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

Um Sistema a Base de Regras para Simulação de Redes de Petri com Marcação Imprecisa

Dissertação submetida à Universidade Federal de Santa Catarina
como requisito parcial à obtenção do grau de

Mestre em Engenharia Elétrica

por

Luciano Lores Caimi

Florianópolis, 24 de abril de 1998

Um Sistema a Base de Regras para Simulação de Redes de Petri com Marcação Imprecisa

Luciano Lores Caimi

Esta dissertação foi julgada adequada para a obtenção do título de **Mestre em Engenharia** na especialidade **Engenharia Elétrica**, área de concentração **Controle, Automação e Informática Industrial**, e aprovada em sua forma final pelo curso de Pós-Graduação.

Florianópolis, 24 de abril de 1998.

Profa. Dra. Janette Cardoso
orientadora

Prof. Dr. Guilherme Bittencourt
co-orientador

Prof. Dr. Adroaldo Raizer
coordenador do curso de Pós-Graduação em Engenharia Elétrica
da Universidade Federal de Santa Catarina.

Banca Examinadora

Profa. Dra. Janette Cardoso
DAS - UFSC

Prof. Dr. Guilherme Bittencourt
DAS - UFSC

Profa. Dra. Sandra Sandri
INPE

Prof. Dr. Jorge Muniz Barreto
INE - UFSC

Prof. Dr. José Eduardo Cury
DAS - UFSC

À minha mãe Lourdes e ao meu falecido pai Lores,

às minhas irmãs, Cláudia, Flávia e Ana,

ao meu "irmão de coração", Dário,

à Carin, minha amada...

Agradecimentos

Inicialmente quero agradecer a minha mãe, Maria de Lourdes, pelo seu incrível espírito de luta. Mãe, com certeza foi graças a você que esta conquista foi possível.

Agradeço aos meus orientadores, profª. Janette Cardoso e prof. Guilherme Bittencourt, pelo desafio que me propuseram e pela confiança depositada no meu trabalho. Obrigado pela presteza e dedicação na orientação.

À todos os professores, analistas, secretárias e demais funcionários do LCMI/DAS, tanto pelo conhecimento repassado como pelas amizades celebradas.

À UFSC e a CAPES pelo suporte material e financeiro.

Aos amigos e colegas do LCMI, que proporcionaram um convívio agradável no trabalho e fora dele, em especial ao Adilson Guelfi, Jézer Pedrosa e Luciano Bertini, verdadeiros irmãos emprestados. Aos amigos do GPEB, da Computação, da Mecânica e do Francês pelo convívio e amizade.

Sou muito grato a minha noiva Carin ♡, que soube suportar com carinho e compreensão a distância física em que nos encontrávamos e ainda me incentivou e me deu forças para continuar em todos os momentos.

E, a todos que de uma forma ou de outra colaboraram para elaboração deste trabalho.

Sumário

Lista de Figuras

Resumo

Este trabalho apresenta um simulador de redes de Petri com marcação imprecisa baseado em sistemas de regras de produção, do tipo sistema especialista. O sistema especialista que simula a rede de Petri utiliza os quadros como método de representação de conhecimento e adota a estratégia progressiva de controle de inferência. Na ferramenta proposta, a rede de Petri com Marcação Imprecisa é representada através de uma base de conhecimento e de quatro conjuntos de regras. A base de conhecimento contém a descrição das classes de objetos e dos lugares da rede de Petri a ser simulada. Os quatro conjuntos de regras fornecem o suporte para as seguintes funcionalidades: i) resolução de conflito de fichas, ii) disparo normal de transições, iii) criação da incerteza e, iv) propagação da incerteza. A ferramenta inclui ainda um tradutor de uma linguagem formal de descrição de redes de Petri para os fatos e regras que formam a base de conhecimento do sistema especialista.

Abstract

This work presents a Petri net simulator based on production rule systems, i.e., expert systems. The expert systems that simulate Petri nets use the frame knowledge representation method and the forward chaining strategy. In the proposed tool, a Petri net with Imprecise Marking is represented through a knowledge base and four rule sets. The knowledge base contains descriptions of the classes of objects and places of the simulated Petri net. The four rule bases provide support for the following functionalities: (i) conflict resolution of tokens, (ii) normal firing of transitions, (iii) uncertainty creation and (iv) uncertainty propagation. The tool includes also a translator from a Petri net formal description language to facts and rules associated with an expert system knowledge base.

Capítulo 1

Introdução

1.1 Considerações Preliminares

A origem da rede de Petri é a tese intitulada *Comunicação com autômatos*, defendida por Carl Adam Petri [Pet62] em 1962 na Universidade de Darmstadt, Alemanha. Ela foi proposta para modelar a comunicação entre autômatos, utilizados, na época, para representar sistemas a eventos discretos, e constitui uma ferramenta gráfica e matemática que se adapta bem a um grande número de aplicações em que a noção de eventos e evoluções simultâneas são importantes.

Desde então as redes de Petri são especialmente utilizadas para modelar Sistemas a Eventos Discretos (SEDs), uma classe de sistemas em que as variáveis de estado variam bruscamente em instantes determinados, e os valores das variáveis nos estados seguintes podem ser calculados diretamente a partir dos valores precedentes sem ter que levar em conta o tempo entre estes dois instantes.

Existem dois aspectos importantes nos sistemas a eventos discretos: sua evolução no tempo e a incerteza das informações com as quais eles trabalham. Por um lado, trabalhos de representação dos aspectos evolutivos raramente têm sido desenvolvidos levando em conta a incerteza e a imperfeição da descrição do mundo e sua evolução. Por outro lado, na maior parte do tempo, os aspectos dinâmicos da informação não são considerados nas pesquisas que tratam com aspectos de incerteza. As duas questões são multidisciplinares, envolvendo aspectos de Engenharia de Software, Inteligência Artificial, Análise de Sistemas, Teoria de Controle e Banco de Dados.

Os aspectos dinâmicos dos SEDs são capturados de maneira formal e natural pelas redes de Petri, permitindo a representação de paralelismo verdadeiro, concorrência, condições de precedência, conflito e, principalmente a noção de recurso, representada por fichas nos lugares da rede. A análise das propriedades gerais do modelo, das propriedades específicas e a avaliação de desempenho constituem um importante auxílio nas fases de especificação e projeto de um SED.

Os aspectos de incerteza são tratados pela teoria de conjuntos nebulosos introduzida por Zadeh em 1965 e pela teoria de possibilidades introduzida por Zadeh em 1978 [Zad78]. A teoria de possibilidades [DP88a] [DP88b] é um método não probabilista para a descrição formal de incertezas sobre eventos. Ela é um modo de dizer o quanto a ocorrência de um evento é possível, mesmo desconhecendo a probabilidade da sua ocorrência.

A rede de Petri a Objetos [SB85] é um tipo de rede de alto nível que permite uma representação de maior abstração. Sua particularidade mais relevante é que as fichas são n-uplas de instâncias únicas de objetos, ou seja, em sua representação cada ficha possui uma identificação.

No modelo de rede de Petri a Objetos, em uma situação normal, cada objeto do sistema possui uma localização, o que se traduz na ficha em um lugar da rede. Quando acontece uma falha pode-se não saber a localização precisa do objeto (físico) no sistema. A rede de Petri com Marcação Imprecisa [RV89], [CVD91] trata esta situação, associando à ficha correspondente a este objeto, um subconjunto do conjunto de lugares da rede, correspondente aos locais do sistema físico onde o objeto pode estar localizado. Existe a certeza da existência do objeto, mas sua localização é imprecisa.

Desta forma, a rede de Petri com Marcação Imprecisa combina o modelo de rede de Petri a Objetos com a teoria de conjuntos nebulosos para representar a imprecisão das informações provenientes do sistema que está sendo modelado, possibilitando flexibilizar a modelagem e permitindo trabalhar com exceções.

1.2 Objetivos

Este trabalho faz parte de um projeto desenvolvido junto ao DAS (Departamento de Automação e Sistemas) de construção de um ambiente de representação, simulação, análise e supervisão de sistemas a eventos discretos, baseado em técnicas de Inteligência

Artificial. Como objetivo geral, propõe-se estender este ambiente desenvolvendo uma ferramenta que utiliza o modelo de redes de Petri.

Especificamente, o objetivo principal do trabalho é desenvolver um simulador para redes de Petri com Marcação Imprecisa, combinando redes de Petri com Marcação Imprecisa e o formalismo de regras de produção.

No simulador desenvolvido, o conhecimento sobre o sistema modelado é armazenado de forma declarativa, possibilitando o acesso ao estado da simulação (e assim às características do sistema modelado) em todos os ciclos de simulação. Isto possibilitará a construção de um módulo de análise e gerenciamento do sistema modelado, que explora este conhecimento armazenado.

1.3 Justificativas

Um primeira justificativa para a utilização do modelo de rede de Petri é o fato de que este formalismo pode ser utilizado em diversas fases do ciclo de vida de um sistema (especificação, modelagem, validação e implementação). Outro aspecto é que o modelo de rede de Petri consegue expressar de forma natural as relações de causalidade existentes em Sistemas a Eventos Discretos, tais como, as ações seqüenciais, a concorrência, o paralelismo verdadeiro, o sincronismo, etc. Além disso, a noção de recurso, necessária neste tipo de sistema, é representada de forma intrínseca pelo formalismo.

Por sua vez, o modelo de rede de Petri a Objetos possibilita o mapeamento direto das classes a partir dos elementos que fazem parte dos processos, tais como, máquinas, ferramentas e peças, facilitando a especificação de tais sistemas [RV89]. Há ainda a flexibilidade oferecida pelo modelo de rede de Petri com Marcação Imprecisa em que as exceções, falhas e possíveis interferências humanas, podem ser interpretadas de forma correta através da noção de marcação imprecisa.

As razões para implementar o simulador utilizando a abordagem de sistemas de regras de produção deve-se ao fato que estes: a) facilitam a integração dos níveis mais baixos da descrição dos sistemas de manufatura (como a supervisão), com níveis mais altos (como o gerenciamento); b) expressam o conhecimento sobre a rede de Petri simulada de forma declarativa (ao invés de armazenar de forma escondida em um procedimento imperativo - como num programa em C, por exemplo). Isto permite a implementação de outras funcionalidades que exploram este conhecimento (que podem

ser implementadas através de outros sistemas de produção ou mesmo através de programas convencionais). Além disso, utilizou-se uma ferramenta desenvolvida no próprio grupo de pesquisa, no caso o FASE, que é um arcabouço para a construção de sistemas especialistas. Com isso pretende-se avançar o trabalho existente na construção de um ambiente de representação de sistemas de manufatura.

1.4 Metodologia

A metodologia proposta para a simulação do modelo de redes de Petri a Objetos com Marcação Imprecisa utiliza um sistema baseado em regras de produção. Para isto, cada transição da rede de Petri é associada a um conjunto de regras. Aproximadamente, os lugares de entrada correspondem as pré-condições das regras e os lugares de saída as pós-condições.

Para o desenvolvimento do simulador propõe-se uma metodologia constituída das seguintes etapas: (*i*) utilizar uma linguagem formal para representar a rede de Petri a Objetos a ser simulada; (*ii*) transformar esta rede de Petri descrita formalmente em uma base de fatos e em um conjunto de regras de produção de maneira a definir um sistema especialista e, (*iii*) oferecer uma interface ao usuário para a execução do sistema especialista, permitindo tanto um controle completo por parte do usuário, como a simulação automática, e até a obtenção das características da rede a qualquer instante da simulação.

A figura 1.1 ilustra, com base na metodologia proposta, os processos pelos quais uma determinada rede deverá passar para que a simulação se inicie. O processo de criação do sistema especialista que fará a simulação da rede especificada é realizado de forma automática. Assim, a rede é especificada em um arquivo utilizando a sintaxe da linguagem de entrada e o sistema especialista que fará a simulação é gerado automaticamente.

As ferramentas utilizadas para o desenvolvimento do simulador, seguindo a metodologia descrita acima são: o LEX-YACC - ferramentas que trabalham de forma integrada proporcionando análise léxica e sintática da especificação a partir da linguagem de entrada; o sistema FASE - ferramenta que constitui um arcabouço para construção de sistemas especialistas; linguagem de programação Common Lisp.

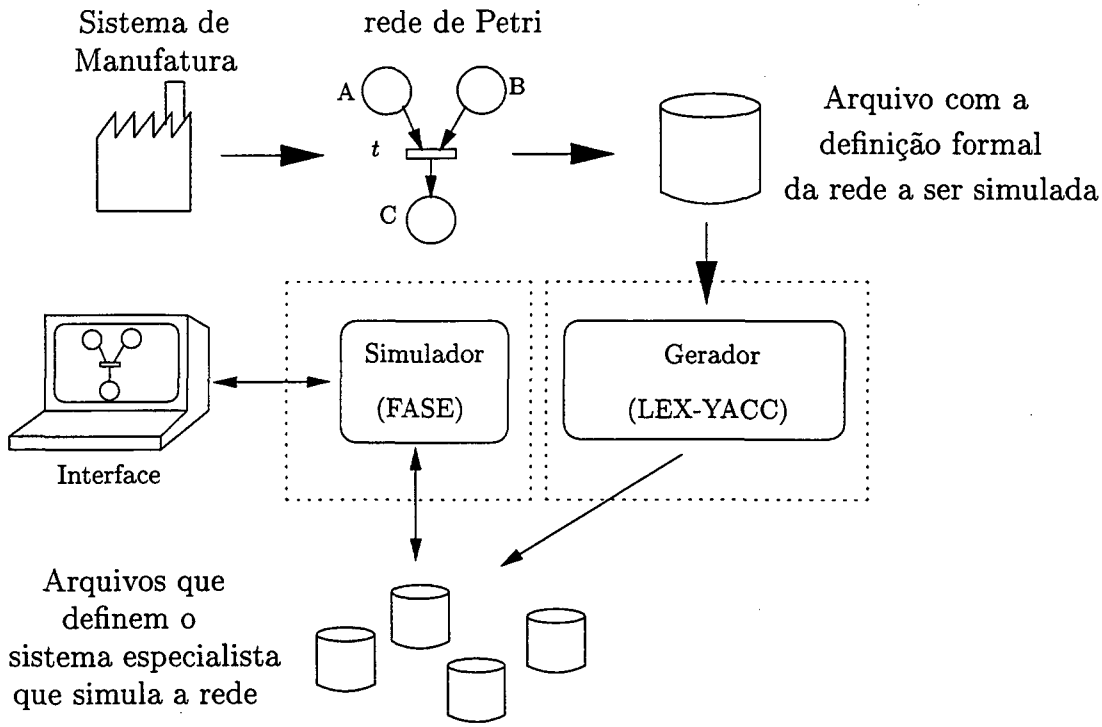


Figura 1.1: Visão geral da metodologia

1.5 Organização do Trabalho

No capítulo 2, é apresentado o modelo da rede de Petri. Inicialmente são discutidas suas principais características. A seguir algumas definições e conceitos da rede de Petri são apresentados de maneira informal. Na seqüência é feita a apresentação formal do modelo. Após isto, o modelo de rede de Petri a Objetos é colocado de forma simplificada e finalmente, o modelo de rede de Petri Nebulosa (no caso rede de Petri com Marcação Imprecisa), utilizado no trabalho, é exposto.

Os sistemas de regras de produção e os sistemas especialistas são apresentados no capítulo 3. No item referente aos sistemas de produção, é apresentada sua definição, os elementos que o compõem e as principais vantagens e desvantagens dos mesmos. No item que trata dos sistemas especialistas são apresentadas suas funcionalidades e elementos dos quais o mesmo é composto. Para finalizar o capítulo é apresentado o sistema FASE, um arcabouço para construção de sistemas especialistas, que fornece a base computacional para o desenvolvimento do simulador.

O capítulo seguinte apresenta de forma mais detalhada a metodologia utilizada na

construção do simulador. As diferenças e semelhanças entre um sistema a base de regras e as redes de Petri são discutidas. A seguir é apresentado o mapeamento a ser realizado para possibilitar a simulação de redes de Petri a Objetos com Marcação Imprecisa utilizando um sistema a base de regras. Cada um dos problemas encontrados (resultantes do mapeamento) são expostos e as soluções obtidas para os mesmos são apresentadas. Ao final do capítulo, a estrutura geral do simulador é apresentada.

A implementação do simulador é apresentada no capítulo 5. A linguagem de entrada desenvolvida é exposta. A seguir a forma como foram representadas as classes de objetos e os lugares da rede de Petri a Objetos é apresentada juntamente com alguns exemplos. Na seqüência cada um dos conjuntos de regras necessários para implementação do simulador de redes de Petri com Marcação Imprecisa é discutido. O capítulo 6 apresenta as conclusões e perspectivas do trabalho.

Para finalizar três apêndices são apresentados. No apêndice A são apresentados alguns conceitos e definições do modelo de redes de Petri ordinárias. No apêndice B são apresentados os arquivos de especificação de uma célula de usinagem, utilizada para auxiliar o entendimento do modelo de rede de Petri a Objetos. São apresentados também os arquivos gerados pela ferramenta para a simulação desta rede. O apêndice C, de maneira análoga ao apêndice anterior, apresenta os arquivos de especificação e os arquivos gerados pela ferramenta para a simulação de um sistema de Veículos Auto-Guiados (AGVs), utilizado para exemplificar o modelo de rede de Petri com Marcação Imprecisa.

Capítulo 2

Redes de Petri

Neste capítulo é apresentado o modelo de rede de Petri. Inicialmente o modelo é apresentado de maneira informal, a seguir sua definição formal é apresentada e, na seqüência, são introduzidas duas extensões ao modelo: a rede de Petri a Objetos e a rede de Petri com Marcação Imprecisa.

Os sistemas flexíveis de manufatura apresentam diversos tipos de relações de causalidade, requerendo formalismos que comportem a descrição destas relações com a máxima fidelidade possível. Para executar esta tarefa diversos formalismos de especificação são utilizados, tais como, máquinas de estado finitas, álgebra de processos, tipos abstratos de dados, entre outros. Entretanto, devido às características próprias dos sistemas de manufatura, como por exemplo a necessidade de representação de paralelismo verdadeiro, muitas vezes estes formalismos se mostram inadequados.

Por outro lado, o modelo de rede de Petri [Pet62] é um formalismo de descrição que apresenta algumas características que se mostram adequadas para a representação dos sistemas de manufatura: a expressividade, a abstração e o formalismo matemático.

Em relação à expressividade a rede Petri permite descrever com precisão as relações de causalidade como a seqüência, a concorrência ou o paralelismo, o conflito, e a sincronização. Uma restrição em relação à expressividade do modelo de rede de Petri diz respeito à representação de dados. O modelo clássico não possibilita este tipo de representação, sendo necessário a introdução de outras estruturas que o estendam.

O modelo clássico oferece um bom nível de abstração, oferecendo formas de representação abrangentes e uma independência entre a especificação de determinado

sistema e sua implementação. Ainda quanto às características da rede de Petri, o formalismo matemático no qual está baseada permite a análise de propriedades específicas (comportamento) e propriedades gerais da especificação (estrutura).

Desta forma, as características apresentadas acima fazem da rede de Petri um formalismo adequado para a especificação de sistemas flexíveis de manufatura. Mais do que isto, será visto posteriormente que a tarefa de supervisão destes sistemas também é realizada de forma apropriada pelo modelo.

2.1 Rede de Petri Ordinária

Inicialmente esta seção apresenta uma descrição informal da rede de Petri, introduzindo os elementos que a constituem e o seu comportamento dinâmico. Após isto, a representação formal do modelo de rede de Petri ordinária é apresentada.

2.1.1 Elementos básicos e comportamento dinâmico

Conforme [CV97], a rede de Petri é um tipo particular de grafo orientado, constituído de três elementos básicos:

- **Lugar** (representado por um círculo): pode ser interpretado como uma condição, um estado parcial, uma espera, um procedimento, um conjunto de recursos, um estoque, uma posição geográfica num sistema de transporte, etc. Em geral, todo lugar tem um predicado associado, por exemplo *máquina livre, peça em espera* (figura 2.1);
- **Transição** (representada por uma barra ou retângulo): é associada a um evento [1] que ocorre no sistema, por exemplo o evento *iniciar a operação*, representado pela transição t na figura 2.1.;
- **Ficha** (representado por um ponto num lugar): é um indicador significando que a condição associada ao lugar é verificada. Pode representar um objeto (recurso ou peça) numa certa posição geográfica ou num determinado estado. Por exemplo, uma ficha no lugar *máquina livre* indica que a máquina está livre (predicado

[1] Entende-se por evento os instantes de observação e de mudança de estado do sistema

verdadeiro). Se não existe ficha no lugar significa que o predicado é falso, o que no caso anterior, significa que a máquina não está livre. Se no lugar *peça em espera* houvesse três fichas, estaria sendo indicado que existem três peças em espera.

As interpretações destes elementos podem ser bastante variadas, na medida em que podem ser utilizadas para descrever entidades abstratas como condições ou estados ou entidades físicas como depósitos ou peças.

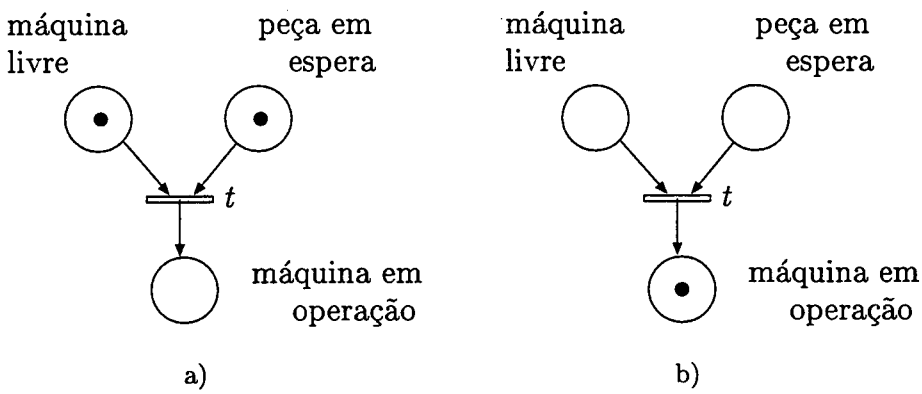


Figura 2.1: Os elementos da rede de Petri

O estado do sistema é dado pela repartição de fichas nos lugares da rede de Petri, cada lugar representando um estado parcial do sistema. A cada evento que ocorre no sistema, é associada uma transição no modelo de rede de Petri. A ocorrência de um evento no sistema (que faz com que este passe do estado atual ao próximo estado) é representada no modelo pelo *disparo* da transição ao qual este está associado.

O disparo de uma transição consiste em dois passos:

- retirar as fichas dos lugares de entrada, indicando que esta condição não é mais verdadeira após a ocorrência do evento;
- depositar fichas em cada lugar de saída, indicando que estas atividades estarão, após a ocorrência do evento, sendo executadas.

Por exemplo, a ocorrência do evento *iniciar a operação*, associado à transição t (figura 2.1.a), só pode acontecer se houver (ao menos) uma ficha no lugar *máquina livre* e (ao menos) uma ficha no lugar *peça em espera*. A ocorrência do evento *iniciar a operação* no sistema é representado na rede de Petri pelo tiro (ou disparo) da transição

t: é retirada uma ficha do lugar *máquina livre* e uma ficha do lugar *peça em espera*, e é colocada uma ficha no lugar *máquina em operação* (figura 2.1.b).

O disparo de t corresponde à ocorrência de um evento e no sistema real, que o faz passar de um estado atual E_i ao próximo estado E_{i+1}. O estado E_i é representado na rede pela distribuição de fichas nos lugares, chamada marcação M_i. Do mesmo modo que o sistema só atingirá o estado E_{i+1} após a ocorrência do evento e se estiver no estado E_i, assim também a transição t só será disparada se a marcação for M_i (marcação em que, em particular, os lugares de entrada de t estão marcados). A marcação M_{i+1}, correspondente ao estado E_{i+1}, será atingida após o disparo da transição t.

Como colocado anteriormente, o desaparecimento das fichas nos lugares de entrada de t indica que as condições ou predicados associados àqueles lugares não são mais verdadeiros, e o surgimento de fichas nos lugares de saída indica que os predicados associados a estes lugares são verdadeiros. O comportamento dinâmico do sistema é, assim, traduzido pelo comportamento da rede de Petri.

