é 1-limitada, ou seja, se nenhum lugar em qualquer marcação alcançável possui mais que uma ficha. Neste caso especial, uma transição T está habilitada se cada lugar de entrada possui exatamente uma ficha, portanto

$$m'(p) = \begin{cases} 0 & se\ p \in \bullet T \setminus T \bullet \\ 1 & se\ p \in T \bullet \setminus \bullet T \\ m(p) & qualquer\ outro \end{cases}$$

Devido a abordagem adotada nesta dissertação todas as redes consideradas devem ser limitadas e seguras. Entretanto, para os casos em que a rede é limitada mas não é segura um método para modelagem de buffers seguros é descrita logo a seguir.

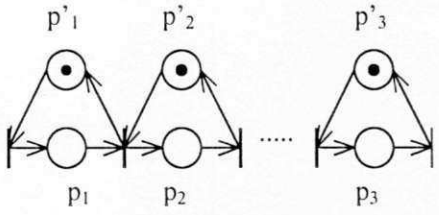
2.2 Modelando Buffers com Redes de Petri

No contexto de sistemas de manufatura, áreas de armazenagem, estoques, ou buffers são muito comuns. Por exemplo, em uma linha de produção podem existir vários buffers entre as máquinas. Esta seção apresenta um método para modelagem de buffers para sistemas utilizando redes de Petri seguras [ZD93]. Este método é de extrema importância no contexto desta dissertação, uma vez que o processo de verificação de modelo utilizado para a síntese do supervisor é baseado em redes de Petri seguras. As redes de Petri resultantes do método de modelagem para buffers mantêm as propriedades de vivacidade, limitação e reversabilidade.

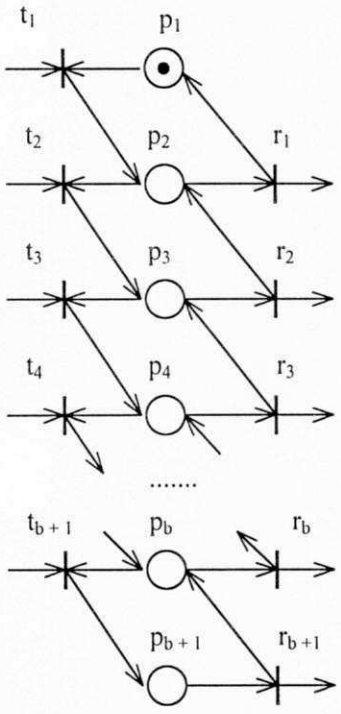
2.2.1 Módulos de Redes de Petri Seguras para Buffers

Agerwala [Ager79] fornece a implementação de um buffer de tamanho dois com rede de Petri segura usando quatro lugares e três transições. O buffer está localizado entre dois estágios de um *pipeline*. Desta maneira este caso pode ser facilmente generalizado para um buffer de capacidade b. O módulo com rede segura está ilustrado na Figura 2.2(a) utilizando $2b$ lugares e $b + 1$ transições. Deve ser observado que um módulo assume que os elementos são ordenados e estão sujeitos a uma política de First In First Out (FIFO). Neste modelo cada ficha (peça em um sistema de manufatura) que entra no buffer para alcançar sua próxima operação necessita do disparo de b transições, i.e., t_2 até t_{b+1}.

Para evitar o determinismo existente no modelo anterior, um outro módulo de rede de Petri para um buffer com capacidade b é apresentado na Figura 2.2(b) com marcação inicial $[1, 0, 0, ...,0]^T$ usando b+1 lugares e $2(b + 1)$ transições.



(a)



(b)

Figura 2.2 – Implementação de um módulo de buffer seguro com capacidade b. (a) FIFO; (b) Não determinístico

Se $b = 1$, então o buffer já é modelado por uma rede de Petri segura. Na descrição que segue, assume-se $b > 1$. Para $i = 1, 2, ..., b+1$,

- t_i : transição que liga o lugar (operação em sistema de manufatura) precedente ao buffer;
- r_i : transição que liga o lugar (operação em sistema de manufatura) sucessora do buffer;
- p_i : lugar modelando buffer, se este está marcado o buffer contém i-1 peças.

Desta forma, se p_1 está marcado, então o buffer está vazio. Se p_{b+1} está marcado, então o buffer está cheio. As seguintes propriedades podem ser verificadas para a rede de Petri apresentada na Figura 2.2(b).

- Se p_1 está marcado, então nenhuma transição r_i não está habilitada, ou nenhuma operação sucessora pode ocorrer. Desta forma, apenas uma operação antecessora pode ser executada (buffer vazio)
- Se p_{b+1} está marcado, então nenhuma transição t_i está habilitada e apenas r_{b+1} está habilitada. Desta forma, apenas uma operação sucessora pode ser executada (buffer cheio)
- Para $i = 2, 3, ..., b+1$, t_i e r_{i-1} estão habilitadas, mas as outras transições não estão habilitadas.
- Apenas uma peça no buffer pode ser entregue para a próxima operação a cada ocorrência de qualquer transição r_i e apenas uma peça pode entrar no buffer para cada disparo de qualquer transição t_i.

É importante observar que diferente do modelo apresentado na Figura 2.2(a), o modelo da Figura 2.2(b) preserva o não determinismo inerente ao modelo de rede de Petri.

Todos estes modelos de redes de Petri seguras, para um buffer, oferecem a possibilidade de exploração das operações internas de um buffer. Elas também fornecem outro esquema de implementação para buffer usando apenas lugares seguros.

2.3 Teoria de Controle Supervisório (TCS)

Em Sistema de Controle Supervisório [RW89], o comportamento do sistema é representado por uma quintupla $G = (\Sigma, Q, \delta, q_0, Q_m)$, chamado gerador. O símbolo Σ representa o conjunto de eventos, ou alfabeto de eventos; Q é o conjunto de estados para uma dada planta; $\delta : \Sigma \times Q \rightarrow Q$ é a função de transição definida para qualquer $q \in Q$, tal que $\delta(\sigma, q)$ é definida para alguns subconjuntos $\Sigma(q) \subset \Sigma$ que depende de q. Os estados marcados Q_m representam o fim de uma seqüência de eventos, e também representam a execução de uma tarefa pelo sistema. O conjunto de todas as *palavras* formadas por algum número de símbolos de Σ, incluindo o símbolo vazio ε, é denotado por Σ^*. A função de transição δ é extensível a palavras de Σ^*.

Um estado q é dito ser *acessível* se e somente se $\exists s \in \Sigma^* \mid \delta(s, q_0) = q$. Um estado q é dito ser *coacessível* se e somente se $\exists s \in \Sigma^* \mid \delta(s, q) \in Q_m$. G é *ajustado* se e somente se este é acessível e coacessível.

Cada gerador G tem duas linguagens associadas: $L(G)$ é a linguagem gerada por G e $L_m(G)$ é a linguagem marcada de G. Estas linguagens são conjuntos de *palavras* definidas em Σ. A linguagen $L(G)$ representa o comportamento fisicamente possível do sistema, enquanto $L_m(G)$ representa as tarefas que o sistema é capaz de completar.

Para controlar um SED é necessário admitir que alguns eventos podem ser desabilitados quando necessário. Para modelar tal ação de controle, o conjunto Σ é particionado em: i) Σ_c o conjunto de eventos *controláveis* e ii) Σ_u conjunto de eventos *não-controláveis*, tal que $\Sigma = \Sigma_c \cup \Sigma_u$ e $\Sigma_c \cap \Sigma_u = \varnothing$. Todos os eventos em Σ_c podem sempre ser desabilitados, enquanto aqueles em Σ_u não sofrem influência da ação de algum controle externo.

Uma *entrada de controle* para G consiste de um subconjunto $\Gamma \subseteq \Sigma$, satisfazendo $\Sigma_u \subseteq \Gamma$. Se $\sigma \in \Gamma$, então σ é habilitado por Γ, de outra forma σ está desabilitado. A condição $\Sigma_u \subseteq \Gamma$ significa que os eventos em Σ_u estão sempre habilitados.

Vamos chamar $\Gamma \subseteq 2^n$ o conjunto de entradas de controle. Um SED representado por *G*, com um conjunto de entradas de controle Γ é um SED controlado (SEDC).

Controlar um SEDC *G*, consiste na geração de uma seqüência de entradas de controle γ, γ', γ", ... $\in \Gamma$, em resposta à observação prévia dos eventos gerados por *G*. Desta maneira um controlador será chamado de *supervisor*.

Um supervisor é um mapeamento $f : L \rightarrow \Gamma$, especificando para cada possível palavra de eventos gerada ϖ, a entrada de controle $f(\varpi)$ que será aplicada.

Um supervisor é representado por um autômato e um mapa de saída, i.e., $S = (\sum, X, \xi, x_0)$ é um autômato e $\Theta : X \rightarrow \Gamma$ é o mapa de saída.

Diz-se que o par (S, Θ) realiza o supervisor *S* se para cada $s \in L(G/S)$, $\Theta(\xi(s, x_0))$, em que $L(G/S)$ representa o comportamento fechado do sistema modelado por *G* e supervisionado por *S*.

O problema básico de controle supervisório pode ser circunstanciado como segue: Dado um *SED G* com comportamento em malha aberta definido por *L* e a especificação desejada *K*, qual comportamento em malha fechada $K \subset L$ pode ser realizado pelo supervisor? Para resolver este problema é necessário definir o conceito de *controlabilidade*.

Dadas duas linguagens arbitrárias *L*, $K \subseteq \Sigma^*$ e um alfabeto $\Sigma = \Sigma_c \cup \Sigma_u$, diz-se que *K* é *L-fechado* se e somente se $K = \overline{K} \cap L$, e *K* é *L-controlável* se e somente se: $\overline{K} \Sigma_u \cap L \subseteq \overline{K}$, onde $\overline{K}$ é um prefixo de *K*, i.e., o conjunto de todos os prefixos de uma palavra em *K*.

Dado um gerador *G*, a linguagem $K \subseteq L_m(G)$ modela a especificação desejada. A linguagem *K* representa as tarefas a serem executadas sobre supervisão. Então, o objetivo é encontrar o supervisor próprio *S* para *G* tal que o sistema em malha fechada satisfaça a condição $L(S/G) = K$.

Existe um supervisor *próprio* tal que $L(S/G) = K$ se e somente se $L_m(G)$-fechado e $L(G)$-*controlável* [RW89].

Quando, em alguma situação, a linguagem especificada K não satisfaz às condições estabelecidas para a existência do supervisor, sempre é possível encontrar uma sublinguagem $K^{\uparrow} \subseteq K$ que satisfaz às condições de maneira minimamente restritiva. Esta linguagem é denominada *suprema linguagem controlável* $K^{\uparrow}$ ou sup $C(L)$ [RW87b].

2.4 Verificação de Modelo

Verificação de Modelo é um método de verificar se uma dada especificação (em uma fórmula lógica) é válida em um sistema finito ("modelo"). Para este propósito o sistema deve primeiro ser convertido em um formato simples "verificável", chamado *modelo de Kripke*. Esta conversão é usualmente executada por um programa de verificação, que recebe como entrada uma descrição dos sistemas em uma linguagem de alto nível.

Um modelo de Kripke é uma tripla (Q, R, L), onde

- Q é um conjunto finito de estados;
- $R \subseteq Q \times Q$ é a relação de transição, em que $(q,t) \in R$ significando que t é um estado *imediatamente sucessor* de s;
- $L : Q \rightarrow (AP \rightarrow B)$ é a avaliação de uma proposição atômica em cada estado, onde AP é o conjunto finito de proposições atômicas podendo ser Booleano (verdadeiro ou falso)

Um modelo de Kripke de um sistema pode ser ilustrado por um sistema de transição de estados rotulado, em que os rótulos de um estado são os valores da proposição atômica neste estado. O sistema de transição de estados pode ser transformado em uma árvore infinita iniciando em um estado inicial arbitrário s_0 do sistema, denominada *árvore computacional*. A Figura 2.3 mostra um sistema de transição de estado de um sistema e sua

correspondente árvore computacional. O estado inicial é s_0 e neste caso, o sistema pode executar duas ações diferentes e alcançar ou s_1 ou s_2.

Um caminho de um modelo $K = (Q, R, L)$ é uma sequência infinita de estados $(q_0, q_1,...)$ tal que $(q_i , q_{i+1}) \in R$., que corresponde ao caminho na árvore computacional iniciando em s_0 e representa um possível comportamento para o sistema.

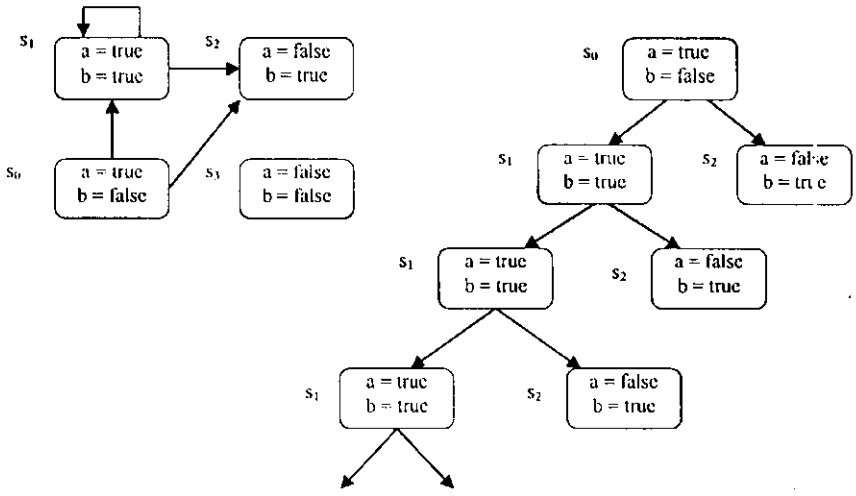


Figura 2.3 Modelo de Kripke e correspondente árvore computacional

2.4.1 Lógica Temporal CTL

Lógica temporal é um sistema para especificar mudanças no tempo, podendo ser empregada para especificar momentos passados ou futuros com relação a momentos presentes [Bur84].

A *Lógica Temporal CTL* (Computation Tree Logic) é um caso particular de *Branching Time Logic* [CE81b] no qual o ordenador temporal < (exemplo, seja s e t dois estados no tempo, então $s < t$, significa que s é inferior a t no tempo) define uma árvore que ramifica-se em direção ao futuro. Desta forma, todo instante tem apenas um passado, mas um futuro indeterminado. As fórmulas em CTL são construídas de proposições atômicas (correspondendo a variáveis), operadores booleanos como $\neg$ (! ou NOT), $\vee$ (| ou OR), $\wedge$ (& ou AND), $\rightarrow$ (INPLY), e operadores temporais. Cada operador temporal consiste de

duas partes, um quantificador de caminho (*A* ou *E*) seguido pelos modificadores temporais (*F, G, X, U*). Os operadores temporais são interpretados com relação ao "estado corrente" *s*. O quantificador de caminho indica se o operador temporal expressa uma propriedade que deve conter em todos os caminhos partindo de *s* (denotado pelo quantificador universal de caminho *A*), ou o menor destes caminhos (denotado pelo quantificador de caminho existencial *E*). Desta forma, operadores de tempo passado não são permitidos.

As modalidades temporais são interpretadas como seguem:

- $X\phi$ ("neXt time ϕ") é verdadeiro se a fórmula ϕ é verdadeira no estado alcançável imediatamente após o estado corrente no caminho.
- $F\phi$ ("Future ϕ") é verdadeiro se existe um estado no caminho em que a fórmula ϕ é verdadeira
- $G\phi$ ("Globally ϕ") é verdadeiro se ϕ é verdadeira em todos os estados no caminho
- $\phi U \psi$ ("ϕ Until ψ") é verdadeiro se existe um estado no caminho no qual ψ é verdadeira e ϕ é verdadeira em todos estados precedentes (se algum).

A Figura 2.4 ilustra o significado de operadores temporais através dos modelos de Kripke no qual *p* é verdadeiro em alguns estados tal que a fórmula correspondente seja satisfeita.

As fórmulas *CTL* podem ser utilizadas para expressar propriedades e/ou o comportamento de sistemas, a seguir são listadas algumas fórmulas *CTL* frequentemente usadas:

- *AGp*, significando "nada mau (i.e.,$\neg p$) sempre acontece". *p* especifica um invariante, i.e. *p* deve ser verdadeiro em todos os estados. Esta fórmula pode ser usada para verificar exclusão mutua (dois processos nunca estão em uma seção crítica ao mesmo tempo) ou ausência de *deadlock*.

- *AFp*, "algo bom, *p*, deve eventualmente acontecer" (i.e.. o sistema deve produzir a resposta correta).
- *AG AFp*, significando "infinitamente *p* livre". Esta fórmula pode ser usada para verificar vivacidade, por exemplo, um processo está no estado "executando" em um número infinito de vezes para alguma execução.
- $AG(p \rightarrow AFq)$, significando "de todos os estados alcançáveis no qual *p* é verdadeiro, algo bom, *q*, deve eventualmente acontecer" (por exemplo., uma requisição de mensagem será eventualmente reconhecida).
- *EFp*, significando "*p* é possível".
- *AG EFp*, significando "de todos os estados alcançáveis, *p* é possível" (por exemplo, é sempre possível reiniciar o sistema).

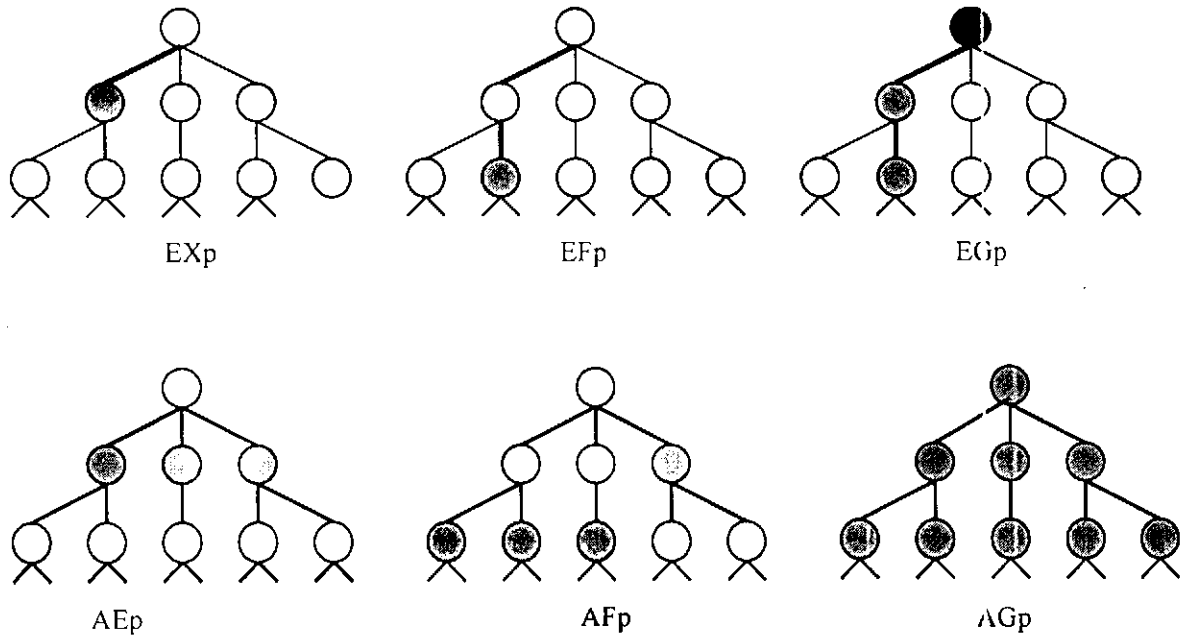


Figura 2.4 – Operadores Temporais de CTL

2.4.2 O Algoritmo de Verificação de Modelo

O algoritmo de verificação de modelo é o principal módulo do programa de verificação formal. O mesmo avalia uma dada fórmula CTL pela computação de um conjunto de estados para o qual esta é verdadeira. Na verdade, uma fórmula CTL e o conjunto de estados que a satisfaz são identificados.

Se o estado inicial s_0 do sistema está contido no conjunto dos estados que satisfazem à fórmula f (note que pode ser escrito como q_0 satisfaz f ($q_0 \models f$)), a fórmula CTL é verdadeira para o sistema, do contrário esta é falsa. Em alguns casos é possível determinar um contra-exemplo (um caminho de execução mostrando que uma fórmula não é verdadeira).

O algoritmo avalia uma fórmula CTL pela divisão recursiva em subfórmulas como segue:

- As fórmulas verdadeiras e falsas são identificadas com $\varnothing$ resp. Q. No qual Q é um conjunto finito de estados .
- Uma proposição atômica p simplesmente denota o conjunto de estados para os quais a avaliação de p é verdadeira.
- Fórmulas como $\neg f$, $f \vee g$, $f \wedge g$ são avaliadas em seus componentes f e g usando os operadores de conjunto \ (restrição), $\cup$ (união) e $\cap$ (intercessão), ou seja, $S \backslash f$ (tudo menos f), $f \cup g$, $f \cap g$ respectivamente.

Para as fórmulas AXf ou EXf, a relação de transição é usada (e.g. AXf é verdadeiro em todos os estados nos quais um sucessor imediato satisfaz f).

Os operadores temporais restantes são avaliados usando uma técnica chamada *interação funcional*. Para exemplificar esta técnica será considerada a fórmula $EF\,f$. Os outros operadores temporais são tratados similarmente.

É intuitivamente claro que EFf (existe um caminho no qual f existe em algum momento no futuro) é verdadeiro se f é verdadeiro para o estado inicial ou EFf existe para

um de seus sucessores imediatos. Isto pode ser denotado como $EFf = f \vee EX\ EFf$. Se definimos uma função $\tau[Y] = f \vee EX\ Y$ (algum conjunto de estados pode ser substituído por Y), então $\tau[EFf] = f \vee EX\ EFf = EFf$. A fórmula EFf é chamado de um *ponto fixo* de τ. Neste caso, esta é realizada iniciando $Y = \varnothing$ e interagindo com $Y = \tau[Y]$ até $Y = \tau[Y]$ ser alcançado. Desta forma, EFf é computado como $f \vee EX(f \vee EX(f \vee EX(...)))$.

Como o tamanho de Y incrementa a cada passo e Q é finito, então esta converge. Mais informações sobre o algoritmo de verificação simbólica e teoria de ponto fixo pode ser encontrado em [McM93].

2.4.3 Verificação simbólica de modelo e BDDs

O algoritmo de verificação de modelo descrito na seção anterior sofre do problema de explosão de estados, porque o conjunto de estados gerado é usualmente muito grande para ser representado explicitamente. No caso de sistemas complexos um sistema com k processos independentes, cada um dos quais podendo definir um espaço com n *estados*. Este sistema então possui n^k estados na totalidade. Em geral, o número de estados cresce exponencialmente com o número de componentes e pode facilmente exceder 10^{20}. Desta forma, faz-se necessário buscar uma técnica para representar o conjunto de estados mais eficientemente. Uma solução para este problema é a *verificação de modelo simbólica*, que torna possível verificar sistemas com 10^{20} estados ou mais [BCM+90].