2.1.2 Definição formal da rede de Petri

A visão gráfica da rede de Petri apresentada acima pode ser "traduzida" para uma visão matricial, associando um conjunto de números inteiros positivos ou nulos aos lugares e transições da rede.

Neste ponto de vista a rede de Petri é uma quádrupla

$$R = < P, T, Pre, Post > \tag{2.1}$$

onde:

- P é um conjunto finito de lugares de dimensão n;
- T é um conjunto finito de transições de dimensão m;
- $Pre : P \times T \longrightarrow I\!N$ é a aplicação de *entrada* (lugares precedentes ou incidência anterior), com $I\!N$ sendo o conjunto dos números naturais;
- $Post : P \times T \longrightarrow I\!N$ é a aplicação de *saída* (lugares seguintes ou incidência posterior).

Uma rede marcada N é uma dupla

$$N = < R, M > \tag{2.2}$$

onde:

- R é uma rede de Petri,
- M é a marcação inicial dada pela aplicação

$$M : P \longrightarrow I\!N. \tag{2.3}$$

$M(p)$ é o número de marcas ou fichas (*tokens* em inglês) contidas no lugar p. A marcação M é a distribuição das fichas nos lugares, sendo representada por um vetor coluna, cuja dimensão é o número de lugares e elementos $M(p)$. É a marcação da rede (distribuição das fichas nos lugares) que caracteriza o estado do sistema em determinado momento. No apêndice A são apresentados outros conceitos e definições relativos ao modelo de rede de Petri ordinárias.

Sistemas complexos de manufatura caracterizam-se pela capacidade de fabricar vários tipos de produtos simultaneamente. Existe, portanto, um componente de dados importante (tipo de peça a ser fabricada, estado atual da peça, etc.), em que a dinâmica tem um papel fundamental (seqüência de operações possíveis sobre um processo de fabricação, por exemplo). A repartição dos recursos (máquinas, sistema de transporte) e a complexidade dos mecanismos de alocação destes põem em evidência o paralelismo (tanto a cooperação como a concorrência) existente no sistema.

Entretanto, em tais sistemas complexos, alguns problemas aparecem. Esta complexidade significa, as vezes, a composição de vários processos semelhantes. Segundo [CV97], neste caso, quando se utiliza a rede de Petri ordinária (com a marcação dos lugares dada por fichas indiferenciadas, e com os lugares comportando-se como contadores), têm-se duas escolhas:

- modelar o comportamento geral sem precisar a identidade de cada processo, mas somente seu número;
- modelar, individualmente, cada um dos processos que constituem o sistema e modelar a interação existente entre eles, o que consiste, muitas vezes, em desdobrar o modelo que representa o comportamento geral.

No primeiro caso, obtém-se uma descrição compacta, mas não detalhada o suficiente: há uma falta de informação. No segundo caso, o modelo obtido pode ser pouco prático

de se trabalhar, seja pelo tamanho da rede, seja pelo número de interações existentes. É necessário, portanto, estruturar parte dos dados do sistema fora da estrutura da rede.

De fato, a modelagem completa de um sistema flexível de manufatura necessita de uma maior expressividade no que se refere à descrição dos dados, daquela fornecida pela rede de Petri ordinária que permite a representação apenas da estrutura e do controle do sistema.

Estas limitações foram sanadas a partir de extensões da rede de Petri que permitem a individualização das fichas, fazendo com que as mesmas carreguem informações consigo. As principais propostas de extensão são: a rede de Petri Colorida, a rede Predicado/Transição [Gen86] e a rede de Petri a Objetos [SB85].

2.2 Rede de Petri a Objetos

Antes de discutir o modelo de rede de Petri a Objetos [SB85] é conveniente apresentar brevemente o modelo Predicado/Transição [Gen86], uma vez que muitos dos conceitos utilizados no primeiro modelo foram definidos inicialmente para o segundo.

A rede Predicado/Transição é fundamentada no formalismo matemático da lógica de predicados de primeira ordem. Neste modelo o conceito de ficha é estendido, sendo permitido aos lugares conter fichas parametrizadas por n-uplas de constantes e variáveis. Estas n-uplas armazenam a identidade de cada um de seus elementos e também representam claramente as relações dinâmicas entre os mesmos. Os arcos são etiquetados por somas formais de variáveis, definindo a configuração de fichas que será produzida ou consumida em um lugar. Sobre as transições podem ser escritas fórmulas lógicas utilizando as variáveis dos arcos de entrada das transições. Assim, para disparar, a transição deve estar sensibilizada por uma substituição consistente de variáveis pelas n-uplas que constituem a marcação e o predicado associado deve ser verificado.

A rede de Petri a Objetos é um modelo semelhante as modelo Predicado/Transição, mas que utiliza a noção de objetos para representação de dados. Em particular, a rede de Petri a Objetos é adequada para a modelagem de aplicações ligadas à Automação Industrial, na qual o uso da noção de objeto permite uma melhor estruturação dos dados do sistema bem como o mapeamento direto dos objetos físicos manipulados [Val88].

Neste sentido, o conceito de classes de objetos, definida como um conjunto de

propriedades e atributos (da mesma forma que no modelo de orientação a objetos), é conveniente porque a modelagem da área trabalho é freqüentemente baseada nos objetos existentes no mesmo (máquinas, ferramentas, etc).

No modelo de rede de Petri a Objetos a marcação é dada por fichas individualizadas, representadas por objetos, que modelam os elementos manipulados pelo sistema. Nesta abordagem, uma classe de objetos é definida por um conjunto de atributos e um conjunto de operações ou métodos que permitem a manipulação dos valores dos atributos. Cada objeto possui um nome e é uma instância da classe de objetos.

A principal característica da rede de Petri a Objetos é que cada objeto é uma instância única da classe, ou seja, há um atributo de identificação do objeto e este atributo deverá receber um valor diferente para cada objeto instanciado.

2.2.1 Definição formal simplificada

Abaixo é colocado, de maneira simplificada, o modelo formal da rede de Petri a Objeto. A rede de Petri a Objeto é definida pela 9-upla:

$$N_o = < C_{class}, P, T, V, Pre, Post, A_{tc}, A_{ta}, M_0 >$$

onde:

- C_{class} é um conjunto finito de classes de objetos, eventualmente organizado em uma hierarquia e definindo para cada classe um conjunto de atributos;
- P é um conjunto finito de lugares cujos tipos são dados por C_{class};
- T é um conjunto finito de transições;
- V é um conjunto de variáveis cujos tipos são dados por C_{class};
- Pre é a função *lugar precedente* que a cada arco de entrada de uma transição faz corresponder uma soma formal de n-uplas de elementos de V;
- $Post$ é a função *lugar seguinte* que a cada arco de saída de uma transição faz corresponder uma soma formal de n-uplas de elementos de V;
- A_{tc} é uma aplicação que a cada transição associa uma condição fazendo intervir as variáveis formais associadas aos arcos de entrada aos atributos das classes correspondentes;

- A_{ta} é uma aplicação que a cada transição associa uma ação fazendo intervir as variáveis formais associadas aos arcos de entrada aos atributos das classes correspondentes;

- M_0 é a marcação inicial que associa a cada lugar uma soma formal de n-uplas de instâncias de objetos (os objetos devem ser representados por identificadores, como por exemplo, seu nome). Formalmente temos:

$$M : P \longrightarrow P(O^*) \tag{2.4}$$

onde O é o conjunto de objetos e O^* é o conjunto de tupla de objetos $o^* = \langle o_1, o_2 \ldots o_n \rangle$ que indica, para cada lugar $p \in P$ a tupla de objeto nele contida.

A cada lugar da rede de Petri a Objetos é associado um conjunto de n-uplas de classes de objetos e somente instâncias destas classes podem ser manipuladas (depositadas e removidas) neste lugar. Para poder manipular os objetos é declarado um conjunto de variáveis, explicitando para cada variável a classe de objetos que ela deverá capturar.

As transições da rede de Petri a Objeto permitem selecionar os objetos nos lugares de entrada, através das variáveis sobre os arcos. O disparo destas transições depende da existência desses objetos e de condições (filtros fornecidos por A_{tc}) a serem verificadas sobre os objetos capturados. Se as condições forem satisfeitas, então a transição pode ser disparada, o que resulta em ações que modificam a semântica desses objetos (conforme definidas em A_{ta}) e uma nova distribuição dos objetos nos lugares de saída da transição.

A análise no modelo de rede de Petri a Objetos é feita sobre a rede de Petri ordinária subjacente [2]. Os invariantes da rede de Petri subjacente fornecem resultados sobre a conservação de números de objetos independentemente de suas classes.

Com o intuito de auxiliar a compreensão do modelo de rede de Petri a Objetos é apresentado a seguir um exemplo, cuja descrição é apresentada em [Can90], que modela uma célula flexível de usinagem.

Considere o sistema apresentado na figura 2.2. O sistema representa uma célula flexível de usinagem formada por um par de Máquinas de Comandos Numéricos (M1 e

[2]Pode-se utilizar, por exemplo, o ARP [Maz90] um analisador de redes de Petri ordinárias desenvolvido no LCMI, disponível via Internet no endereço *ftp://ftp.lcmi.ufsc.br*

M2), que processam peças transportadas por uma mesa rotativa composta de quatro bandejas ("pallets") nas quais as peças são depositadas.

As peças são retiradas de um estoque e carregadas na mesa na posição 1, e descarregadas após o término da usinagem da posição 4. As peças são usinadas pela máquina M1 ou M2 quando presentes na bandeja 2 ou na bandeja 3, respectivamente. A cada peça é associado um conjunto de tarefas, definidos por uma seqüência de usinagem sobre as máquinas M1 e M2, como por exemplo a seqüência (M1 M2 M1).

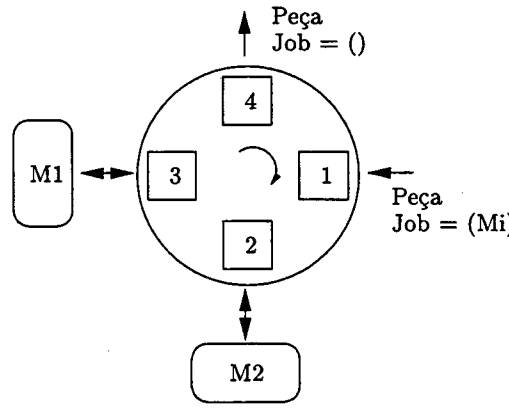


Figura 2.2: Célula flexível de usinagem

O resultado desta especificação obtida através do modelo de rede de Petri a Objetos é apresentado na figura 2.3, onde o lugar *peca_estoque* representa o estoque das peças, *pallet_estoque* representa o estoque de pallets, *mesa* representa a mesa rotativa, *maq_livre* representa as máquinas livres e *usinando* representa as máquinas usinando. As transições *Entrada* e *Saída* representam as operações de carga e descarga das peças na mesa. A rotação da mesa é representada pela transição *Rotação*, e o início e fim de usinagem pelas transições *Início* e *Fim*.

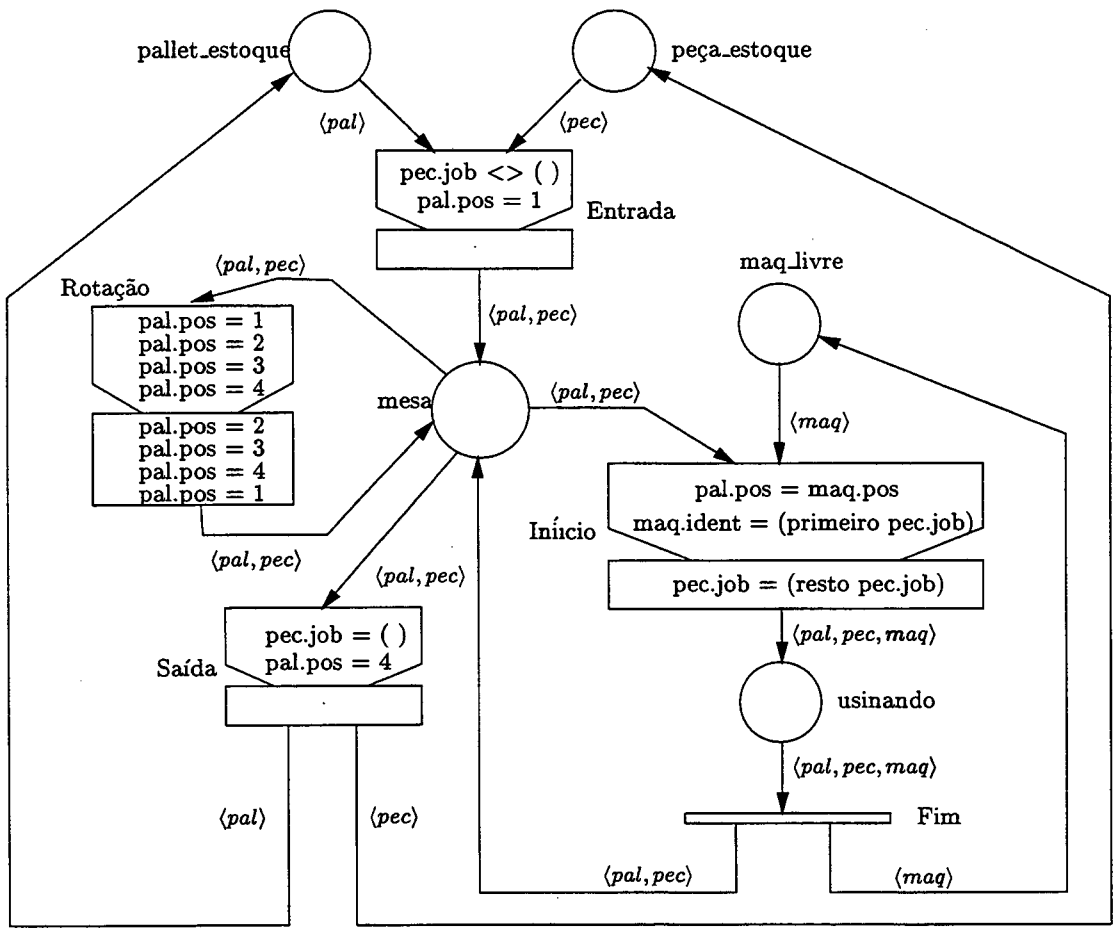
Pode-se perceber no exemplo que as três classes de Objetos (Peças, Pallets e Máquinas), são obtidas pelo simples mapeamento dos objetos físicos do sistema. Para cada transição são apresentadas as condições e ações da mesma. Cada objeto ao ser instanciado preenche a lista de atributos e já é colocado em um dos lugares da rede, compondo a marcação inicial.

C_{class} = { *Pallets, Peças, Máquinas* }
P = { *peça_estoque, pallet_estoque, mesa, maq_livre, usinando* }
T = { *Entrada, Rotação, Início, Fim, Saída* }
V = { *pal, pec, maq* }

Atributos:
Peças: ident, job;
Pallets: pos;
Máquinas: ident, pos;

Classe das Variáveis:
pec: Peças
pal: Pallets
maq: Máquinas

Classe dos lugares:
peça_estoque: ⟨*Peças*⟩
pallet_estoque: ⟨*Pallets*⟩
mesa: ⟨*Pallets, Peças*⟩
maq_livre: ⟨*Máquinas*⟩
usinando ⟨*Pallets, Peças, Máquinas*⟩



Marcação:
pal_1.Pallets: (posição 1), (location pallet_estoque);
pal_2.Pallets: (posição 2), (location pallet_estoque);
pal_3.Pallets: (posição 3), (location pallet_estoque);
pal_4.Pallets: (posição 4), (location pallet_estoque);
M1.Máquinas: (posição 2), (nome M1), (location maq_livre);
M2.Máquinas: (posição 3), (nome M2), (location maq_livre);
P1.Peças: (ident P1), (job (M1 M2 M1)), (location peça_estoque);
P2.Peças: (ident P2), (job (M2 M1)), (location pecça_estoque);

Figura 2.3: Modelagem da célula de usinagem utilizando rede de Petri a Objetos

2.2.2 Redes de Petri e Supervisão

Quando a rede de Petri é utilizada para a modelagem de sistemas físicos, as transições denotam a mudança de estado. Em geral, sistemas físicos interagem com o ambiente. Neste caso, na descrição com rede de Petri, cada evento que ocorre no sistema é associado com uma transição como uma variável booleana. Tal variável é chamada interpretação ou condição extra. Ela pode ser:

- *externa*, como o valor de um sensor, uma mensagem recebida ou enviada. Por exemplo, se o evento *peça chegando na máquina* é detectado por um sensor, a condição extra é uma variável booleana P associada com a transição;
- *interna*, como o valor do atributo de um objeto. Por exemplo, uma informação temporizada ou uma informação geral (diâmetro da peça $>$ 3cm).

A descrição através da rede de Petri permite uma supervisão sistemática estrita. Se um erro ocorre no sistema, o estado da rede de Petri (marcação + valores lógicos das condições extras) não permite representar o estado do sistema. Considere o jogador (*token player* em inglês) de rede de Petri interpretada descrito na figura 2.4, que atualiza a marcação a cada ocorrência de evento. Enquanto está sendo executado duas situações podem ocorrer:

- a transição está habilitada pela marcação e, devido a interpretação ser falsa, nunca dispara. Este caso corresponde à verificação que o evento escalonado ocorreu antes de algum *"deadline"*. Um objeto permanecendo em um lugar (que representa a operação de uma máquina, por exemplo) mais que esta duração significa que a máquina falhou ou o tempo atribuído estava incorreto;
- o evento ocorre e a transição correspondente não está habilitada pela marcação. Este caso corresponde a ocorrência de um evento vinculado a uma transição que não pode ser disparado pelo estado atual do sistema. Nesta situação um alarme deve ser ativado.

No último caso, a marcação que não representa o estado atual do sistema pode ser o resultado de uma possível intervenção humana (por exemplo a retirada manual de uma peça de um depósito). É desejável levar em consideração a intervenção humana, ao invés de acionar um alarme. Neste sentido, é melhor tentar atualizar o estado do sistema de forma correta.

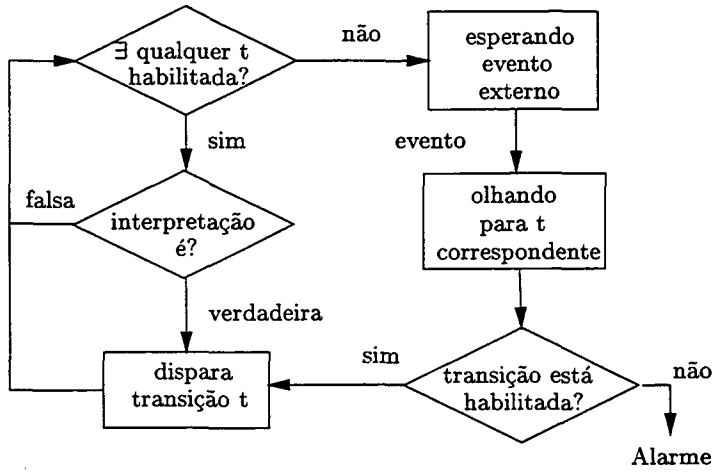


Figura 2.4: Jogador de rede de Petri interpretada

A nível de supervisão existem alguns eventos possíveis (que aqui serão chamados de *anormais*), cuja ocorrência não é desejável, mas que devem ser vistos de forma natural, no sentido que podem eventualmente acontecer, como panes, quebras, etc.

É desejável que o processamento seja realizado com a mínima assistência do operador. Entretanto, esta assistência constitui um elemento essencial no sistema de manufatura. De fato, intervenções humanas devem ser raras, mas em algumas situações elas podem ser essenciais.

Os requisitos para uma função de supervisão que leve em conta eventos anormais são ambíguos e contraditórios: devem ser estritos para evitar a propagação de qualquer falha; mas devem ser tolerantes à intervenções humanas no sentido de evitar o disparo de alarmes em tais situações.

O problema é ser tolerante em casos específicos. Desta forma, os eventos podem ser decompostos em três casos:

- operação normal;
- aceitáveis uma vez;
- proibidos.

Por exemplo, considere um Veículo Auto Guiado (AGV - *Automatic Guided Vehicle*) parado entre dois contatos em um sistema de transporte. Depois de uma intervenção humana para resolver um problema, o veículo é reiniciado; um evento normal é a

chegada do AGV no próximo contato, o proibido é a chegada em qualquer outro contato, entretanto um evento aceitável é a chegada do AGV a uma estação de manutenção.

O comportamento normal e o proibido são considerados de uma forma natural pela teoria de rede de Petri. Mas, se somente estes eventos são considerados, na ocorrência de eventos inesperados, contradições entre o modelo e as informações do mundo real irão aparecer.

O modelo de rede de Petri com Marcação Imprecisa combina a teoria de conjuntos nebulosos e a teoria de rede de Petri a Objetos para dotar este modelo da capacidade de representação de conhecimento incerto sobre o estado do sistema. A seguir, na seção 2.3 é apresentado o modelo de rede de Petri com Marcação Imprecisa.

2.3 Redes de Petri com Marcação Imprecisa

Alguns trabalhos tem sido realizados no sentido de dotar as redes de Petri da capacidade de trabalhar com informações imprecisas ou imperfeitas [BB94][CS93][GAG91][GG95][LG94][Loo88][PG94][SG93]. Estes modelos, apesar de serem chamados genericamente de redes de Petri Nebulosas (*Fuzzy Petri nets*), são bastante diferentes entre si. Por exemplo, em [Loo88] a rede de Petri é vista como uma rede Neural sem os aspectos dinâmicos da rede de Petri; em [SG94] e [CKC90], a rede de Petri Nebulosa é modelo para o raciocínio sobre proposições; em [RV89] ela é um modelo para representar e raciocinar sobre o tempo e os estados do sistema na presença de informação incompleta. Uma discussão sobre estes trabalhos pode ser vista em [JC96].

Antes de apresentar formalmente o modelo de rede de Petri Nebulosa utilizado, será feita uma apresentação informal, com base no exemplo apresentado em [CV97] e mostrado na figura 2.5. Ela descreve um sistema de transporte com um depósito de peças, oito máquinas $M_1 \ldots M_8$, uma esteira circular e um robô que retira uma peça no depósito e a coloca no sentido horário (M_1 a M_8) nas máquinas.

O modelo de rede de Petri Nebulosa apresentado associa a distribuição de possibilidade $\pi^*{}_o(p)$ para a localização de um objeto o^*, p sendo um lugar da rede. Este modelo permite representar:

- *uma marcação nebulosa*: a localização de cada objeto possui uma distribuição de

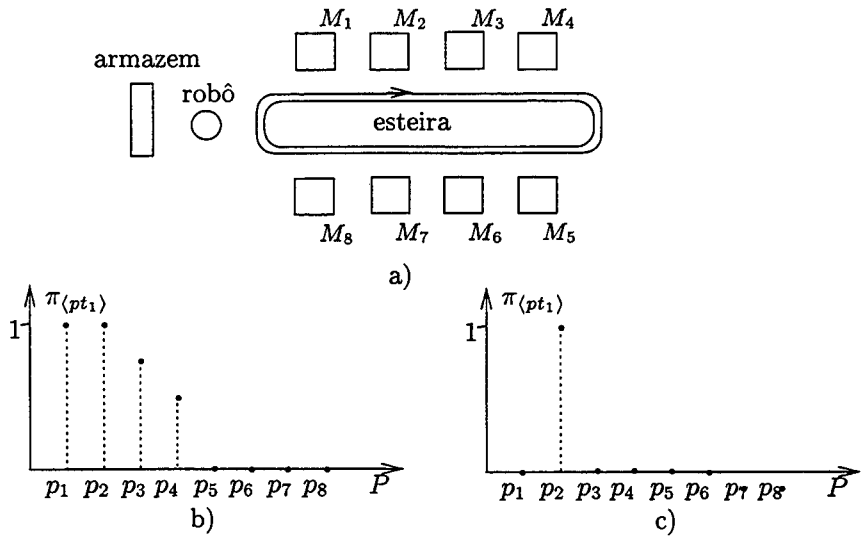


Figura 2.5: Sistema de transporte

possibilidade sobre o conjunto de lugares. Não é possível afirmar que um dado objeto está em um dado lugar, mas somente que ele está em um lugar entre um dado conjunto de lugares;

- *marcação precisa*: cada objeto está localizado em um e somente um lugar da rede (o estado atual é bem conhecido).

Considere que o sistema da figura 2.5 é modelado por uma rede de Petri onde cada máquina M_i é representada por um lugar p_i e uma peça é representada por uma ficha (uma instância de objeto) $\langle pt_j \rangle, j = \{1..8\}$ da classe ou tipo peça.

Suponha que depois de uma pane, a localização de uma peça $\langle pt_1 \rangle$ não é conhecida. A distribuição de possibilidade $\pi_{pt_1}(p)$ na figura 2.5.b representa a imprecisão da informação sobre o estado atual e caracteriza uma marcação nebulosa, indicando que:

- a peça não está nas máquinas M_5 a M_8 ($\pi_{pt_1}(p) = 0$ para estes lugares);
- a peça pode estar nas máquinas M_1 a M_4 ($\pi_{pt_1}(p) \neq 0$ para estes lugares).

Deve-se registrar que não há ambigüidade entre a informação imprecisa sobre o estado de uma peça (ela pode estar entre a máquina M_1 e M_4) e duas peças diferentes (pt_1 está em M_5 e pt_2 está em M_2).

Se depois de um certo tempo uma informação precisa é recebida do sistema (como um sensor indicando que pt_1 está na máquina M_2), $\pi_{pt_1}(p)$ passa a ter um único elemento (figura 2.5.c): a marcação é então conhecida com certeza.

Neste sentido, uma marcação precisa representa o comportamento normal do sistema e a marcação imprecisa indica que alguma dedução está sendo feita.

Como na rede de Petri clássica, o disparo da transição em uma rede de Petri com Marcação Imprecisa modifica a marcação, representando o conhecimento sobre o sistema. Mas para cada tipo de marcação a transição está habilitada de uma forma diferente, como será visto na seção 2.3.2. Mas o que provoca uma marcação nebulosa? No início da operação do sistema, o estado atual do mesmo é conhecido. Mas se há uma perda de informação não é possível saber se uma peça não chegou na máquina (o operador pode ter retirado a peça) ou ela chegou, mas o sensor está com problema. Isto quer dizer que a interpretação associada com esta transição não pode somente ter os valores *verdadeiro* e *falso* como na rede de Petri clássica, mas também um valor *incerto*, expressando desta forma eventos aceitáveis (a retirada da peça pelo operador ou a falha do sensor).

De acordo com o tipo de marcação (precisa ou imprecisa), e o valor da interpretação (precisa - verdadeira ou falsa -, ou incerta), o disparo da transição em uma rede de Petri com Marcação Nebulosa é preciso (disparo clássico) ou incerto (*pseudo-disparo*). No primeiro caso uma marcação precisa é obtida, e no último uma marcação nebulosa é obtida. As condições para o disparo preciso e para o pseudo-disparo são discutidas na seção 2.3.2.

2.3.1 O Modelo da rede de Petri com Marcação Imprecisa

Uma forma equivalente à apresentada na equação 2.4 para definir a marcação de uma rede de Petri a Objetos é:

$$M : O^* \times P \longrightarrow \{0, 1\} \tag{2.5}$$

onde O é o conjunto de objetos e O^* é o conjunto de tupla de objetos.

Em [Car98] a marcação descrita pela equação 2.5 é estendida, considerando os valores no intervalo $[0, 1]$ assim, obtém-se $M : O^* \times P \longrightarrow [0, 1]$. Nesta abordagem, ao lado dos valores impossíveis ($M = 0$), e dos valores possíveis ($M = 1$), os valores mais ou menos possíveis ($0 < M < 1$) também são definidos. Esta representação constitui uma marcação nebulosa, uma vez que todos os valores do intervalo $[0, 1]$ são considerados, entretanto, convém ressaltar, neste trabalho é utilizado a representação

da marcação descrita pela equação 2.5, a qual permite a representação de marcação imprecisa. Assim, pode-se perceber, a marcação imprecisa se caracteriza por associar valores do conjunto $\{0,1\}$ a localização dos objetos (ou tuplas de objetos) aos lugares da rede de Petri.

Como foi discutido na apresentação informal, no sentido de representar a imprecisão é introduzido a distribuição de possibilidade $\pi_{o^*}(p)$ que é vinculada a cada tupla $o^* \in O^*$ e definida no conjunto P.

Seja $l(o^*) \in L(O^*)$ a variável indicando a localização de o^*, C_p e C_o os tipos dos lugares e das tuplas de objetos. A distribuição de possibilidade $\pi_{l(o^*)}$ é a estimação numérica da possibilidade que a localização de o^*, $l(o^*)$, seja exatamente p,

$$l(o^*) \in \{p \mid C_p(p) = C_o(o^*)\} \quad \pi_{l(o^*)} : P \rightarrow \{0,1\}$$

O conjunto de todas estas funções (para todos os objetos e todos os lugares da rede) descrevem a marcação. Deste ponto em diante será utilizado π_{o^*} para denotar de forma reduzida $\pi_{l(o^*)}$

$$o^* \in \{p \mid C_p(p) = C_o(o^*)\} \quad \pi_{o^*} : P \longrightarrow \{0,1\} \tag{2.6}$$

O valor de $\pi_{o^*}(p)$ indica a possibilidade de que a localização da tupla o^* seja p. A partir de um valor inicial dado na marcação inicial, novos valores são computados após a ocorrência de cada evento através do disparo das transições.

O valor da distribuição de possibilidade, vinculado a qualquer elemento o_i, pertencente a tupla o^*, é o mesmo ao da própria tupla. Se um dado elemento aparece em mais de uma tupla em um determinado lugar, então o valor da sua distribuição de possibilidade será o máximo valor entre as tuplas:

$$\forall o^* \mid o_i \in o^* \pi_{o_i} = max\{\pi_{o^*}(p)\}.$$

Considerando a supervisão, enquanto nenhum disparo de transição acontecer, será trabalhado diretamente com a distribuição de possibilidades sobre as instâncias de objetos (e não sobre as tuplas):

$$o \in O \; \pi_o : P \longrightarrow \{0,1\} \tag{2.7}$$

e será considerado como uma representação parcial da marcação.

É possível fazer as seguintes afirmativas: $\pi_o(p) = 1$ representa o fato que p é a possível localização de o, e $\pi_o(p) = 0$ expressa a certeza que o não está presente no lugar p.

O conhecimento sobre o estado do sistema, em particular a localização dos objetos, é então expressa desta forma. Deste conhecimento é possível deduzir se uma instância de objeto pertence a um dado conjunto de lugares. O interesse em obter esta informação é grande já que a partir do momento que a marcação é imprecisa, em algum momento é necessário voltar para um estado preciso ou, ao menos, ajudar o operador indicando a possível localização atual dos objetos no sistema.

A teoria de possibilidades permite executar estas tarefas, calculando a possibilidade Π [DP88b], que $l(o)$, a localização de o, seja um lugar que pertence a um conjunto A de lugares, onde $A \subseteq P$.

Considere que o conjunto de referência P seja particionado em subconjuntos A_i que são unitários, ou seja $A_i = \{p_i\}$. A *medida de possibilidade* Π [DP88b] pode ser definida a partir destes valores nos subconjuntos:

$$\Pi(A) = \max_{p \in A} \pi_o(p) \; \forall A \subseteq P, \tag{2.8}$$

onde $\pi_o(p) = \Pi(\{p\})$. Seja $\bar{A} = P - A$. A *medida de necessidade* $N(A)$ pode ser obtida a partir da medida de possibilidade $\Pi(A)$:

$$N(A) = 1 - \Pi(\bar{A}). \tag{2.9}$$

É este valor da medida de necessidade N que define se a marcação é precisa ou imprecisa. Se $\Pi(A) = 0$, é impossível que o^* esteja no lugar $p \in A$. Mas se $\Pi(A) = 1$, existem dois resultados possíveis: se $N(A) = 1$ a marcação é precisa e, se $N(A) \neq 1$ a marcação é imprecisa.

Se $\pi_o(p) = 1$, e $\forall p_i \neq p \; \pi_o(p_i) = 0$, utilizando a equação 2.8 obtém-se $\Pi(A) = 1$ e, utilizando a equação 2.9 obtém-se $N(A) = 1$, já que $\Pi(\bar{A}) = \max_{p \in \bar{A}} \pi_o(p) = 0$. Então o objeto necessariamente pertence ao lugar p e portanto a marcação é precisa.

2.3.2 O disparo da transição na rede de Petri com Marcação Imprecisa

Uma marcação imprecisa corresponde a um conhecimento sobre uma dada situação em um determinado instante. Pretende-se apresentar nesta seção como uma marcação evolui a partir do disparo de uma transição e como uma nova distribuição de possibilidade é calculada.

A marcação evolui quando uma nova informação permite que se conclua que o evento correspondente à transição ocorreu ou é possível que ele ocorra. Esta informação pode ser considerada de duas formas:

1. o tempo é considerado de forma implícita e aparece na interpretação, associado com a transição;
2. o tempo é considerado explicitamente no modelo e o tempo atual corresponde a uma possível data para o disparo da transição.

No presente trabalho foi utilizada a primeira abordagem que considera o tempo de forma implícita. Ou seja, está associado a uma interpretação ou uma condição extra vinculada à transição.

Conforme foi visto na seção 2.3.1, em uma rede de Petri com Marcação Imprecisa a marcação pode representar o estado do sistema tanto quando o mesmo é preciso, como o estado onde o conhecimento é impreciso. Assim, o disparo das transições que faz a marcação evoluir, atualiza os valores da distribuição de possibilidades.

Do ponto de vista da marcação, uma transição está habilitada se existe pelo menos uma instância de objeto o em cada lugar de entrada, tal que $\pi_{o^*}(p) > 0$. A transformação da marcação será diferente conforme a certeza (ou não) da ocorrência do evento associado com o disparo da transição, na data atual. Assim, do ponto de vista da marcação, há duas formas de disparar a transição:

- *disparo preciso*: quando $\forall p \in Pre(p,t), N(\{p\}) = 1$, ou seja, a marcação atual dos lugares de entrada da transição é precisa. Corresponde ao disparo normal da transição e, desta forma, a localização de todas as instâncias de objetos envolvidas deve ser precisa: $N(A) = 1$. A ação associada com a transição é executada; um novo estado do sistema é obtido e a nova marcação é também precisa;
- *disparo incerto* ou *pseudo-disparo*: quando $\exists p \in Pre(p,t), |N(\{p\}) \neq 1$, isto é, um dos objetos que está no lugar de entrada da transição tem sua localização

imprecisa. Isto não corresponde a evolução esperada do sistema, o que significa que está sendo realizada uma inferência sobre o estado do sistema.

Para ajustar o modelo aos dois tipos de disparo existentes em uma rede de Petri com Marcação Imprecisa, o disparo (preciso ou incerto) de uma transição é decomposto em 2 passos:

- início do disparo: as fichas (objetos) são colocadas no lugar de saída mas não são removidas do lugar de entrada;
- final do disparo, que pode ser:
  - cancelamento do disparo: as fichas são removidas do lugar de saída de t;
  - confirmação do disparo: as fichas são removidas do lugar de entrada de t.

Um disparo preciso consiste do início do disparo, imediatamente seguido da confirmação do disparo.

O pseudo-disparo pode ser considerado o início de um disparo porque não há nenhuma informação que permita afirmar com certeza que o evento associado com a transição ocorreu ou não naquele determinado instante. A imprecisão será propagada enquanto não ocorrer um evento que permita deduzir a localização do objeto. Enquanto isto não ocorre deve-se levar em conta ambas as marcações:

- a marcação M existente antes do disparo de t;
- a marcação M', com $M \xrightarrow{t} M'$, que seria obtida depois do disparo da transição t, correspondente à ocorrência do evento.

Esta marcação corresponde a marcação imprecisa $\mathcal{M} = \{M, M'\}$, que é um conjunto disjuntivo de marcações.

O pseudo-disparo é uma forma de fazer uma dedução para frente. Fazendo isto, evita-se a contradição em caso de comportamento anormal do sistema. Assim que ocorre um incidente opera-se com conhecimento impreciso no sentido de estar habilitado a levar em conta um largo conjunto de eventos possíveis (mensagens de atualização do sistema de manufatura). A ocorrência de um destes eventos permite retornar a um conhecimento preciso (ou, ao menos, um conhecimento menos impreciso). Neste instante, um procedimento de recuperação é chamado para atualizar a distribuição de possibilidades de todas as fichas (objetos). Alguns pseudo-disparos serão confirmados (se o evento correspondente ocorreu) e alguns serão cancelados (se o evento correspondente não ocorreu).

2.3.3 Incerteza introduzida pela interpretação

Como foi visto na seção 2.2.2, a interpretação ou condição extra pode envolver dados (por exemplo, atributos dos objetos na rede de Petri a Objetos) ou a ocorrência de um evento externo. Uma transição somente pode ser disparada (de forma precisa) quando está habilitada por uma marcação precisa e quando a interpretação é verdadeira.

A interpretação I de uma rede de Petri com Marcação Imprecisa é definida vinculando, à cada transição, uma *função de autorização* $\eta_{x_1 \ldots x_n}$ que implementa a parte da condição extra de disparo:

$$\eta_{x_1 \ldots x_n} : T \longrightarrow \{falso,\ incerto,\ verdadeiro\}$$

onde $x_1 \ldots x_n$ são as variáveis associadas aos arcos de chegada da transição t.

Assumindo-se que $o_1 \ldots o_n$ é uma possível substituição para $x_1 \ldots x_n$ para disparar t, por meio da interpretação η, é possível diferenciar as situações para as quais a transição t disparará de forma precisa ou com pseudo-disparo.

No modelo de rede de Petri com marcação imprecisa a função de autorização desempenha um papel fundamental, visto que através dela é introduzida a noção de incerteza na interpretação, possibilitando assim levar em conta eventos anormais que possam acontecer no sistema modelado.

Utiliza-se a figura 2.6 para verificar cada uma das situações, para cada um dos tipos de disparo. Conforme foi colocado de maneira informal, a marcação pode ser precisa ou imprecisa, e a interpretação pode ser incerta ou verdadeira. Se a transição t:

- não está habilitada pela marcação, mas a interpretação é verdadeira: é uma situação proibida e um alarme é ativado;
- está habilitada por uma marcação precisa e a interpretação é verdadeira: a transição é disparada de forma precisa, como no caso de rede de Petri clássica;
- está habilitada de forma precisa e a interpretação é incerta: o pseudo-disparo é executado e a imprecisão do sistema é criada;
- é habilitada por uma marcação imprecisa; se a interpretação é incerta, t é pseudo-disparada (propagação da imprecisão);

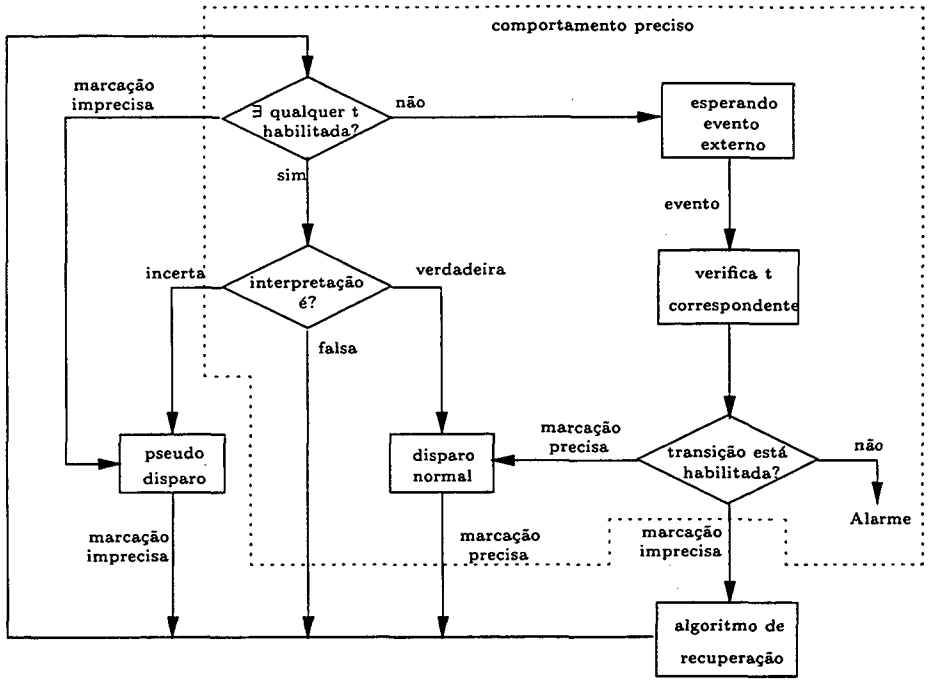


Figura 2.6: Jogador de rede de Petri com Marcação Imprecisa

- é habilitada por uma marcação imprecisa (o que significa que alguma dedução já foi feita) e a interpretação é verdadeira (informação chegando do sistema): o *algoritmo de recuperação* é chamado (uma nova distribuição de possibilidades destas fichas - objetos - é calculada de forma a retornar para um estado preciso).

Na figura 2.6, a parte interna do pontilhado corresponde ao comportamento normal (preciso) do sistema. O jogador de rede de Petri desta parte interna é equivalente ao da figura 2.4.

Recuperação de um estado preciso

A mudança de um estado incerto para um estado preciso é realizada quando, para uma transição t que estava pseudo-disparada, temos uma interpretação verdadeira ($\eta(t) = verdadeiro$) para a tupla contendo o objeto que está com a localização imprecisa.

No sentido de preservar uma marcação coerente, um tratamento permitindo transformar a possível presença do objeto no lugar de entrada em uma certeza deve ser

obtido. Isto é feito por um procedimento de *recuperação de estado preciso.*

Se este procedimento conseguir retornar para um estado preciso, onde a localização de todas as instancias é conhecida ($\forall o^* \in O^*\ N_{o^*}(q) = 1$ e $N_{0^*}(p) = 0\ \forall p \neq q$), o sistema continua funcionando de forma normal. Se isto não é possível, o algoritmo dá ao operador (ou ao sistema de diagnóstico) o conjunto de todas as possíveis localizações do objeto o, que corresponde a todos os lugares p tais que $\pi_o(p) \neq 0$.

2.3.4 Utilização da rede de Petri com Marcação Imprecisa

Neste item será apresentado um exemplo de modelagem, utilizando o modelo de rede de Petri com Marcação Imprecisa, para tal, considere a figura 2.7 apresentada em [CVD91] que descreve um sistema de Veículos Auto Guiados (AGV - *Automatic Guided Vehicle*). Estes veículos percorrem automaticamente alguns circuitos. Suas localizações são conhecidas somente em alguns pontos chamados contatos (C). Eles só recebem comandos (parar, ir, mudar itinerário) quando estão nos contatos. No sentido de evitar colisões os circuitos são decompostos em sessões (S), e os veículos são controlados de forma que em um dado instante haja somente um veículo por sessão.

O exemplo descreve a coordenação de um sistema de transporte baseado em AGVs: as transições *trans_a* e *trans_b* e os lugares MS (veículo movendo-se ao longo da sessão), C (veículo sobre o contato) e S (sessão livre) representam o comportamento normal do sistema.

Como as variáveis associadas aos arcos são substituídas por instâncias de objetos, elas também são tipadas pela classe de objetos correspondente. No exemplo, a variável v representa um veículo, s é alguma sessão e ns é a próxima sessão. A transição *trans_b* indica que o veículo $\langle v \rangle$ esta começando a move-se ao longo de $\langle s \rangle$. O lugar C representa o veículo $\langle v \rangle$ no contato da sessão $\langle s \rangle$. O lugar S contém as sessões livres.

As transições *trans_d* e *trans_f* descrevem o comportamento anormal do sistema. Este comportamento anormal pode ser provocado pela falta de bateria no AGV (que faz com que o mesmo pare enquanto está atravessando a sessão). Assim quando a bateria do veículo $\langle v \rangle$ falha, este é retirado manualmente da sessão $\langle s \rangle$ e dirigido para a estação de bateria (*lugar LB*). O lugar MOS representa o movimento de $\langle v \rangle$ fora das sessões.

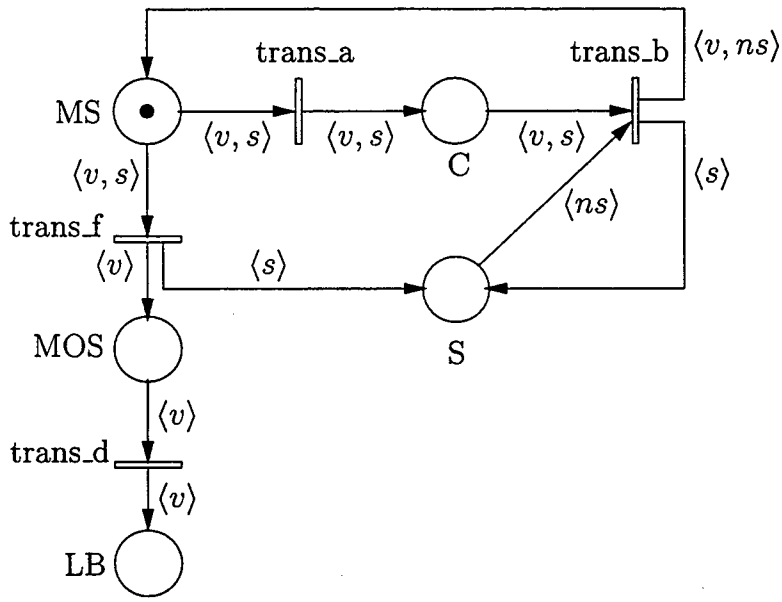


Figura 2.7: Exemplo de rede de Petri com Marcação Imprecisa

A transição *trans_a* representa a chegada do veículo $\langle v \rangle$ no contato. Sua *função de autorização* é a seguinte:

$$\eta_v(trans\_a) = \forall\ v \begin{cases} verdadeira & se & sinal(x) \\ incerta & se & v.time > t_{max_2} \\ falsa & & demais\ casos \end{cases} \tag{2.10}$$

onde *sinal(x)* é a mensagem enviada pelo contato quando $\langle v \rangle$ chega.

Assume-se que o veículo $\langle v \rangle$ possui um atributo que armazena a data quando ele entrou na sessão $\langle s \rangle$ (atributo "arrived", por exemplo). Este atributo é usado como cão de guarda (*watch dog* em inglês), determinando quanto tempo o veículo permanece movendo-se na sessão.

Eventualmente o veículo pode parar porque detecta algo bloqueando seu trajeto (por exemplo, um operador cruzando seu caminho), ou ainda, como colocado, pode acontecer da bateria descarregar e o veículo deve ser removido manualmente até a estação de manutenção. A coordenação do sistema de transporte deve tolerar as duas situações e liberar adequadamente a sessão em que o veículo estava quando o segundo caso acontece. Como o movimento fora da sessão para a estação de manutenção não é controlado pela coordenação do sistema de transporte, nenhum sinal indicando que a transição *trans_f* foi disparada. Entretanto é esta a transição que libera a sessão em

que se encontrava o veículo que falhou.

Assim, a seguinte *função de autorização* é vinculada a transição *trans_f*:

$$\eta_v(trans\_f) = \forall\ v \begin{cases} incerta & se \quad v.time > t_{max} \\ falsa & demais\ casos \end{cases} \tag{2.11}$$

onde t_{max} é o tempo máximo que o veículo leva para cruzar a sessão. A *função de autorização* da transição *trans_d* é idêntica a *trans_a*.

Quando um veículo $\langle v \rangle = v_i$ permanece por um longo período de tempo sobre a sessão $\langle s \rangle = s_j$, um pseudo-disparo da transição *trans_f* é realizado (valor do atributo $v.time$ é maior que t_{max_2}). A distribuição de possibilidade π_{vi} é igual a 1 para os lugares MS e MOS e π_{sj} é igual a 1 para MS e S. Assim, a sessão $\langle s_j \rangle$ não pode ser alocada para outro veículo apesar do seu nome aparecer (de forma imprecisa) no lugar S. Isto porque a localização do veículo $\langle v_i \rangle$ é imprecisa, o que significa que ele ainda pode estar na sessão $\langle s_j \rangle$.

Como a marcação é imprecisa, a transição *trans_a*, bem como a transição *trans_d*, estão habilitadas pela tupla $\langle v_i, s_j \rangle$. Supondo que, após um certo tempo $v_i.time > t_{max_2}$, a transição *trans_a* também é pseudo-disparada. Neste instante as transições *trans_a* e *trans_d*, bem como *trans_b* (caso a seção s_{j+1} esteja livre) estão habilitadas. Uma delas irá disparar depois de receber uma mensagem e uma nova computação da distribuição de possibilidades de v_i e s_j será executada.