A idéia principal da verificação simbólica de modelo é que os conjuntos de estados sejam representados simbolicamente, usando fórmulas booleanas. Por exemplo, o conjunto de estados no qual as variáveis a e b são ambas verdadeiras pode ser representada pela fórmula booleana $a \wedge b$.

Todas as operações necessárias ao algoritmo de verificação de modelo podem ser executadas usando operações booleanas, e.g, $a \vee b$ representa a união dos conjuntos de estados no qual a é verdadeiro com o conjunto de estados em que b é verdadeiro.

A relação de transição é representada pela duplicação de variáveis, com next(v) significando o valor da variável v no próximo estado. Suponha que um sistema seja definido para as variáveis $v_1,...,v_n$. A relação de transição é então representada como uma fórmula booleana contendo as variáveis $v_1,...,v_n$, next(v_1), ..., next(v_n). Se esta fórmula converge para verdadeiro através de $v_1 = x_1,..., v_n = x_n$, next(v_1) = y_1, ..., next(v_n) = y_n, então o estado em que $v_1 = y_1,..., v_n = y_n$ é um sucessor imediato do estado em que $v_1 = x_1,..., v_n = x_n$. Por exemplo, na Figura 2.3, observamos que a relação de transição é $a \wedge$ next(b). Desta forma, o algoritmo de verificação simbólica de modelo pode ser implementado inteiramente através da manipulação de fórmula, minimizando o problema de explosão de estados. Todavia, para tornar a verificação de modelo prática, uma estrutura de dados eficiente é necessária para representar fórmulas booleanas. Diagrama de Decisão Binário Ordenado (*Ordered Birary Decision Diagram*, *OBDDs*) é uma estrutura de dados, introduzida por Bryant em 1986 [Bry86], e que será apresentada a seguir.

2.4.4 Diagrama de Decisão Binário (BDD)

Um diagrama de decisão binário representa uma função booleana como uma árvore computacional. Como um exemplo, a Figura 2.5 ilustra a representação de uma função *f*(X1,X2,X3) definida pela tabela verdade dada no lado esquerdo da figura. Cada vértice não-terminal v é rotulado por uma variável *var(v)* e tem arcos diretamente ligados a dois filhos, *lo(v)* (mostrado com linhas pontilhadas) correspondendo ao caso em que à variável é atribuído o valor lógico 0 (zero) e *hi(v)* (mostrado com uma linha sólida) correspondendo ao caso em que à variável é atribuído o valor lógico 1 (um). Cada vértice terminal é rotulado com 0 ou 1. Para uma dada atribuição dada às variáveis, o valor assumido por uma função é determinado traçando-se o caminho desde a raiz até o vértice terminal, seguindo os ramos indicados pelos valores atribuídos às variáveis. O valor da função então é dado pelo rótulo do vértice terminal. Devido ao caminho dos galhos estarem ordenados nesta figura, os valores do vértice terminal, lidos da esquerda para a direita, iguala-se ao da tabela verdade, lidos do topo para baixo.

X1	X2	X3	*f*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

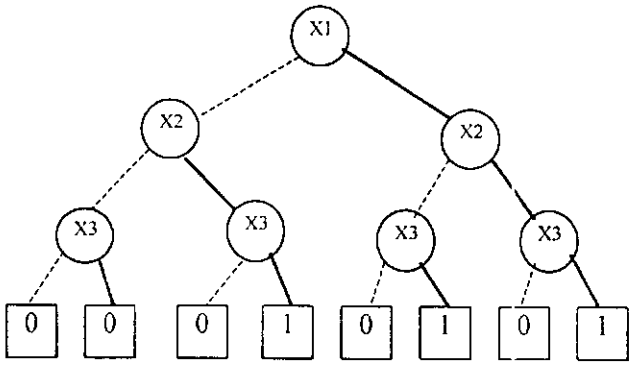


Figura 2.5 - Tabela Verdade e árvore de decisão representando a função booleana. Um galho tracejado (sólido) denota o caso em que a variável de decisão é 0 (1).

2.4.4.1 Ordenação e Redução

Para um *Diagrama de Decisão Binário Ordenado* (OBDD), temos que impor uma ordenação total, em que < significa a ordem de distribuição das e exigimos que para qualquer vértice *u*, e cada filho não-terminal *v*, suas respectivas variáveis devem ser ordenadas de tal forma que *var(u)* < *var(v)*. Para a árvore de decisão apresentada na Figura 2.5, por exemplo, as variáveis são ordenadas de forma que X1 < X2 < X3. A princípio a ordenação das variáveis pode ser selecionada arbitrariamente - o algoritmo operará corretamente para qualquer ordenação. Na prática, a seleção de uma ordenação satisfatória é crítica para uma manipulação simbólica eficiente, como será descrito mais adiante

2.4.4.2 Regras de transformação para os grafos

Remover Terminais Duplicados

Eliminar todos os vértices terminais com o mesmo rótulo, exceto um e redirecionar todos os arcos dos vértices eliminados para o único remanescente. Ver a Figura 2.6(a).

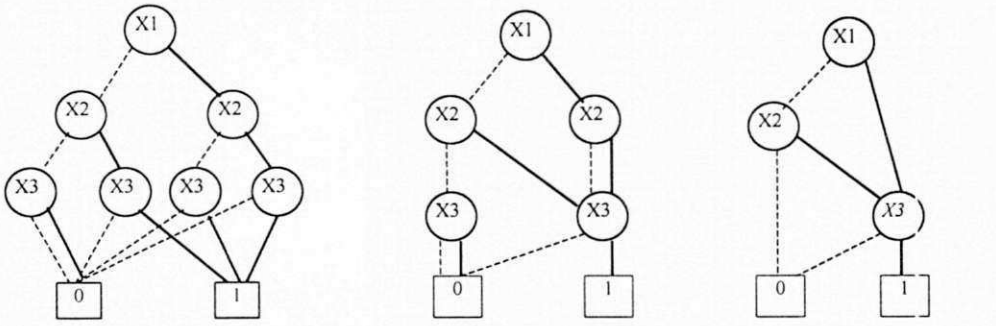
Remover Vértices Não-terminais Duplicados

Se os vértices não terminais *u* e *v* têm *var(u) = var(v)*, *lo(u) = lo(v)*, e *hi(u) = hi(v)*, então eliminar um dos dois vértices e redirecionar todos os arcos chegando para outros vértices. Ver a Figura 2.6(b).

Remover Testes Redundantes

Se o vértice não-terminal *v* tiver *lo(v) = hi(v)*, então elimine *v* e redirecione todos os arcos de entrada para *lo (v)*. Ver a Figura 2.6 (c).

Em geral, a regra de transformação deve ser aplicada repetidamente, desde que cada transformação possa exibir novas possibilidades de reaplicá-las para uma mesma ordem das variáveis no grafo..



a) Terminações duplicadas b)Não-terminações Duplicadas c) Testes Redundantes

Figura 2.6 - Redução da árvore de decisão para OBDD. Aplicação das três regras de redução na árvore da Figura 2.5.

2.4.4.3 Efeito da Ordenação das Variáveis

O tamanho do OBDD, resultante do algoritmo de redução do grafo original que representa uma função booleana depende da ordenação das variáveis. Por exemplo, um OBDD que representa a função $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3) \ldots \vee (a_n \wedge b_n)$, para uma primeira ordenação de variáveis, seja $a_1 < b_2 < \ldots < a_n < b_n$ obtém-se um OBDD com 2n vértices não terminais - um para cada variável pode ser definido. Por outro lado, para um segundo caso de ordenação das variáveis para $a_1 < a_2 < \ldots < a_n < b_1 < b_2 < \ldots < b_n$, obtém-se um OBDD com $2(2^n - 1)$ de vértices não terminais. Para grandes valores de *n*, a diferença entre o crescimento linear da primeira ordenação versus o crescimento da segunda tem um efeito dramático nos requisitos de armazenamento e na eficiência dos algoritmos de manipulação.

Muitas aplicações usando OBDDs escolhem alguma ordenação de variáveis inicial e constroem todo o grafo de acordo com esta ordenação. Esta ordenação é escolhida aleatoriamente ou por uma análise heurística particular do sistema a ser representado. Note que estas heurísticas necessariamente não encontram a melhor ordenação possível.

2.5 Exemplos de CTL Aplicada a Sistemas a Eventos Discretos

Vejamos um exemplo da utilização de fórmulas CTL para expressar propriedades e especificar o comportamento de um sistema a eventos discretos. O sistema ilustrado na Figura 2.7 representa uma célula de manufatura composta por dois robôs: *R1* e *R2* e duas estações de trabalho: *WS1* e *WS2*. Em linguagem temporal a especificação desejada para a execução do sistema consiste em:

- Quando *WS1* está pronto para executar uma tarefa de montagem, esta requisitam o robô *R1* e o adquire se o mesmo estiver disponível.
- Quando *WS2* está pronto para executar uma tarefa de montagem, esta requisitam o robô *R2* e o adquire se o mesmo estiver disponível.
- Após a estação de trabalho *WS1* ou *WS2* adquirir o robô *R1* ou *R2*, esta requisita o robô *R2* ou *R1* e o adquire se o mesmo estiver disponível, respectivamente.

- Quando uma estação inicia uma tarefa de montagem, esta não pode ser interrompida até que seja concluída.
- Quando *WS1* (*WS2*) termina a tarefa, esta libera ambos os robôs.

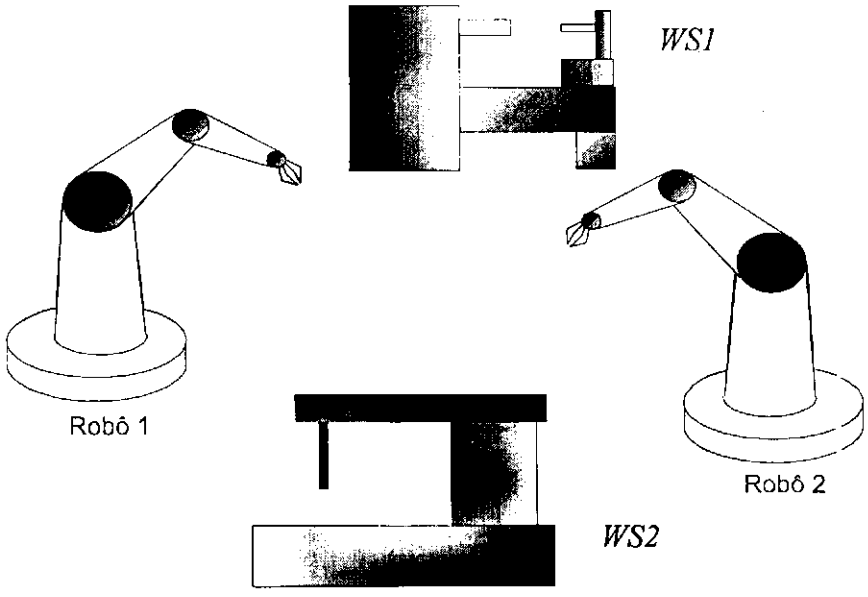


Figura 2.7 - *Layout* para um sistema de montagem com dois robôs e duas estações de trabalho

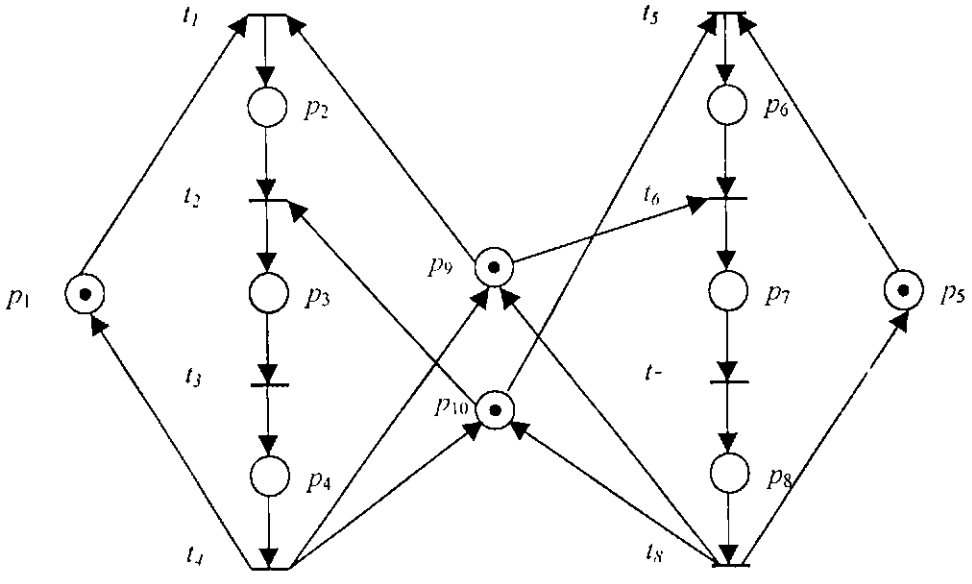


Figura 2.8 - Rede de Petri que modela o sistema apresentadc na Figura 2.7

Lugares	**Transições**
P_1: *WS1 requisita R1*	T_1: *inicia a aquisição de R1 por WS1*
P_2: *WS1 adquire R1*	T_2: *inicia a aquisição de R2 por WS1*
P_3: *WS1 adquire R2*	T_3: *inicia a montagem em WS1*
P_4: *montando em WS1*	T_4: *completa a montagem em WS1*
P_5: *WS2 requisita R1*	T_5: *inicia a aquisição de R1 por WS2*
P_6: *WS2 adquire R1*	T_6: *inicia a aquisição de R2 por WS2*
P_7: *WS1 adquire R2*	T_7: *inicia a montagem em WS2*
P_8: *montando em WS2*	T_8: *completa a montagem em WS2*
P_9: *estado de R1*	
P_{10}: *estado de R2*	

Tabela 2.1 – Referências de Lugares e Transições para a rede de Petri da Figura 2.7

A rede de Petri apresentada na Figura 2.8 modela o sistema de manufatura mostrado na Figura 2.7. A especificação do comportamento do sistema exige a existência de uma estrutura de exclusão mútua entre as estações de trabalho *WS1* e *WS2* para a aquisição dos robôs. A rede de Petri do sistema apresenta duas estruturas de exclusão mútua (exclusão mútua paralela) que são identificadas na rede por (p_9, {(T_1, T_4), (T_6, T_8)}) e (p_{10}, {(T_2, T_4), (T_5, T_8)}). Outras propriedades importantes são a ausência de *deadlock* e a reversibilidade para o sistema. Além destas propriedades teremos que representar a especificação para o sistema em fórmula CTL. Portanto, dividiremos a especificação global para o sistema em:

especificação de propriedades

especificação de execução

Em todos os exemplos mostrados neste trabalho os lugares p_n das redes de Petri serão definidos como as variáveis que compõem as fórmulas booleana, em que são aplicados os modificadores temporais e de caminho do CTL. Desta maneira as seqüências de ocorrência de transições $t_1,\ldots,t_n$ representam os caminhos para os quais as fórmula CTL

são verdadeiras ou falsas. A razão pela qual está regra é adotada está no fato de que os lugares p_n podem estar marcados ou não devido a ocorrência de transições t_m. Em todos os exemplos, devido a padronização utilizada pelo *SMV,* assumiremos a nomenclatura !, & e | representando os operadores booleanos ¬, ∧ e ∨, respectivamente.

2.5.1 Especificação de propriedades

Propriedades	**Fórmula CTL**	**Descrição**
Exclusão Mútua	AG !(p_2 & p_7)	Nunca *WS1* e *WS2* adquirirão *R1* ao mesmo tempo
Exclusão Mútua	AG !(p_3 & p_6)	Nunca *WS1* e *WS2* adquirirão *R2* ao mesmo tempo
Ausência de deadlock	AG (!*deadlock*) , onde *deadlock* = p_1 & p_9 | p_2 & p_{10} | p_3 | p_4 | p_5 & p_{10} | p_6 & p_9 | p_7 | p_8	Sempre existirá pelo menos uma transição habilitada
Reversibilidade	AG EF (P1&P5&P9&P10)	Sempre será possível a marcação inicial

2.5.2 Especificação de execução

Especificação	**Fórmula CTL**	**Descrição**
Especificação WS1	*AG* ((p_1 & p_9) → *AG* ((p_2 & p_{10}) → *AF* p_3)) & *AG*(p_3→*AF* p_1)	Esta especificação pode ser vista em duas fases. A primeira significa que para todos os estados em que os robôs estão disponíveis, *WS1* pode sempre adquirir *R1* seguido de *R2*. A segunda parte avalia se dado que em todos os estados nos quais *WS1* requer *R1* implica que *R2* deverá ser adquirido.
Especificação WS2	*AG* ((p_5 & p_{10}) → *AG* ((p_6 & p_9) → *AF* p_7)) & *AG*(p_7→*AF* p_5)	Esta especificação é similar a especificação referente a *WS1*,

Especificação Global: $AG\ !(p_2\ \&\ p_6)\ \&\ AG\ !(p_3\ \&\ p_7)\ \&\ AG\ (!deadlock)\ \&\ AG\ EF\ (P1\&P5\&P9\&P10)\ \&\ AG\ ((p_1\ \&\ p_9)\ \rightarrow\ AG\ ((p_2\ \&\ p_{10})\ \rightarrow AF\ p_3))\ \&\ AG(p_3 \rightarrow AF\ p_1)\ \&\ AG\ ((p_5\ \&\ p_{10})\ \rightarrow\ AG\ ((p_6\ \&\ p_9)\ \rightarrow AF\ p_7))\ \&\ AG(p_7 \rightarrow AF\ p_5)$.

Em uma primeira análise podemos observar que o sistema não está livre de *deadlock*, pois a ocorrência da seqüência t_1, t_5 leva ao bloqueio total. Uma outra propriedade que pode ser verificada pela ocorrência da mesma seqüência é a ausência de exclusão mútua paralela devido ao robô *R1* e *R2* poderem ser requisitados pelas máquinas *WS1* ou *WS2* respectivamente. Uma solução para este problema será apresentada posteriormente no Capítulo 3 deste trabalho.

Capítulo 3

Ferramentas para Modelagem, Análise e Verificação de Modelos

A teoria de redes de Petri tem sido exaustivamente aplicada para modelar, analisar e simular aplicações de sistema de manufatura [ZD93, DHPSV93], [PX96, MA98]. A análise e a verificação são de fundamental importância na avaliação do comportamento e na especificação de sistemas de manufatura. Neste Capítulo apresentamos o PEP (*Programming Environment based on Petri Nets*), uma importante ferramenta para modelar, simular e analisar redes de Petri. Em seguida apresentamos a ferramenta SMV (*Symbolic Model Verifier*) que é uma ferramenta importante para a verificação de propriedades para sistemas concorrentes, e finalmente apresentamos um exemplo de um sistema de manufatura, na qual as ferramentas PEP e SMV são empregadas para modelagem, análise e verificação.

3.1 A ferramenta PEP

A ferramenta PEP (*Programming Environment based on Petri Nets*) é uma ferramenta abrangente que suporta as principais fases de desenvolvimento de sistemas paralelos como mostrado na Figura 3.1. As ferramentas que integram a PEP são:

- Editor e simulador,

- Compilador para redes de Petri (PN), e
- Analisadores bem como verificadores de modelo para PN.

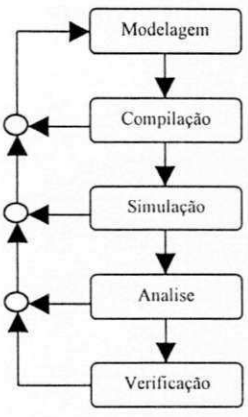


Figura 3.1 – Fases de Desenvolvimento

Com a ferramenta PEP os sistemas paralelos podem ser modelados de cinco maneiras distintas, como apresentado a seguir e esquematizado na Figura 3.2:

- Autômato paralelo finito (PFA).
- Algoritmos paralelos expressos em $B(PN)^2$ [BH], que é uma linguagem de programação imperativa/predicativa. $B(PN)^2$ é chamada *Basic Petri Net Programming Notation* porque esta possui uma semântica composicional em termos de redes de Petri de alto nível (HL) ou baixo nível (LL) [BDH92].
- Expressões de álgebra de processos chamada de PBC (*Petri Box Calculus*) [BDH92], podendo também ser derivada automaticamente de um programa $B(PN)^2$ ou ser projetado independentemente.
- M-nets [BFF+95a], permite projetar e editar redes de alto nível (HL), que é uma semântica alternativa para programas baseados em $B(PN)^2$ [BFF+95b].
- Redes de baixo Nível (***Low Level***), que permite projetar e editar redes de Petri rotuladas de baixo nível.

Além disto, PEP disponibiliza as seguintes funções:

- Fórmulas em Lógica Temporal que podem ser definidas pelo usuário para permitir verificação de modelo de um sistema proprietário.
- Prefixo finito de um processo de ramificação (*prefix*)[ERV], que é necessário para evitar o problema de explosão de estados.

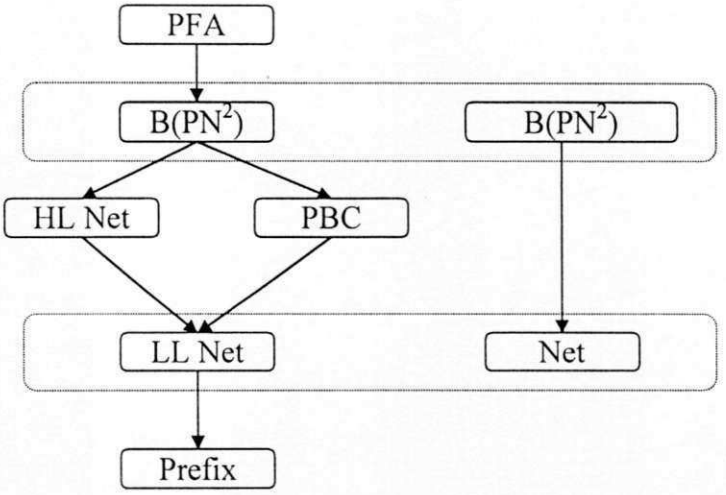


Figura 3.2 – Objetos utilizados pelo sistema PEP

Tipicamente quando um programa em $B(PN)^2$ é escrito, a rede de Petri (LL) correspondente é criada automaticamente e as propriedades de interesse são expressas por fórmulas. Estas propriedades podem então ser verificadas através de um verificador de modelo.

A estrutura de um sistema PEP é mostrada na Figura 3.3. A parte mais importante para esta dissertação é o componente de verificação. Cada objeto PEP corresponde a um arquivo em disco. O verificador de modelo que é utlizado nesta dissertação para obter a síntese de supervisores recebe como entrada a representação de redes de Petri (LL) e as fórmulas temporais na forma de arquivos.