Considere o caso em que quando a transição *trans_d* é disparada. O pseudo-disparo da transição *trans_f* que é vinculado, e da transição *trans_a* são cancelados. A localização do objeto $\langle v_i \rangle$ no lugar *MOS* é conhecida com certeza e a transição *trans_d* é disparada. Por conseqüência, a localização do objeto s_j torna-se precisa no lugar *S*, o que significa que a seção poderá ser alocada para outro veículo (através do disparo preciso de *trans_b*.

2.4 Conclusão

Neste capítulo foi introduzido o modelo da rede de Petri. Inicialmente foram colocadas as principais características do mesmo. A seguir fez-se uma apresentação informal do modelo, descrevendo os elementos que o compõem e seu comportamento dinâmico. Na seqüência o modelo formal das redes de Petri ordinárias foi apresentado; para tal foi apresentada a notação matricial de representação da rede.

Para suprimir as limitações quanto a expressividade da rede de Petri no que se refere a representação de dados foi apresentado o modelo de redes de Petri a Objetos. A descrição dos elementos que compõem o modelo e um exemplo também foram expostos.

A partir da necessidade de uma interpretação correta da descrição dos sistemas na presença de falhas, ou eventos anormais, foi introduzido o modelo de rede de Petri com Marcação Imprecisa. Este modelo estende a noção de disparo da transição, inserindo o pseudo-disparo. No pseudo-disparo as fichas são colocadas nos lugares de saída da transição sem contudo retirá-las dos lugares de entrada, desta forma, é feita uma inferência sobre a localização dos objetos pseudo-disparados pela transição.

Capítulo 3

Sistemas de Regras

Neste capítulo são apresentados inicialmente os chamados *sistemas de produção*, com suas vantagens e desvantagens. Posteriormente o modelo dos sistemas especialistas será discutido. A seguir é apresentado o sistema FASE, uma ferramenta que constitui um arcabouço de sistema especialista.

3.1 Sistemas de Produção

O termo *Sistema de Produção* é usado para descrever uma família de sistemas que têm em comum o fato de serem constituídos por um conjunto de regras na forma de pares de expressões representando uma condição e uma ação. A idéia inicial dos sistemas de produção foi concebida por Emil Post em 1936, quando ele propôs estes que atualmente são chamados *Sistemas de Post* [Pos43].

Um sistema de Post consiste em um conjunto de regras para a especificação sintática de transformações sobre cadeias de caracteres, e representa, como demonstrou Post, um método geral para processamento de dados.

Na sua forma mais simples, um modelo de sistema de produção apresenta dois componentes passivos - o *Conjunto de Regras* e a *Memória de Trabalho* - definidos da seguinte forma:

- Regras: conjunto ordenado de pares (LE, LD)[1], onde LE e LD são seqüências de

[1]LE e LD são a abreviação para lado esquerdo e lado direito, tradução das expressões em inglês,

caracteres;

- Memória de Trabalho: uma seqüência de caracteres;

e um componente ativo - o *Interpretador* - que realiza o seguinte procedimento:

- para cada regra (LE, LD), se a seqüência de caracteres LE está contida na memória de trabalho, então substituir os caracteres de LE na memória de trabalho pelos caracteres LD. Continuar na próxima regra e, caso esta seja a última, retornar à primeira.

Originalmente os sistemas de Post não possuem nenhuma semântica associada aos símbolos armazenados na memória de trabalho. Entretanto, no caso de se fazer alguma classificação quanto a monotonicidade de tais sistemas, os mesmo seriam classificados como não monotônicos, uma vez que os caracteres que satisfazem determinada regra são retirados da memória de trabalho e substituídos por um outro conjunto de caracteres.

Para exemplificar um sistema de produção com o comportamento acima descrito, considere um sistema formado pelo seguinte conjunto de regras:

$$\{1 : P \rightarrow XYX,\ 2 : X \rightarrow X1,\ 3 : X \rightarrow 1,\ 4 : Y \rightarrow Y0,\ 5 : Y \rightarrow 0\}$$

Supondo que inicialmente a memória de trabalho contém o caracter P, a execução do componente interpretador definido anteriormente, resultaria na seguinte seqüência de valores na memória de trabalho:

$$\begin{array}{l} \mathrm{P} \xrightarrow{1} \\ \quad \mathrm{XYX} \xrightarrow{2} \\ \quad\quad \mathrm{X1YX} \xrightarrow{3} \\ \quad\quad\quad \mathrm{11YX} \xrightarrow{4} \\ \quad\quad\quad\quad \mathrm{11Y0X} \xrightarrow{5} \\ \quad\quad\quad\quad\quad \mathrm{1100X} \xrightarrow{2} \\ \quad\quad\quad\quad\quad\quad \mathrm{1100X1} \xrightarrow{3} \\ \quad\quad\quad\quad\quad\quad\quad \mathrm{110011} \end{array}$$

A parte *condição* da regra é também denominada de *lado esquerdo* ou *antecedente*, ou ainda *premissa*, enquanto que a parte que expressa a *ação* a ser executada, é denominada *lado direito* ou *conseqüente*.

Left-Hand-Side (LHS) e Right-Hand-Side (RHS), respectivamente.

De acordo com [Bit96] os sistemas de produção foram redescobertos durante os anos setenta como uma ferramenta para a modelagem da psicologia humana. O formato condição-ação se adapta à modelagem de todos os comportamentos baseados em pares estímulo-resposta. Devido a isso, estes sistemas são a forma de representação do conhecimento mais utilizada em Inteligência Artificial (IA).

Conforme colocado em [Rab95], tais sistemas podem ser formados por mais de uma base de regras, separadas segundo conveniências de processamento. Complementa ainda - junto ao interpretador - uma estratégia de controle que estabelece a ordem com que as regras são aplicadas, bem como critérios de desempate quando houver mais de uma regra candidata a ser aplicada ao mesmo tempo.

Os sistemas de produção apresentam algumas vantagens e desvantagens que convém observar. As principais vantagens apresentadas por [Wat86] são:

Modularidade: as regras dos sistemas de produção podem ser consideradas, para efeito de manipulação, como peças independentes. Como os programas de IA quase sempre estão incompletos, novas regras podem ser acrescentadas ao conjunto já existente sem maiores preocupações.

Naturalidade: pelo seu formato condição/ação a regra é uma forma natural de pensar a solução de problemas: o que deve ser feito em determinada circunstância é diretamente traduzido em regras de produção.

Uniformidade: se for observado um sistema de produção percebe-se que todas as regras são escritas seguindo um mesmo padrão. Esta padronização permite pessoas não familiarizadas com o sistema representar/entender o seu conhecimento. Entretanto, paradoxalmente, pode ser um inconveniente se o usuário necessitar de uma maior flexibilidade.

As desvantagens apresentadas são:

Opacidade: característica resultante da modularidade e da uniformidade que faz com que seja difícil verificar a completude destes sistemas, bem como verificar os possíveis fluxos de processamento que levaram à solução do problema.

Ineficiência: em relação ao tempo de execução, resultante do número de regras a

serem combinadas, e do esforço de *matching*[2] necessário ao suporte de execução das regras.

Existe uma discussão quanto à classificação dos sistemas de produção. Alguns autores argumentam que os sistemas de produção devem ser classificados entre os métodos de representação de conhecimento, ao lado de outros formalismos como as redes semânticas e os quadros (*frames* em inglês), já que possibilitam uma aquisição fácil do conhecimento e permitem o armazenamento de forma coerente destas informações. Por outro lado, alguns autores defendem a idéia de que os sistemas de produção, enquanto base para o desenvolvimento de sistemas especialistas, constituem modelo computacional para a solução de problemas.

3.2 Sistemas Especialistas

Outro tipo de sistema que utiliza o formato de regras de produção como método de representação do conhecimento são os *Sistemas Especialistas (SE)*. Conforme [Bit96] o objetivo dos SE's é ao mesmo tempo mais restrito e mais ambicioso do que o objetivo dos modelos psicológicos: os SE's são concebidos para reproduzir o comportamento de especialistas humanos na resolução de problemas do mundo real, mas o domínio destes é altamente restrito.

Os sistemas especialistas constituem uma generalização dos sistemas de produção de Post, tendo sido desenvolvidos com o objetivo de uma maior facilidade de uso, bem como eficiência e expressividade, do que estes.

Conforme [Wat86] o conhecimento em um sistema especialista é *organizado* de uma forma que separa o conhecimento sobre o domínio do problema dos outros conhecimentos do sistema, tais como o conhecimento geral de como resolver o problema ou o conhecimento de como interagir com o usuário.

Um SE apresenta uma estrutura com três módulos: uma *Base de Regras*, uma *Memória de Trabalho* e um *Motor de Inferência*. A base de regras e a memória de trabalho constituem a *Base de conhecimento* do SE, onde o conhecimento sobre o domínio está representado. O motor de inferência é o mecanismo de controle do sistema

[2]Por *matching* entende-se a verificação das regras que se aplicam ao estado do problema, bem como quais regras antecedem ou sucedem outra regra.

que avalia e aplica as regras de acordo com as informações da memória de trabalho. A figura 3.1 mostra a arquitetura de um SE.

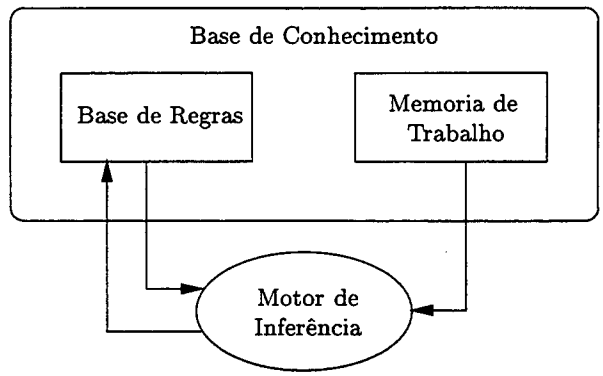


Figura 3.1: Arquitetura de um Sistema Especialista

A memória de trabalho - apenas uma seqüência de caracteres no modelo de Post -, no modelo generalizado, pode conter qualquer tipo de estrutura de dados. Mais do que estruturas de dados, a memória de trabalho dos sistemas especialistas devem respeitar um *Método de Representação do Conhecimento*, ou seja, uma linguagem formal e uma descrição matemática de seu significado. A lógica de primeira ordem é um exemplo de formalismo de representação de conhecimento.

A base de regras passa a conter condições que representam "perguntas" à representação de conhecimento da memória de trabalho. Estas perguntas - limitadas à comparação de caracteres do modelo de Post - podem ser de diferentes tipos, mas em geral envolvem variáveis a serem instanciadas e eventualmente algum tipo de inferência.

O motor de inferência controla a atividade do sistema. Esta atividade ocorre em ciclos, cada ciclo consistindo em três fases:

1. Correspondência de dados, onde as regras que satisfazem a descrição da situação atual são selecionadas;
2. Resolução de conflitos, onde as regras que serão realmente executadas são escolhidas - entre as regras que foram selecionadas na primeira fase - e ordenadas;
3. Ação, onde é realizada a execução propriamente dita das regras.

Conforme colocado em [Bit96] a chave para o desempenho dos SE's está no conhecimento armazenado em suas regras e em sua memória de trabalho. Este conhecimento deve ser obtido junto a um especialista humano do domínio e representado de acordo com as regras formais definidas para a codificação de regras no SE em questão. Este aspecto define um SE em duas partes: a ferramenta de programação que define o formato do conhecimento da memória de trabalho e das regras, além dos aspectos operacionais da sua utilização, e o conhecimento do domínio propriamente dito.

Devido a esta separação, atualmente, os SE's são desenvolvidos em geral a partir de *Arcabouços de Sistemas Especialistas* (ASE - do inglês, *Expert System Shells*). Tais ferramentas suportam todas as funcionalidades de um SE [Gev87], restando ao programador a tarefa de codificar o conhecimento especializado, de acordo com a linguagem de representação de conhecimento disponível. A existência de ASE's facilitou bastante a implementação dos SE's e é um dos fatores responsáveis pela sua disseminação.

Em geral os ASE's se limitam a oferecer uma única linguagem (ou formalismo) de representação do conhecimento. Alguns sistemas dispõem de mais de um formalismo, entretanto estes formalismos devem ser utilizados de forma isolada. Alguns poucos ASE's, possuem os chamados *sistemas híbridos de representação de conhecimento* [NL87] que, além de possuir diversos formalismos de representação, dispõem também de algoritmos de acesso que integram os conhecimentos representados nos diversos formalismos de maneira a permitir sua utilização de maneira integrada.

Nos itens seguintes são discutidos alguns aspectos referentes às funcionalidades existentes nos ASE's. Em particular são tratados os formalismos de representação do conhecimento e o motor de inferência.

3.2.1 Métodos de representação de conhecimento

A parte mais importante de um projeto de SE é a escolha do método de representação do conhecimento. A linguagem associada ao método deve ser suficientemente expressiva para permitir a representação do conhecimento a respeito do domínio escolhido de maneira completa e eficiente. A princípio, uma representação geral como a lógica seria suficientemente expressiva para representar qualquer tipo de conhecimento. No entanto, problemas de eficiência, facilidade de uso e a necessidade de expressar conhecimento incerto e incompleto levaram ao desenvolvimento de diversos formalismos de representação de conhecimento. A seguir são apresentados alguns formalismos de

representação de conhecimento mais utilizados.

Lógica

A lógica é a base para a maioria dos formalismos de representação de conhecimento, seja de forma explícita (como nos SE's baseados na linguagem Prolog), seja disfarçada na forma de representações específicas, que podem facilmente ser interpretadas como proposições ou predicados lógicos.

Mesmo os formalismos não lógicos têm, em geral, seu significado formal descrito através de uma especificação lógica de seu comportamento.

Redes Semânticas

O termo rede semântica é usado para descrever um conhecimento baseado numa estrutura de grafo e é utilizado para definir um conjunto heterogêneo de sistemas. Em última análise, a única característica comum a todos estes sistemas é a notação utilizada: uma rede semântica consiste em um conjunto de *nós* conectados por um conjunto de *arcos*. Os nós em geral representam objetos e os arcos relações binárias entre estes objetos. Mas os nós podem também ser utilizados para representar predicados, classes, palavras de uma linguagem, entre outras possíveis interpretações, dependendo do sistema de redes semânticas em questão.

A *herança de propriedades* através de caminhos formados por arcos é uma das principais características do método de representação por redes semânticas. Esta característica permite que propriedades de um nó sejam especificadas somente uma vez, no nível mais alto de uma hierarquia de conceitos, sendo herdadas por todos os conceitos derivados, implicando uma economia substâncial de memória.

Os algoritmos de herança utilizados em redes semânticas na forma de árvores são bastante simples e bastante eficientes, mas se heranças múltiplas forem permitidas, especialmente de arcos que definam exceções, o problema de determinação de caminhos de herança torna-se bastante complexo. Há ainda o fato de que, na presença de herança múltipla e exceções, a intuição sobre o que é uma política de herança coerente passa a ser discutível, levando diferentes sistemas a implementarem diferentes políticas de herança [THT87].

Quadros

Os quadros (*frames* em inglês), e sua variação, os roteiros (*scripts* em inglês), foram introduzidos para permitir a representação de estruturas internas dos objetos, mantendo a possibilidade de representar herança de propriedades como a rede semântica.

As idéias fundamentais destes métodos foram introduzidas por Marvin Minsky [Min75]. O método de quadros também está na origem das idéias que levaram às linguagens de programação orientadas a objetos. Os roteiros foram introduzidos por Schank e Abelson [SA75], [SA77], e consistem em sistemas de quadros especializados na descrição de seqüências de eventos.

Em geral, um quadro consiste de um conjunto de *atributos* (*slots* em inglês) que, através de seus valores, descrevem as características do objeto representado pelo quadro (por exemplo, os atributos fabricante, modelo, capacidade, com os valores: Eisenmann, AGV-C3PO, 5). Os valores atribuídos aos atributos podem também ser outros quadros, criando uma rede de dependência entre quadros. Os quadros são também organizados em uma hierarquia de especialização, criando uma outra dimensão de dependência entre eles.

Da mesma maneira que as redes semânticas, os sistemas baseados no método de quadros não são um conjunto homogêneo, no entanto, alguns pontos fundamentais, que dizem respeito ao mecanismo de inferência do formalismo, são compartilhados por estes sistemas. Um primeiro ponto comum é a herança de propriedades, que permite a especificação de uma classe de objetos através da declaração que esta classe é uma subclasse de uma outra que possui a propriedade em questão. A herança pode ser um mecanismo de inferência muito eficiente em domínios que apresentem uma taxonomia natural de conceitos.

Outro ponto comum aos sistemas baseados em quadros é o *raciocínio baseado em expectativas*. Um quadro contém atributos, e esses atributos podem ter valores típicos, ou a priori, os chamados *valores default*. Ao tentar instanciar um quadro para que corresponda a uma situação dada, o processo de raciocínio deve tentar preencher os valores dos atributos com informações disponíveis na descrição da situação. Caso alguma informação não esteja disponível, o processo de raciocínio deve buscar esta informação de forma adequada, através de valores previamente estipulados ou através da herança de valores de atributos.

Uma terceiro ponto é a chamada *ligação procedimental.* Além dos valores default, um atributo pode ser associado a um procedimento que deve ser executado quando certas condições forem satisfeitas, tipicamente: ao ser criado o atributo (*demon if-create*), ao ser lido o valor do atributo (*demon if-necessary*), ao ser modificado o valor do atributo (*demon if-modify*), ou ao ser destruído o valor do atributo (*demon if-delete*).

3.2.2 Motor de inferência

As principais características dos motores de inferência disponíveis em um ASE dizem respeito às seguintes funcionalidades: método de raciocínio, estratégia de busca e resolução de conflito. Cada uma destas funcionalidades é discutida a seguir.

Existem basicamente dois modos de raciocínio aplicados a regras de produção: *encadeamento para frente* (ou progressivo, *forward* em inglês) e *encadeamento para trás* (ou regressivo, *backward* em inglês).

No encadeamento para frente, também chamado encadeamento dirigido por dados, a parte esquerda da regra é comparada com a descrição da situação atual, contida na memória de trabalho. As regras que satisfazem esta descrição têm sua parte direita executada, o que em geral significa a introdução de novos fatos na memória de trabalho.

No encadeamento para trás, também chamado de encadeamento dirigido por objetivos, o comportamento do sistema é controlado por uma lista de objetivos. Um objetivo pode ser satisfeito diretamente por um elemento da memória de trabalho, ou podem existir regras que permitam inferir algum dos objetivos correntes, isto é, contenham a descrição do objetivo em suas partes direitas. As regras que satisfazem esta condição têm suas partes esquerdas adicionadas à lista de objetivos correntes.

O tipo de encadeamento normalmente é definido de acordo com o tipo de problema a ser resolvido. Problemas de planejamento, projeto e classificação, tipicamente utilizam encadeamento para frente, enquanto problemas de diagnóstico, onde existem apenas algumas saídas possíveis, mas um grande número de estados iniciais, utilizam encadeamento para trás.

Uma característica importante do modo de raciocínio se refere a monotonicidade ou não do método de inferência. Sistemas monotônicos não permitem a revisão dos fatos, isto é, uma vez declarado verdadeiro, o fato não pode mais tornar-se falso. Sistemas não

monotônicos, por outro lado, permitem a alteração dinâmica dos fatos. O preço desta capacidade é a necessidade de um mecanismo de revisão de crenças, pois uma vez que um fato, antes verdadeiro, torna-se falso, todas as conclusões, já tiradas anteriormente, baseadas neste fato, devem tornar-se falsas.

Uma vez definido o tipo de encadeamento, o motor de inferência necessita de uma estratégia de busca para guiar a pesquisa na memória de trabalho e na base de regras. Este tipo de problema é conhecido como busca em um espaço de estados. Este tópico foi um dos primeiros estudados em IA e descrições detalhadas dos algoritmos de busca podem ser encontrados em livros texto de IA, como Charniak e McDermott [CM85], Nilsson [Nil71], Rich [Ric83], Winston [Win84], entre outros.

Ao terminar o processo de busca, o motor de inferência possui um conjunto de regras que satisfazem à situação atual do problema, o chamado *conjunto de conflito*. Caso este conjunto seja vazio, a execução é terminada, caso contrário é necessário escolher as regras que serão realmente aplicadas e em que ordem.

Os métodos de resolução de conflitos mais utilizados ordenam as regras de acordo com os seguintes critérios: prioridades atribuídas estaticamente; características da estrutura das regras como complexidade, simplicidade e especificidade; características dos dados associados as regras como o tempo decorrido desde sua obtenção, sua confiabilidade, ou seu grau de importância; e, finalmente, a seleção ao acaso (aleatória).

Em geral a utilização de um destes critérios é insuficiente para resolver conflitos. Neste caso o ASE pode combinar mais de um método na forma de método primário, secundário, etc.

A seguir é apresentado o FASE (Ferramenta para construção de Arcabouços de Sistemas Especialistas), um ASE de domínio público, implementado na linguagem de programação Common Lisp, cujos fontes estão disponíveis e podem ser obtidos via Internet (*ftp://ftp.lcmi.ufsc.br*).

3.3 O Sistema FASE

O sistema FASE [BM93] [Bit95] consiste em uma ferramenta para geração de arcabouços de sistemas especialistas baseados em regras e fornece o suporte computacional para o desenvolvimento deste trabalho. O objetivo da ferramenta é integrar em um

único sistema várias funcionalidades para desenvolvimento de sistemas especialistas oferecendo diversos recursos como representação de conhecimento, estratégias de controle, técnicas de resolução de conflito e de raciocínio incerto, de maneira a permitir a geração de arcabouços de sistemas especialistas adaptados a problemas específicos. O sistema é implementado em Common Lisp [SJ84] e consiste de uma biblioteca de funções, distribuídas em diversos arquivos, um para cada funcionalidade de sistema especialista disponível no sistema. Os arquivos desta biblioteca podem ser utilizados como blocos para implementar um arcabouço de sistema especialista cujas funcionalidades são adaptadas às características do problema a ser resolvido.

A biblioteca de representação de conhecimento é formada por uma base de conhecimento interna e um conjunto de funções de manipulação, que implementam formalismos de representação de conhecimento. Os três formalismos de representação de conhecimento discutidos anteriormente (lógica, redes semânticas e quadros) estão disponíveis.

Ao lado do formalismo de representação de conhecimento a ferramenta também oferece duas estratégias de controle: inferência para frente e inferência para trás. A ferramenta fornece ainda uma biblioteca de representação de incerteza, permitindo a escolha das teorias de raciocínio incerto (como teoria de possibilidades [Zad78] e Mycin), e uma biblioteca de resolução de conflitos (com a possibilidade de utilização dos seguintes critérios: ordem das regras, caso especial, regra mais breve, regra mais específica).

3.3.1 As estratégias de controle

Como foi colocado no item anterior encontram-se disponíveis no FASE dois tipos de estratégias de controle para encadeamento de regras: para frente e para trás. Na estratégia para frente formula-se o problema sob a forma de uma situação inicial, armazenando-o na memória de trabalho. Os padrões esquerdos das regras são comparados com o conteúdo da memória de trabalho. As regras que forem satisfeitas serão selecionadas e ordenadas em uma lista através do método de resolução de conflito, os padrões direitos associados das regras serão combinados com as instâncias resultantes da comparação e utilizados como novos fragmentos de conhecimento. O encadeamento progressivo é adequado a problemas orientados a dados.

Na estratégia para trás o problema consiste em um objetivo a ser atingido. Armazena-se o objetivo na memória de trabalho e comparam-se os padrões direitos

das regras com o objetivo. As regras cujas conclusões corresponderem ao objetivo, serão selecionadas e ordenadas segundo o método de resolução de conflito adotado. Os padrões esquerdos das regras selecionadas serão armazenados na memória de trabalho, como novos objetivos. As aplicações típicas deste tipo de estratégia são os problemas de diagnósticos e detecção de falhas.

3.3.2 Os formalismos de representação do conhecimento

Conforme foi colocado, encontram-se disponíveis no FASE três tipos de formalismo: lógica, quadros e rede semânticas. Cada formalismo é definido por uma linguagem formal, um mecanismo de raciocínio e uma interface entre o formalismo e a base de regras. A seguir serão discutidos somente os aspectos referentes ao formalismo de quadros.

Os quadros consistem de uma hierarquia de estruturas de dados. Cada quadro possui um conjunto de atributos onde um valor pode ser associado a cada atributo. Este valor pode ser qualquer informação primitiva sobre o conceito representado pelo quadro, ou um ponteiro para um outro quadro.

Os três mecanismos de inferência do formalismo de quadros, descritos na página 39 estão disponíveis, são eles:

- herança de valores de atributos através da hierarquia;
- valores previamente escolhidos (*default*);
- ligação procedural para permitir a execução de uma função LISP externa ao formalismo.

A ligação procedimental é um mecanismo especialmente útil porque permite associar funções aos dados armazenados pelo formalismo. Assim, quando da manipulação de um atributo (criação, modificação, leitura e destruição) é possível executar uma função previamente definida.

As funções são associadas aos atributos através de um saci (do inglês, *demon*) que fica monitorando os procedimentos sobre o atributo. Quando a ação que o *demon* esta monitorando acontece a função previamente construída é invocada.

Os procedimentos monitorados e os respectivos *demons* são: i) criação de atributo (*demon if-create*); ii) leitura de atributo (*demon if-necessary*); iii) modificação de

atributo (*demon if-modify*) e, iv) destruição de atributo (*demon if-delete*).

O formalismo de quadros é adequado para domínios estruturados taxonomicamente, onde o mecanismo de herança pode ser explorado eficientemente.

3.3.3 O funcionamento do FASE

O funcionamento do arcabouço de sistemas especialistas, construído a partir das bibliotecas do FASE pode ser descrito, de forma simplificada, pelo seguinte procedimento:

- 1. Para cada regra :

  1.1. Selecionar o lado esquerdo ou o lado direito, de acordo com o tipo de estratégia de controle escolhido.

  1.2. Utilizar cada padrão do lado escolhido como uma pergunta à memória de trabalho, de acordo com o tipo de representação de conhecimento associado ao padrão, obtendo como resposta uma lista de instâncias.

  1.3. Caso todos os padrões resultem em listas de instâncias não vazias realizar a combinação destas listas obtendo uma lista única. Associar esta lista de instâncias à regra em questão.

- 2. Utilizando as listas de instâncias associadas às regras no passo anterior, escolher quais regras serão realmente executadas e em que ordem, de acordo com o método de resolução de conflito escolhido.

- 3. Para cada regra a ser executada:

  3.1. Combinar cada instância resultante aos padrões presentes no lado oposto da regra e utilizar estas combinações como novos fragmentos de conhecimento a serem armazenados na memória de trabalho de acordo com os tipos de representação de conhecimento associados aos padrões.

Disparo paralelo das regras O procedimento descrito pelas etapas 1, 2 e 3 é realizado para cada uma das regras existentes no sistema especialista em um mesmo ciclo de inferência. Todas as instâncias das regras são disparadas neste ciclo, caracterizando assim um disparo paralelo.

A priori isto pode parecer incomum visto que na etapa de resolução de conflitos as regras são ordenadas antes de disparar. Entretanto, do ponto de vista do ciclo de inferência como um todo, o conjunto de regras questiona o mesmo conteúdo da memória de trabalho (que já está definida no início do ciclo) e depois de selecionadas e ordenadas (internamente) são executadas; assim, do ponto de vista da memória de trabalho todas atuaram sobre o mesmo estado inicial e, ao final do ciclo, um novo estado foi atingido. Para cada ciclo de inferência o disparo das regras é indivisível.

3.4 Conclusão

Neste capítulo foram introduzidos os sistemas a base de regras. Inicialmente foram apresentados os Sistemas de Produção, propostos por Post em 1936. Foram descritos os componentes que formam os Sistemas de Produção, bem como as vantagens e desvantagens de tais sistemas.

Os Sistemas Especialistas, que são uma generalização dos Sistemas de Produção, também foram descritos neste capítulo. Conforme foi visto, este mecanismo é constituído de uma estrutura com três módulos (base de regras, memória de trabalho e motor de inferência). As características principais do motor de inferência (estratégia de encadeamento das regras, métodos de resolução de conflitos e monotonicidade), bem como três métodos de representação de conhecimento na memória de trabalho (lógica, redes semânticas e quadros) foram apresentados.

Por último foi apresentado o FASE, um arcabouço para a construção de Sistemas Especialistas que serve de suporte computacional para a construção do simulador desenvolvido. Em relação ao FASE foram apresentadas as estratégias de controle da inferência que ele admite, os formalismos de representação que a ferramenta suporta e a forma como se dá o funcionamento do mesmo.

Capítulo 4

Metodologia da Simulação

Neste capítulo são apresentados os principais aspectos relativos à metodologia da simulação. Inicialmente são colocados alguns aspectos referentes à representação de redes de Petri utilizando sistemas de produção. A seguir, é apresentado o mapeamento do modelo de rede de Petri para o sistema FASE, onde os problemas em relação ao comportamento do modelo de rede de Petri e de Sistemas de Produção são discutidos. Na seqüência a forma geral do simulador desenvolvido é apresentada.

4.1 Redes de Petri e Sistemas de Produção

A rede de Petri (apresentada no capítulo 2), também pode ser representada através de um sistema de regras. A principal característica que permite tal analogia é a representação da evolução de estados da rede de Petri, descrita por transições com a forma

se condição *então* ação.

De uma maneira geral, nesta abordagem de representação, as transições com seus lugares de entrada e saída (a estrutura da rede), são representadas por regras e a marcação da rede é representada pela memória de trabalho. De forma mais específica, pode-se visualizar as seguintes correspondências:

- o conjunto de transições da rede, com suas condições e ações, representadas, respectivamente, pelos vetores $Pre(.,t)$ e $Post(.,t)$ (apêndice A, página 93), constituem a base de regras;

- os lugares da rede de Petri e suas relações com as transições, juntamente com a marcação inicial, podem ser vistos como a base de fatos inicial;
- o jogador da rede de Petri, que determina qual transição é disparada: se as transições são paralelas, a ordem de disparo é indiferente; se as transições estão em conflito efetivo, apenas uma poderá ser disparada. O jogador é, portanto, análogo ao motor de inferência de um sistema de produção.

A tabela 4.1 mostra esta correspondência:

Rede de Petri	Sistemas de Regras
Lugar de Entrada	Condição da regra
Lugar de Saída	Ação da regra
Marcação Inicial	base de fatos inicial
Transição	Regra
Jogador	Motor de Inferência

Tabela 4.1: Correspondência Rede de Petri × Sistema de Regras

Os aspectos dinâmicos da rede de Petri são descritos utilizando as chamadas *regras de reescrita*, que obedecem a seguinte sintaxe [CV97]:

$$\mu(t) : p_i^m \rightarrow p_j^n$$

onde p_i e p_j definem os nomes dos lugares de entrada e saída, respectivamente, e os índices superiores (m e n) definem o número de fichas que devem ser retiradas e colocadas no lugar, respectivamente. Por exemplo poderíamos ter

$$\mu(c) : p_2^3 \rightarrow p_3$$

como representação da transição c da figura A.2, apresentada no apêndice A.

As vantagens da utilização de um sistema de regras para a descrição da rede de Petri são a concorrência (várias regras podem ser disparadas em qualquer ordem) e a escolha (seleção de uma regra em um conjunto em conflito).

Embora haja uma correspondência no que diz respeito à representação do modelo de rede de Petri utilizando sistema de regras, existem diferenças fundamentais no que se refere às semânticas vinculadas à transição e à regra em cada um dos modelos.

Em um sistema de regras de produção, quando uma regra é satisfeita por várias instâncias, todas as instâncias estão em princípio habilitadas para disparar a regra. Já

em uma rede de Petri, se existem diversos objetos em um determinado lugar de entrada de uma transição, em geral apenas um será disparado.

A seguir são colocadas outras questões referentes ao mapeamento do modelo de rede de Petri para o sistema FASE e mais adiante as soluções para os problemas encontrados.

4.2 Redes de Petri e o Sistema FASE

Dado o fato da utilização de um arcabouço de sistema especialista para a simulação da rede de Petri, faz-se necessário algumas definições referentes à forma de representação de conhecimento e à estratégia de encadeamento das regras a serem utilizadas para a representação e funcionamento da simulação.

Em função dos elementos que compõem a rede de Petri a Objetos (lugares, transições, classes de objetos, instâncias de objetos, etc), o formalismo de representação de conhecimento adotado são os quadros. Conforme visto na item 3.2.1 os quadros já implementam alguns conceitos de orientação a objetos, como os conceitos de hierarquia e herança. Além disso, as classes de objetos e suas instâncias, declaradas na especificação da rede de Petri podem ser mapeadas diretamente para este formalismo.

Quanto ao tipo de encadeamento a ser adotado, a escolha recai sobre a estratégia para frente, visto que o sistema especialista que implementa a simulação define um sistema dirigido por dados, neste caso, a marcação inicial da rede. Desta forma, a cada disparo das regras uma nova marcação para a rede de Petri é obtida, habilitando novas regras e simulando o comportamento dinâmico da rede.

No item 2.2 foi definido o modelo de rede de Petri a Objetos como uma 9-upla $< C_{class}, P, T, V, Pre, Post, A_{tc}, A_{ta}, M_0 >$ e no item 4.1 foram colocadas as correspondências entre o modelo de rede de Petri e um sistema de regras. A tabela 4.2 apresenta o mapeamento de cada um dos elementos.

Através da tabela pode-se perceber que os lugares, a marcação, as classes e mesmo as instâncias das classes, são transpostas para o sistema especialista como elementos formadores da base de fatos. Os demais elementos da 9-upla são de alguma forma mapeados nas regras, alguns constituindo os antecedentes e conseqüente das regras, outros em elementos de filtragem para aplicação das mesmas, ou variáveis para comparação de atributos de objetos.

Rede de Petri a Objetos		FASE
Estrutura	P - lugares	base de fatos
	T - transições	base de regras
	Pre	antecedente das regras
	Post	conseqüente das regras
	V	variáveis das regras
	A_{tc}	filtro das regras
	A_{ta}	conseqüente das regras
Marcação	objetos	base de fatos
Classes	C_{class}	base de fatos
Jogador		motor de inferência

Tabela 4.2: Mapeamento do modelo de rede de Petri a Objetos no Sistema FASE

Além dos elementos apresentados na tabela 4.2 existe a necessidade de representar os dados externos da simulação (sensores) e também utilizar algum tipo de representação do tempo. Na ferramenta proposta, estes dois itens são representados na base de fatos. A seguir, na seção 4.3, será tratada a forma de representação e atualização destes itens.

O mapeamento descrito pela tabela 4.2 necessita encontrar meios de adaptar corretamente os modelos para resolver os seguintes problemas que se apresentam:

(i) *Conteúdo dos lugares*: resultante da diferença semântica entre redes de Petri (os objetos nos lugares representam recursos consumíveis), e sistemas de regras (o disparo das regras não retira a instância utilizada da memória de trabalho) - ver item 4.1;

(ii) *Inconsistência de dados*: causado pelo disparo paralelo de regras no sistema FASE;

(iii) *Resolução de conflitos*: inerente à implementação do simulador, visto que, de alguma maneira, é necessário informar ao mecanismo de inferência qual objeto entre vários possíveis deve ser utilizado em um disparo de transição e, quando for o caso, com qual transição;

(iv) *Pseudo-disparo das transições*: como implementar o mecanismo de pseudo-disparo e o mecanismo de recuperação da imprecisão.

Cada problema possui a sua especificidade (adaptação entre os modelos, necessidade própria do simulador, característica do FASE, modelo de rede de Petri a ser

simulado), entretanto, todos devem ser resolvidos de forma integrada, já que a partir da solução destes problemas é possível obter um simulador de redes de Petri com Marcação Imprecisa.

O problema da resolução de conflitos pode ser definido a partir de duas questões: a) como selecionar um objeto para ser disparado por uma transição quando mais de um objeto está habilitando a regra; b) como selecionar a transição que deve disparar quando existe apenas um objeto habilitando duas transições. A solução do conflito existente na primeira questão será chamado de *resolução de conflito das fichas* e da segunda *resolução de conflito das transições.*

Cabe aqui um parênteses para chamar a atenção ao termo *resolução de conflitos.* Na leitura que se segue deve-se ter especial atenção, visto que este termo é comum tanto ao formalismo de redes de Petri quanto aos sistemas especialistas; entretanto, possuem significados diferentes em cada um destes.

Supondo, numa rede de Petri, duas transições em conflito efetivo (apêndice A) habilitadas por um único objeto, ao disparar a primeira transição com o objeto a outra não dispara, pois o disparo representa o consumo de objetos. Entretanto, num sistema de regras todas as regras habilitadas em um determinado ciclo podem disparar. Portanto este conflito (que corresponde ao conflito entre transições), também deve ser tratado para se obter o comportamento da rede de Petri durante a execução do sistema especialista.

O problema da inconsistência dos dados é causado pelo disparo paralelo das regras no sistema FASE (item 3.3.3). O problema pode ser percebido quando duas transições em seqüência são disparadas paralelamente pelo simulador. Observe o caso simples apresentado na figura 4.1.a. Na rede de Petri da figura percebe-se que tanto *trans_1* quanto *trans_2* estão sensibilizadas e portanto podem ser disparadas de forma paralela, levando à nova marcação, mostrada pela figura 4.1.b.

Em um ciclo de inferência qualquer do sistema FASE, o motor de inferência verifica a base de fatos (inicialmente, $lug\_a = 1$ e $lug\_b = 1$) e dispara as transições numa dada ordem (por exemplo, *trans_1*, *trans_2*) . Quando a primeira transição dispara (*trans_1*), é retirada uma ficha do lugar de entrada (*lug_a*) e adicionada uma ficha no lugar de saída (*lug_b*), o que significa escrever $lug\_a = 0$ e $lug\_b = 2$. Como a modificação pelo disparo da *trans_1* de fato será conhecida somente no final do ciclo de inferência, a segunda transição é disparada (*trans_2*) com a base de fatos conhecida no início

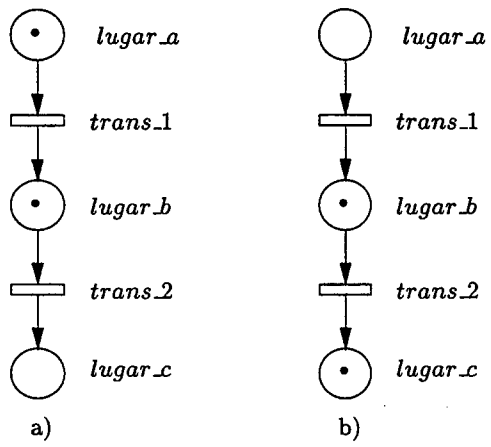


Figura 4.1: Exemplo de inconsistência de dados

do ciclo (relembrando, *lug_a = 1* e *lug_b = 1*): uma ficha é retirada de *lug_b* o que resulta em *lug_b = 0* e colocada em *lug_c*. Após este ciclo de inferência a base de fatos possuirá os seguintes dados: *lug_a = 0*, *lug_b = 0* e *lug_c = 1*. Desta forma, a ficha que foi adicionada em *lug_b* pela *transição_1* é perdida pois somente a última modificação sofrida por cada lugar é que é percebida. Fica, assim, caracterizada uma inconsistência nos dados, devendo-se, portanto, providenciar mecanismos adequados para evitar este problema.

O problema do pseudo-disparo de transições, assim como o problema de resolução de conflitos é inerente à construção do simulador visto que o objetivo é desenvolvê-lo com aptidão para representar redes de Petri com Marcação Imprecisa. A solução de tal problema passa pela adaptação do jogador de rede de Petri descrito pela figura 2.6 para o sistema especialista que faz a simulação da rede de Petri (ver itens 4.2.1 e 4.3).

A solução dos quatro problemas descritos acima determinam a estrutura e o comportamento do simulador desenvolvido, uma vez que o sistema especialista para simular o modelo de rede de Petri a Objetos com Marcação Imprecisa será constituído de uma base de fatos e quatro bases de regras.

Nos itens seguintes serão apresentadas as soluções para os quatro problemas levantados anteriormente.

4.2.1 Os problemas de monotonicidade e inconsistência na base de fatos

Dois problemas se apresentam em relação ao comportamento do lugar quando do disparo das transições: o conteúdo dos lugares e a inconsistência da base de fatos. O primeiro se refere à necessidade de expressar o consumo do objeto e o segundo diz respeito à perda de informação quando um lugar é manipulado duas vezes em um mesmo ciclo de inferência. A chave para a solução destes problemas é dotar a representação do lugar no simulador de um comportamento condizente com o modelo de rede de Petri.

A representação da rede de Petri no sistema especialista está baseada nos lugares. Desta forma, para cada lugar é definido um quadro, contendo uma série de atributos como o conteúdo do lugar, as transições posteriores e anteriores ao lugar e um atributo *interface*, dentre outros.

Cada lugar apresenta depósitos distintos (atributos) para a colocação dos objetos que entram no lugar e para os que saem do lugar. O depósito de entrada é único para todas as transições de entrada do lugar. Quanto aos depósitos de saída, há um para cada transição de saída do lugar.

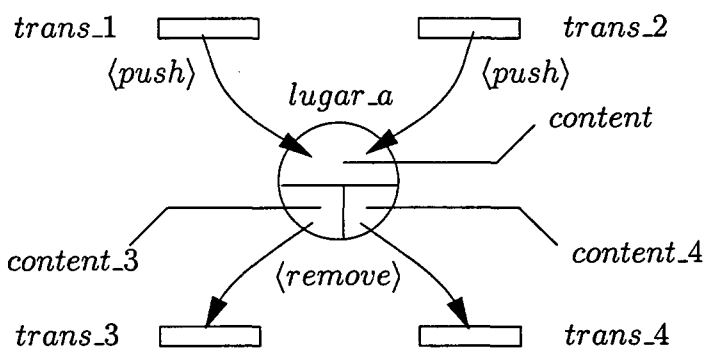


Figura 4.2: Representação dos lugares no FASE

Observe a figura 4.2. Os objetos colocados no *lugar_a* tanto pela transição *trans_1* quanto por *trans_2* serão armazenados em um único depósito (denominado *content*). Já os objetos que sairão do lugar, pelo disparo de *trans_3* ou pelo disparo de *trans_4*, estarão armazenados nos depósitos específicos para cada uma delas (*content_t3* e *content_t4*, respectivamente). A retirada do objeto do depósito de entrada e colocação em um dos depósitos de saída são realizadas na etapa de resolução de conflito das fichas, que será explicada no item 4.2.2.

Os atributos do lugar que armazenam os objetos que entram e saem não são modificados diretamente. Todos os dados são enviados para o atributo *interface*, onde são manipulados pelo *demon* vinculado a este atributo. Como visto na seção 3.3.2 o FASE permite a monitoração dos procedimentos executados sobre os atributos dos quadros. No atributo *interface* é utilizado o *demon if-modify*. Assim, cada vez que se desejar manipular o conteúdo dos lugares deve-se fazê-lo através do atributo *interface*, o qual possui o *demon* associado.

A manipulação dos objetos pelo *demon* é feita a partir de uma pequena linguagem, constituída das seguintes primitivas:

- `(push <objeto>)` adiciona o *objeto* na lista de objetos do depósito entrada *content* de um dado lugar.
- `(move <objeto> <content_j>)` remove o objeto da lista de objetos do depósito de entrada *content* e coloca no depósito de saída *content_j* de determinado lugar.
- `(remove <objeto> <content_j>)` remove o objeto do depósito de saída *content_j* de um dado lugar.
- `(uncert <objeto>)` remove o objeto do depósito de entrada *content* e coloca no depósito de objetos disparados de forma imprecisa *content_unc*. É utilizado no pseudo-disparo das transições (ver seção 4.2.3).

A primitiva *push* ao colocar um objeto em determinado lugar, primeiro verifica a lista de objetos já existentes e então adiciona este à lista. O *demon* permite que a verificação seja feita no momento exato da aplicação da regra, desta forma, mesmo os objetos colocados em um mesmo ciclo de inferência são percebidos, o que impede a inconsistência na base de fatos.

Para permitir a noção de consumo de recurso no disparo de uma transição, obtendo com isto uma semântica não monotônica para o sistema especialista, é utilizada a primitiva *remove*. Esta primitiva é utilizada na ação da regra, removendo o objeto do lugar em que se encontra, fazendo com que um fato antes verdadeiro (a presença do objeto na entrada da transição) se torne falso, no momento de disparo da própria regra.

4.2.2 A resolução de conflito das fichas e transições

Para o caso do disparo normal das transições (lembre-se que a rede de Petri com Marcação Imprecisa define dois tipos de disparo) é necessário fazer a resolução de

conflito das fichas e das transições. Para implementar este mecanismo corretamente é feita a separação da base de regras do sistema. Assim, para o disparo normal das transições, define-se uma base de regras responsável pela resolução de conflito das fichas e outra para o disparo da transição propriamente dito.

Esta solução é utilizada em conjunto com a linguagem definida para a manipulação dos objetos, através do *demon* vinculado ao atributo *interface*. A seleção da ficha para o disparo de uma dada transição faz a retirada do objeto do depósito de entrada do lugar (indicado por *content* na figura 4.3) e coloca o objeto no depósito de saída para a transição (por exemplo *content_t3* na figura 4.3). Esta tarefa é realizada pela primitiva *move*, apresentada anteriormente.

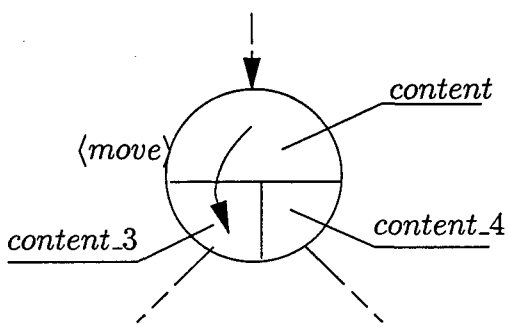


Figura 4.3: Atuação da primitiva *move*

Portanto, no disparo de determinada transição duas regras são utilizadas: a primeira faz a resolução de conflitos das fichas e a segunda faz o disparo da transição propriamente dito. Vale salientar que estas regras são executadas em seqüência, constituindo assim duas etapas. A etapa de resolução de conflito das fichas faz o "gatilho" da transição e a etapa seguinte faz o disparo, colocando o objeto escolhido no lugar de saída e retirando do lugar de entrada (no caso de disparo normal).

A utilização de duas etapas de regras para o disparo normal de uma transição fica transparente para o usuário. Isto porque quando o controle da simulação é feito por ele, somente a resolução de conflito das fichas lhe é apresentada. Uma vez escolhida(s) a(s) ficha(s) (retirada da ficha de *content* e colocação em *content_j*), o disparo é realizado automaticamente sem nova consulta ao usuário.

Deve-se salientar que a resolução de conflito das fichas (e somente ela) é implementada através de um mecânismo proprio (o qual foi apresentado acima). A resolução de conflito das transições utiliza o próprio método de resolução de conflito do FASE.

Isto acontece porque a resolução de conflito em sistemas especialistas é implementada para escolher qual regra deve disparar primeiro quando mais de uma está habilitada. Portanto, somente a resolução de conflito entre transições é resolvida diretamente pelos mecanismo de resolução de conflito do FASE; a resolução de conflitos das fichas deve realmente ser feita em separado.

Quando a "simulação" está sendo realizada de forma gráfica, através de um papel, ou mesmo, apenas mentalmente, a escolha de qual transição será disparada e qual objeto será usado para tal, acontece de forma natural e automática pelo cérebro humano.

Entretanto, quando da implementação de um simulador, a forma de escolher tanto a transição a ser disparada, quanto o objeto que será utilizado, deve estar definida a partir de alguma estratégia. Ou seja, deve-se expor explicitamente o mecanismo pelo qual a escolha deve ser feita.

Quando determinada regra possui uma marcação não nula nos lugares de entrada, é possível que esta seja disparada. Neste ponto, sabe-se apenas que as condições da regra foram satisfeitas, sem o conhecimento de qual e quantas n-uplas de objetos diferentes estão fazendo isto. A questão é então diferenciar cada uma das fichas que estão sensibilizando a regra, para permitir que apenas uma delas seja disparada.