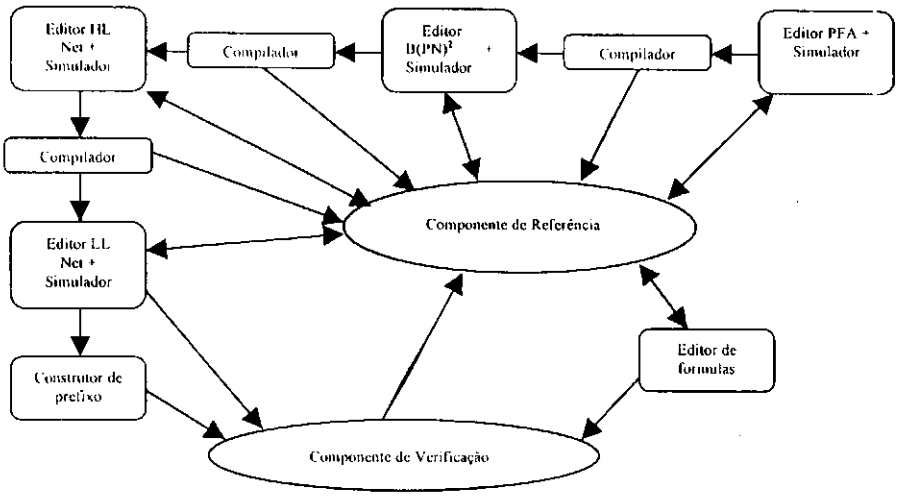


Figura 3.3 – Estrutura do sistema PEP

3.2 O Sistema SMV

O Sistema SMV (*Symbolic Model Verifier*) é uma ferramenta de verificação de modelo simbólica baseada em BDD que pode ser executada nos sistemas operacionais UNIX e DOS. Para verificar um sistema de estados finito com o SMV, primeiramente este tem que ser descrito pela linguagem de entrada do SMV, em conjunto com a especificação das propriedades desejadas para o comportamento do modelo, na forma de fórmulas CTL como introduzido no Capítulo 2. A lógica temporal CTL admite uma rica classe de propriedades temporais, incluindo segurança, vivacidade, justiça e ausência de *deadlock*, especificada em uma sintaxe concisa. O SMV traduz a descrição em um modelo de Kripke e usa o algoritmo de verificação simbólica baseado em OBDD para verificar se a especificação do modelo é verdadeira ou falsa. A Figura 3.5 ilustra a estrutura do sistema SMV.

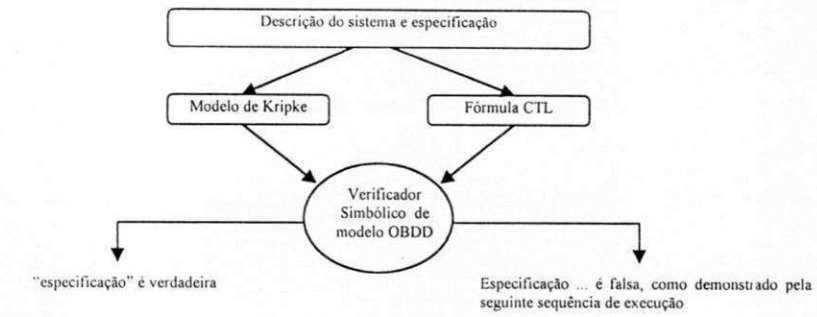


Figura 3.4 – Estrutura de funcionamento do SMV

3.2.1 A linguagem de entrada do SMV

O propósito principal da linguagem de entrada do SMV é descrever as relações de transições de uma estrutura de Kripke finita que representa o sistema. A descrição do sistema é uma coleção de módulos reunidos em um módulo principal que especifica o comportamento global do sistema. Um módulo consiste de um cabeçalho `MODULE module_name(parameters)` (a lista de parâmetros é opcional) e as seguintes declarações:

- `VAR`: contém as declarações de variáveis de um módulo, escrito como

  ```
  Var: type;
  ```

  Os tipos associados a uma variável podem ser booleanos ou uma enumeração (e.g. {1,2,3} ou {vermelho, verde, azul}). Também é possível criar uma instância definida em um outro modulo. As atribuições nestes submódulos são executadas ao mesmo tempo nos módulos parentes. Módulos são instanciados usando

  ```
  Module_inst: module_name(parameters);
  ```

  na declaração `VAR` . Variáveis dentro destes módulos podem ser referenciadas usando '.'(e.g. `module_inst.var`). Módulos também podem ser instanciados

como *processos* (com a palavra chave "`process`"). Neste caso apenas uma atribuição destes processos é executada a cada passo.

- `ASSIGN`: contém atribuições para as variáveis. As atribuições são de
  `Init(var):=expr; var:=expr; ou next(var):=expr`
  Onde `Init(var)` representa o valor inicial da variável, `var` é o valor corrente e `next(var)` é o valor no próximo estado. Usando `case`, a expressão do lado direito pode ser feita dependente de certas condições, e também pode estimar o valor de um conjunto, e neste caso à variável é atribuída não deterministicamente um dos valores deste conjunto. Se uma variável não é atribuída em um módulo, o SMV fica livre para escolher algum valor para a mesma a cada passo, isto pode ser usado como uma entrada aleatória do sistema.
- `DEFINE`: declara um símbolo que pode ser utilizado com expressões comumente usadas, e.g. `cor := Vermelho|Verde|Azul`.
- `SPEC`: fornece a especificação do sistema como fórmula CTL. A sintaxe usada para as operações booleanas são !, &, |, –>, <–> significando $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$ respectivamente.
- `FAIR`: especifica uma restrição de equidade, i.e. uma fórmula CTL que deve ser assumida como verdadeira com frequência.
- `TRANS` e `INIT`: estas declarações são uma alternativa ao `ASSIGN` e torna possível especificar relações de transição e estados iniciais diretamente como expressões boleanas.

A sintaxe completa da linguagem SMV é descrita na documentação SMV [McM92]. O SMV também possui um manual o qual descreve as varias opções que podem ser aplicadas ao algoritmo de verificação de modelo.

3.2.2 Modelos de Rede de Petri em SMV

Existem pelo menos três maneiras diferentes para codificar redes de Petri usando módulos e declarações em SMV. Estas alternativas são discutidas com detalhes em [Win97]. Embora as alternativas que utilizam módulo *process* e declarações `ASSIGN` sejam bastante razoáveis, estas podem ser ineficientes e até incorretas em algumas situações [Wim97]. No entanto, as codificações de rede de Petri cujas relações de transição são especificadas diretamente usando `TRANS` e `INIT` são fáceis de entender e apresentam maior eficiência[Wim97].

A especificação SMV que é gerada para rede de Petri consiste apenas de um módulo principal. Na parte `VAR`, uma variável booleana P_i é declarada para cada lugar p_i. A parte `INIT` especifica o estado inicial do sistema. Este contêm uma fórmula booleana das variáveis de lugar que apenas são verdadeiras se o valor de cada variável de lugar é a marcação inicial:

$$INIT = \bigwedge_{p_i \in P} P_i = m_0(p_i) \tag{3.1}$$

A parte `TRANS` especifica a relação de transição como uma fórmula booleana. Esta consiste de uma subfórmula `TRANS`, para cada transição t:

$$TRANS_t = t.enabled \wedge \bigwedge_{p_i \in t^\bullet \setminus {}^\bullet t} next(P_i) = 1 \wedge \bigwedge_{p_i \in {}^\bullet t \setminus t^\bullet} next(P_i) = 0 \wedge \bigwedge_{p_i \in P \setminus ({}^\bullet t \cup t^\bullet)} next(P_i) = P_i \tag{3.2}$$

t.enabled é definido como

$$t.enabled = \bigwedge_{p_i \in {}^\bullet t} P_i \tag{3.3}$$

Esta fórmula é verdadeira se a transição está habilitada e o próximo valor da variável do lugar corresponde a marcação da rede de Petri depois da transição ter ocorrido. A relação de transição é então a disjunção de todas estas subfórmulas. Desta forma, se a fórmula evolui para verdadeiro, uma subfórmula evoluiu para verdadeiro, portanto os próximos valores das variáveis de um estado sucessor válido.

Contudo, a semântica do CTL contém a hipótese que todos os caminhos no sistema são infinitos (i.e, cada estado deve possuir um sucessor imediato). Neste caso, a relação de transição é chamada "completa". Portanto, o SMV elimina todos os estados sem sucessor do modelo (estados de *deadlock* em que nenhuma transição é habilitada). Para possibilitar verificar se redes de Petri contêm *deadlocks*, uma subfórmula tem de ser adicionada para a relação de transição que permite ao sistema permanecer em seus estados correntes se existe um *deadlock*. Deste modo,

$$deadlock = \neg \bigvee_{t \in T} t.enabled \tag{3.4}$$

e finalmente

$$TRANS = \bigvee_{t \in T} TRANS_t \vee (deadlock \wedge \bigwedge_{p_i \in P} next(P_i) = P_i) \tag{3.5}$$

Como uma fórmula TRANS contêm uma subfórmula para cada transição e estas contêm todas as variáveis de lugar , o tamanho do arquivo de entrada do SMV é propocional a |P|•|T|. Portanto a idéia é definir `u_Pi_Pj` de abreviações para o `next(Pi)=Pi&...&next(Pj)=Pj` e os utilizar sempre que possível. A expressão para a qual são definidas abreviações é obtida por divisão recursividade do conjunto de lugares ao meio. Isto fornece abreviações que podem ser usadas muito freqüentemente, e não são precisadas de muitos definições adicionais.

<table>
<tr><td colspan="4">u_P1_P10</td></tr>
<tr><td colspan="3">u_P1_P5</td><td>u_P6_P10</td></tr>
<tr><td colspan="2">u_P1_P3</td><td>u_P4_P5</td><td></td></tr>
<tr><td>u_P1_P2</td><td>next(P3)=P3</td><td></td><td></td></tr>
<tr><td>next(P1)=P1</td><td>next(P2)=P2</td><td></td><td></td></tr>
</table>

A seguir apresentamos um programa SMV que representa a rede de Petri para o sistema de manufatura apresentado na Seção 2.5

```
-- generated by PEP2SMV from exclusao.ll_net (model 5)
-- (C) 1996-1997 Guido Wimmel
```

```
MODULE main
VAR
P1: boolean;
P2: boolean;
P3: boolean;
P4: boolean;
P5: boolean;
P6: boolean;
P7: boolean;
P8: boolean;
P9: boolean;
P10: boolean;
INIT
(P1=1&P2=0&P3=0&P4=0&P5=1&P6=0&P7=0&P8=0&P9=1&P10=1)
TRANS
(
--T1
 T1.enabled&next(P2)=1&next(P1)=0&next(P9)=0&next(P10)=1&next(P3)=P3&u_P4_P5&u_P6_P8
|
--T2
 T2.enabled&next(P3)=1&next(P10)&next(P2)=0&next(P1)=P1&u_P4_P5&u_P6_P10
|
--T3

T3.enabled&next(P4)=1&next(P9)=1&next(P3)=0&u_P1_P2&next(P5)=P5&u_P6_P8&next(P10)=P1
0
|
--T4

T4.enabled&next(P1)=1&next(P10)=1&next(P4)=0&next(P2)=P2&next(P3)=P3&next(P5)=P5&u_P
6_P8&next(P9)=P9
|
--T5

T5.enabled&next(P6)=1&next(P5)=0&next(P10)=0&u_P1_P3&next(P4)=P4&next(P7)=P7&next(P8
)=P8&next(P9)=P9
|
--T6
 T6.enabled&next(P7)=1&next(P6)=0&next(P9)=0&u_P1_P5&next(P8)=P8&next(P10)=P10
|
--T7
 T7.enabled&next(P8)=1&next(P7)=0&u_P1_P5&next(P6)=P6&u_P9_P10
|
--T8
T8.enabled&next(P5)=1&next(P9)=1&next(P10)=1&next(P8)=0&u_P1_P3&next(P4)=P4&u_P6_P7
|
-- selfloop for deadlocks
deadlock&u_P1_P10
)
DEFINE
T1.enabled:=P1&P9&P10;
T2.enabled:=P2;
T3.enabled:=P3;
T4.enabled:=P4;
T5.enabled:=P5&P10;
T6.enabled:=P6&P9;
T7.enabled:=P7;
T8.enabled:=P8;
deadlock:=!(T1.enabled|T2.enabled|T3.enabled|T4.enabled|T5.enabled|T6.enabled|T7.ena
bled|T8.enabled);
u_P1_P2:=next(P1)=P1&next(P2)=P2;
u_P1_P3:=u_P1_P2&next(P3)=P3;
u_P4_P5:=next(P4)=P4&next(P5)=P5;
```

```
u_P1_P5:=u_P1_P3&u_P4_P5;
u_P6_P7:=next(P6)=P6&next(P7)=P7;
u_P6_P8:=u_P6_P7&next(P8)=P8;
u_P9_P10:=next(P9)=P9&next(P10)=P10;
u_P6_P10:=u_P6_P8&u_P9_P10;
u_P1_P10:=u_P1_P5&u_P6_P10;
SPEC
AG !(p2 & p6)
SPEC
AG !(p3 & p7)
SPEC
AG (!deadlock)
SPEC
AG EF (P1&P5&P9&P10)
SPEC
AG ((p1 & p9)  →  AG ((p2  & p10) → AF p3)) & AG(p3→AF p1)
SPEC
AG ((p5 & p10)  →  AG ((p6  & p9) → AF p7)) & AG(p7→AF p5).
```

3.2.3 Otimizações do Verificador de Modelos

O SMV possui algumas opções que podem ser aplicadas ao algoritmo de verificação de modelo. Estas opções e seus efeitos são apresentados a seguir.

`-f`: indica que uma pesquisa prévia deverá ser realizada antes da verificação da especificação. O algoritmo de verificação de modelo considera apenas os estados alcançáveis. Esta técnica é muito utilizada para espaços de estado em que apenas um subconjunto deste é alcançável. Desta forma, esta opção sempre é utilizada para verificar modelos de redes geradas pelo PEP.

`-inc`: causa uma avaliação incremental da relação de transição. Isto significa que a cada passo na pesquisa prévia, a relação de transição está sendo restrita ao conjunto de estados alcançáveis, tornando a relação de transição menor, no entanto, envolve maior esforço computacional devido a relação de transição ter que ser reavaliada a cada passo.

Todavia, a sobrecarga adicional causada pela opção `-inc` melhora o desempenho para redes de Petri muito grandes.

`-reorder`: como mencionado no Capítulo 2, o tamanho da representação BDD de uma relação de transição depende crucialmente da ordenação de variáveis. BDD menores

podem influenciar para uma considerável melhoria de desempenho de uma verificação de modelo.

O sistema SMV quando com a opção **-reorder** causa "reordenação dinâmica das variáveis". Isto significa que de tempos em tempos o SMV tenta modificar a ordem das variáveis para reduzir o número de nós do BDD. Além disso, a ordem das variáveis que foi encontrada pelo SMV pode ser escrita para um arquivo (opção **-o filename**) para depois ser lida (opção **-i filename**) sem a sobrecarga causada pela reordenação dinâmica.

-c: determina o tamanho do cache para as operações do BDD que são executadas no SMV. Infelizmente, nem sempre o aumento do cache significa melhor desempenho de verificação. Os tamanhos dos caches sugeridos são 16381 (default), 65521, 262063, 522713 e 1046429 entradas (16 bytes por entrada).

-v 1: faz com que o SMV apresente informações de saída para os cálculos executados enquanto verificando uma especificação de modelo.

-r: quando esta opção é utilizada o SMV mostra o número de estados alcançáveis no modelo e desta forma informa o tamanho do sistema que está sendo modelado.

3.2.4 O Tradutor pep2smv

pep2smv [Win97] é o programa de tradução desenvolvido para gerar os arquivos de entrada para o SMV a partir dos arquivos de rede de Petri da ferramenta PEP. O pep2smv é executado através do comando:

```
pep2smv inputfile {opções}
```

em que o arquivo de entrada é o nome de descrição do arquivo da rede de Petri de baixo nível (LL-Petri net) que possui a extensão .ll.net. Se pep2smv é executado com alguma opção, este apresenta informações sobre o seu uso:

-m=<model>: especifica qual modelo (1-7) deverá ser usado quando gerando a representação SMV da rede de Petri. O modelo default é 5.

-o=<outputfile>: especifica o nome do arquivo SMV para ser criado.

`-f=<specification file name>`: especifica o nome do arquivo que deverá conter uma fórmula CTL por linha.

`-s=<specification>`: a especificação fornecida nesta opção é adicionada ao arquivo de entrada do SMV.

`-c`: esta opção faz com que o pep2smv chame o SMV imediatamente após a entrada ter sido gerada.

3.2.5 O programa transsmv

transsmv [Wim97]traduz a saída de verificação do SMV para um modelo de rede de Petri em SMV, em uma seqüência de disparo de transições se esta contém um contra-exemplo para a fórmula avaliada como falsa. transsmv deverá normalmente ser utilizado como parte de um *pipe*, por exemplo. `smv -f exclusao.smv | transsmv -n=exclusao.ll.net`.

3.2.6 Exemplo da Aplicação do SMV para Verificação de Modelo

No Capítulo 2 apresentamos um exemplo de um sistema de manufatura, em que discutimos como a especificação desejada deveria ser definida em termos de fórmula CTL. O programa descrito na Seção 3.2.2 mostra o programa SMV para a rede de Petri que modela o sistema do nosso exemplo. Nesta seção mostraremos e discutiremos os resultados da verificação de modelo aplicada ao referido programa, para algumas propriedades de comportamento da rede e para a especificação desejada para o sistema.

Em sistemas de manufatura representados por redes de Petri, a semântica de modelagem é muito importante para que a rede resultante possa representar a especificação desejada para o sistema. Portanto, propriedades como alcançabilidade, limitação, vivacidade e reversibilidade devem ser consideradas no processo de modelagem [ZD93]. Em sistemas de manufatura de tamanho moderado a complexidade de projeto, no nível de detalhe, pode vir a ser irracional, desta maneira, é de fundamental importância uma metodologia eficiente de modelagem e verificação das propriedades.

Uma das maiores dificuldades no processo de modelagem é o compartilhamento de recursos. Em sistemas de manufatura estes recursos podem ser robôs, sistema para transporte de material, ou centros de processamento (máquina). Segundo [ZD93] a existência de compartilhamento de recursos é a principal razão para que um sistema possa conter *deadlock*.

O sistema de manufatura modelado pela rede de Petri da Figura 3.5 é um exemplo clássico do compartilhamento de recursos, em que duas máquinas disputam concorrentemente dois robôs. Podemos observar a existência de uma estrutura de exclusão mútua paralela [ZD93]. Para verificar as propriedades comportamentais e a especificação de execução desejada para o modelo utilizamos o método de verificação simbólica discutido no Capítulo 2.

O programa SMV que representa a rede de Petri para o sistema foi obtido a partir da aplicação do pep2smv ao arquivo de saída que representa a rede de Petri, modelada no editor para redes de baixo nível (LL) da ferramenta PEP, conforme a seguinte linha de comando: `pep2smv exclusao.ll_net -c -- -f`

As propriedades de comportamento estudadas foram:

Propriedades	**Fórmula CTL**	**Resposta do SMV**	**Trace**
Exclusão Mútua	AG !(p_2 & p_7)	Falso	T_1, T_5
Exclusão Mútua	AG !(p_3 & p_6)	Falso	T_1, T_5
Ausência de deadlock	AG (!*deadlock*) , onde *deadlock* = p_1 & p_9 | p_2 & p_{10} | p_3 | p_4 | p_5 & p_{10} | p_6 & p_9 | p_7 | p_8	Falso	T_1, T_5
Reversibilidade	AG EF (P1&P5&P9&P10)	Falso	T_1, T_5
Alcançabilidade	EF P4	Verdadeiro	
Justiça	AF P4	Falso	*Loop:* T_5,T_6,T_7,T_8
Vivacidade	AG AF (P2 | P6)	Falso	T_1, T_5

Tabela 3.1 – Conjunto de Propriedades para a rede da Figura 3.7

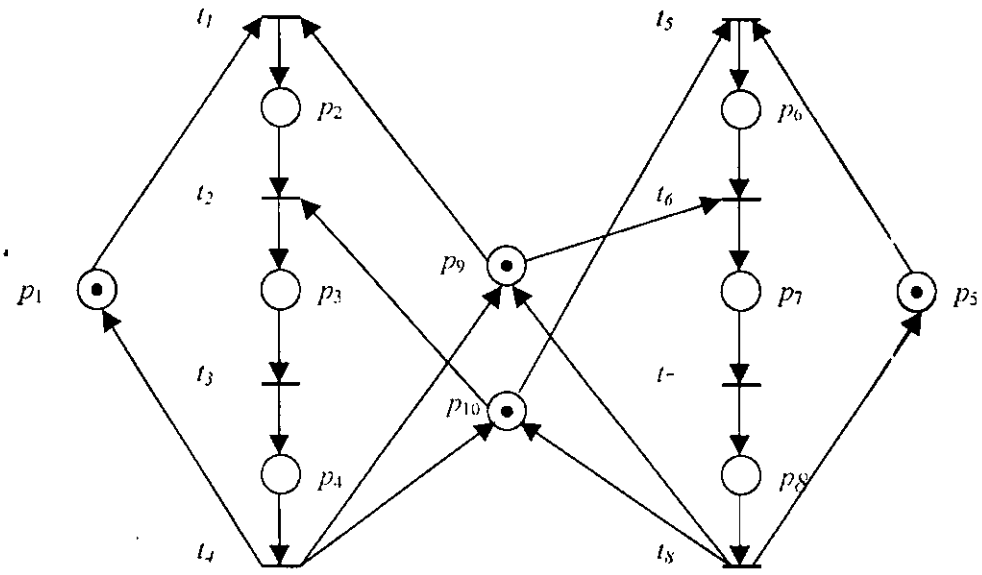


Figura 3.5 –Rede de Petri apresentada no Capítulo 2

A sequência de disparo composta pelas transições T_1 ,T_5 conduz o sistema a um estado de *deadlock*. Podemos observar na Tabela 3.1 que todas as propriedades que foram negadas pelo SMV são decorrentes da mesma seqüência de ocorrência. A seqüência T_1, T_5 indica que o robô *R1* é requisitado por *WS1* e o robô *R2* é requisitado por *WS2*. Esta seqüência de ocorrência não representa a aquisição conjunta de *R1* e *R2* por *WS1* ou *WS2* para a montagem de uma peça, significando a ausência de exclusão mútua paralela. Pelo mesmo motivo observamos que esta seqüência impossibilita qualquer outro disparo de transição levando o sistema ao *deadlock* e por conseguinte a não reversibilidade.

Especificação WS1

A especificação, $AG\ ((p_1\ \&\ p_9) \rightarrow AG\ ((p_2\ \&\ p_{10}) \rightarrow AF\ p_3))\ \&\ AG(p_3 \rightarrow AF\ p_1)$, pode ser dividida em duas partes. A primeira significa que para todos os estados em que os robôs estão disponíveis, *WS1* pode adquirir sempre *R1* seguido de *R2*. A segunda parte avalia se dado que em todos os estados nos quais *WS1* requer *R1* implica que *R2* deverá ser adquirido. Na primeira avaliação podemos afirmar que a especificação é falsa devido a

WS2 poder adquirir *R2* provocando um *deadlock* no sistema. Da mesma forma, a segunda avaliação também não é possível devido a *WS2* poder sempre adquirir *R2* e *R1*.

Especificação WS2

A especificação para WS2, $AG\ ((p_5\ \&\ p_{10}) \rightarrow AG\ ((p_6\ \&\ p_9) \rightarrow AF\ p_7))\ \&\ AG(p_7 \rightarrow AF\ p_5)$, é similar a especificação referente a *WS1*, em que os resultados obtidos na verificação também são similares.

Como sugestão para resolver o problema de *deadlock* causado pelas marcações simultâneas de p_2 e p_6, detectadas na verificação das especificações das propriedades de comportamento, *WS1* e *WS2*, a rede de Petri da Figura 3.6 é apresentada. No entanto, esta rede não resolve o problema da justiça de aquisições de *R1* e *R2* pelas estações de trabalho *WS1* e *WS2*, podendo qualquer uma das estações monopolizar totalmente os robôs. A aplicação da especificação à rede da Figura 3.6 através do SMV ratifica esta expectativa.

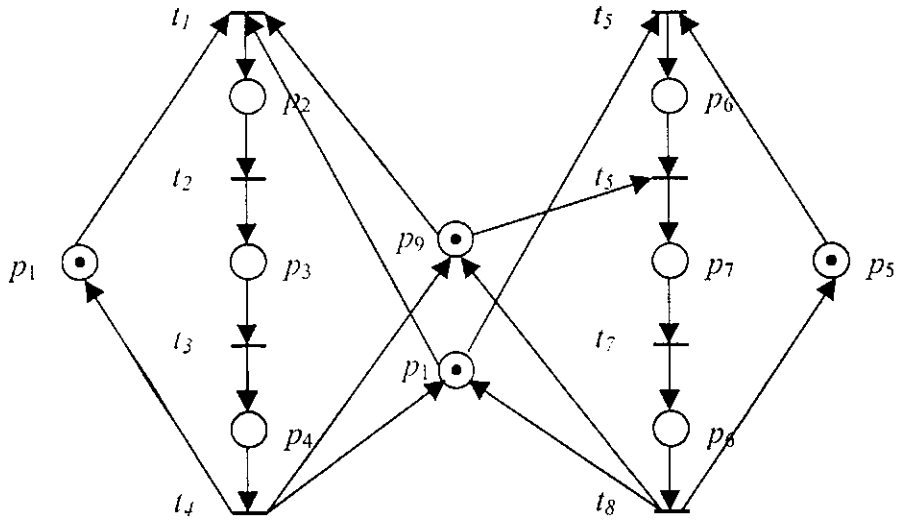


Figura 3.6 – Rede de Petri com exclusão mútua paralela

Capítulo 4

Síntese de Supervisor Utilizando Verificação Simbólica de Modelo

Neste Capítulo apresentamos o algoritmo de síntese de supervisor baseado em verificação simbólica de modelo, utilizando o SMV como ferramenta de verificação. Em seguida dois exemplos simples de SEDs são apresentados em conjunto com as especificações desejadas em lógica temporal CTL, e o algoritmo proposto neste capítulo é aplicado. O resultado obtido é a especificação dos supervisores.

4.1 Introdução

A teoria de geradores mostra que é possível representar um sistema a eventos discretos (SED) por intermédio de um gerador G com linguagem gerada $L(G) = L$ e linguagem marcada $L_m(G) = L_m$, na qual estas linguagens pertencem à classe de linguagens regulares. Este é um resultado importante para a teoria de controle supervisório (TCS), desenvolvida por Ramadge e Wonhan, que não se limita apenas a modelos representados por geradores finitos, embora os resultados computacionais mais precisos e definidos se limitem a estes geradores.

Conforme descrito na Seção 2.2 (Capítulo 2), dado um SED, representado por um gerador G, a teoria de controle supervisorio (TCS) estabelece as condições necessárias para a existência de um supervisor S, baseado no comportamento desejável, representado pela linguagem K.

Um SED representado por um gerador G é simplesmente um gerador espontâneo de eventos, sem controle externo, portanto, G é a representação do sistema em malha aberta. Na maioria das vezes é desejado que um SED realize seqüências de eventos específicos, para os quais alguns eventos devem ser desabilitados. Para que a especificação desejada do SED seja alcançada é necessário estabelecer ações de controle sobre o gerador G (sistema em malha fechada). O princípio de controle estabelecido pela TCS consiste em chavear entradas de controle em resposta a seqüências de eventos gerada pelo sistema, em que a linguagem gerada sob a ação de controle, represente um conjunto de tarefas especificadas pelo comportamento desejável. O sistema em malha fechada é a composição do supervisor S com o gerador controlado G_c, denotado por S/G, desta maneira, a TCS separa explicitamente o sistema a ser controlado (malha aberta) do controlador.

Originalmente, a TCS foi desenvolvida utilizando autômatos e linguagens formais, em que várias abordagens para a síntese de supervisor foram desenvolvidas com base nestes formalismos. No entanto, estas abordagens restringem-se a exemplos simples de controle. Principalmente pelo fato de que a modelagem de sistemas reais por autômatos é dificultada devido ao número de estados crescer exponencialmente com o aumento da estrutura do sistema. Uma das maneira utilizada para resolver este problema é a modelagem do sistema em módulos, entretanto, esta prática pode não preservar todas as propriedades necessárias para a existência do supervisor [Bar96].

Por outro lado, as redes de Petri, pela sua facilidade de modelar SEDs, são uma alternativa que tem sido bastante empregada para síntese de supervisores. Entre as várias vantagens apresentadas palas redes de Petri introduzidas no Capítulo 1, destaca-se a possibilidade da modelagem de sistemas reais, pois as redes de Petri que modelam estes sistemas são limitadas. A classe de linguagens geradas por estes modelos são equivalentes

à classe de linguagens regulares, portanto todos os resultados obtidos pela TCS com relação a existência de supervisores, também se aplicam às linguagens geradas por redes de Petri limitadas. Por estas razões, é possível determinar as condições necessárias e suficientes para a existência de supervisores modelados por redes de Petri limitadas. Um diagrama simplificado da utilização das redes de Petri em conjunto com a TCS, para a análise e controle de um SED proposto por [Bar96] pode ser observado na Figura 4.1.

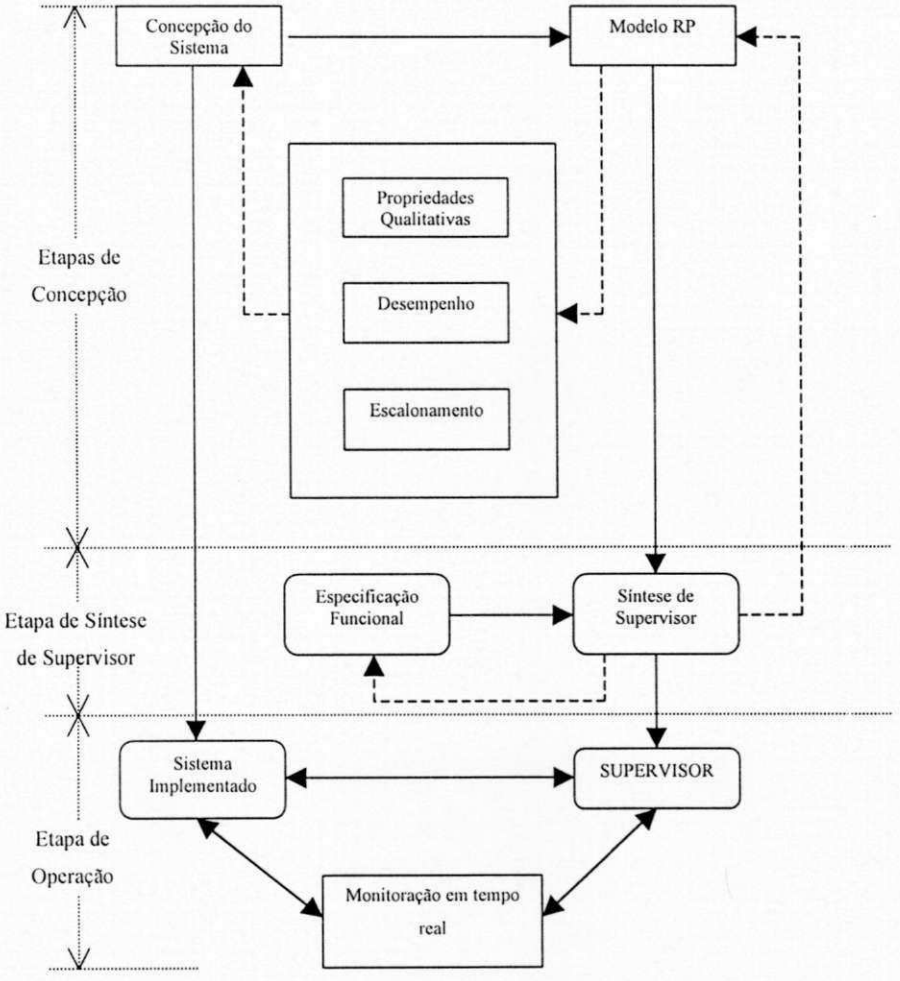


Figura 4.1: Utilização de RP e TCS para análise e controle de SEDs.

Baseado na exposição acima, Barroso [Bar96] propõe uma metodologia de síntese de supervisor composta do algoritmo modificado da árvore de alcançabilidade (AMArA) mais o algoritmo para construção do gerador da suprema linguagem controlável Sup$C(L)$, definida no Capítulo 2, (ACGS). A síntese é realizada tendo como base o modelo do sistema em rede de Petri e a especificação desejada, ilustrado pelo diagrama de blocos da Figura 4.2. Apesar desta abordagem resolver o problema da síntese do supervisor, não resolve o problema da explosão de estados uma vez que se utiliza uma enumeração explícita do espaço de estados, representado pela árvore de alcançabilidade. O problema de explosão de estados torna a análise de redes de Petri, computacionalmente cara. Desta forma é necessário empregar um método mais eficiente que torne a análise viável também para sistemas complexos, notadamente no contexto de sistemas de manufatura.

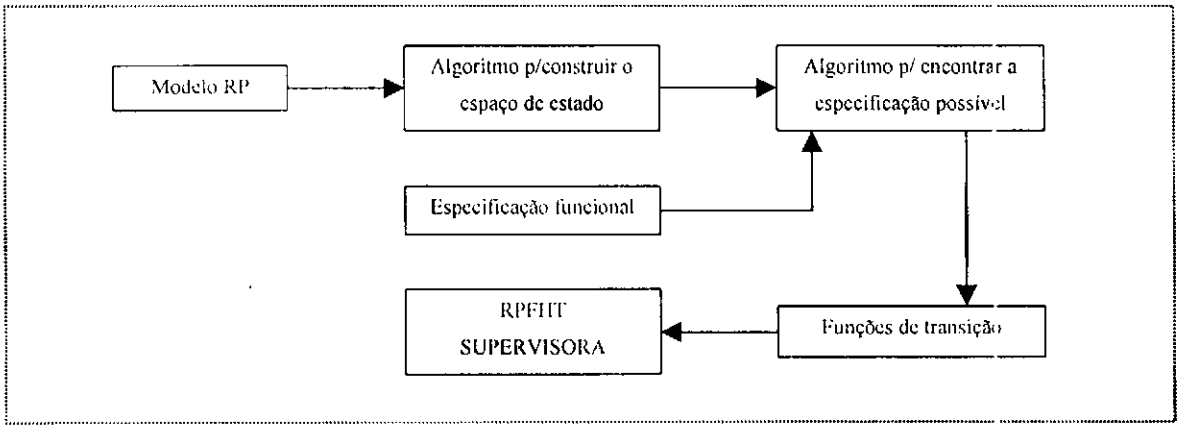


Figura 4.2: Diagrama em blocos do procedimento de síntese do supervisor

A verificação de modelo simbólica, que é uma representação implícita do espaço de estado, através de OBDD [Bry86], tornando possível verificar propriedades de sistemas com 10^{20} estados ou mais, conforme descrito no Capítulo 2. A aplicação do método é restrito a redes de Petri seguras. Deste fato decorre que para analisar redes com lugares cuja capacidade é superior a um, é necessário recorrer a técnicas de modelagem para buffers como as utilizadas por Zhou e DiCesare [ZD93], descritas no Capítulo 2. Uma decorrência direta da utilização desta técnica é o aumento do número de lugares e

transições para a rede de Petri, proporcional à capacidade do lugar da rede original, acarretando um aumento da representação do espaço de estados. Entretanto, este aumento pode ser compensado pelo uso de OBDD que possibilita reduzir em valores consideráveis a representação do espaço de estados.

Nas seções seguintes apresentamos um algoritmo de síntese para supervisor utilizando verificação de modelo simbólica, reduzindo o problema da explosão de estados. Por fim, para ilustrar a aplicação do algoritmo, apresentamos um exemplo de um sistema de manufatura simples.

4.1.1 Síntese de Supervisor

Como mencionado na seção anterior os SEDs reais podem ser modelados por redes de Petri limitadas. As linguagens geradas por estas redes são equivalentes as linguagens regulares, satisfazendo todas as condições para a existência de um supervisor. A TCS estabelece como condições necessárias para a existência de um supervisor: o fechamento da linguagem do sistema supervisionado *L(S/G)* com relação a linguagem $\overline{L(S/G)}$, significando que para cada seqüência de eventos gerada pelo sistema supervisionado, todos os seus prefixos também pertencem a linguagem, $L(S/G) = \overline{L(S/G)}$; e a *controlabilidade* do sistema, significando que dada uma seqüência de eventos s desejada para o comportamento do sistema e um evento não controlável σ, a existência de $s\sigma$ é uma seqüência de eventos fisicamente possível. Satisfeitas as condições de fechamento e controlabilidade, existe um supervisor *próprio* (completo, não bloqueável e não rejeitável), representado por um gerador S e o mapa de controle $\Theta(x)$ que pode ser encontrado a partir das equações:

$$L(S/G) = K, \text{ e}$$

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

cuja a linguagem K representa a especificação desejada para o sistema, Σ_x^0 (ou $\overline{Y}$) é o conjunto de eventos σ que, por serem fisicamente possíveis após a ocorrência de $s \in K$ e se

$s\sigma \notin K$, precisam ser desabilitados quando S se encontrar no estado x, de modo que não pertencem a $\Theta(x)$. $\Sigma_x^!$ (ou Y) é o conjunto de eventos σ que são continuações da palavra s tais que $s\sigma \in K$ e precisam estar habilitados quando S se encontrar no estado x, ou seja $\sigma \in \Theta(x)$.

O algoritmo de síntese da especificação do supervisor proposto nas seções seguintes identifica para cada estado x pertencente a especificação desejada, quais os eventos que devem ser controlados, ou seja, o conjunto Σ_x^0 será encontrado. Vale salientar que os eventos pertencentes a este conjunto são aqueles contidos no alfabeto Σ_c (eventos controláveis). Os eventos não controláveis Σ_u sempre estarão contidos no conjunto $\Sigma_x^!$, pois estes eventos não sofrem ação de controle externo, portanto devem estar sempre habilitados. Logo, a especificação desejada para o sistema tem que prever em sua estrutura a ocorrência destes eventos.

4.1.2 Princípio da Síntese

Como já mencionado na seção introdutória deste capítulo e como introduzido no Capítulo 2, o supervisor S atua sobre o gerador G_c com chaveamento das entradas de controle Γ, em resposta a eventos previamente gerados por G com o propósito de permitir que apenas os eventos desejáveis ocorram. Esta ação de controle, por outro lado, inibe a ocorrência de eventos não desejáveis ao sistema, ou seja, é como se estes eventos deixassem de existir para o estado em um determinado instante.

Para um determinado estado pertencente a especificação desejável, dois conjuntos de eventos disjuntos existem: eventos desejáveis e eventos não desejáveis. O algoritmo proposto na seção seguinte obtém para cada estado da especificação quais os eventos que devem ser inibidos. O princípio pelo qual o algoritmo está estabelecido é a técnica de programação por diferença [Kum92][Tsa93], que consiste em obter um conjunto Y a partir da retirada de $\overline{Y}$ do conjunto Σ. A Figura 4.3 mostra o diagrama de Venn para os conjuntos Y, $\overline{Y}$ e Σ referente a cada estado x da especificação desejável. A região limitada por Y

define o espaço de operação determinado pela especificação, enquanto que a região limitada por $\overline{Y}$ define o espaço que contém os eventos não desejáveis ao comportamento do sistema. $\Sigma = Y \cup \overline{Y}$ define o espaço de estado com todos os eventos possíveis para o sistema.

A utilização desta técnica é conseqüência dos resultados oferecidos pelo SMV (ver Capítulo 3), o qual fornece para uma dada especificação em lógica temporal (CTL) descrita no Capítulo 2, um caminho, caso este exista, em que a especificação não é verdadeira. Desta maneira, o algoritmo obtém o conjunto $\overline{Y}$ e por eliminação obtém também o conjunto Y.

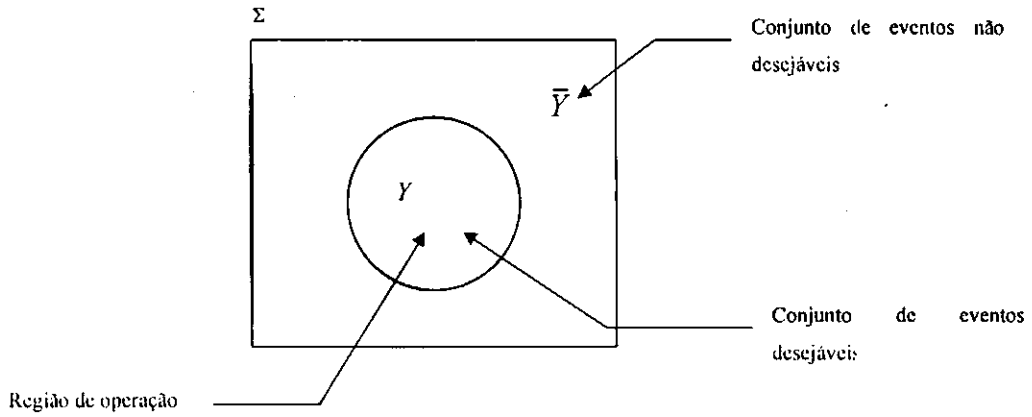


Figura 4.3: Diagrama de Venn do espaço de comportamento de um sistema

O resultado obtido pelo algoritmo de síntese é uma lista de eventos que corresponde ao comportamento do controlador, ou seja, sua especificação. O algoritmo não faz a distinção entre eventos controláveis e não-controláveis, no entanto, no projeto de sistemas a eventos discretos fisicamente realizáveis, estes eventos devem ser considerados na estruturação do modelo. O algoritmo proposto neste trabalho oferece a possibilidade do projetista identificar os eventos que não se ajustam a especificação funcional do projeto e os eventos que podem levar o sistema a um estado de bloqueio. A modificação da especificação com base nos resultados da aplicação do algoritmo, considerando os eventos não-controláveis, e a aplicação interativa do algoritmo de síntese fornece a máxima

especificação para o controlador do sistema. Vale salientar que, a síntese do supervisor é possível uma vez que todo sistema a eventos discretos fisicamente realizável possui capacidade finita [Bar96], tornando a verificação de modelo factível.

4.1.3 O Algoritmo SSS (Síntese de Supervisor com SMV)

O algoritmo SSS utiliza a verificação de modelo para avaliar a especificação funcional de um sistema, através da análise iterativa de fórmulas CTL que representam os estados desejáveis. O SMV avalia se uma determinada fórmula CTL é verdadeira ou falsa como descrito na seção 3.2. No caso em que a fórmula é avaliada como falsa, a ferramenta poderá apresentar como resposta ao procedimento, um caminho (*trace*) contendo todos os estados intermediários a partir do estado inicial s_0. O algoritmo utiliza o quantificador de caminho *A* (*all path*) nas fórmulas para forçar o SMV a fornecer o *trace* caso a fórmula seja avaliada como falsa.

O *trace* fornecido pelo SMV pode não ser o único em que a formula é falsa. Portanto, é imprescindível que o algoritmo seja capaz de conduzir o SMV a encontrar todos os *traces* para os quais a fórmula CTL não é verdadeira. Este problema é solucionado realizando interações recursivas com o SMV, em que a cada resposta negativa, a fórmula CTL é modificada para conter o *trace* que a conduziu a esta resposta. Quando todos os *traces* estiverem contidos na fórmula CTL o SMV irá avaliar a fórmula como verdadeira. A Figura 4.4 ilustra o diagrama funcional para algoritmo SSS.

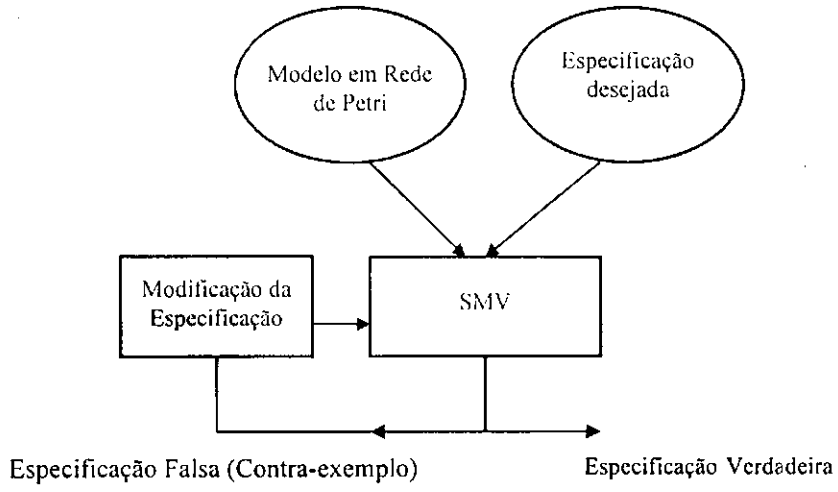


Figura 4.4: Diagrama funcional do algoritmo SSS

4.1.4 O Algoritmo

A especificação funcional do sistema deve ser definida em termos de transições de estados, em que uma transição de estado define a mudança do estado atual para um estado futuro, para os estados alcançáveis pelo sistema. A união destas transições de estado, especificadas em CTL, constituem a especificação funcional que será executada pelo SED. A especificação desejada é definida como:

$$Esp = \bigvee_{i=1}^{n} Esp_i$$

em que :

Esp_i é a especificação que define uma transição de estado.