Como colocado, a resolução de conflito das fichas é implementada utilizando um mecanismo específico. Tal mecanismo permite que sejam adotadas quatro estratégias diferentes para a resolução de conflito das fichas: *(i)* escolha por prioridade atribuída aos objetos; *(ii)* escolha pela primeira ficha encontrada na lista; *(iii)* escolha aleatória e; *(iv)* escolha pelo usuário.

As três primeiras estratégias permitem a escolha automática do objeto, sem a interferência do usuário no momento da simulação. A estratégia *(iv)* dá ao usuário o controle da simulação permitindo que este escolha quais transições devem disparar e com quais objetos.

Como colocado, para fazer a resolução de conflito das transições é utilizado o próprio mecanismo de resolução de conflito do FASE, juntamente com o *demon if-modify* associado ao atributo *interface*.

Entre as diferentes estratégias de resolução de conflito definidas pelo FASE (item 3.3), para implementar a resolução de conflito entre transições, a ferramenta utiliza a que classifica as regras pela ordem de entrada, o que permite definir uma

prioridade estática às transições. As transições de mais alta prioridade devem ser declaradas antes das transições de prioridade mais baixa.

Caso haja conflito entre duas regras por um mesmo objeto, a regra ordenada em primeiro pelo FASE tem o direito de uso sobre o objeto e, caso ela seja disparada, a outra regra não será mais disparada.

4.2.3 O pseudo-disparo de transições

O pseudo-disparo é implementado através de dois conjuntos de regras específicos para tal fim. Com estes conjuntos ficam definidos os quatro conjuntos de regras necessários para implementação do simulador.

Os conjuntos de regras definidos para o pseudo-disparo devem providenciar a criação e a propagação da imprecisão ao longo da rede simulada. A criação da imprecisão acontece quando a marcação é precisa, mas a interpretação é incerta ($\eta = incerto$), e a propagação da imprecisão acontece quando a marcação já é imprecisa.

Vale ressaltar que as regras de pseudo-disparo não estão sujeitas a nenhum tipo de resolução de conflito, ou seja, se duas regras indicam que um mesmo objeto, ou n-upla de objetos deve ser disparada de forma imprecisa, tanto uma como a outra devem ser disparadas.

O primeiro destes conjuntos de regras (denominado regras de criação do pseudo-disparo), é responsável pela criação da imprecisão. Estas regras fazem a verificação da condição de disparo impreciso, definido na *função de autorização*.

Uma vez satisfeita a condição de disparo impreciso, a regra utiliza o *demon* vinculado ao atributo *interface* para retirar o objeto a ser disparado do atributo *content* e colocá-lo no atributo *content_unc*, responsável pelo armazenamento de objetos imprecisos em cada lugar da rede. Além disso, um atributo auxiliar, denominado *"fire"* é utilizado para indicar que o objeto ainda não foi propagado pelo pseudo-disparo da transição, o que deve ser feito pelo segundo conjunto de regras de pseudo-disparo.

A retirada do objeto do atributo *content* e colocação em *content_unc*, bem como a modificação do atributo *fire*, são realizadas pelo *demon* associado ao atributo *interface* através da primitiva *uncertain*, definida no item 4.2.1.

O segundo conjunto de regras definido para o disparo impreciso das transições, encarrega-se de propagar os objetos que se encontram em *content_unc*, colocando os mesmos nos lugares de saída da transição (mais especificamente no depósito *content_unc* destes lugares de saída), sem entretanto retirá-los do lugar de entrada (tal conjunto é denominado regras de propagação do pseudo-disparo).

Assim, fica caracterizada a existência de dois depósitos de entrada para cada transição: o primeiro (já explicado no item 4.2.1), é utilizado para armazenar os objetos precisos da rede simulada; o segundo para armazenar os objetos imprecisos.

Além das operações descritas acima, o segundo conjunto de regras de pseudo-disparo modifica o atributo *fire*, indicando que o objeto já foi propagado, evitando que no ciclo seguinte isto se repita.

Deve-se ter presente que as regras que implementam o pseudo-disparo são definidas somente para as transições cuja *função de autorização* define as condições para o disparo incerto. Isto significa que transições cuja *função de autorização* está completa (isto é, definem condições tanto para o disparo normal, quanto para o pseudo-disparo), apresentam 4 regras para implementá-la no sistema desenvolvido. Já as transições que definem apenas as condições para o disparo normal apresentam somente 2 regras associadas.

Para retornar a rede simulada a um estado preciso, é realizado um procedimento de recuperação da imprecisão (implementado através de funções Lisp) que, a cada ciclo de inferência, analisa a necessidade de chamar o algoritmo de recuperação da precisão.

4.3 A Estrutura do Simulador

A estrutura do simulador proposto é definida a partir da solução dos problemas apresentados no item 4.2. Assim, a ferramenta desenvolvida apresenta uma estrutura em que cada ciclo de inferência é composto de quatro etapas. Cada uma destas etapas associa a base de fatos do sistema a um dos quatro conjuntos de regras definidos anteriormente (regras de resolução de conflito das fichas, regras de disparo normal, regras de criação da imprecisão e regras de propagação da imprecisão).

A figura 4.4 apresenta os possíveis fluxos de execução das regras. O conjunto de regras a ser executado para cada transição, em um ciclo qualquer de inferência, varia

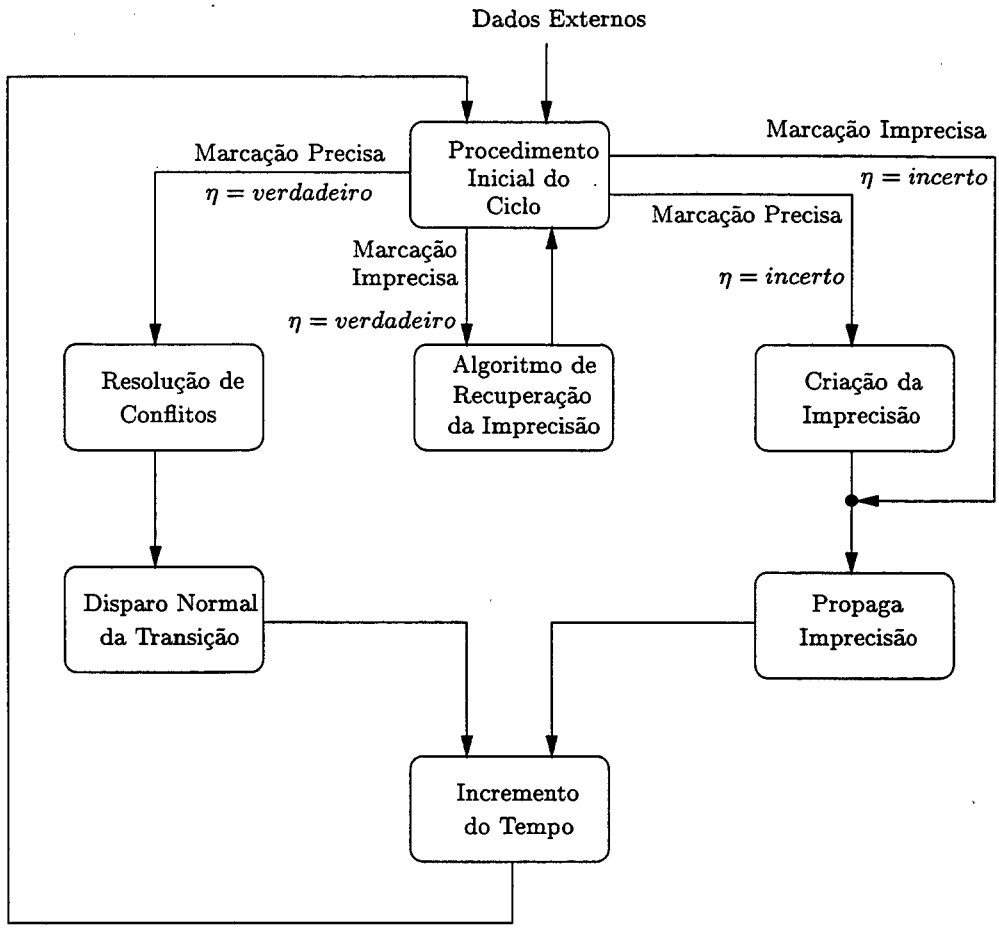


Figura 4.4: Forma geral do simulador

com o estado da marcação e com a interpretação associada. Por sua vez, a interpretação é dependente dos dados externos da simulação. A seguir, de maneira análoga à mostrada na página 26 e figura 2.6, é apresentado o conjunto de regras possíveis de serem executados para determinada transição de acordo com a variação da marcação (precisa ou imprecisa) e da interpretação (verdadeira ou incerta) no sistema FASE. Se a transição:

- está habilitada por uma marcação precisa e a interpretação é verdadeira ($\eta = verdadeiro$): a transição deve ser disparada de forma precisa, assim, primeiro passa pela resolução de conflito das fichas e em seguida é realizado o disparo normal;
- está habilitada de forma precisa e a interpretação é incerta ($\eta = incerto$): o pseudo-disparo é executado e a imprecisão é criada, desta forma, primeiro será executada a regra de criação da imprecisão e a seguir a regra de propagação da

imprecisão;

- é habilitada por uma marcação imprecisa e a interpretação é incerta, t é pseudo-disparada (propagação da imprecisão), neste caso, somente as regras de propagação da imprecisão atuam visto que a marcação já é imprecisa;
- é habilitada por uma marcação imprecisa (o que significa que alguma dedução já foi feita) e a interpretação é verdadeira (informação chegando do sistema): o *algoritmo de recuperação* é chamado para tentar retornar a marcação para um estado preciso.

Na figura 4.4, o chamado procedimento inicial do ciclo tem como função básica determinar a necessidade de se chamar a função que implementa o algoritmo de recuperação da imprecisão. Esta decisão é dependente do estado atual da marcação e dos valores atribuídos aos dados externos no início do ciclo de inferência.

Os dados externos estão associados às transições da rede de Petri e, na ferramenta desenvolvida, são atualizados no início de cada ciclo de inferência, permitindo emular o comportamento dos sensores presentes no processo simulado. É a partir dos mesmos que se determina a localização de determinado objeto e, consequentemente, a possibilidade de disparo (preciso ou impreciso) e a recuperação do estado preciso da marcação. Na ferramenta desenvolvida os dados externos são representados na base de fatos utilizando a lógica como formalismo de representação de conhecimento.

No item 2.3.2 foi colocado que a ferramenta considera o tempo de forma implícita. Assim, na representação utilizada, é possíverl considerar restrições temporais sobre o disparo de determinada transição (por exemplo, especificar que a transição somente poderá disparar se o objeto estiver no lugar de entrada por três unidades de tempo ou mais).

Para expressar a noção de passagem do tempo (evolução temporal), na ferramenta desenvolvida cada ciclo de inferência representa uma unidade de tempo de simulação. Desta forma, no início, o "relógio da simulação" apresenta-se com tempo igual a zero e a cada ciclo de simulação é acrescido uma unidade de tempo ao "relógio da simulação" (após 5 ciclos de inferência o tempo de simulação é igual a 5).

A unidade tempo de simulação é unitária, podendo o usuário interpretar livremente o que isto significa (segundos, minutos, horas, etc). Como pode ser visto na figura 4.4, ao final de cada ciclo de inferência incrementa-se o tempo de simulação.

Com o propósito de facilitar a representação e a utilização do tempo na rede de Petri especificada, cada objeto possui um atributo que armazena o tempo de simulação em que foi colocado no lugar onde se encontra atualmente. Caso seja necessário saber o tempo que determinado objeto se encontra em um determinado lugar (para, por exemplo, utilizar como condição de disparo da transição), basta solicitar o atributo *time* do objeto instanciado pela regra. Com este recurso é possível implementar facilmente uma *função de autorização* do tipo mostrado na equação 2.10.

Cada vez que um objeto é colocado em um lugar pela primitiva *move*, implementada no *demon* associado ao atributo *interface*, é setado um atributo deste objeto, que armazena o tempo de simulação atual. Na ferramenta, o tempo geral da simulação é armazenado utilizando a lógica como formalismo de representação de conhecimento.

4.4 Conclusão

Neste capítulo foi apresentado o mapeamento entre o modelo de rede de Petri com Marcação Imprecisa e o sistema FASE. Como foi visto, para a correta representação do modelo de rede de Petri no FASE, é necessário solucionar algumas questões decorrentes das diferenças entre os modelos no que se refere a execução de regras, e também questões inerentes a construção do simulador.

As questões referentes ao comportamento dos lugares e a inconsistência de dados foram resolvidas através da vinculação do *demon if-modify* ao atributo *interface*. A resolução de conflito das fichas é tratado com a criação de dois conjuntos de regras, um para resolver o conflito e outro para retirar o objeto do lugar de entrada e colocar no lugar de saída. A resolução de conflito entre transições é realizada pelo próprio mecanismo de resolução de conflito do FASE. O pseudo-disparo das transições é implementado com a utilização de outros dois conjuntos de regras: o primeiro é responsável pela criação da imprecisão e o outro pela propagação da imprecisão.

Capítulo 5

Implementação

Neste capítulo são apresentados os diversos aspectos referentes à implementação do Simulador de redes de Petri a Objetos com Marcação Imprecisa. Inicialmente são apresentadas as características e a sintaxe da linguagem de especificação desenvolvida e como a mesma é tratada no sistema. A seguir é apresentada a representação dos objetos e dos lugares na base de fatos do sistema. Na seqüência, cada um dos quatro conjuntos de regras do sistema é descrito e, finalmente a interface para o controle da simulação pelo usuário é apresentada.

5.1 A Linguagem de Entrada

Para realizar a simulação de uma determinada rede deve ser fornecido ao simulador um arquivo com a descrição desta rede. A descrição utiliza uma linguagem de entrada que possibilita especificar os elementos da rede de Petri a Objetos (lugares, transições, classes, etc), as condições para o disparo normal e para o disparo impreciso das transições, as ações a serem executadas quando da aplicação da regra e as variáveis externas do processo simulado.

A especificação de uma rede de Petri é feita a partir da definição de oito elementos da linguagem de entrada. Cada um destes elementos constitui uma palavra chave da linguagem e a definição deles determina a especificação da rede. Os elementos da linguagem são:

- *Class*: onde cada uma das classes existentes no sistema são descritas a partir de um nome de classe e uma série de atributos;
- *Places*: onde são declarados os lugares da rede de Petri a ser simulada. Além do nome deve ser definida a classe de objeto (ou n-upla de classes) que o lugar manipula. Cada lugar pode manipular apenas uma classe (ou n-upla de classes) de objetos;
- *Objects*: os objetos que serão manipulados no sistema são descritos neste elemento. Cada objeto constitui uma instância de determinada classe anteriormente declarada. Além do nome, podem ser definidos uma série de atributos para os objetos;
- *Variables*: as variáveis constituem os elementos de captura de objeto na aplicação da transição. Elas são declaradas a partir da definição do seu nome e a classe dos objetos que a mesma irá capturar;
- *Structure*: a estrutura da rede de Petri a Objetos é definida através deste elemento. Para cada transição são especificados os lugares de entrada e os lugares de saída, bem como as variáveis associadas aos arcos de cada um deles. Além disso, deve ser declarado o método de resolução de conflito das fichas da transição;
- *Extern*: as variáveis externas (sensores) são definidas neste elemento. Caso haja algum dado externo vinculado a determinada transição, o mesmo deve ser definido neste ponto.
- *Conditions*: neste elemento é definida a função de autorização de cada uma das transições. Assim, as condições para o disparo normal e para o pseudo-disparo (se houver) devem ser declaradas;
- *Actions*: as ações a serem executadas pela transição quando do seu disparo são definidas neste elemento. Estas ações envolvem a alteração dos valores dos atributos dos objetos.

Na definição das classes os atributos podem receber valores que posteriormente podem ser herdados pelas instâncias de objeto. Na definição dos objetos da rede a ser simulada deve-se especificar o nome do objeto e a classe a partir do qual o mesmo é instanciado. A partir da instanciação dos objetos é definida a marcação inicial da rede, visto que, cada objeto instanciado possui um atributo obrigatório (denominado *location*) que define o lugar da rede de Petri onde o objeto é colocado inicialmente. Para

lugares que representam alguma relação dinâmica entre objetos (que manipulam uma n-upla de classes de objeto) deve-se ter o cuidado de instanciar todos objetos formadores da relação dinâmica, caso isto não seja feito uma mensagem de erro é produzida. O nome do objeto instanciado deve ser único na especificação.

As variáveis declaradas na especificação permitem a captura dos atributos dos objetos para serem utilizados nas funções que especificam as condições e as ações das transições. A utilização das variáveis permitirá a filtragem da regra quando os atributos dos objetos capturados por elas não satisfizerem determinadas condições (por exemplo, diâmetro do objeto *peça_1* maior que 5 cm).

A estrutura da rede de Petri a ser simulada é capturada através do elemento *structure*. O formato desta declaração é semelhante a regra de reescrita apresentada no item 4.1, ou seja, o nome da transição com os lugares de entrada e os lugares de saída. Para cada lugar de entrada ou saída são especificadas as variáveis associadas aos arcos. Um dos quatro métodos de resolução de conflito de fichas definidos pela ferramenta (prioridade, primeira encontrada, aleatória e do usuário) também deve ser especificado para cada um das transições declaradas.

Como colocado no capítulo 2, as variáveis externas do sistema permitem emular o comportamento do ambiente modelado, através da passagem de mensagens que indicam as mudanças de estado ocorridas no mesmo. Na ferramenta as variáveis externas são definidas no elemento *extern*. Assim, as transições que possuem um ou mais sensores vinculados devem ser declaradas neste elemento, e o nome das variáveis externas associadas devem ser especificadas.

O elemento *conditions* implementa a *função de autorização* descrita no item 2.3.3. Atualmente a função de autorização não é descrita diretamente na especificação, isto é, ao invés de se colocar diretamente na linguagem de entrada as condições para o disparo normal e o pseudo-disparo, descreve-se a transição e o nome das funções que implementam cada um dos tipos de disparo. O usuário é o responsável pela implementação de cada uma das funções declaradas. Estas funções são implementadas em um arquivo cujo nome é especificado na linguagem de entrada. As funções devem ser implementadas em Lisp, e o retorno das mesmas deve ser verdadeiro ou falso.

No elemento *actions* são definidas as ações a serem executadas quando da aplicação de determinada regra. Três formatos de aplicação das ações são definidos. O primeiro permite a atribuição de um valor fixo a determinado atributo (este valor pode ser um

número, uma lista, um caracter ou uma seqüência de caracteres). O segundo formato permite atribuir o valor de outro atributo para o atributo a ser modificado. O terceiro formato permite a definição de uma função a ser executada quando da aplicação da regra; esta função deve retornar o valor a ser atribuído. Caso seja utilizada uma função para realizar a ação, a mesma deve ser implementada no mesmo arquivo em que estão as funções definidas para as condições.

A seguir a descrição sintática da linguagem de entrada é apresentada na forma BNF (Backus-Naum Form). A BNF consiste em uma série de meta-símbolos (ou seja, símbolos que não pertencem a linguagem descrita) utilizados para definir meta-identificadores (notados por símbolos entre os caracteres "≪" e "≫"), que correspondem a classes de elementos da linguagem a ser definida.

A notação será a seguinte: símbolos não terminais serão limitados por "≪" e "≫"; campos opcionais limitados por "[" e "]"; elementos em negrito caracterizam caracteres e símbolos definidos na linguagem. Também são utilizados os meta-símbolos "::=" (é definido por), "*" (repetição zero ou mais vezes), "+" (repetição uma ou mais vezes) e "|" (indicando ou). Os símbolos "(" e ")" utilizados na descrição BNF, serão, neste trabalho, substituídos por "{" e "}", pois são utilizados na linguagem de entrada desenvolvida. Os símbolos "<" e ">" são utilizados na linguagem de entrada, e deve-se ter o cuidado de não confundí-los com os símbolos "≪" e "≫" utilizados pela notação BNF. A especificação de uma rede qualquer é definida da seguinte forma:

```
<<especificação>>::=
                <<declaração_das_classes_objetos>> <<declaração_dos_lugares>>
                <<declaração_dos_objetos>> <<declaração_das_variáveis>>
                <<declaração_das_transições>>
                [<<declaração_das_variáveis_externas>>]
                [<<declaração_das_condições>>] [<<declaração_das_ações>>]
                [<<declaração_do_arquivo_de_funções>>]
                 endnet
```

A seguir a sintaxe de cada um dos elementos apresentados acima é descrita. Para auxiliar a compreensão desta sintaxe, após cada definição é apresentado um pequeno exemplo. Estes exemplos estão baseados no exemplo da célula de usinagem, apresentada no item 2.2. No apêndice B é apresentada a descrição completa da especificação de entrada, o arquivo que implementa as funções Lisp definidas, a base de fatos e os conjuntos de regras gerados.

```
<<declaração_das_classes_objetos>>::=
```

Class:= {<<nome_da_classe>> :
{ (<<nome_do_atributo>> [<<valor_do_atributo>>])}+ ;}+

onde:
≪nome_da_classe≫ é o nome da classe de objeto;
≪nome_de_atributo≫ é o nome do atributo da classe de objeto;
≪valor_de_atributo≫ é o valor do atributo da classe de objeto.

Cada classe de objeto declarada possui um nome e uma lista variável de atributos. Cada atributo é definido por um nome e opcionalmente pode ter um valor. Para separar o nome da classe da declaração dos atributos, utiliza-se o símbolo ":". Cada atributo é definido entre parênteses. A declaração da classe termina com o símbolo ";". A descrição das classes do exemplo é a seguinte:

```
Class:=
    Pallet: (posicao );
    Peca: (job );
    Maquina: (posicao );
```

Os lugares da rede de Petri são definidos de acordo com a seguinte sintaxe:

<<declaração_dos_lugares>>::=
Places:= {<<nome_do_lugar>> :
< <<nome_de_classe_de_objeto>> {[, <<nome_de_classe_de_objeto>>]}*> ;}+

onde:
≪nome_do_lugar≫ é o nome do lugar;
≪nome_da_classe_de_objeto≫ é o nome de uma das classes de objeto previamente declarada.

Os lugares da rede são definidos a partir do seu nome e a n-upla de classes de objetos que manipula. Esta n-upla pode ser formada por apenas uma classe de objetos ou por uma lista de classes. Embora a n-upla possa ser composta, cabe lembrar que cada lugar manipula apenas uma n-upla. No exemplo tem-se:

```
Places:=
   peca_estoque: <Peca>;
   pallet_estoque: <Pallet>;
   mesa: <Pallet, Peca>;
   maq_livre: <Maquina>;
   usinando: <Pallet, Peca, Maquina>;
```

A sintaxe para a declaração dos objetos é definida assim:

```
<<declaração_dos_objetos>>::=
  Objects:= {<<nome_da_classe>> .<<nome_do_objeto>> :
                    { (<<nome_de_atributo>> [<<valor_de_atributo>>] )}*
                      (location <<lugar_do_objeto>> )
                    { (<<nome_de_atributo>> [<<valor_de_atributo>>] )}* ;}+
```

onde:

≪nome_da_classe≫ é o nome da classe de objeto a ser instanciada;

≪nome_do_objeto≫ é a identificação do objeto instanciado;

≪nome_de_atributo≫ é o nome de um atributo do objeto;

≪valor_de_atributo≫ é o valor de um atributo do objeto.

Cada objeto é instanciado a partir de uma determinada classe. Para instanciar uma classe, coloca-se o nome da classe seguida de um ponto e a identificação (nome) do objeto criado. A lista de atributos do objeto instanciado deve obrigatoriamente conter o atributo *location*. Isto se deve ao fato de que a marcação inicial da rede é definida a partir dos objetos instanciados. Desta forma, quando um objeto é instanciado ele já deve ser alocado à um lugar da rede. Não é necessário especificar nenhum atributo com o nome do objeto, isto é feito automaticamente pela ferramenta. Quando for necessário obter a identificação de determinado objeto basta solicitar o atributo *ident*. A declaração dos objetos do exemplo da célula de usinagem é a seguinte:

```
Objects:=
    Pallet.pal_1: (location pallet_estoque) (posicao 1) ;
    Pallet.pal_2: (location pallet_estoque) (posicao 2) ;
    Pallet.pal_3: (location pallet_estoque) (posicao 3) ;
    Pallet.pal_4: (location pallet_estoque) (posicao 4) ;
    Maquina.M1: (posicao 2) (location maq_livre) ;
    Maquina.M2: (posicao 3) (location maq_livre) ;
    Peca.P1: (job (M1 M2 M1)) (location peca_estoque) ;
    Peca.P2: (job (M2 M1))   (location peca_estoque) ;
```

As variáveis são definidas a partir da seguinte sintaxe:

```
<<declaração_das_variáveis>>::=
  Variables:=  {<<classe_de_objeto>> :
                    <<nome_de_variável>> {[, <<nome_de_variável>>]}* ;}+
```

onde:

≪classe_de_objeto≫ define o nome da classe de objetos que a variável pode receber;

≪nome_de_variável≫ é o nome da variável do arco de entrada ou saída da transição.