$Esp_i = AG(x_n \rightarrow AC(x_{n+1}))$ é a transição de estados em fórmula CTL

onde :

C representa os modificadores temporais F, G e X,

x_n é o estado atual do sistema, e

x_{n+1} é o estado futuro do sistema.

O algoritmo SSS deve ser aplicado a cada transição de estado Esp_i pertencente a especificação Esp. O algoritmo procura obter de cada transição de estado qual o *trace* que conduz o sistema a um estado não desejado. O *trace* não desejado é então comparado ao *trace* que conduz o sistema ao estado desejável, o resultado da comparação entre os *traces* pode ser um evento não desejado. Por exemplo, seja $s\sigma$ o *trace* de eventos não desejáveis, e $s\rho$ o *trace* de eventos desejáveis a uma transição de estados, a comparação dos *traces* revela que a seqüência desejada pode não ser alcançada, dado que, antes do estado x_{n+1} ser alcançado através de $s\rho$, o evento σ poderá conduzir o sistema a um estado x_y não pertencente a especificação funcional. Uma mesma transição de estado pode conduzir o sistema a vários estados indesejáveis, portanto, a avaliação de cada transição deve ser executada até que todos os eventos indesejáveis sejam encontrados.

Descrição do Algoritmo

O algoritmo SSS mostrado na Figura 4.5 realiza a síntese de supervisor utilizando como entrada uma lista de estados $S[NE]$ que descreve a especificação desejada para o SED. A variável NE refere-se ao número de transições de estados da especificação. O processo de síntese inicia com um laço definido pelo comando enquanto, este laço é executado até que todas as transições de estado tenham sido avaliadas. A avaliação de cada transição de estado é iniciada pelo SMV através da verificação da fórmula $AG\ (x_n \rightarrow !EC\ (x_{n+1}))$, que verifica se o estado x_{n+1} poderá ser atingido a partir do estado x_n. Caso a resposta seja falsa significa que existe pelo menos um caminho para o qual x_{n+1} é alcançavel a partir de x_n, caso contrário o algoritmo deverá ser interrompido devido a especificação não ser fisicamente realizável. A lista *TraceP* armazena o caminho possível para a transição de estado corrente de índice i. A seqüência do algoritmo avalia para cada transição de estado se o estado x_{n+1} sempre poderá ser alcançado a partir de x_n. A fórmula CTL: $AG\ (x_n \rightarrow AC\ (x_{n+1}))$ é avaliada pelo SMV como falsa ou verdadeira, então o caminho, para o qual a fórmula é falsa, fornecido pelo SMV é armazenado na lista *TraceL* para ser comparada evento a evento com os itens da lista *TraceP*. A comparação dos itens das listas *TraceP* e *TraceL* identifica qual evento σ_r (uma transição da rede de Petri) conduz o sistema a um estado diferente de x_{n+1}. Este evento é armazenado na lista $Super[i]$ referente a cada transição de estado analisada. Para identificar outros eventos indesejáveis ao sistema, o algoritmo modifica o estado x_{n+1}, incluindo no mesmo o evento E_r originado pela ocorrência do evento σ_r. O algoritmo repete a análise da fórmula $AG\ (x_n \rightarrow AC\ (x_{n+1}))$ até que todos os eventos σ_r sejam encontrados.

Os eventos contidos em $Super_i$ correspondem aos eventos não desejáveis ao comportamento do sistema, em que o índice i está relacionado com cada transição de estado da especificação *Esp*. A determinação destes eventos relacionados a cada transição de estado é o conjunto Σ_v^0 (ou $\overline{Y}$), em que os eventos não inclusos pertencentes ao mapa de entradas de controle $\Theta(x)$ do supervisor S.

```
Algoritmo
    Declare
        var NE ; {Número de transições de estado}
        {Lista que contém as transições de estado para a especificação funcional S}
        lista S[NE:n];
        {Lista que contém os eventos não desejáveis para cada transição de estado}
        lista Super{NE:n];
        {Lista que contém um caminho possível para a transição de estado Si }
        lista TraceP [n];
        {Lista que contém a transição de estado quando a transição de estado é falsa}
        lista TraceL[n];
        numérico i, j, k;
    enquanto i < NE {Enquanto todas as transições de estado não forem avaliadas}
        Ler S[i]; {Ler os elementos xn e xn+1 da especificação Si }
        Se AG (xn→ !EC (xn+1)) é verdadeiro {Avaliação do SMV}
            então interrompa;
        senão
            salvar o caminho em TraceP; {Resposta do SMV}
            repita
                se AG (xn → AC xn+1) é verdadeiro {Avaliação do SMV}
                    então interrompa;
                senão
                    salvar o caminho em TraceL;  {Resposta do SMV}
                    enquanto j < n { Enquanto não correr todos os elementos de TraceL}
                        se TraceP[j] ≠ TraceL[j]
                            então
                            se TraceL[j] ∉ Super[i]
                                então Super[i][k] ← TraceL[j];
                                    incrementa k;
                            fim se
                             xn+1 ← xn+1 | xr ; {Modificar o estado xn+1 de acordo com o
                             estado definido por TraceL[j]}
                        fim se
                        incrementa j;
                    fim enquanto
                fim senão
            fim repita
        fim senão
        incrementa i;
    fim enquanto
Fim Algoritmo
```

Figura 4.5: Algoritmo para Síntese de Supervisor através de SMV

Na seção seguinte mostraremos um exemplo simples para ilustrar a aplicação do algoritmo. No Capítulo 5 dois outros exemplos mais complexos são apresentados.

4.2 Síntese de um Supervisor para Sistemas de Manufatura

Nesta Seção apresentaremos dois exemplos simples de sistemas de manufatura para ilustrar a aplicação do algoritmo SSS na obtenção da especificação do supervisor.

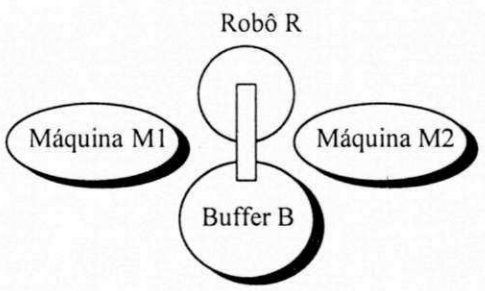
4.2.1 Exemplo 1

A Figura 4.6 ilustra um exemplo de um SED composto por duas máquinas, *M1* e *M2*, um robô compartilhado, *R*, para descarregar e carregar peças intermediárias de/para um buffer *B*. As peças devem ser processadas primeiramente pela máquina *M1* e em seguida pela máquina *M2*. As peças entram no sistema fixadas em bandejas e são carregadas para a máquina *M1*. Após o processamento, o robô descarrega a peça intermediária de *M1* para o buffer *B*. Na máquina *M2* as peças intermediárias são automaticamente descarregadas e processadas. Quando a máquina *M2* termina o processamento de uma peça, o robô *R* descarrega o produto final, desprende a mesma da bandeja que é devolvida a máquina *M1*.

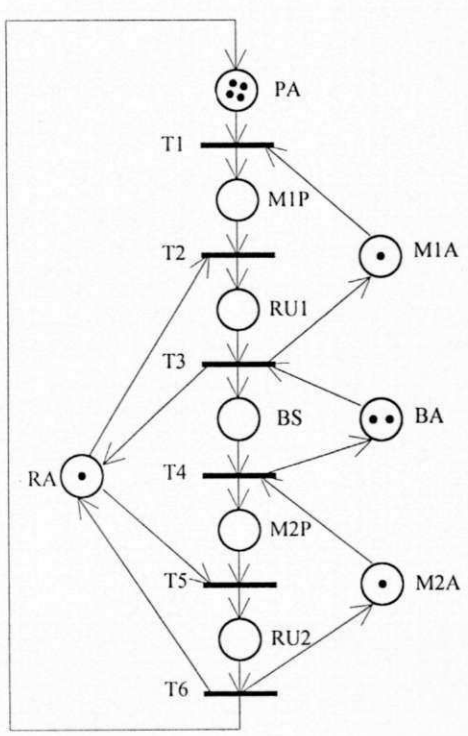
Neste exemplo assumimos que 4 bandejas de fixação estão disponíveis para a condução de peças dentro do sistema de manufatura. No modelo de rede de Petri, apresentado na Figura 4.7 estas bandejas são representadas por quatro fichas no lugar PA ($M_0(PA) = 4$), além disso o buffer tem capacidade para duas peças ($M_0(BA) = 2$). O SMV restringe sua aplicação a redes seguras, o que não acontece com a rede em questão. Para tornar possível a utilização do algoritmo é necessário converter esta rede em uma rede segura. Portanto, devemos aplicar as técnicas de modelagem de buffers introduzidas por Zhou e DiCesare [ZD93] e descritas no Capítulo 2. A Figura 4.8 mostra a rede de Petri equivalente modelando os buffers.

Modificada a rede de Petri original, através da substituição dos lugares cuja capacidade é superior a um, pelos seus equivalentes de rede segura, podemos então aplicar o algoritmo SSS para determinar os eventos não desejáveis ao funcionamento do sistema de manufatura. A rede de Petri proposta na Figura 4.7 corresponde ao modelo de

funcionamento do sistema. Desta forma, a aplicação do algoritmo SSS tem como principal objetivo verificar se o comportamento da rede de Petri proposta atende aos requisitos determinados pela especificação funcional definida para o sistema.



4.6: Um exemplo de sistema a eventos discretos



PA	:bandejas disponíveis;
M1P	:*M1* carrega, fixa e processa uma peça;
RU1	:*R* descarrega uma peça intermediária;
BS	:*B* contém peças intermediárias armazenadas;
M2P	:*M2* carrega e processa uma peça intermediária;
RU2	:*R* descarrega produto final e retorna a bandeja para *M1*;

Figura 4.7: Rede de Petri para o sistema da Figura 4.6

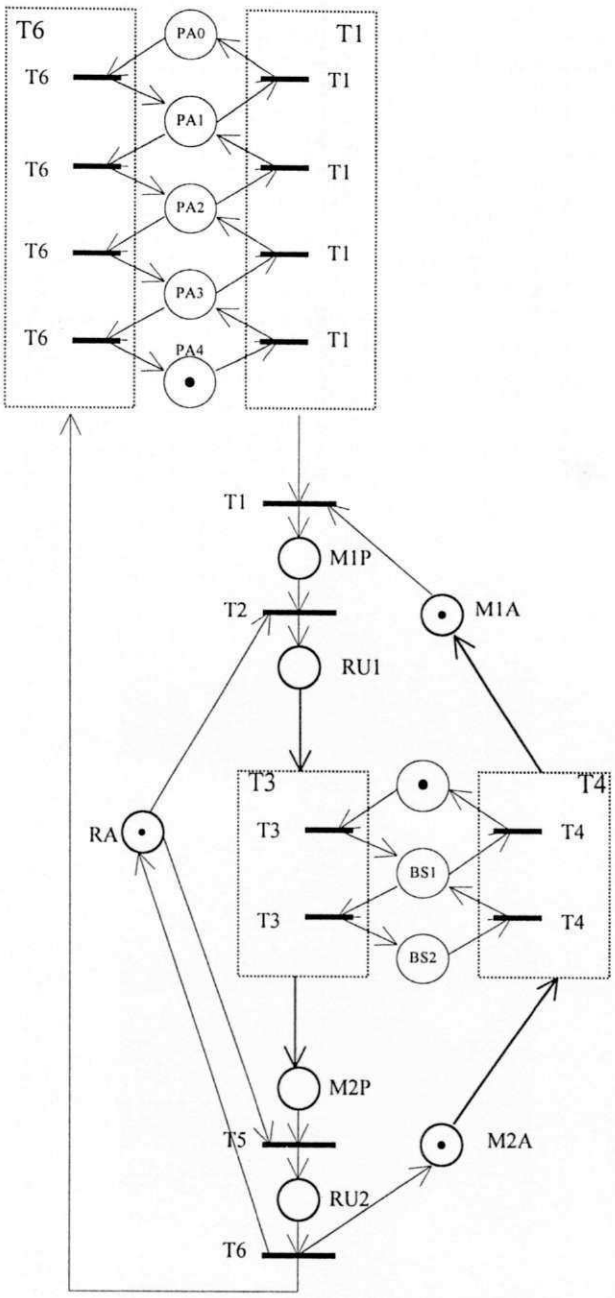


Figura 4.8: Rede de Petri equivalente a rede de Petri da Figura 4.7

A especificação funcional do sistema de manufatura apresentado na Figura 4.6, determina que uma peça para ser produzida deve passar pelo processamento na máquina *M1* e máquina *M2*, respectivamente. A observação individual do fluxo de uma bandeja dentro do modelo de rede de Petri indica que a especificação funcional é atendida. No entanto, se levarmos em consideração que as máquinas *M1* e *M2* podem estar processando concorrentemente e que peças intermediárias podem ser armazenadas no buffer, nada podemos afirmar sobre o comportamento do sistema. Para aplicar o algoritmo SSS é necessário transformar a especificação funcional do sistema em várias fórmulas CTL, que expressam as transições de estado do sistema. O conjunto de todas as transições de estado em fórmula CTL define a especificação para ser utilizada pelo algoritmo SSS.

O algoritmo SSS determina que as transições de estado sejam definidas conforme a fórmula CTL : $AG\ (x_n \rightarrow AF\ x_{n+1})$. Esta fórmula indica que sempre é possível alcançar o estado x_{n+1} desde que o estado x_n seja verdadeiro. Aplicando esta regra obtemos a seguinte especificações apresentada na Tabela 4.1:

1	*AG (!PA0 → AF M1P)*:	Sempre é possível que a máquina *M1* venha a processar uma peça, dado que existe pelo menos uma bandeja disponível na sua entrada;
2	*AG (M1P → AF RU1)*:	Sempre é possível para o robô *R* descarregar uma peça intermediária, dado que existe uma peça sendo processada em *M1*;
3	*AG (RU1 → AF !BS0)*:	Sempre é possível para o robô *R* descarregar uma peça intermediária para o buffer BS, dado que o robô já retirou a peça de *M1*;
4	*AG (!BS0 → AF M2P)*:	Sempre é possível que a máquina *M2* venha a processar uma peça, dado que existe pelo menos uma peça intermediária no buffer *B*;
5	*AG (M2P → AF RU2)*:	Sempre é possível para o robô *R* descarregar uma peça, dado que existe uma peça sendo processada em *M2*;
6	*AG (RU2 → AF !PA0)*:	Sempre é possível para o robô *R* recolocar uma bandeja na entrada de M1, dado que o robô já retirou a peça da máquina *M2*;

Tabela 4.1 – Especificação funcional para o sistema ilustrado na Figura 4.6.

Aplicando o algoritmo SSS e a especificação S descrita na Tabela 4.1, verificamos que a mesma não é possível de ser executada, devido a transição de estado $AG(M2P \rightarrow AF\ RU2)$ nunca poder ser realizada se a seqüência de eventos *T1D,T2,T3A,T4A,T1C,T2,T3A,T1B,T2,T3B,T1A,T2* _LOOP9:(none) (*trace* fornecido pelo SMV) ocorrer a partir da marcação inicial. Esta sequência de eventos leva o sistema a um estado de bloqueio (*deadlock*), em que nenhuma outra transição de estado poderá ocorrer. A Tabela 4.2 mostra todos os passos da aplicação do algoritmo SSS para o qual identificamos *T2* como o evento a ser controlado no estado *RU2*. O algoritmo também sugere o controle sobre o evento *T1C*, no entanto, o controle deste evento significa a ausência de peças sendo processadas concorrentemente no sistema de manufatura. Observando a especificação 5 na Tabela 4.2, podemos verificar que a transição de estado representada pela fórmula $AG\ (M2P \rightarrow AF\ RU2)$ não é verdadeira se controlarmos o estado *M1P* a partir do evento *T1C*, no entanto, passa a ser verdadeira quando controlamos o estado *RU1* a partir do evento *T2*. Portanto, podemos concluir que o controle de *T2* a partir de *RU2* torna possível a transição de estado $AG\ (M2P \rightarrow AF\ RU2)$.

Especificação	Resposta do SMV	Trace
As especificações 1, 2, 3 e 4 descrevem o comportamento do sistema iniciando no processamento de uma peça na Máquina *M1* até o processamento de uma peça na Máquina *M2*. Podemos observar que para cada especificação o algoritmo avalia primeiramente se existe um caminho possível, a partir do estado inicial, para que a especificação seja verdadeira. Conforme descrito pelo algoritmo, para verificar este caminho cada especificação de transição de estado deve ser descrita pela fórmula CTL $AG\ (x_n \rightarrow !EF\ (x_{n+1}))$. Em todos os casos o SMV avaliou que existe caminhos possíveis, indicados pelos *traces*. Na Segunda fase da aplicação do algoritmo, é avaliado, para cada especificação, se existe apenas um caminho para qual a especificação $AG\ (x_n \rightarrow AF\ (x_{n+1}))$ é verdadeira. A avaliação desta fórmula para as especificações 1, 2, 3 e 4 é sempre verdadeira, significando a inexistência de eventos que possam levar o sistema a um estado indesejado ou de bloqueio.		
Especificação 1		
$AG\ (!PA0 \rightarrow !EF\ M1P)$	Falso	T1D
$AG\ (!PA0 \rightarrow AF\ M1P)$	Verdadeiro	
Especificação 2		

AG (M1P →!EF RU1)	Falso	T1D,T2
AG (M1P → AF RU1)	Verdadeiro	
Especificação 3		
AG (RU1 →!EF (!BS0))	Falso	T1D,T2,T3A
AG (RU1 → AF (!BS0))	Verdadeiro	
Especificação 4		
AG (!BS0 →!EF M2P	Falso	T1D,T2,T3A,T4A
AG (!BS0 → AF M2P)	Verdadeiro	
A aplicação do algoritmo SSS à especificação 5 mostra que o estado *RU2* poderá ser alcançado a partir do estado *M2P* através da seqüência de eventos *T1D,T2,T3A,T4A,T5*. No entanto esta seqüência de eventos representa apenas um caminho para o qual a especificação *AG (M2P →!EF RU2)* é verdadeira. A Segunda fase de aplicação do algoritmo revela que o sistema poderá evoluir para um estado de bloqueio total a partir da seqüência de estados *T1D,T2,T3A,T4A,T1C,T2,T3A,T1B,T2,T3B, T1A,T2*. Na seqüência da avaliação da especificação o algoritmo compara o caminho gerado por *AG (M2P →!EF RU2)* com o trace fornecido *AG (M2P → AF RU2)*, e isola o evento *T1C* como um dos eventos que conduzem o sistema ao bloqueio. A ocorrência deste evento gera o estado *M1P*, desta forma a especificação deve ser alterada para *AG (M2P → AF (RU2 | M1P))*, fazendo com que este evento não seja mais considerado na avaliação da transição de estado. A avaliação da transição de estado modificada revela a existência de uma outra seqüência de eventos, para a qual o sistema permanece em estado de bloqueio *T1D,T2,T3A,T1C,T2,T3B,T1B,T2,T4B,T3B,T1A,T2*. Podemos observar que *T2* é um outro evento inconvejável pois leva o sistema ao bloqueio. A ocorrência do evento *T2* implica na inclusão do estado *RU2* na transição de estado modificada anteriormente. Uma nova avaliação do algoritmo conduz a uma resposta do SMV em que podemos concluir que os eventos *T1C* e *T2* devem ser controlados para eliminar a possibilidade de bloqueio do sistema		
Especificação 5		
AG (M2P →!EF RU2)	Falso	T1D,T2,T3A,T4A,T5
AG (M2P → AF RU2)	Falso	T1D,T2,T3A,T4A,T1C,T2,T3A,T1B,T2,T3B, T1A,T2 _LOOP9:(none)
AG (M2P → AF (RU2 | M1P))	Falso	T1D,T2,T3A,T1C,T2,T3B,T1B,T2,T4B,T3B,T1A,T2 _LOOP10:(none)
AG (M2P → AF (RU2 | M1P | RU1))	Verdadeiro	
Especificação 6		
AG (RU2 →!EF (!PA0))	Falso	T1D,T2,T3A,T4A,T5
AG (RU2 → AF (!PA0))	Verdadeiro	

Tabela 4.2: Aplicação do algoritmo SSS a especificação desejado para o modelo da Figura 4.6

Duas soluções de controle para evitar o estado de bloqueio para o sistema da Figura 4.6 são consideradas por [Bar96]. A primeira delas é a modificação da estrutura da rede de Petri que modela o sistema, através da inclusão de um lugar de controle *lc* com uma marcação inicial $(M_0(lc)) = 3)$. Veja a Figura 4.9. Nesta solução existe um aumento da estrutura da rede pois são acrescentados um lugar e dois arcos.

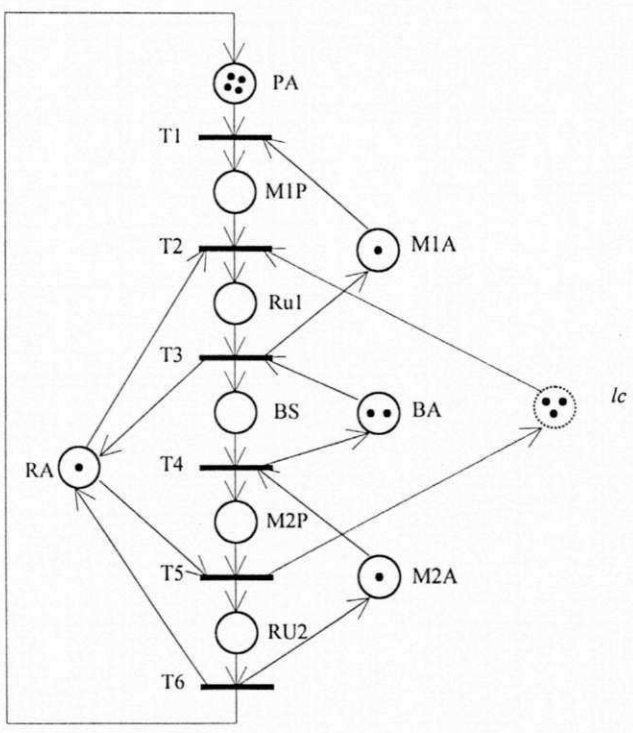


Figura 4.9: Sistema de Manufatura com lugar de controle *lc*

Uma segunda alternativa seria a utilização de uma *RPFHT* (*Redes de Petri com Função para Habilitação de Transições*) [PC92] com a mesma estrutura da rede que modela o sistema original. O bloqueio do sistema é evitado aplicando-se uma restrição ao disparo da transição *T2* através da função de habilitação $\vartheta_2 = [M(bs) \leq 1 \vee M(m_2 p) = 0]$. Desta forma, *T2* irá ocorrer apenas se o número de fichas de *BS* for menor ou igual a 1 ou se o número de fichas em *M2P* for igual a zero. A rede de Petri supervisora é apresentada na Figura 4.10.

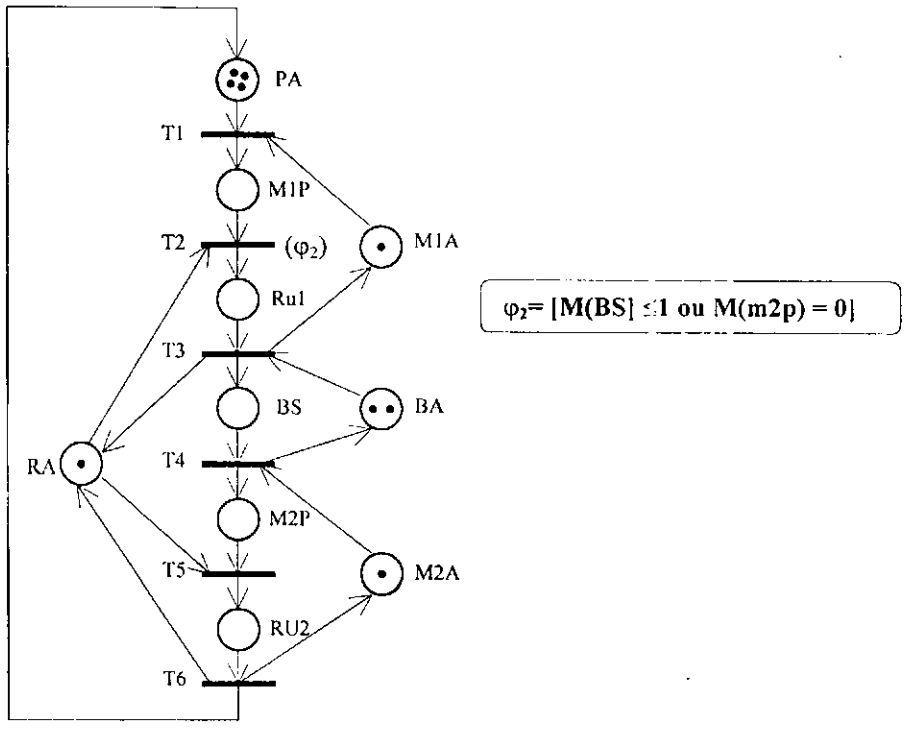


Figura 4.10: *RPFTH* supervisora para o sistema de manufatura

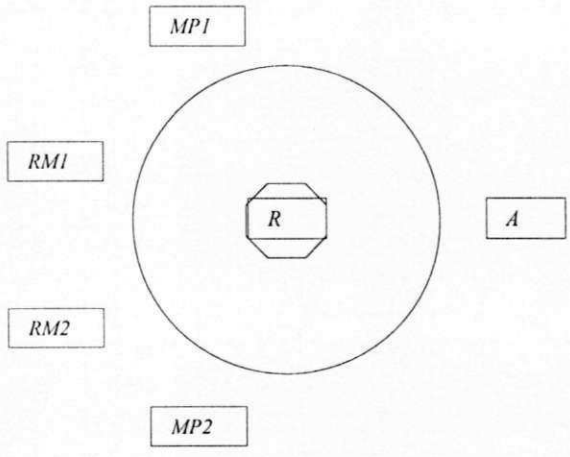
4.2.2 Exemplo 2

Considere uma célula de manufatura [BLP95], como mostrada na Figura 4.11(a), nesta é possível identificar dois depósitos de matéria prima *RM1* e *RM2*; dois centros de processamento *MP1* e *MP2*; e um centro de montagem *A*. Um robô *R* move matéria prima dos depósitos *RM1* e *RM2* para os centros de processamento *MP1* e *MP2*, respectivamente, e destes centros para a o centro de montagem *A*. A Figura 4.11(b) mostra a rede de Petri que modela esta célula, em que cada três peças montadas constitui um item desta célula. Os itens são transportados para outra célula. A saída de um item de *A*, é modelada pela transição T_5 (evento μ). O maior número de peças que o centro de montagem *A* suporta é três. Estas peças são suficientes para compor um item, portanto, isto siguinifica que o número de fichas máximo que deve marcar o lugar *A* é três.

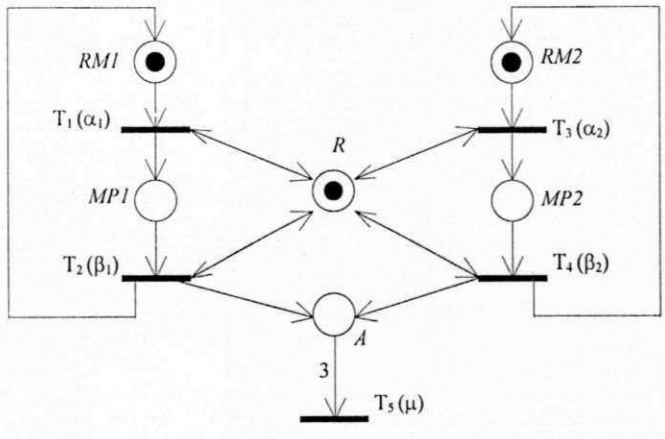
Seja $\Sigma = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \mu\}$, onde $\Sigma_c = \{\beta_1, \beta_2, \mu\}$ e $\Sigma_u = \{\alpha_1, \alpha_2\}$. Para ser possível aplicar o algoritmo *SSS*, no sentido de obter o supervisor que atenda a especificação de

comportamentro do sistema, é necessário aplicarmos a técnica de modelagem de buffers para o lugar A da rede de Petri da Figura 4.11(b), resultando na Figura 4.12.

Quando modelamos o lugar A com um buffer de três lugares resolvemos o problema de não limitação do lugar A, uma vez que se analisarmos a marcação do mesmo dentro da árvore de alcançabilidade, identificamos que a rede não é limitada. Para criarmos as condições de aplicabilidade de *SSS*, vamos considerar que a capacidade do lugar A é de quatro fichas. Desta maneira, será possível estabelecermos a restrição de que o mesmo só poderá conter três fichas. A Figura 4.12 ilustra a rede de Petri da Figura 4.11(b) modificada.



(a)



(b)

Figura 4.11: (a) Exemplo de uma célula de manufatura; (b) modelo de uma célula de manufatura

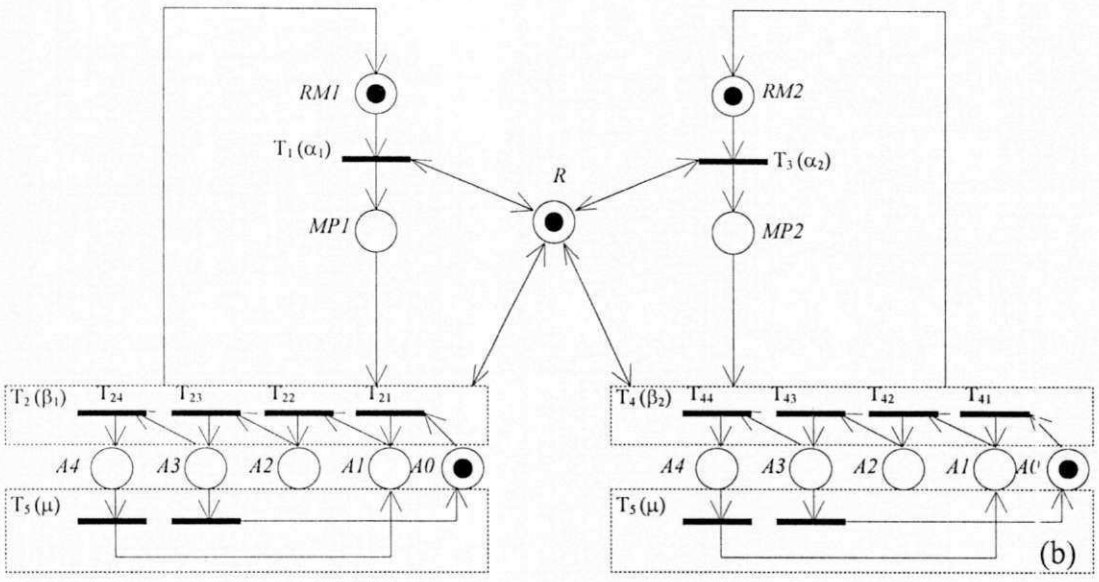


Figura 4.12: Rede de Petri da Figura 4.11(b) modificada

O comportamento desejado para o sistema modelado pela rede de Petri apresentada na Figura 4.12 resume-se apenas a uma transição de estado, que significa: uma vez que existe matéria prima nos depósitos *RM1* e *RM2* o sistema nunca colocará quatro peças no buffer *A*. Portanto, em fórmula CTL, a transição de estado pode ser assim definida: $AG(RM1 \mid RM2 \rightarrow AG(!A_4))$. O resultado da aplicação do algoritmo *SSS* pode ser observado na Tabela 4.3.

Especificação	Resposta do SMV	Trace
$AG(RM1 \mid RM2 \rightarrow !EG(!A_4))$	Falso	$T_1, T_3, T_{41}, T_3, T_{42}, T_{23}, T_5$
$AG(RM1 \mid RM2 \rightarrow !AG(!A_4))$	Falso	$T_1, T_3, T_{41}, T_3, T_{42}, T_3, T_{43}, T_{24}$
$AG(RM1 \mid RM2 \rightarrow !AG(!A_4 \mid MP2))$	Falso	$T_1, T_3, T_{41}, T_3, T_{42}, T_3, \underline{T_{43}}, T_{24}$
$AG(RM1 \mid RM2 \rightarrow !AG(!A_4 \mid MP2 \mid A_3))$	Falso	$T_1, T_3, T_{41}, T_3, T_{42}, T_3, T_{43}, \underline{T_{24}}$
$AG(RM1 \mid RM2 \rightarrow !AG(!A_4 \mid MP2 \mid A_3 \mid A4))$	Verdadeiro	

Tabela 4.3: Trace da aplicação do algoritmo *SSS* na especificação da célula de manufatura

Portanto, a especificação para o supervisor é dada por :

Transição de Estado	**Eventos indesejáveis**
$RM1 \mid RM2 \rightarrow A3$	T_{43} e T_{24}

Observe, que pelos *traces* apresentados na Tabela 4.3 a transição T_3 (evento α_2) deveria ser controlada, sugerindo que se este evento for controlado o sistema poderia executar a especificação sem problemas. Entretanto, o evento α_2 não é controlável, impossibilitando o controle da transição T_3 da rede de Petri que modela o sistema. Por outro lado, o algoritmo também indica o controle dos eventos controláveis β_1 e β_2, que na rede de Petri da Figura 4.12 são representados pelas transições T_{21}, T_{22}, T_{23}, T_{24} *e* $T_{41}, T_{42}, T_{43}, T_{44}$, respectivamente. As transições que devem ser controladas são T_{43} e T_{24}, representando que uma vez ocorrido duas vezes o evento β_2, o evento β_1 poderá ocorrer apenas uma vez. Uma outra possibilidade é a ocorrência de três eventos β_1 sem que β_2 ocorra..

Capítulo 5

Exemplos de Aplicação do Algoritmo SSS em Sistema a Eventos Discretos

Neste Capítulo apresentamos dois exemplos da aplicação do algoritmo de síntese SSS a SEDs. O primeiro SED consiste de um Sistema de Produção em Lote (SPL) capaz de executar um conjunto de especificações distintas (receitas). No exemplo apresentado neste tralho a síntese do supervisor é realizada apenas para uma receita *R*. Uma definição precisa de SPL pode ser encontrado em [BPF98]. O segundo exemplo representa o Sistema Flexível de Manufatura (SFM) proposto por [BPF97]. O modelo permite que uma grande quantidade de especificações possam ser executadas com o objetivo de produzir peças ou produtos diferentes. A aplicação do algoritmo SSS tem por objetivo determinar qual é o supervisor que deverá ser aplicado ao SFM para que este possa produzir uma peça ou produto determinado pela especificação.

5.1 Um sistema de Produção em Lote

O sistema de transferência de fluídos ilustrado na Figura 5.1 é formado por dois tanques interligados por uma válvula *V2*. O tanque A pode ser alimentado através da válvula *V1* e o tanque B pode ser escoado através da válvula *V3*. As vazões das válvulas

V1, *V2* e *V3* são tais que $q_a > q_b > q_c$, respectivamente. Os fluídos são transferidos, sempre no sentido do tanque A para o tanque B, conforme a receita (especificação) determinada de acordo com o controle de abertura das válvulas. A Figura 5.2 mostra a rede de Petri que modela os componentes do sistema. O modelo do sistema é obtido através da fusão das transições comuns aos modelos dos tanques (Figuras 5.2(a) e 5.2(c)), válvulas (Figura 5.2(d)) e redes de ligação (Figuras 5.2(b) e 5.2(e)). Considera-se transições comuns aquelas que possuem o mesmo nome.

O sistema de transferência de fluídos deste exemplo pode executar várias receitas sem a necessidade de modificar a estrutura da rede de Petri. Para que esta rede possa modelar a receita *R*, alguma forma de controle tem que ser introduzida. Observe que a execução da receita *R* restringe o espaço de estados que representa comportamento do sistema. Existem várias formas de implementação para o supervisor, no entanto não é o objetivo deste trabalho modelar o supervisor, mas fornecer a especificação para o mesmo através da identificação dos eventos que devem ser controlados para cada transição de estado do sistema em malha aberta.

Neste exemplo utilizamos o algoritmo SSS para obter o supervisor que executa a receita *R* de acordo a seguinte especificação:

O líquido do tanque Ta é transferido para o tanque Tb quando a válvula V2 está aberta. Quando o nível do líquido no tanque atingir $x_2(t) = k_2$ *a válvula V2 deve ser fechada e após o termino desta operação, as válvulas V1 e V3 devem ser abertas para encher o tanque Ta e esvaziar Tb. Quando o tanque Ta estiver cheio* ($x_1(t) = k_1$), *e o tanque Tb estiver vazio* ($x_2(t) = 0$), *V1 e V3 devem ser fechadas e V2 aberta para transferir outra vez o líquido do tanque Ta para o tanque Tb. O controlador deve operar sobre as válvulas a fim de que a operação de transferência ocorra conforme o especificado. Assumimos que inicialmente o tanque Ta está cheio e que o tanque Tb está vazio, assim como todas as válvulas estão fechadas.*

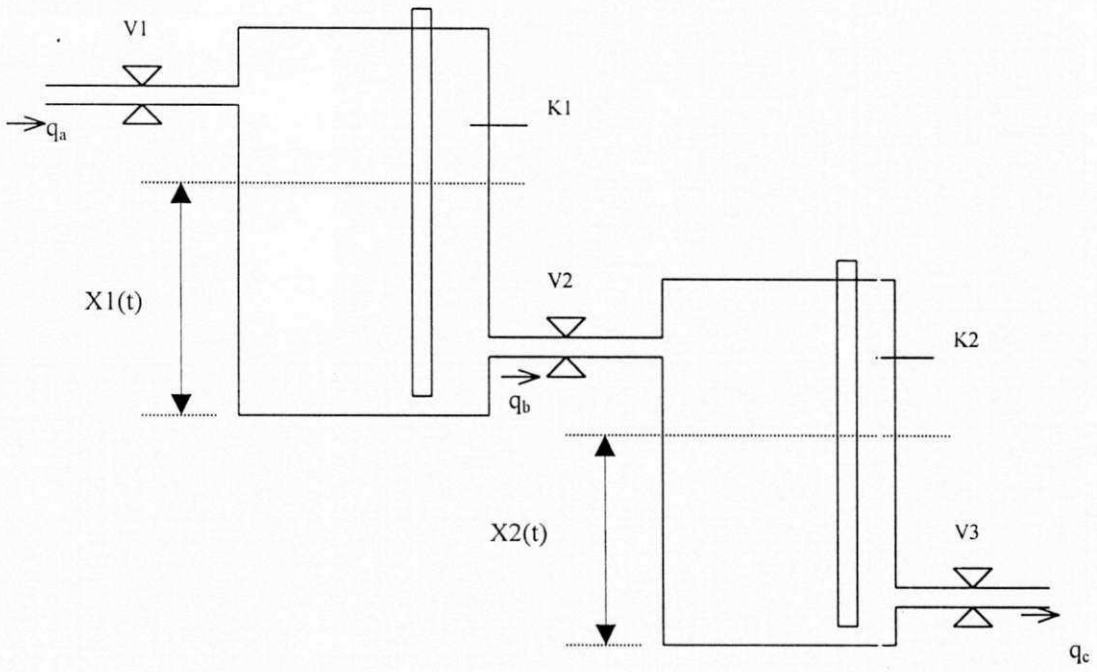


Figura 5.1: Sistema de transferência de fluídos

Os lugares dos modelos da Figura 5.2 tem o seguinte significado:

Tanque *Ta* (ver a Figura 5.2(a)):

- estado E1 (lugar P1) significa que *Ta* está enchendo. O estado E1 é alcançado quando *V1* e *V2* estão abertas.
- estado E2 (lugar P2) significa que *Ta* está enchendo. O estado E2 é alcançado quando *V1* está aberta e *V2* está fechada.
- estado E3 (lugar P3) significa que *Ta* está esvaziando. O estado E3 é alcançado quando *V1* está fechada e *V2* está aberta.
- estado E4 (lugar P4) significa que *Ta* está constante. O estado E4 é alcançado quando *V1* e *V2* estão fechadas.

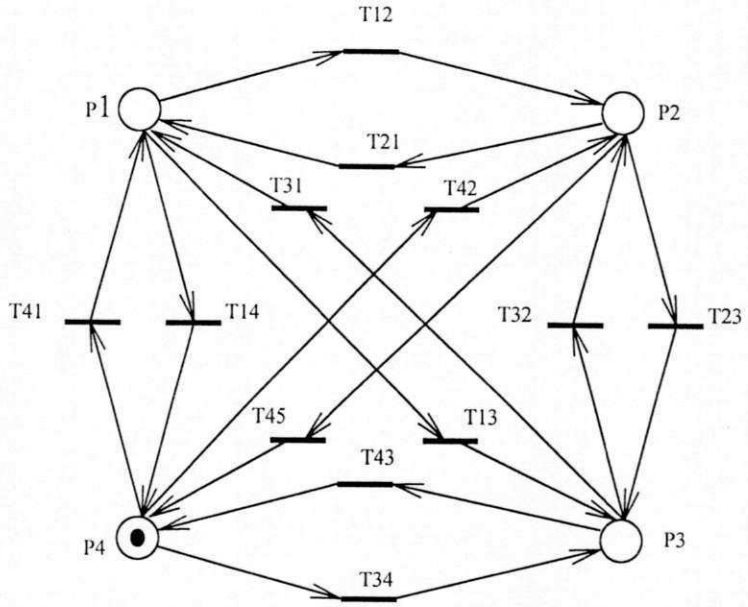


Figura 5.2 (a): Tanque A

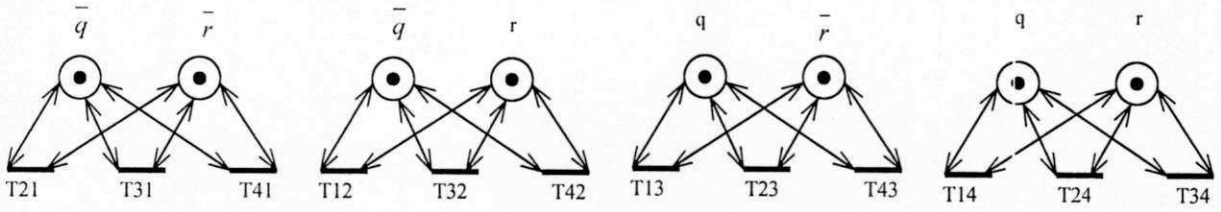


Figura 5.2(d) – Redes de Ligação do tanque A

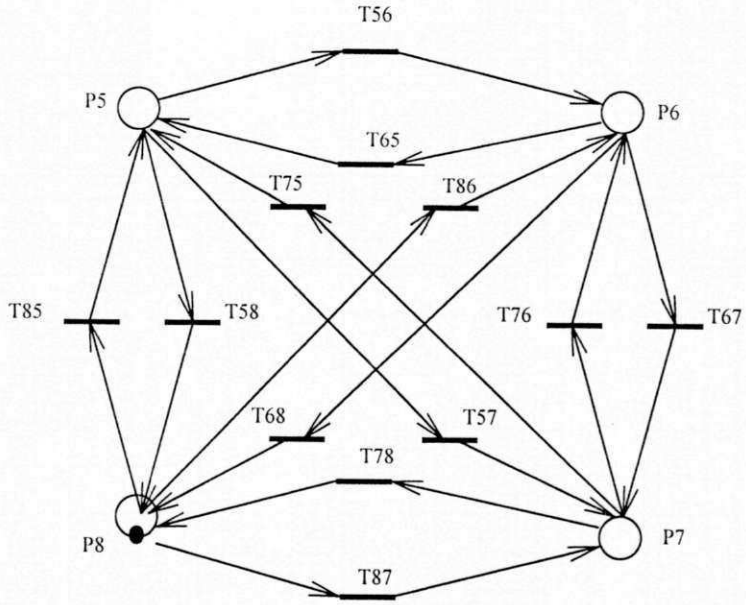


Figura 5.2(c): Tanque B

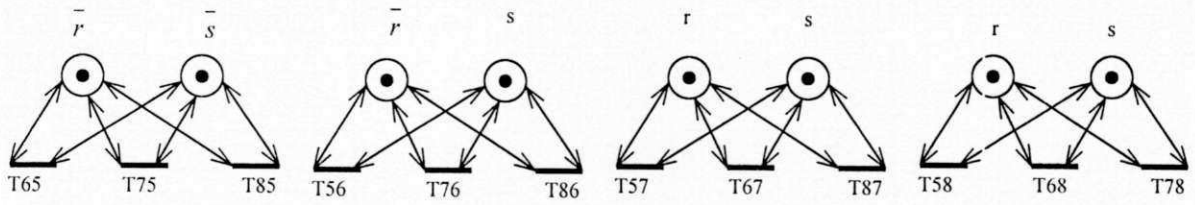


Figura 5.2(d) – Redes de Ligação do tanque B

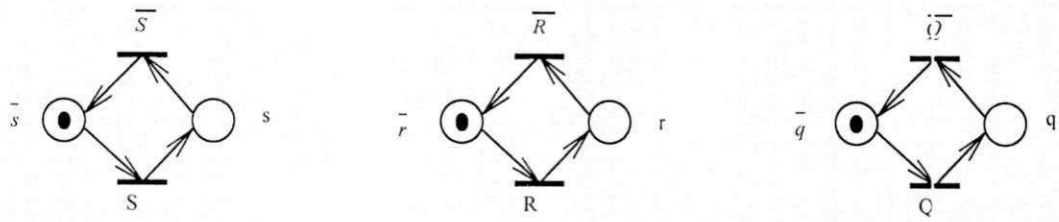


Figura 5.2(e): Redes de Petri para as válvulas V1, V2, e V3

Tanque *Tb* (ver a Figura 5.2(c):

- estado E5 (lugar P5) significa que *Tb* está enchendo. O estado E5 é alcançado quando *V2* e *V3* estão abertas.
- estado E6 (lugar P6) significa que *Tb* está enchendo. O estado E6 é alcançado quando *V2* está aberta e *V3* está fechada.
- estado E7 (lugar P7) significa que *Tb* está esvaziando. O estado E7 é alcançado quando *V2* está fechada e *V3* estão aberta.
- estado E8 (lugar P8) significa que *Tb* está constante. O estado E8 é alcançado quando *V2* e *V3* estão fechadas.