Para cada umas das classes de objetos declaradas podem ser definidas diversas variáveis. Tantas quantas sejam as necessárias nos arcos que ligam os diferentes lugares às diferentes transições. No exemplo são definidas as seguintes variáveis:

```
Variables:=
    Pallet: pal;
    Peca: pec;
    Maquina: maq;
```

As transições devem ser declaradas respeitando a seguinte sintaxe:

```
<<declaração_das_transições>>::=
  Structure:=
    {<<nome_da_transição>> (<<método_resolução_conflito>> ):
      (<<nome_do_lugar>>  < <<nome_de_variável>> [{, <<nome_de_variável>>}*]>
      <<outros_lugares>> )
        {-> | , }
      (<<nome_do_lugar>> < <<nome_de_variável>> [{, <<nome_de_variável>>}*]>
      <<outros_lugares>> )}+
```

onde:

≪nome_da_transição≫ é o nome da transição;

≪método_resolução_conflito≫ define o método de resolução de conflito;

≪nome_do_lugar≫ é o nome do lugar de entrada ou saída da transição;

≪nome_de_variável≫ é o nome da variável.

e, ≪outros_lugares≫ é definido assim:

```
<<outros_lugares>> ::= [{, <<nome_do_lugar>> < <<nome_de_variável>>
                       [{, <<nome_de_variável>>}*] >}*]
```

Cada transição é especificada de forma semelhante à regra de reescrita. Define-se o seu nome, seguido do método de resolução de conflito das fichas a ser adotado. A seguir é colocado a lista dos lugares de entrada com suas respectivas variáveis e a lista dos lugares de saída o com suas respectivas variáveis. No exemplo da célula de usinagem tem-se a seguinte especificação para as transições:

```
Structure:=
    Entrada (user): (pallet_estoque <pal>, peca_estoque <pec>)
                                        -> (mesa <pal, pec>);
    Rotacao (user): (mesa <pal, pec>) -> (mesa <pal, pec>);
    Inicio  (user): (mesa <pal, pec>, maq_livre <maq>)
                        -> (usinando <pal, pec, maq>);
    Fim     (user): (usinando <pal, pec, maq>)
                    -> (mesa <pal, pec>, maq_livre <maq>);
    Saida   (user): (mesa <pal, pec>)
                    -> (peca_estoque <pal>, peca_estoque <pec>);
```

A especificação das variáveis externas deve seguir a seguinte sintaxe:

```
<<declaração_das_variáveis_externas>>::=
    Extern:=
        {<<nome_da_transição>> :
            {<<nome_da_variável_externa>> = { 0 | 1} ;}+}+
```

onde:
≪nome_da_transição≫ é o nome da transição ao qual a variável exetrna está vinculada;
≪nome_da_variável_externa≫ é o nome da variável externa.

No modelo de redes de Petri as variáveis externas permitem o recebimento de mensagens do sistema. Na ferramenta desenvolvida este comportamento é emulado através da atribuição variáveis externas vinculadas às transições. Cada variável externa está vinculada a uma transição, sendo que uma mesma transição pode possuir mais de uma variável externa vinculada. O exemplo considerado não possui nenhum dado externo, mas uma especificação válida seria:

```
Extern:=
  trans_a: sensor_a = 0;
```

A definição das condições que filtram as regras utiliza a seguinte sintaxe:

```
<<declaração_das_condições>>::=
    Conditions:=
        {<<nome_da_transição>> :
            true: <<nome_da_função>> (<<parâmetros>> )
            [ uncertain: <<nome_da_função>> (<<parâmetros>> )] ;}+
```

onde:

≪nome_da_transição≫ é o nome da transição;

≪nome_da_função≫ é o nome da função que faz o teste para verificar se o os objetos atendem as condições de disparo ;

≪parâmetros≫ são definidos os parâmetros que a função necessita para realizar o teste, pode ser o valor de um atributo, de um dado externo ou do tempo em que o objeto se encontra no lugar de entrada;

As condições de disparo são especificadas por transição. Em cada uma delas define-se o nome da função que implementa a condição de disparo normal da transição (*true*) e opcionalmente define-se o nome da função que implementa a condição de pseudo-disparo da transição (*uncertain*). Quando houver alguma função definida para as condições, ou para as ações, deve-se especificar o nome do arquivo onde as funções estão implementadas. A seguir é apresentada a especificação das condições de disparo das transições do exemplo considerado.

```
Conditions:=
  Entrada:
    true: ver_entrada(pec.job, pal.posicao);
  Rotacao:
    true: ver_rotacao(pal.posicao);
  Inicio:
    true: ver_inicio(pal.posicao, maq.posicao, maq.ident, pec.job);
  Saida:
    true: ver_saida(pec.job, pal.posicao);
```

A declaração das ações a serem executadas pela transição deve respeitar a seguinte sintaxe:

```
<<declaração_das_ações>>::=
 Actions:=
   {<<nome_da_transição>> :
        [ function:
            {<<nome_de_atributo>>  = <<nome_da_função>> (<<parametros>> )}+]
        [ attribute:
            {<<nome_de_atributo>>  = <<nome_de_atributo>> ;}+]
        [ constant: {<<nome_de_atributo>>  = <<valor_constante>> ;}+]}+
```

A especificação das ações a serem executadas no disparo da regra permite atribuir valores aos atributos dos objetos diretamente, desde que não seja necessário nenhuma

operação sobre os dados (utilizando as entradas *attribute* e *constant*), ou fazer a atribuição utilizando uma função Lisp. Para o exemplo, as seguintes ações são definidas:

```
Actions:=
    Rotacao:
      functions:
         pal.posicao = exec_rotacao(pal.posicao);
    Inicio:
      functions:
         pec.job = exec_inicio(pec.job);
```

Como colocado, no caso de se utilizar funções Lisp para implementar as condições ou as ações da regra, é necessário definir o nome do arquivo onde as mesmas estão implementadas. Para tal deve-se seguir a seguinte sintaxe:

file: <<nome_do_arquivo>> ;

No caso do exemplo:

```
file: usinagem.lsp;
```

A linguagem de entrada desenvolvida define algumas palavras chaves para determinar o início da definição de cada um dos elementos da linguagem, outras que informam o método de resolução de conflito das fichas a ser adotado, etc. A lista completa das palavras reservadas da linguagem é apresentada a seguir na tabela 5.1.

Class	Objects	Places
Variables	Structure	Conditions
Location	Aleatory	Priority
User	First	True
Uncertain	File	Time
Ident	Actions	Extern
Fire	Ident	Function
Attribute	Constant	Arrived
Endnet		

Tabela 5.1: Palavras reservadas da linguagem

Para permitir uma maior flexibilidade, as palavras chaves apresentadas na tabela 5.1 podem ser escritas de três formas distintas: toda a palavra em letras minúsculas, toda a palavra em letras maiúsculas e a primeira letra da palavra em maiúscula e as demais minúsculas.

A verificação sintática da especificação é feita através das ferramentas LEX [Les75] e YACC [Joh75]. O Lex (*a Lexical analyzer generator*) consiste de uma ferramenta para a construção de um analisador léxico de cadeias de caracteres. A ferramenta permite definir classes de caracteres (expressões regulares) e identificadores para estas classes.

O YACC (*Yet Another Compiler-Compiler*) é uma ferramenta para construção de um analisador sintático. Ele permite a definição de regras que determinam a sintaxe da entrada. Além disso, permite a definição de um procedimento (escrito na linguagem C) a ser processado quando as regras sintáticas são reconhecidas.

As duas ferramentas trabalham de forma cooperativa. Cada palavra fornecida ao analisador sintático é entregue ao analisador léxico que a identifica e retorna um identificador ao analisador sintático. Se a seqüência de palavras está de acordo com a sintaxe exigida pelas regras, ela é identificada e o procedimento vinculado a regra sintática é executado.

No caso do analisador desenvolvido, os procedimentos a serem executados pelas regras consistem basicamente do armazenamento dos nomes das classes, lugares, objetos, etc., em estruturas de dados a serem manipuladas posteriormente.

Quando um arquivo com a especificação de uma dada rede é passado para o programa o analisador sintático é invocado. Se houver algum erro em relação à sintaxe do programa deve-se corrigir o problema e entrar novamente com a especificação da rede. Algumas mensagens de erro são geradas para auxiliar o usuário na correção da especificação. As mensagens de erro indicam quando um lugar foi utilizado por uma transição mas não foi declarado, quando uma variável foi utilizada sem ter sido declarada, entre outras.

Ao final da análise sintática de uma rede sem erros, obtém-se as estruturas que armazenam os nomes de classes, atributos, transições, etc, devidamente preenchidas. Então, a partir destas estruturas gera-se um arquivo intermediário que será manipulado por funções Lisp para gerar a base de fatos do sistema especialista e cada um dos conjuntos de regras necessários.

A figura 5.1 mostra que, a partir de uma arquivo com a descrição na linguagem de entrada desenvolvida, obtém-se 7 arquivos. O arquivo *intermed.llc* é um arquivo temporário que possibilita a transposição de todos os elementos da rede especificada para o Lisp, onde serão manipulados adequadamente.

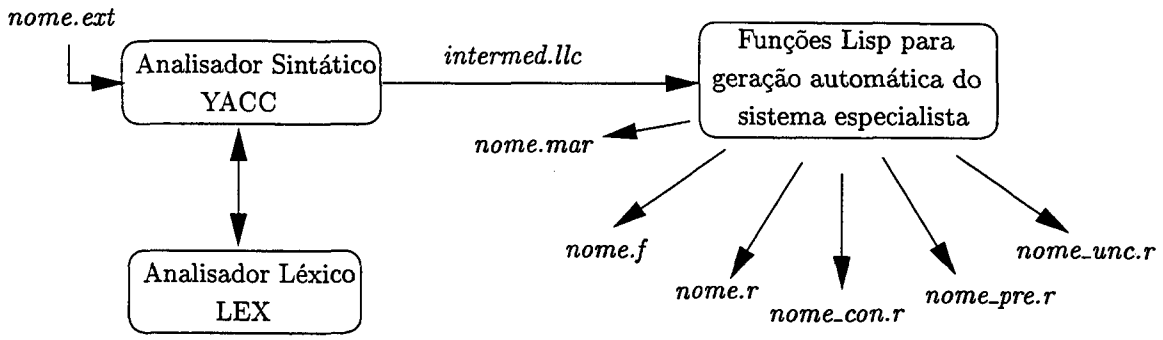


Figura 5.1: A criação automática do Sistema Especialista

A partir da leitura do arquivo intermediário pelas funções Lisp, inicia-se o processo de criação automática do sistema especialista que fará a simulação da rede de Petri especificada.

Nas funções Lisp um novo arquivo temporário, contendo a marcação inicial da rede, é gerado. A função deste arquivo é apenas de facilitar a geração da base de fatos do sistema.

No arquivo com extensão ".f" (*arquivo.f*) está definida a base de fatos do sistema especialista. Todas as definições de classes, objetos, marcação inicial, etc., estarão contidos nesta base de fatos (ver também tabela 4.2).

O arquivo *nome.r* define o conjunto de regras que realiza o disparo normal das transições. Em *nome_con.r* está definido o conjunto de regras que implementa a resolução de conflito das fichas, no arquivo *nome_pre.r* está definido o conjunto de regras de pseudo-disparo responsável pela criação da imprecisão e, no arquivo *nome_unc.r*, está definido o conjunto de regras de pseudo-disparo que realiza a propagação da imprecisão. Nas seções seguintes, é feita a descrição do conteúdo de cada um destes arquivos.

Como colocado anteriormente, os arquivos *intermed.llc* e *nome.mar* são temporários e, ao final do processo de geração dos arquivos que definem o sistema especialista, eles são apagados.

5.2 Representação de Classes de Objetos e dos Lugares

O arquivo que implementa a base de fatos iniciais do sistema possui a representação dos elementos que descrevem a rede a ser simulada (classes e respectivos atributos, objetos e respectivos atributos, lugares e respectivas transições para o qual é lugar de entrada e/ou de saída, marcação inicial, e relações dinâmicas entre objetos instanciados, transições com os nomes das funções que implementam as condições e as ações das regras).

Conforme colocado na seção 4.2, o formalismo de representação de conhecimento adotado são os quadros. A seguir, na figura 5.2, é apresentado o diagrama de quadros que define a estrutura da representação de conhecimento de uma determinada rede.

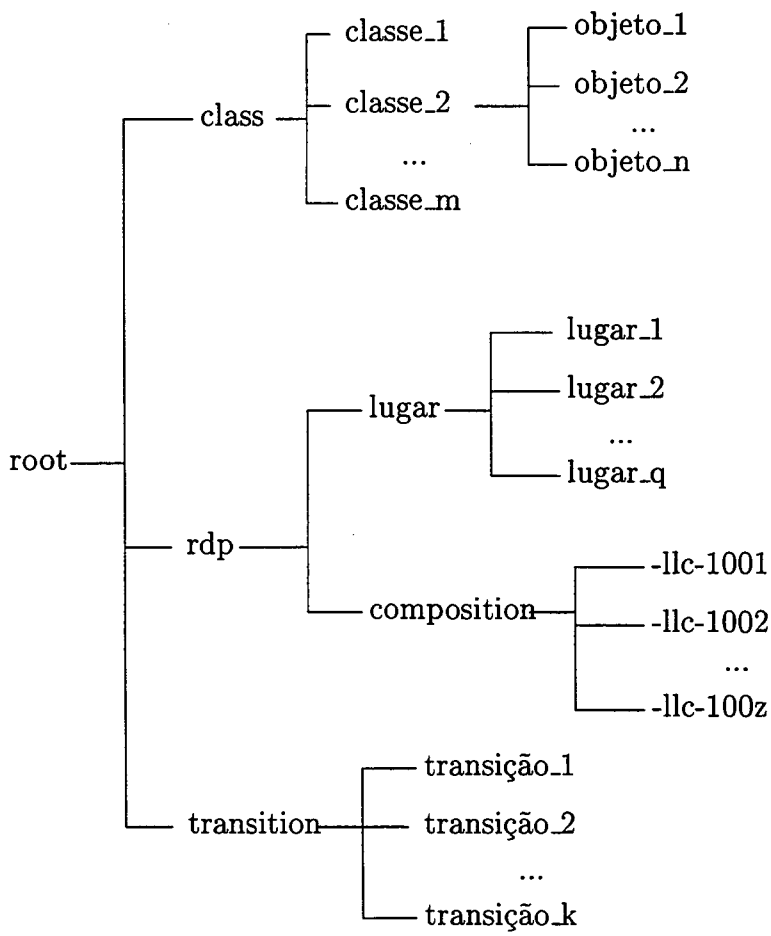


Figura 5.2: Diagrama de quadros do Sistema Especialista

Toda a estrutura de quadros da figura 5.2, descende de um quadro principal denominado *root* que não possui nenhum atributo. A partir de *root* três quadros são derivados: *class*, *rdp* e *transition*. Estes quadros armazenam os diferentes elementos do modelo de rede de Petri. A seguir, é apresentada a forma de representação no arquivo de fatos dos principais elementos que definem a rede de Petri a Objetos. O exemplo do item 2.3.3 (página 28) é utilizado como referência. De maneira a tornar o exemplo mais completo, é apresentado a descrição na linguagem de entrada e, em seguida, a descrição correspondente na base de fatos.

Quadro Class: A partir do quadro *class* são armazenadas cada uma das classes de objetos definidas na linguagem de entrada. Haverá tantos quadros descendentes de *class* quantas forem as classes declaradas na linguagem de entrada. Os atributos das classes definidas na entrada, serão atributos dos quadros que representam esta classe. No exemplo:

```
Class:=
   veiculo: (trajeto) (proxima_s);
   sessao: (status);
```

Assim, cada uma das classes (*veiculo* e *sessao*) é representada por um quadro, descendente de *class*:

```
(veiculo class
   (variables (v))
   (trajeto nil)
   (proxima_s nil))
```

```
(sessao class
   (variables (s ns))
   (status nil))
```

Pode-se perceber que os atributos da classe de objetos declarada na linguagem de entrada são atributos do quadro que representa a respectiva classe. Além disso, o nome das variáveis que capturam objetos desta classe (que é especificado no elemento *variables*), também é armazenado.

Objetos: Quando uma determinada classe é instanciada, o novo objeto criado será descendente desta classe. Na representação de quadros isto significa que o objeto instanciado será representado por um quadro descendente da classe que o objeto instanciou. Haverá tantos quadros descententes da classe, quantos forem os objetos instanciados a

partir dela. Observe no exemplo a instanciação das classes *veiculo* e *sessao*. Na linguagem de entrada declara-se (estas instâncias de objetos são diferentes das apresentadas no exemplo do capítulo 2; no que diz respeito às instâncias das classes, daqui para frente deve ser considerada a definição apresentada aqui):

```
Objects:=
  veiculo.v1: (location C) (proxima_s s3) (trajeto (s1 s3 s4 s2));
  veiculo.v2: (location C) (proxima_s s2) (trajeto (s4 s2 s3 s1));
  sessao.s1: (status t) (location C) ;
  sessao.s2: (status t) (location S) ;
  sessao.s3: (status t) (location S) ;
  sessao.s4: (status t) (location C) ;
```

A representação no arquivo de fatos será a seguinte:

```
(v1 veiculo                          (v2 veiculo
    (ident v1)                           (ident v2)
    (arrived 0)                          (arrived 0)
    (location c)                         (location c)
    (proxima_s s3)                       (proxima_s s2)
    (trajeto (s1 s3 s4 s2)))             (trajeto (s4 s2 s3 s1)))

(s1 sessao                           (s2 sessao
    (ident s1)                           (ident s2)
    (arrived 0)                          (arrived 0)
    (status t)                           (status t)
    (location c))                        (location s))

(s3 sessao                           (s4 sessao
    (ident s3)                           (ident s4)
    (arrived 0)                          (arrived 0)
    (status t)                           (status t)
    (location s))                        (location c))
```

Quando o valor de determinado atributo consistir de mais de um elemento, este valor deve ser colocado na forma de uma lista, como pode ser visto no objeto *v1* cujo atributo *trajeto* consiste da lista *(s1 s3 s4 s2)*. Além dos atributos declarados para cada objeto, é criado o atributo *ident*, que armazena o nome do objeto, e o atributo

arrived, que armazena o tempo da simulação em que o objeto foi colocado no lugar. Como pode ser visto na tabela 5.1, o nome destes atributos estão entre as palavras reservadas da linguagem de entrada.

Quadro transition: O quadro *transition* têm como quadros descendentes cada uma das transições especificadas no sistema. Cada quadro descendente armazena o nome dos lugares e variáveis dos arcos de entrada da transição. Além disso, quando existir, é armazenado o nome da função que faz a verificação da condição de disparo normal e seus respectivos parâmetros, e também o nome da função que verifica as condições para o pseudo-disparo e seus parâmetros. Para exemplificar considere as seguintes definições na linguagem de entrada:

```
Conditions:=
    trans_a:
        true: true_trans_a(v.time, sensor_a);
        uncertain: unc_trans_a(v.time);
    trans_b:
        true:  true_trans_b(v.proxima_s, ns.ident);
```

A representação no arquivo com a base de fatos é a seguinte:

```
(trans_a transition
     (certain (true_trans_a (v time) sensor_a))
     (uncert  (unc_trans_a (v time)))
     (lug_entr (ms))
     (var_entr ((v s))))
(trans_b transition
     (certain (true_trans_b (v proxima_s) (ns ident)))
     (uncert  nil)
     (lug_entr (c s))
     (var_entr ((v s) (ns))))
```

Quadro rdp: Como colocado anteriormente no item 4.2.1, representação da estrutura da rede de Petri a Objetos é baseada nos lugares que servem como referência para dizer quais as transições de entrada e saída.

Para a representação correta da estrutura da rede e dos próprios lugares da rede de Petri, cada um dos lugares da rede de Petri especificado na linguagem de entrada é mapeado em um quadro (descendente do quadro *lugar*), com os seguintes atributos:

```
(lugar_i lugar
     (identificacao <<nome>>)
     (transicoes_anteriores (<<lista de momes de transições>>))
     (transicoes_posteriores (<<lista de momes de transições>>))
     (tipo_objeto <<n-upla manipulada>>)
     (interface place if-modify)
     (content <<depósito de entrada>>)
     (content_1 <<depósito de saída 1>>)
     ...
     (content_m <<depósito de saída m>>))
     (content_unc )
```

onde os três primeiros atributos (*identificacao*, *transicoes_anteriores* e *transicoes_posteriores*) contém símbolos que identificam o lugar e suas transições de entrada e de saída. O atributo *tipo_objeto* informa a n-upla de objetos manipulada pelo lugar.

O *demon* vinculado ao atributo *interface* (ver item 4.2.1), permite eliminar as dificuldades na implementação do comportamento da rede de Petri. Cada lugar possui um depósito de saída como atributo, associado com cada uma das transições de saída. Em um dado momento, cada um destes conterá no máximo um objeto.

Independente do número de entradas, cada lugar possui um único depósito de entrada de objetos precisos, o atributo *content* que irá conter a lista de objetos precisos do lugar (seção 4.2.1). Condizente com o modelo de rede de Petri a Objetos, todos os objetos em um determinado depósito (de entrada ou de saída) são da classe ou n-uplas de classes associadas ao lugar.

Composition: O quadro *composition* é criado quando existe alguma relação dinâmica entre objetos em um determinado lugar e esta relação é desfeita quando os objetos saem do lugar. Cada lugar que representa algum tipo de relação dinâmica possui em seu atributo *content* uma referência (ponteiro) para o quadro descendente de *composition*, onde estão armazenados os nomes dos objetos que compõem esta relação. Este artifício de representação da composição de dois (ou mais) objetos é totalmente transparente para o usuário, pois quando o conteúdo do lugar deve ser mostrado, existem funções que manipulam os quadros de forma que, para o usuário, seja mostrado os próprios objetos e não o nome do quadro onde eles se encontram.

Para exemplificar a representação dos lugares e da relação dinâmica entre objetos,

considere os lugares *MS*, *C* e *S* do exemplo de referência. Os lugares *MS* e *C* expressam uma relação dinâmica entre os objetos do tipo *veiculo* e *sessao*.

Seis instâncias de objetos estão presentes nos diferentes lugares (conforme definido no elemento *Objects* da linguagem de entrada): os objetos <*s2*> e <*s3*> estão no lugar *S*, e os objetos <*v1, s1*> e <*v2, s4*> apresentam uma relação dinâmica e estão localizados no lugar *C*. A representação dos lugares e da composição dinâmica para esta descrição é apresentada a seguir.

```
(MS lugar
  (identificacao MS)
  (tipo_objeto (veiculo sessao))
  (transicoes_anteriores (trans_b))
  (transicoes_posteriores
    (content_trans_a content_trans_f))
  (interface )
  (content nil)
  (content_trans_a )
  (content_trans_f ))
  (content_unc )
```

```
(C lugar
  (identificacao C)
  (tipo_objeto (veiculo sessao))
  (transicoes_anteriores
                (trans_a))
  (transicoes_posteriores
          (content_trans_b))
  (interface )
  (content (-llc-1002 -llc-1001))
  (content_trans_b ))
  (content_unc )
```

```
(S lugar
   (identificacao S)
   (tipo_objeto (sessao))
   (transicoes_anteriores
            (trans_b trans_f))
   (transicoes_posteriores
            (content_trans_b))
   (interface )
   (content (s3 s2))
   (content_trans_b ))
   (content_unc )
```

```
(-llc-1001 composition
   (arrived 0)
   (componente_1 v1)
   (componente_2 s1))

(-llc-1002 composition
      (arrived 0)
      (componente_1 v2)
      (componente_2 s4))
```

Nos lugares *C* e *S* a transição de saída é a mesma, em função disto o atributo *transicoes_posteriores* possui o mesmo valor *content_trans_b*. Este valor indica o nome do depósito de saída do lugar e não a transição em si. Isto é necessário para que a primitiva *move* (explicada no item 4.2.1) saiba onde colocar o objeto após a resolução de conflito das fichas.