Válvula *V1* (ver a Figura 5.2(e)) :

- lugar $\overline{q}$ significa que a válvula *V1* está aberta.
- lugar q significa que a válvula *V1* está fechada.

Válvula *V2* (ver a Figura 5.2(e)) :

- lugar $\overline{r}$ significa que a válvula *V2* está aberta.
- lugar r significa que a válvula *V2* está fechada.

Válvula *V3* (ver a Figura 5.2(e)) :

lugar $\overline{s}$ significa que a válvula *V3* está aberta.

lugar s significa que a válvula *V3* está fechada.

Podemos concluir que a rede de Petri originada pela fusão dos modelos de tanques, válvulas e redes de ligação é uma rede segura, uma vez que, em qualquer condição de disparo de uma transição o número máximo de fichas em um lugar é sempre igual a um, desta maneira não é necessário utilizar técnicas de modelagem para buffers e podemos utilizar o algoritmo SSS para obtenção do supervisor, que irá restringir o comportamento do sistema ao especificado pela receita *R*.

A receita *R* pode ser representada pelo diagrama de transição de estados ilustrado na Figura 5.3. O diagrama indica como ocorrerá a mudança de estados do sistema através da seqüência de eventos determinada pela receita. Ei,Ej representam os estados concorrentes assumidos pelos tanques *Ta* e *Tb*, respectivamente. Enquanto β_n representa os estados determinados pela abertura ou fechamento das válvulas. Comparando o diagrama com a receita *R* observamos que:

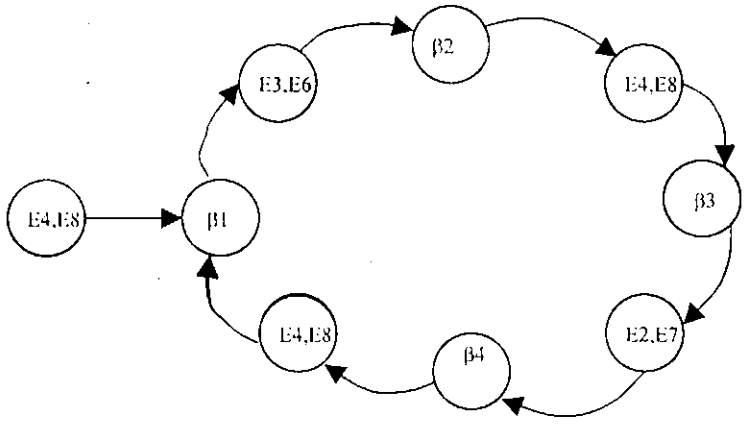


Figura 5.3: Diagrama de transição de estados para a receita R

Onde,

- E4,E8 representa o estado inicial dos tanques, significando *Ta* e *Tb* constantes. Assumimos que as válvulas *V1*, *V2*, e *V3* estão fechadas.
- *β1* significa a abertura da válvula *V2*.
- E3, E6 significa que o tanque *Ta* está esvaziando e o tanque *Tb* está enchendo.

- β2 significa o fechamento da válvula *V2*.
- E4, E8 significa que os níveis dos tanques *Ta* e *Tb* estão constantes.
- β2 significa o fechamento da válvula *V2*.
- β3 significa a abertura das válvulas *V1* e *V3*.
- E2, E7 significa que o tanque *Ta* está enchendo e que o tanque *Tb* está esvaziando.
- β4 significa o fechamento das válvulas *V1* e *V3*.
- E4, E8 significa que os níveis dos tanques *Ta* e *Tb* estão constantes.
- β1 significa a abertura da válvula *V2*.

Para obter o supervisor que irá restringir o comportamento do sistema devemos transformar a receita em transições de estado utilizando lógica temporal CTL. Uma vez especificado o comportamento desejável, o algoritmo SSS encontrará quais os eventos que devem ser controlados no sistema. A especificação em lógica temporal para o comportamento do sistema é mostrado na Tabela 5.1 conforme relações de transições descritas na seção 4.1.4 do Capítulo 4.

O resultado obtido da aplicação do algoritmo SSS sobre a especificação mostrada na Tabela 5.1, resulta em uma lista de eventos, relativas a cada transição de estado, que devem ser controladas. A Tabela 5.2 mostra os conjuntos dos eventos desejáveis e não desejáveis ao comportamento do sistema, observe que o supervisor poderá ser obtido a partir de qualquer um dos dois conjuntos. Neste caso optamos por utilizar o conjunto de eventos desejáveis para obter uma rede que será sincronizada a rede do sistema, com o objetivo de sincronizar a sua execução, restringindo o comportamento do sistema a especificação. A Figura 5.4 [BPF98], representa a rede de controle do sistema, se compararmos esta rede com o conjunto *Y* podemos constatar que as transições desta rede de Petri são as mesmas identificadas pelo algoritmo SSS. Vale salientar que o sistema de produção em lote apresentado acima é um sistema híbrido, composto por um componente

discreto mais um componente contínuo. Neste exemplo foi abordada apenas a componente discreta do sistema. Uma abordagem completa pode ser encontrada em [BP98].

Transição de Estado	**Fórmula CTL**
E4,E8 → β1	$AG(E4 \mid E8 \rightarrow AF(\bar{r}))$
β1 → E3,E6	$AG(\bar{r} \rightarrow AF(E3 \mid E6))$
E3,E6 → β2	$AG(E3 \,\&\, E6 \rightarrow AF(r))$
β2→ E4,E8	$AG(r \rightarrow AF(E4 \mid E8))$
E4,E8 → β3	$AG(E4 \,\&\, E8 \rightarrow AF(\bar{q} \mid \bar{s}))$
β3→ E2,E7	$AG(\bar{q} \,\&\, \bar{s} \rightarrow AF(E2 \mid E7))$
E2,E7 → β4	$AG(E2 \,\&\, E7 \rightarrow AF(q \mid s))$
β4→ E4,E8	$AG(q \,\&\, s \rightarrow AF(E4 \mid E8))$

Tabela 5.1: Especificação funcional (receita) para o sistema de transferência de fluídos.

Transição de Estado	**Eventos indesejáveis**	**Eventos desejáveis**
E4,E8 → β1	$\bar{Q}, \bar{S}$	$\bar{R}$
β1 → E3,E6	$\bar{Q}, \bar{S}, R$	*T86,T43*
E3,E6 → β2	$\bar{Q}, \bar{S}$	R
β2→ E4,E8	$\bar{Q}, \bar{S}, \bar{R}$	*T34,T68*
E4,E8 → β3	$\bar{R}$	$\bar{Q}, \bar{S}$
β3→ E2,E7	$Q, S, \bar{R}$	*T42,T87*
E2,E7 → β4	$\bar{R}$	Q, S
β4→ E4,E8	$\bar{Q}, \bar{S}, \bar{R}$	*T24,T78*

Tabela 5.2: Resultado obtido do algoritmo SSS sobre a especificação da Tabela 5.1

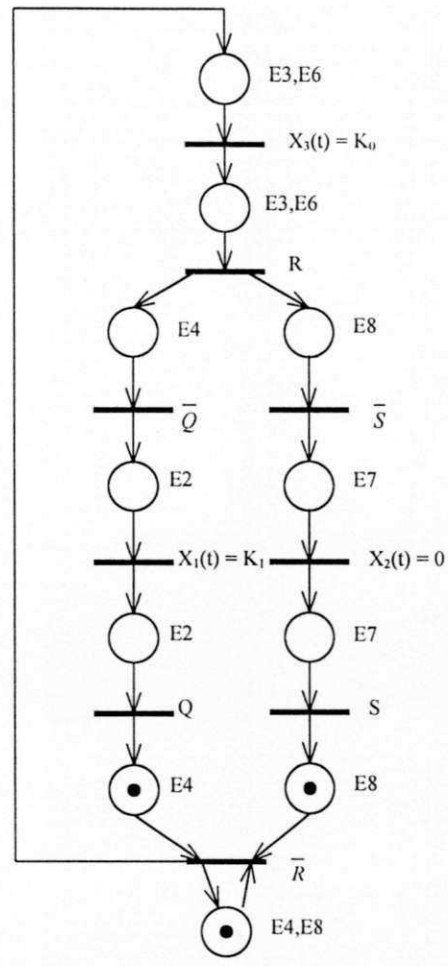


Figura 5.4: Rede de controle para a execução da receita *R* do SPL da Figura 6.1

De fato, com este resultado podemos constatar que para o sistema de transferência de fluídos executar a receita *R*, é suficiente controlar as válvulas, representadas pelos eventos da segunda coluna da Tabela 5.2. No entanto, a terceira coluna da Tabela 5.2 representa os eventos que devem ocorrer para que a receita R seja satisfeita. Na verdade estes são conjuntos complementares, mas que representam o mesmo controle, ou seja, representam a mesma informação de controle.

5.2 Um Sistema Flexível de Manufatura (SFM)

Um Sistema Flexível de Manufatura é definido como uma 8-upla (P, B, V, C, M, R, H, $F(h)$), formada por conjuntos finitos e não vazios, onde:

- $A = \{a_1, a_2, ..., a_n\}$ é um conjunto de produtos, que é um resultado das transformações que a matéria prima sofre desde a entradas do SFM até alcançar a saída;
- $B = \{b_1, b_2, ..., b_n\}$ é um conjunto de buffers, que são armazéns temporários de subprodutos;
- $V = \{ v_1, v_2, ..., v_n\}$ é um conjunto de elementos de transporte, que são veículos automatizados ou robôs utilizados para transporte de subprodutos entre os buffers do sistema;
- $C = \{c_1, c_2, ..., c_n\}$ é um conjunto de células, que são regiões dentro do SFM que pode conter uma ou mais máquinas;
- $X = \{x_1, x_2, ..., x_n\}$ é um conjunto de máquinas, o qual são máquinas que podem conter um ou mais recursos;
- $R = \{r_1, r_2, ..., r_n\}$ é um conjunto de recursos, que determina qual a atividade de processamento de matéria prima ou subproduto á maquina irá executar;
- $H = \{h_1, h_2, ..., h_n\}$ é um conjunto de trajetórias, que são as rotas de produção dentro do SFM que definem um ou mais produtos de acordo com a sequência de alocação, sem repetição, de recursos.
- $F(h)$ é uma função $F:H \rightarrow (\varnothing, A)$ definida da seguinte forma:

$$F(h) = \begin{cases} a_i \in A & se\ h \in V \\ \varnothing & de\ outra\ forma \end{cases}$$

Em que, G é o conjunto de todas as possíveis rotas fisicamente realizáveis do sistema.

O SFM ilustrado na Figura 5.5 é formado por quatro células, em que cada célula contém duas máquinas com um recurso cada. Para simplificar o diagrama, o sistema de transportes não é ilustrado. Portanto os componentes que formam o SFM são:

$A = \{a_1, a_2\}$

$B = \{be, bs, bec_1, bec_2, bsc_1, bsc_2, bem_1, ..., bem_4, bsm_1, ..., bsm_4\}$

$C = \{c_1, c_2\}$

$M = \{x_1, x_2, x_3, x_4\}$

$R = \{r_1, r_2, r_3, r_4\}$

$H = \{r_1, r_2, r_3, r_4, ..., (r_1 r_4), ..., (r_4 r_3), ..., (r_2 r_4 r_3), ..., (r_1 r_2 r_3 r_4)\}$

Observe que o sistema poderá produzir qualquer produto especificado pelas rotas definidas em H. O número de rotas possíveis para o sistema é definido por:

$$\# H = A_1^4 + A_2^4 + A_3^4 + A_4^4$$

Para este exemplo podemos concluir que o sistema em malha aberta é um gerador com uma linguagem H. Para que o SFM seja controlado de maneira a executar uma especificação desejada é necessário estabelecer um controle sobre o conjunto de rotas, de maneira que, apenas as rotas definidas pela especificação sejam conhecidas pelo controlador. Neste contexto, [BPF97] define V como o conjunto de rotas que devem ser reconhecidas pelo sistema controlado, enquanto, $\overline{V}$ é o conjunto de rotas que não resultam em nenhum produto. Portanto, o conjunto $H = V \cup \overline{V}$ define o conjunto de rota fisicamente realizáveis.

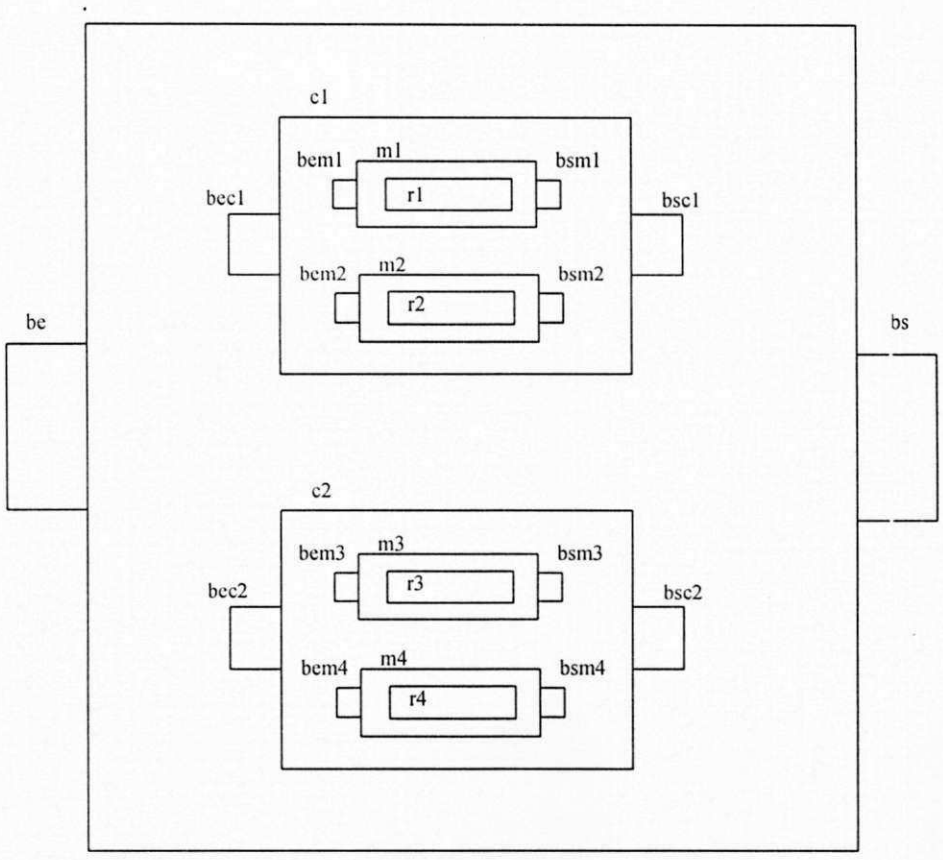


Figura 5.5: Layout para um SFM

Para o exemplo iremos considerar a seguinte especificação funcional:

$F(h_1) = F((r_1 r_3)) = p_3$
$F(h_2) = F((r_4 r_1)) = p_1$
$F(h_3) = F((r_1 r_2 r_3 r_4)) = p_2$
$F(h) = \varnothing$ *para qualquer outra trajetória* $h \in H$

Portanto a região de operação definida para o supervisor S é formada pelas sequências $(r_1 r_3)$, $(r_4 r_1)$,($r_1 r_2$ $r_3 r_4$). Na verdade, para cada produto p_i deverá existir um supervisor S. Desta forma, surge uma questão: como determinar qual produto será produzindo, uma vez depositada a matéria prima no buffer de entrada do sistema? Este

problema é solucionado pelo Sistema Alocador e Gerenciador de Recursos (SAGR) proposto por [BPF97], que é um módulo pertencente ao SFM que determina quais recursos deverão ser alocados e utilizados para uma dada ordem de produção. A Figura 5.6 ilustra a rede de Petri equivalente ao SFM apresentado na Figura 5.5. Observe que com esta rede de Petri é possível modelar qualquer rota definida em *H*. Neste modelo lugares e transições possuem o seguinte significado:

Lugares:

- *PE*: modela o buffer **be** de entrada do sistema,
- *PS*: modela o buffer **bs** de saída do sistema,
- *PEC*: modela os buffers **bec** de entrada das células,
- *PSC*: modela os buffers **bsc** de saída das células,
- *PEM*: modela os buffers **bem** de entrada das máquinas,
- *PSM*: modela os buffers **bsm** de saída das máquinas,
- *PA*: modela o estado *recurso em operação*,
- *PF*: modela o estado *recurso livre*,

Transições

- *TEC*: modela o transporte de subproduto do buffer de entrada (**be**) para os buffers (**bec**) de entrada das células,
- *TSC*: modela o transporte de subproduto do buffer de saídas das células (**bsc**) para os buffers (**bs**) de saída do sistema ,
- *TEM*: modela o transporte de subproduto do buffer de entrada das células (**bec**) para os buffers (**bem**) de entrada das máquinas,
- *TSM*: modela o transporte de subproduto do buffer de saídas (**bsm**) das máquinas para os buffers (**bsc**) de saída das células,
- *TPC*: modela o transporte de subproduto do buffer de saída das células (**bsc**) para os buffers (**bec**) de entrada das demais células,

O SFM apresentado na Figura 5.5 pode executar concorrentemente qualquer rota da especificação desejável. A existência dos buffers de entrada e saída para armazenamento de subprodutos determina que a rede não é segura. Portanto, para permitir a utilização do algoritmo SSS é indispensável a utilização do método de conversão de buffers para redes seguras mostradas no Capítulo 2. A rede de Petri segura equivalente para buffers de capacidade com duas fichas é ilustrada na Figura 5.8. Observe que o modelo equivalente é uma rede maior, em que para buffers de tamanho dois, o número de lugares e transições dobrou. O aumento da rede implica em um aumento do OBDD, no entanto, uma boa ordenação das variáveis (ver Capítulo 2) pode reduzir significativamente a representação do espaço de estados. É evidente que em sistema de manufatura a ordem de eventos é sempre bem definida o que possibilita uma boa ordenação das variáveis (lugares), garantindo uma representação compacta do OBDD.

Para ilustrar a aplicação do algoritmo SSS vamos obter o supervisor para a especificação funcional que resulta na produção do produto p_1, ou seja, habilitar apenas a rota r_4r_1. Como já discutido no Capítulo 4 é necessário definir a especificação funcional, correspondente ao comportamento desejável, em termos de transição de estados. A rota definida para o produto p_1 implica nas transições de estado ilustradas na Figura 5.7 e descritas em CTL na Tabela 5.3.

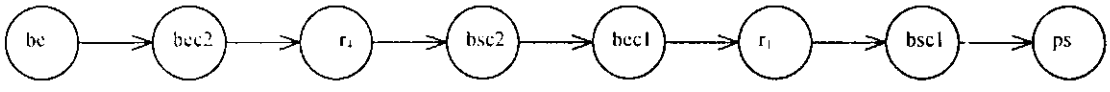


Figura 5.7: Transições de estado que especificam a rota para produção do produto a_1

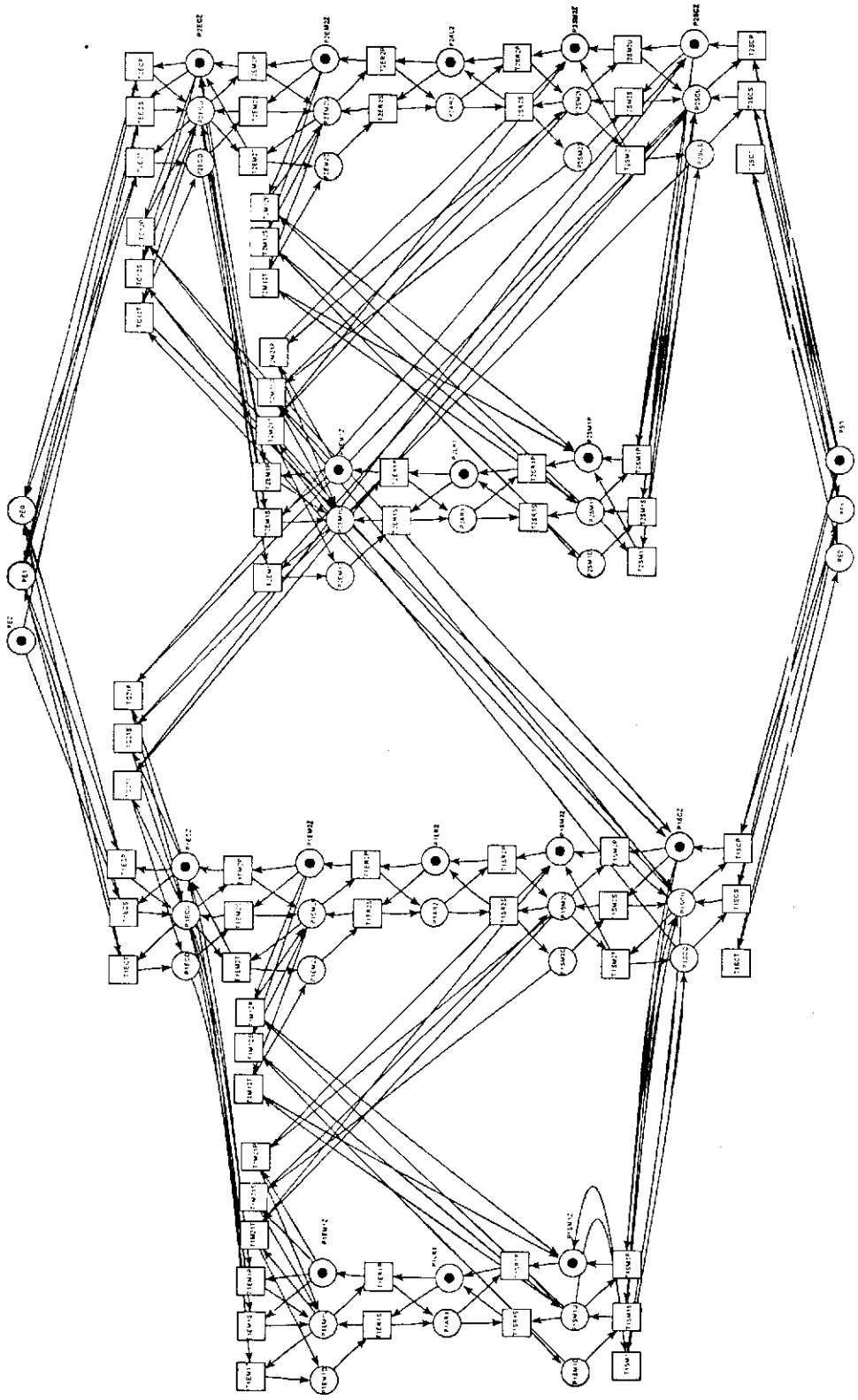


Figura 5.8: Rede de Petri para SFM com buffer seguro para duas fichas

Transição de Estado	Fórmula CTL
be → bec2	*AG (!PE0 → AF (!P2ECZ))*
bec2 → r_4	*AG (!P2ECZ → AF (P2AR2))*
r_4 → bsc2	*AG (P2AR2 → AF (!P2SCZ))*
bsc2→ bec1	*AG (!P2SCZ → AF (!P1ECZ))*
bec1 → r_1	*AG (!P1ECZ → AF (P1AR1))*
r_1→ bsc1	*AG (P1AR1 → AF (!P1SCZ))*
bsc1→ ps	*AG (!P1SCZ → AF (!PS0))*

Tabela 5.3: Especificação funcional (CTL) para o SFM.

Aplicando o algoritmo SSS à especificação da tabela 5.3 obtemos o conjunto $\overline{Y}$ com todas as transições da rede de Petri que não devem estar habilitadas para cada transição de estado. A especificação do supervisor S para o produto p_1 pode ser observada na Tabela 5.4. No Anexo 1 pode ser observado o *trace* completo da aplicação do algoritmo SSS para este exemplo.

Transição de Estado	Eventos indesejáveis
be → bec2	*T1ECS T1EM1P T2EM1P*
bec2 → r_4	*T2EM1P T1ECP T1ECS T1EM1P*
r_4 → bsc2	*T2M21P T1ECP T1EM1P*
bsc2→ bec1	*T2SCP*
bec1 → r_1	*T1EM2P T2ECP TEECS T2EM1P*
r_1→ bsc1	*T1M12P T1ECP T2ECP*
bsc1→ ps	*TC21P T2ECP T1ECS T2ECP T2EM2P T2ER2P T2SM2U*

Tabela 5.4: Especificação do supervisor S para o produto p_1.

Encontrar a melhor ordenação de variáveis para o OBDD é uma tarefa complicada, dependendo do número de lugares e transições que possui o modelo de rede de Petri. A ferramenta SMV, estudada no Capítulo 3, disponibiliza o recurso opcional de reordenação para variáveis. Na aplicação do algoritmo SSS para o exemplo da rede de Petri da Figura 5.6, o SMV foi empregado de três maneiras diferentes. No primeiro caso a síntese do supervisor foi realizada com o SMV sem utilizar a reordenação de variáveis, em que não foi possível realizar a verificação das transições de estado em um tempo inferior a 48 horas. Na segunda aplicação do SSS foi utilizado o opcional de busca da melhor ordenação de variáveis (`smv -f -reorder -o sfmorder.ord sfmbuff.smv | transsmv -n=sfmbuff.ll_net`), em que foi obtido um resultado, com relação ao primeiro, bastante satisfatório, correspondendo ao tempo total para a síntese completa do supervisor igual a 133.31 segundos. Na terceira etapa foi empregada a melhor ordenação encontrada na etapa anterior, eliminando-se a busca da melhor ordenação (`smv -f -i sfmorder.ord sfmbuff.smv | transsmv -n=sfmbuff.ll_net`), cujo o tempo de verificação de todas as transições de estado foi reduzido para 21.27 segundos, representando uma melhora de desempenho de 6.25 vezes.

Capítulo 6

Conclusões e Trabalhos Futuros

Nesta dissertação introduzimos uma abordagem para síntese de supervisores de sistemas a eventos discretos, utilizando a teoria de controle supervisório e as redes de Petri. O trabalho realizado foi desenvolvido para definir um algoritmo, que dado um sistema modelado por uma rede de Petri e sua especificação desejada em lógica temporal CTL, obtenha a especificação do supervisor que controla o sistema modelado pela rede de Petri.

O algoritmo de síntese obtido é baseado na verificação iterativa de transições de estados, expressas em lógica temporal, através da verificação simbólica de modelo com a ferramenta SMV (*Simbolic Model Verifier*). O uso desta ferramenta, que executa a verificação de modelo através da representação simbólica do espaço de estado (*OBDD – Diagrama de Decisão Binário Ordenado*) do modelo, possibilita sintetizar supervisores de sistemas reais, cujo espaço de estado ultrapassa 10^{20} estados.

Um outro resultado importante é que todo o processo de síntese ocorre com as ferramentas integradas PEP (*Programming Enviroment based Petri Nets*) e SMV, que são ferramentas já consolidadas no mundo acadêmico e industrial. No entanto, o processo de aplicação do algoritmo proposto nesta dissertação envolve trabalho manual, dificultando a sua aplicação para sistemas maiores do que apresentado nos exemplos do Capítulo 5. Desta forma uma proposta para trabalhos futuros é a implementação do algoritmo como uma

ferramenta integrada ao PEP e ao SMV, a qual executa o algoritmo iterativamente e automaticamente a cada resposta fornecida pelo SMV. Uma outra importante contribuição seria o trabalho de determinação de uma heurística para a ordenação de variáveis, que se aplique a sistemas de produção em lote ou sistemas flexíveis de manufatura.

Referências Bibliográficas

[Ager79] T. Agerwala. Putting Petri nets to work. *Computer*, 12(12), pages 85-94.

[Bar96] G. C. Barroso. *Uma nova Abordagem para a Síntese de Supervisores de Sistemas a Eventos Discretos*. Tese de Doutorado, Universidade federal da Paraíba – Campus II, Campina Grande, PB-BR, 1996.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Howang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proc. of Fifth Annual Symposium on Logic in Computer Science,* 1990.

[BDH92] E. Best, R. Devillers and J. G. Hall. The Box Calculus: a New Causal Algebra with Multi-Label Communication. *Advances in Petri Nets 92*, Springer *LNCS 609.*

[BFF+95a] E. Best, H. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. An M-Nets Semantics of $B(PN)^2$. *Proc.* of *STRICT'95*, Springer.

[BFF+95b] E. Best, H. Fleischhack, W. Fraczak, R. P. Hopkins, H. Klaudel, and E. Pelz. A Class Composable High Level Petri Nets.. *Proc. of* ATPN'95..

[BG] E. Best and B. Grahlmann. *PEP: Documentation and User Guid.* Universität Hildesheim. Avaliable toggether with the tool via: http://www.informatik.uni-hildesheim.de/~pep/.

[BH] E. Best and R. P. Hopkins. $B(PN)^2$ – a Basic Petri Net Programming Notatiion. *Proc. of PARLE,* Springer *LNCS* 694.

[BHG+93] S. Balemi, G. J. Hoffman, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid themal multiprocessor. *IEEE Transaction on Automatic Control*, 38(7):1040-1059, Julho 1993.

[BLNB95] K. Boel, L. Bem-Naoun, and V. Van Breusegem. On forbidden state problems for class of controlled Petri Nets. *IEEE Transactions on Automatic Control*, 40(10):1717-1731, october 1995.

[BLP95] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Transition enabling petri nets to Supervisory Control theory. In *Procedings of IEEE/ECLA/IFIP Iternational Conference on Architetures and Design Methods for Balanced Automation Systems*, pages 56-63, Vitoria, ES, BR, 1995.

[BLP96a] G. C. Barroso, A. M. N. Lima, and A. Perkusich. Petri nets with trasition enabling functions in the supervision of discrete event systems. XI Congresso Brasileiro de Automática, 1996.

[BPF98] T.C. Barros and A. Perkusich. Redes de petri: Um procedimento de modelagem aplicado aos sistemas de produção em lotes. In Anais do XII Congresso Brasileiro de Automática, pages 1377-1382, Uberlândia, Brasil, September 1998.

[BPF97] T.C. Barros, A. Perkusich, and J.C.A. de Figueiredo. A coloured petri net based approach for resource allocation and fault tolerance for flexible manufacturing systems. In Proc. of 2nd Workshop on Manufacturing and Petri Nets, 18th Int. Conference on Application and Theory of Petri Nets, pages 77-96, Toulouse,France, June 1997.

[Bra83] G. W. Brams. *Reseaux de Petri: Théoric et Pratique*, *Tomo II.* Masson S. A., Paris, FR, 1983.

[Bry86] R.E. Bryant. Graph-Based algorithms for boollean funciton manipulation. *IEEE Transactions on Computers*, C-35980:677-691, 1986.

[Bur84] J. P. Burgess. Basic tense logic. In D. Gadday and F. Guenthner editors, *Handbook of Philosophical Logic. Volume II: Extension of Classical Logic*, pages 89-134. D. Reidel, 1984.

[BW94] B. A. Brandin and W. M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transaction on Automatic Control*, 39(2):293-341, Fevereiro 1994.

[Car89] Carrol, J. *Theory of Finite Automata*. Prentice Hall, Engelwood Cliffs, NJ.

[CE81b] E. M. Clarke and E. A. Emerson. Synthesis of syncronization skeletons for branching time temporal logic. In Dexter Kozen, editor. *Logic of Programs: Workshop*. Volume 131 of *Lecture Notes in Computer Science*. Yorktown Heights. New York, May 1981. Springer-Verlag.

[DHPSV93] F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva and F. B. Vernadat. *Practice of Petri Nets in Manufactoring*. Chapman & Hall, 1993.

[ERV] J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Proc. of TACAS'96,* springer *LNCS 1055*.

[GD91] A. Giua anf F. DiCesare. Supervisory design using Petri nets. In *Procedings of 30th IEEE International Conference on Decision and Control*, pages 92-97, 1991.

[GD92] A. Giua anf F. DiCesare. On the existence of Petri net supervisors . In *Procedings of 31th IEEE International Conference on Decision and Control*, 1992.

[GD94a] A. Giua anf F. DiCesare. Blocking and controllability of Petri nets in supervisory control. *IEEE Transactions on Automatic Control,* 39(4):818-823, 1994.

[HK90] L. E. Holloway and B. H. Krogh. Sysntesis of feedback control logic for a class of controlled Petri nets. *IEEE Transactions on Automatic Control,* 35(5):514-523, 1990.

[IV88] K. Inan and P. Varaiya. Finitely recursive process models for discrete event systems. *IEEE Transaction Automatic Control,* 33(7):626-639, 1998.

[Jen92] K. Jensen. *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use*, volume 1 : Basic Concept. Springer-Verlag, 1992.

[Kum92] V. Kumar. Algorithms for Constraint-Specification Problems: A Survey. *Jornal: AI Magazine*. Volume 13, Pages 32-44, 1992.

[LW88a] Y. Li and W. M. Wonham. On supervisory control of real-time discret-event systems. *Information Sciences*, (44):173-198, 1998.

[LW88b] F. Lin and W. M. Wonham. On observabily of discrete event systems. *Information Sciences*, (44):173-198, 1988.

[MA98] J. O. Moody and P. J. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998

[McM92] K. L. McMillan. *The SMV system (DRAFT)*. Carnegie-Mellon University, February 1992.

[McM93] K. L. McMillan. *Simbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.

[Mur89] T. Murata. Petri net: Proprieties, analysis and applications. *Procedings of the IEEE*, 77(4):541-580, 1989.

[PC92] Y. E. Papelis and T. L. Casavant. Specification and analysis of parallel/distributed software and system by Petri net with trasition enabling functions. *IEEE Transactions on Software Enginieering*, 18(3):252-261, 1992.

[PdFC94] A. Perkusich, J. C. A. de Figueiredo, and S. K. Chang. Embedding fault-tolerant properties in the design of complex systems. *Journal of System and Software*, 25(2):23-37, 1994.

[Per94] A. Perkusich. *Análise de Sistemas Complexos Baseada na Decomposição de Sistemas de* G-Nets. Tese de Doutorado, Universidade Federal da Paraíba – Campus II, Campina Grande, PB – BR, 1994.

[Pet81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, N.J. USA, 1981.

[PX96] J. M. Proth and X. Xie. *Petri Nets: A tool for Design and Management of Manufacturing Systems*. John Wiley & Sons, 1996.

[Rei85] W.Reisig. *Petri Nets: Na Introduction*. Springer-Verlag, 1985

[RW87a] P. J. Ramadge and W. M. Wonham. Modular feedback logic discrete event systems. *SIAM Jornal of Control and Optimization,* 25(5):1202-1218, 1987.

[RW87b] P. J. G. Ramadge and W.M.Wonhan. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optmization,* 25(3):637-659, 1987.

[RW87c] P. J. G. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM. Joenal of Control and Optimization,* 25(1): 206-230, 1987.

[RW88] P. J. G.Ramadge and W.M.Wonhan. Modular supervisory control of discret event system. *Matheematics control, Signals and Systems,* 1(1):13-30, 1988.

[RW89] P. J. G.Ramadge and W.M.Wonhan. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81-97, 1989.

[SK92] R. S. Sreenivas and B. H. Krogh. On Petri net models of infinite state supervisors. *IEEE Transaction 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 language l. *IEEE Transaction on Automatic Control*, 38(4):659-662, 1993.

[Tsa93] E. Tsang. *Fioundations on Constraint Satisfaction.* Academic Press. 1993.

[VSS] T. Villa, G. Swamy and T. Shiple. *VIS User's Manual*.University of Berkeley California. Avaliable together with the VIS toll via *http//www-cad.eecs.berkeley.edu/respep/research/vis/.*

[Win97] G. Wimmel. *A BDD-based Model Checker for the PEP Tool*. Major thesis, University of Newcastle, USA, 1997.

[ZC94] R. M. Ziller and J. E. R. Cury. On the supremal L-controllable sublanguage of a non prefix-closed language. In *Proceedings of 10th Congresso Brasileiro de Automática*, pages 260-265, Rio de Janeiro, RJ, BR, 1994.

[ZD93] M. C. Zhou and F. DiCesare. *Petri Net Syntesis for Discrete Event Control of Manufacturing Systems.* Kluwer Academic Publishes, Boston, USA, 1993.

Apêndice A

Trace da aplicação do algoritmo SSS ao exemplo do Capítulo 5. Um Sistema Flexível de Manufatura (SFM)

```
-- specification AG (!PE0 -> !EF (!P2ECZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE:
T2ECS

-- specification AG (!PE0 -> AF (!P2ECZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE1:
T1ECS,T1ECT,T1EM1S
_LOOP1:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!PE0 -> AF (!P2ECZ | P1ECU)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE2:
T1ECS,T1EM1P
_LOOP2:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!PE0 -> AF (!P2ECZ | P1ECU | P1EM1U)... is false
-- as demonstrated by the following execution sequence
_SEQUENCE3:
TEECS,T2EM1P
_LOOP3:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!PE0 -> AF (!P2ECZ | P1ECU | P1EM1U ... is true

-- specification AG (!P2ECZ -> !EF P2AR2) is false
-- as demonstrated by the following execution sequence
_SEQUENCE5:
TEECS,T2EM2P,T2ER2P
```

```
-- specification AG (!P2ECZ -> AF P2AR2) is false
-- as demonstrated by the following execution sequence
_SEQUENCE6:
TEECS
_LOOP6:
T2EM1P,T2ER1P,T2SR1P,T2SM1P,TC21P,T1EM2P,T1ER2P,T1SR2P,T1SM2P,TC12P

-- specification AG (!P2ECZ -> AF (P2AR2 | P2EM1U)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE7:
TEECS,T1ECP,T2EM2P,T1EM1P
_LOOP7:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2ECZ -> AF (P2AR2 | P2EM1U | P1ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE8:
T1ECS,T2ECP,T1EM1P
_LOOP8:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2ECZ -> AF (P2AR2 | P2EM1U | P1ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE9:
T1ECS,T2ECP,T1EM1P
_LOOP9:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2ECZ -> AF (P2AR2 | P2EM1U | P1ECU... is true

-- specification AG (P2AR2 -> !EF (!P2SCZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE11:
TEECS,T2EM2P,T2ER2P,T2SR2P,T2SM2U

-- specification AG (P2AR2 -> AF (!P2SCZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE12:
TEECS,T2EM2P,T2ER2P
_LOOP12:
T2SR2P,T2M21P,T2ER1P,T2SR1P,T2M12P,T2ER2P

-- specification AG (P2AR2 -> AF (!P2SCZ | P2EM1U)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE13:
TEECS,T2EM2P,T2ER2P,T1ECP,T2SR2P,T1EM1P
_LOOP13:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (P2AR2 -> AF (!P2SCZ | P2EM1U | P1ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE14:
T1ECS,T2ECP,T2EM2P,T2ER2P,T1EM1P
_LOOP14:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (P2AR2 -> AF (!P2SCZ | P2EM1U | P1ECU... is true

-- specification AG (!P2SCZ -> !EF (!P1ECZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE16:
TEECS,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T1ECP
```

```
-- specification AG (!P2SCZ -> AF (!P1ECZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE17:
TEECS,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T2ECP,T2EM1P
_LOOP17:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P2SCZ -> AF (!P1ECZ | P2SCU)) is true

-- specification AG (!P1ECZ -> !EF P1AR1) is false
-- as demonstrated by the following execution sequence
_SEQUENCE19:
T1ECS,T1EM1P,T1ER1P

-- specification AG (!P1ECZ -> AF P1AR1) is false
-- as demonstrated by the following execution sequence
_SEQUENCE20:
T1ECS
_LOOP20:
T1EM2P,T1ER2P,T1SR2P,T1SM2P,TC12P,T2EM2P,T2ER2P,T2SR2P,T2SM2U,TC21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE21:
T1ECS,T2ECP,T2EM1P
_LOOP21:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U | P2ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE22:
T1ECS,T2ECP,T2EM1P
_LOOP22:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U | P2ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE23:
T1ECS,T2ECP,T2EM1P
_LOOP23:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U | P2ECU... is true

-- specification AG (P1AR1 -> !EF (!P1SCZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE25:
T1ECS,T1EM1P,T1ER1P,T1SR1P,T1SM1P

-- specification AG (P1AR1 -> AF (!P2SCZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE26:
T1ECS,T1EM1P,T1ER1P
_LOOP26:
T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P,T1ER1P

-- specification AG (P1AR1 -> AF (!P2SCZ | P1EM2U)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE27:
T1ECS,T1EM1P,T1ER1P,T1ECP,T1SR1P,T1SM1P,TC12P,T2EM1P
_LOOP27:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (P1AR1 -> AF (!P2SCZ | P1EM2U | P1ECU... is false
```

```
-- specification AG (!P2SCZ -> AF (!P1ECZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE17:
TEECS,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T2ECP,T2EM1P
_LOOP17:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P2SCZ -> AF (!P1ECZ | P2SCU)) is true

-- specification AG (!P1ECZ -> !EF P1AR1) is false
-- as demonstrated by the following execution sequence
_SEQUENCE19:
T1ECS,T1EM1P,T1ER1P

-- specification AG (!P1ECZ -> AF P1AR1) is false
-- as demonstrated by the following execution sequence
_SEQUENCE20:
T1ECS
_LOOP20:
T1EM2P,T1ER2P,T1SR2P,T1SM2P,TC12P,T2EM2P,T2ER2P,T2SR2P,T2SM2U,TC21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE21:
T1ECS,T2ECP,T2EM1P
_LOOP21:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U | P2ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE22:
T1ECS,T2ECP,T2EM1P
_LOOP22:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U | P2ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE23:
T1ECS,T2ECP,T2EM1P
_LOOP23:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P1ECZ -> AF (P1AR1 | P1EM2U | P2ECU... is true

-- specification AG (P1AR1 -> !EF (!P1SCZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE25:
T1ECS,T1EM1P,T1ER1P,T1SR1P,T1SM1P

-- specification AG (P1AR1 -> AF (!P2SCZ)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE26:
T1ECS,T1EM1P,T1ER1P
_LOOP26:
T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P,T1ER1P

-- specification AG (P1AR1 -> AF (!P2SCZ | P1EM2U)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE27:
T1ECS,T1EM1P,T1ER1P,T1ECP,T1SR1P,T1SM1P,TC12P,T2EM1P
_LOOP27:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (P1AR1 -> AF (!P2SCZ | P1EM2U | P1ECU... is false
```

```
-- as demonstrated by the following execution sequence
_SEQUENCE28:
T1ECS,T1EM1P,T1ER1P,T2ECP,T2EM1P
_LOOP28:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (P1AR1 -> AF (!P2SCZ | P1EM2U | P1ECU... is false
-- as demonstrated by the following execution sequence
_SEQUENCE29:
T1ECS,T2ECP,T2EM1P,T1EM1P,T1ER1P
_LOOP29:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (P1AR1 -> AF (!P2SCZ | P1EM2U | P1ECU... is true

-- specification AG (!P2SCZ -> !EF (!PS0)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE31:
TEECS,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T2SCP

-- specification AG (!P2SCZ -> AF (!PS0)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE32:
TEECS,T2EM2P,T2ER2P,T2SR2P,T2SM2U
_LOOP32:
TC21P,T1EM2P,T1ER2P,T1SR2P,T1SM2P,TC12P,T2EM2P,T2ER2P,T2SR2P,T2SM2U

-- specification AG (!P2SCZ -> AF (!PS0 | P1ECU)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE33:
TEECS,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T2ECP,T2EM1P
_LOOP33:
T2ER1P,T2SR1P,T2M12P,T2ER2P,T2SR2P,T2M21P

-- specification AG (!P2SCZ -> AF (!PS0 | P1ECU | P2ECU)) is false
-- as demonstrated by the following execution sequence
_SEQUENCE34:
T1ECS,T2ECP,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T1EM1P
_LOOP34:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2SCZ -> AF (!PS0 | P1ECU | P2ECU |... is false
-- as demonstrated by the following execution sequence
_SEQUENCE35:
T1ECS,T2ECP,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T1EM1P
_LOOP35:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2SCZ -> AF (!PS0 | P1ECU | P2ECU |... is false
-- as demonstrated by the following execution sequence
_SEQUENCE36:
T1ECS,T2ECP,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T1EM1P
_LOOP36:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2SCZ -> AF (!PS0 | P1ECU | P2ECU |... is false
-- as demonstrated by the following execution sequence
_SEQUENCE37:
T1ECS,T2ECP,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T1EM1P
_LOOP37:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2SCZ -> AF (!PS0 | P1ECU | P2ECU |... is false
-- as demonstrated by the following execution sequence
```

```
_SEQUENCE38:
T1ECS,T2ECP,T2EM2P,T2ER2P,T2SR2P,T2SM2U,T1EM1P
_LOOP38:
T1ER1P,T1SR1P,T1M12P,T1ER2P,T1SR2P,T1M21P

-- specification AG (!P2SCZ -> AF (!PS0 | P1ECU | P2ECU |... is true

resources used:
user time: 25.43 s, system time: 0.1 s
BDD nodes allocated: 18415
Bytes allocated: 2097152
BDD nodes representing transition relation: 4172 + 1
```