O atributo *content* é o depósito de entrada de todos os objetos precisos de um dado lugar. Para o lugar S o atributo contém o próprio nome dos objetos que estão presentes no mesmo (no exemplo *s4* e *s3*) e, para o lugar C, o conteúdo do atributo é o nome do quadro que expressa a relação dinâmica entre os objetos do lugar. Neste caso, os nomes dos quadros são *-llc-1001* e *-llc-1002*; eles são descendentes de *composition* e possuem como atributos o nome de cada um dos componentes da relação dinâmica, além do instante de tempo em que foram colocados no lugar atual.

5.3 Representação das Transições

O simulador desenvolvido apresenta quatro conjuntos de regras, cuja finalidade é expressar de forma adequada o comportamento dinâmico do modelo de redes de Petri com Marcação Imprecisa. Como foi visto na seção 4.2, um conjunto de regras implementa a resolução de conflito das fichas, um segundo conjunto de regras implementa o disparo normal das transições, um terceiro conjunto de regras implementa a criação da imprecisão no pseudo-disparo da transição e, um quarto conjunto de regras implementa a propagação da imprecisão.

As transições do modelo de rede de Petri são representadas no simulador através de pelo menos dois conjuntos de regras: as regras que definem a resolução de conflito das fichas para a transição e as regras que executam o disparo da transição propriamente dito. Quando o pseudo-disparo é definido na linguagem de entrada (através da função declarada em *uncertain*), outros dois conjuntos de regras implementam este tipo de disparo. A seguir cada um destes conjuntos de regras é detalhado, sendo apresentado o formado das regras e o significado dos elementos que às compõem. Para exemplificar cada uma das bases de regras do sistema, será utilizado o exemplo do sistema de AGVs apresentado na seção 2.2. Além disso, no apêndice C o exemplo é apresentado de forma completa, através da apresentação do arquivo de especificação com a linguagem de entrada, o arquivo com as condições (para disparo normal e pseudo-disparo) e as ações a serem executadas pelas regras, o arquivo com a base de fatos desta rede, e os quatro arquivos que constituem as bases de regra para esta especificação.

Regras de resolução de conflito das fichas: O conjunto de regras que implementa a resolução de conflito das fichas é denominado de *regras de resolução de conflitos* e sua tarefa é, para cada transição, selecionar um dos objetos que estão no depósito de

entrada do lugar e colocar este objeto no depósito de saída para a transição em questão. Cada regra de resolução de conflito possui o seguinte formato:

```
(nome_da_transição_i
 ((frame
   (nome_do_lugar_de_entrada_a (content variável_interna_a))
   ...
   (nome_do_lugar_de_entrada_n (content variável_interna_n))))
  (include (z . (ask-instance 'estratégia 'nome_da_transição_i
                 '(nome_do_lugar_de_entrada_a ...
                        nome_do_lugar_de_entrada_n)
                 'content_nome_da_transição_i))))
 ())
```

Os elementos *(nome_do_lugar_de_entrada_a (content variável_interna_a)) ... (nome_do_lugar_de_entrada_n (content variável_interna_n))* verificam a existência de objetos que habilitem a transição. O valor n representa o número de lugares de entrada da transição. O comando *(nome_do_lugar_de_entrada (content variável_interna))* verifica se o lugar de entrada da transição está marcado. No caso de todas as variáveis internas obterem algum objeto como instância, a regra chama o *demon* associado ao lugar (vinculado ao atributo *interface*) que executa a resolução de conflito conforme a *estratégia* escolhida, e coloca os objetos selecionados nos depósitos de saída *(content_nome_da_transição_i)* para a transição em questão, em cada um dos lugares de entrada.

A regra de resolução de conflito das fichas da transição *trans_b* do exemplo definido pela figura 2.7 é dado por:

```
(trans_b
 ((frame (C (content x1))
         (S (content x2)))
  (include (z . (ask-instance 'user 'trans_b
                  '(C S) 'content_trans_b))))
 ())
```

No caso de haver apenas um objeto habilitando duas transições, no início da etapa ambas estarão sensibilizadas, entretanto se objeto for retirado pela primeira regra que faz a resolução de conflito o *demon* da segunda regra verificará que isto aconteceu e não estará mais sensibilizada a segunda regra.

Neste conjunto de regras o *demon* verifica o estado da base de fatos no momento da aplicação da regra (e não apenas no início do ciclo). No caso de existir algum objeto habilitando a regra, o *demon* aplica a função de disparo normal (especificada no campo *true* da linguagem de entrada) para selecionar os objetos que satisfazem a condição da regra.

O objeto somente é retirado do atributo *content* e é colocado no atributo que representa o depósito de saída *conten_trans_b* se, tanto a marcação quanto as condições definidas pela função que indica o disparo normal da transição permitirem.

Regras de disparo normal das transições: Para completar o processo de disparo normal das transições existe o conjunto de regras denominado *regras de disparo normal*, que é responsável pela retirada dos objetos dos lugares de entrada da transição e colocação dos mesmos nos lugares de saída desta. Além disso, este conjunto de regras é responsável por toda a manipulação das relações dinâmicas entre objetos.

As regras de disparo normal são responsáveis pela composição e decomposição das relações dinâmicas entre os objetos. Elas não possuem um formato muito rígido, pois conforme a existência ou não de relações dinâmicas nos lugares de entrada e de saída da transição, há a necessidade de manipular estes objetos de forma diferente. Além disso, as ações definidas para a regra são executadas na etapa de disparo normal das transições, o que significa que, no caso de ter sido especificada uma função que realiza a ação, deve-se preparar os parâmetros solicitados pela função.

Para exemplificar um possível formato para as regras de disparo normal da transição é utilizada a transição *trans_b* do exemplo do sistema de AGVs. No disparo desta transição será necessário desfazer uma relação dinâmica já existente no lugar *C*, e montar outra no lugar *MS*. A representação da regra de disparo normal para a transição *trans_b* é dada por:

```
(trans_b
  ((frame
     (C (content_trans_b ?-c))
     (?-c composicao (componente_1 ?-v) (componente_2 ?-s))
     (S (content_trans_b ?-ns))
     (?-v ?-y1)
     (?-v (trajeto ?-p2))
     (?-v (proxima_s ?-p3))
```

```
    (?-ns (ident ?-p4))   )
  (include (z1 . (exec_troca '?-p2 '?-p3 '?-p4)))
  (include (z2 . (gensym "-LLC-"))))
 ((frame
    (off (?-c))
    (C (interface (remove ?-c content_trans_b)))
    (S (interface (remove ?-ns content_trans_b)))
    (?-v ?-y1 (proxima_s z1))
    (z2 composicao (componente_1 ?-v) (componente_2 ?-ns))
    (MS (interface (push z2)))
    (S (interface (push ?-s))))))
```

Os elementos *(C (content_trans_b ?-c)) (?-c composicao (componente_1 ?-v) (componente_2 ?-s))* são utilizados para verificar as condições da regra, o que para este conjunto de regras, significa verificar se os objetos foram "gatilhados" pelas regras de resolução de conflito.

As variáveis *?-c, ?-ns* capturam os objetos a serem disparados. Caso nenhum objeto seja capturado por qualquer uma das variáveis a regra não é disparada. O elemento *(?-c composicao (componente_1 ?-v) (componente_2 ?-s))* auxilia na decomposição da relação dinâmica existente no lugar *C*, capturando cada um dos componentes da relação, para serem utilizados de maneira independente na parte ação da regra.

A composição e decomposição de objetos é utilizada para facilitar a manipulação dos objetos em lugares que manipulam n-uplas de classes de objetos. O elemento *(include (z2 . (gensym "-LLC-"))* auxilia a tarefa de composição criando um nome de quadro para representar a nova composição a ser montada na saída da regra. Do ponto de vista do usuário este quadro não existe uma vez que em nenhum momento ele toma conhecimento da existência deste. Mesmo quando da apresentação da marcação da rede simulada, para o usuário são apresentados somente os elementos que formam a composição.

Na parte direita da regra a relação dinâmica existente é desfeita; apaga-se o quadro que a representa (comando *(off ?-c)*), os objetos disparados são removidos do lugar de entrada pela primitiva *remove* e colocados no lugar de saída (através da primitiva *push*), e a relação dinâmica no lugar *MS* é construída (através de *(z2 composicao ... ?-ns)*). Deve-se perceber que no atributo *content* do lugar *?-MS* é colocado o nome do frame que representa a relação dinâmica existente no lugar, e não os objetos que

formam a relação dinâmica. Estes objetos são colocados como atributos do quadro que representa a relação dinâmica entre eles (atributos *componente_1* e *componente_2*). As primitivas *remove* e *push* são aquelas definidas sobre o atributo *interface* (item 4.2.1, página 53.

As variáveis *?-p2 ?-p3 ?-p4* capturam os valores dos atributos que são utilizados como parâmetros pela função que verifica as condições para o disparo normal. O valor retornado por esta função é armazenado em *z1* e na ação da regra este valor é atribuído ao atributo *próxima_s* (classe veículo).

Regras de criação da imprecisão: O conjunto de regras que implementa a criação da imprecisão possui um formato semelhante ao das regras de resolução de conflito.

Sua tarefa é, para cada transição que possui a condição de pseudo-disparo, verificar se as condições são satisfeitas e por quais objetos. Os objetos que satisfizerem estas condições são retirados do atributo *content* e colocados no atributo *content_unc*. Esta operação é realizada através do *demon* vinculado ao atributo *interface* que, com o uso da primitiva *uncertain*, retira o objeto do atributo *content* e coloca em *content_unc*. Cada regra de criação da imprecisão possui o seguinte formato:

```
(nome_da_transição_i
 ((frame
   (nome_do_lugar_de_entrada_a (content variável_interna_a))
   ...
   (nome_do_lugar_de_entrada_n (content variável_interna_n))))
  (include (z . (ask-precision  'nome_da_transição_i
                  '(nome_do_lugar_de_entrada_a  ...
                          nome_do_lugar_de_entrada_n)))))
  ())
```

Os elementos *(nome_do_lugar_de_entrada_a (content variável_interna_a)) ... (nome_do_lugar_de_entrada_n (content variável_interna_n))* verificam a existência de objetos que habilitem a regra. No caso de todas as variáveis internas obterem algum objeto como instância, a regra chama a função *ask-precision* que resgata o nome e os atributos da função que define as condições para o disparo impreciso e, em seguida, verifica os objetos a serem pseudo-disparados. Os objetos selecionados serão retirados do atributo *content* pela primitiva *uncertain* definida no *demon*, e colocados no atributo *content_unc* do lugar.

A regra de criação da imprecisão da transição *trans_a* do exemplo é:

```
(trans_a
 ((frame (MS (content x1)))
 (include (z . (ask-precision 'trans_a '(MS)))))
 ())
```

O objeto só é retirado do atributo que representa o depósito de entrada *content* e é colocado no atributo que armazena os objetos pseudo-disparados *conten_unc* para a transição em questão se, tanto a marcação quanto as condições impostas pela função que indica o pseudo-disparo permitirem.

Regras de propagação da imprecisão: As regras de propagação da imprecisão atuam sobre os objetos já pseudo-disparados. Assim, a verificação da habilitação da regra consulta os objetos que estão no atributo *content_unc* dos lugares de entrada. O formato das regras é mais simples em relação às regras de disparo normal visto que não é necessário retirar os objetos do lugar de entrada, nem executar as ações definidas para a regra. Também não possuem um formato definido, visto que, do mesmo modo que as regras de disparo normal, devem montar e desmontar as composições que representam as relações dinâmicas entre objetos.

Um exemplo do formato das regras que representam a propagação da imprecisão é apresentado com base na transição *trans_f* do exemplo que serve como referência.

```
(trans_f
  ((frame (MS (content_unc ?-MS)))
   (include (z1 . (propagar '?-MS)))
   (include (z2 . (car z1)))
   (include (z3 . (cadr z1))))
  ((frame (MOS (content_unc z2))
          (S (content_unc z3)))))
```

O elemento *(MS (content_unc ?-MS))* verifica se existe algum objeto presente em MS que possa habilitar a regra. Havendo algum objeto é chamada a função *propagar* que verifica se os objetos (ou n-uplas de objetos) presentes no atributo *content_unc* já foram propagados. Os objetos (ou n-uplas) que ainda não foram propagados são retornados pela função e armazenados em *z1*. Uma vez que é necessário desfazer a composição, as variáveis *z2* e *z3* separam os objetos de maneira que os objetos da

classe *veiculo* ficam em *z2* e os objetos da classe *sessão* ficam em *z3*. A seguir, na parte ação da regra, cada um dos conjuntos de objetos são colocados no atributo *content_unc* dos respectivos lugares de saída.

A partir da descrição pode-se perceber que cada lugar a ser representado deve possuir um atributo para colocar os objetos pseudo-disparados, cuja localização é imprecisa (para tal é definido o atributo *content_unc*). Além disto, os próprios objetos pseudo-disparados devem possuir um atributo que informe se ele já foi propagado ou não (para tal é definido o atributo *fire*).

5.4 A interface com o usuário

A ferramenta desenvolvida apresenta uma interface no formato texto. Através da interface a ferramenta informa a marcação da rede e as transições que estão habilitadas em determinado ciclo. Além disso, solicita que o usuário informe os valores dos dados externos e os objetos que devem disparar com cada transição.

Quando a rede simulada possui algum dado externo, no início de cada ciclo de inferência a ferramenta atualiza os valores dos dados externos, para isto cada um dos dados externos definidos é apresentado com seu respectivo valor atual e o usuário deve informar o novo valor. Caso não esteja definido nenhum dado externo este procedimento não é realizado.

Depois do procedimento com os dados externos a ferramenta apresenta a marcação atual da rede. Para tal é construída uma tabela contendo o nome do lugar, os objetos presentes no lugar e o tempo que o objeto está no lugar. A composição de objetos utilizada para representar a relação dinâmica fica transparente para o usuário já que é realizado um tratamento nos dados armazenados no atributo *content* e ao invés de mostrar o nome da composição para o usuário, são mostrados os objetos que fazem parte da relação (ver página 79).

Após ser apresentada a marcação atual da rede são apresentadas as transições habilitadas pela marcação naquele ciclo de inferência. É mostrada uma tabela que contém o nome das transições e cada um dos objetos que habilitam a transição pela marcação.

Com estas duas tabelas define-se para o usuário o panorama da rede simulada em cada ciclo inferência. Após apresentar a marcação atual e as transições habilitadas, o

usuário deve fazer a resolução de conflito das fichas para as transições cuja estratégia de resolução de conflito foi especificada como "user".

Assim para cada uma destas transições é apresentado o nome da transição e a lista de instâncias que habilitam a transição. Os objetos que estão nos lugares de entrada da transição, mas seus atributos não satisfazem a condição de disparo, e os objetos que não satisfazem alguma condição temporal, não são apresentados entre as instâncias de objetos à disparar a transição.

As transições cuja interpretação é falsa, mesmo possuindo objetos no lugar de entrada, não são apresentadas para a resolução de conflito das fichas. Quando duas transições estão habilitadas por um único objeto e a primeira transição apresentada escolhe este objeto para ser disparado, a segunda transição não é mais apresentada.

5.5 Conclusão

Neste capítulo foram tratados os aspectos referentes a implementação do simulador de rede de Petri com Marcação Imprecisa. Inicialmente foi apresentada a linguagem de entrada desenvolvida para a especificação da rede de Petri. Foram descritos os elementos que compõem a linguagem e a sintaxe dos mesmos. Mostrou-se também como que, a partir do arquivo com a especificação da rede de Petri a ser simulada, obtém-se a base de fatos do sistema especialista e os quatro conjuntos de regras necessários para o correto funcionamento do simulador.

Foi apresentado a seguir a representação dos diversos elementos da rede de Petri na base de fatos do sistema. Como foi visto, a representação do conhecimento define uma estrutura de quadros que armazena as informações da rede de Petri simulada.

Na seqüência foi apresentado o formato das regras de cada um dos 4 conjuntos de regras definidos no sistema. Para cada um dos conjuntos foram explicados os elementos que compõem a regra e um exemplo foi apresentado. Ao final, foi apresentada a interface de controle da simulação que a ferramenta apresenta para o usuário.

Capítulo 6

Conclusões e Perspectivas

O presente trabalho apresentou a descrição da construção de uma ferramenta para a simulação de redes de Petri com Marcação Imprecisa, bem como os modelos teóricos nos quais a mesma está fundamentada.

O modelo de rede de Petri com marcação imprecisa permite uma representação correta do estado do sistema na presença de falhas. Este modelo, além do disparo clássico das transições, introduz o pseudo-disparo, que permite expressar a falta de conhecimento em relação ao estado de determinado objeto. O modelo de base utilizado pela ferramenta é a rede de Petri a Objetos, o qual supre uma das deficiências do modelo clássico que é a representação dos dados.

A abordagem utilizada para a construção do simulador utiliza um sistema a base de regras (no caso um sistema especialista) para fazer a representação da rede de Petri e também do comportamento dinâmico da mesma, isto é, o disparo das transições.

O suporte computacional para a construção da ferramenta é o FASE, um sistema que constitui um arcabouço para a construção de sistemas especialistas, apresentando diversas funcionalidades para construção destes, como estratégias de controle da inferência, formalismos de representação de conhecimento e resolução de conflito, dentre outros.

No sistema desenvolvido, cada rede de Petri a ser simulada é transformada em uma base de conhecimento baseada no formalismo de quadros e quatro conjuntos de regras. Os aspectos estruturais da rede, bem como as classes de objetos e instâncias de objetos são representadas em uma base de fatos que utiliza o formalismo de quadros.

Os conjuntos de regras fornecem as funcionalidades de resolução de conflitos de fichas, disparo normal das transições, criação da imprecisão e propagação da imprecisão.

Para a especificação da rede a ser executada pelo simulador foi desenvolvida uma linguagem de entrada. Esta linguagem permite expressar os diversos elementos que definem uma rede de Petri a objetos, tais como, as classes de objetos, as instâncias de objetos, as variáveis associadas aos arcos, a estrutura da rede, as condições para o disparo normal e para o pseudo-disparo, entre outros.

Ao receber o arquivo com a especificação da rede, a ferramenta verifica se a especificação está correta, sob o ponto de vista sintático e, não havendo nenhum erro, gera automaticamente os arquivos que definem o sistema especialista que controlará a simulação.

A ferramenta oferece diferentes estratégias para a resolução de conflito das fichas (prioridade, primeira encontrada, aleatória, pelo usuário), o que em última instância determina o modo de execução da simulação: automática ou controlada pelo usuário. Para a resolução de conflito das transições a ferramenta utiliza o próprio mecanismo de resolução de conflito do FASE (no caso, a estratégia é a ordem das regras).

A interface com o usuário da ferramenta está implementada em modo texto. No início de cada ciclo de inferência são apresentadas a marcação atual da rede e a lista das transições habilitadas pela marcação com as respectivas n-uplas de objetos. Após isto, cada transição habilitada pela marcação e pela interpretação, é apresentada ao usuário para ele definir se a transição vai disparar e com qual n-upla de objetos.

Como perspectivas para trabalhos futuros pode-se apontar primeiramente a criação de um módulo para o gerenciamento da simulação onde poderiam ser implementadas funcionalidades como modificação on-line da marcação, roteiros de disparo de transições, armazenamento de estatísticas de disparo, dentre outras. Este módulo contará com uma interface gráfica, o que facilitará tanto a especifição da rede a ser simulada, como o controle da simulação por parte do usuário.

Outra perspectiva relevante é estender a distribuição de possibilidades associada à localização dos objetos, objetivando aumentar ainda mais a sua flexibilidade e capacidade de representação. Atualmente a ferramenta comporta a distribuição de possibilidades limitada ao conjunto $\{0, 1\}$, o que poderia ser estendido para suportar todo o intervalo $[0, 1]$.

Uma vez que a simulação evolui levando em conta os dados externos da simulação (sensores), outra perspectiva que se apresenta, é a definição de um módulo que implemente uma interface entre o simulador e a planta de um sistema de manufatura. Esta interface deve estar apta à receber valores de sensores e à alterar valores de atuadores. Com isso a ferramenta poderia ser utilizada como elemento de supervisão de sistemas de manufatura, com a vantagem de suportar falhas e intervenções humanas sobre a planta.

Apêndice A

Conceitos e definições do modelo de rede de Petri

Este apêndice pretende dar continuidade a apresentação do modelo de rede de Petri ordinária iniciada no capítulo 2. Neste sentido, inicialmente serão abordados alguns questões referentes as características do modelo e em seguida serão apresentados alguns conceitos e definições pertinentes ao modelo.

Como foi visto no início do capítulo 2, o modelo de rede de Petri se destaca pela sua expressividade. A rede de Petri captura de maneira natural as diversas relações de causalidade (sincronização, paralelismo, conflito) existentes em sistemas que utilizam a noção de recurso, como os sistemas de manufatura.

Em relação a esta expressividade, conforme podemos observar na figura A.1, a rede Petri permite descrever com precisão as relações de causalidade como a seqüência (A.1.a); a concorrência ou paralelismo (A.1.b); o conflito (A.1.c) e a sincronização (A.1.d).

Os diferentes tipos de interações existentes nos sistemas de manufatura (repetição de operações, evolução com sincronismo e sem sincronismo, desvios, caminhos alternativos, etc.) podem ser modelados através da composição das diferentes relações mostradas na figura A.1.

Entretanto, existe uma restrição em relação à expressividade do modelo de rede de Petri que diz respeito à representação de dados envolvidos na especificação (como características de uma peça que está sendo manipulada, ou a quantidade de produtos

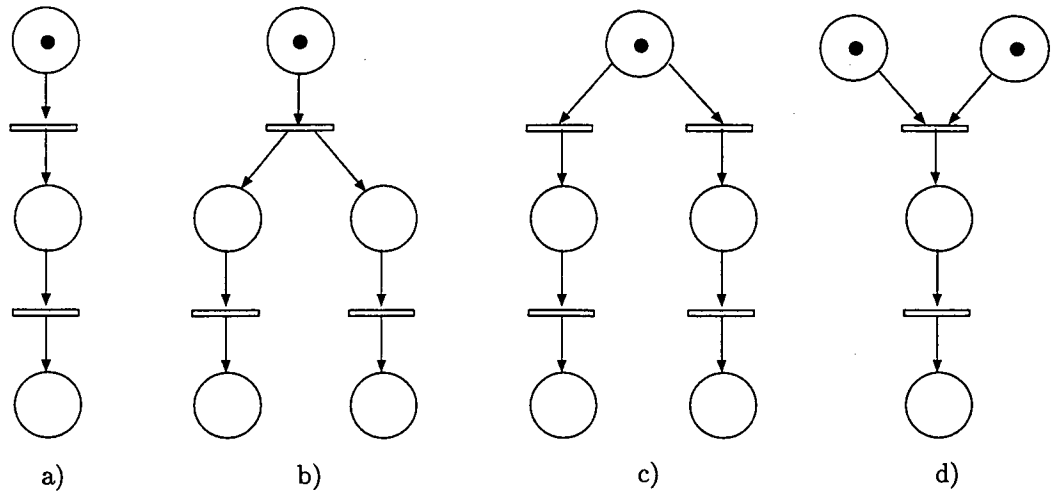


Figura A.1: Relações de causalidade Modeladas com rede de Petri

que um pallet está armazenando, por exemplo). O modelo clássico não possibilita este tipo de representação, sendo necessário a introdução de outras estruturas que estendem o modelo clássico.

Apesar desta limitação do modelo clássico, conforme foi visto, o modelo apresenta outras características positivas como o nível de abstração e o formalismo matemático oferecidos.

Este formalismo disponibiliza a verificação de algumas propriedades do sistema modelado, que são especialmente úteis em sistemas de manufatura, como a vivacidade, a limitabilidade e a reiniciabilidade. Estas chamadas *boas propriedades* são obtidas através da enumeração das marcações e são específicas para a o estado inicial em que se encontra a rede analisada.

É possível também análise das propriedades gerais da rede (*análise estrutural*). Esta análise é realizada através do cálculo dos componentes conservativos e repetitivos estacionários da rede. Tal cálculo fornece características da estrutura da rede de Petri analisada (independente da marcação) e fornece: 1) os lugares onde a soma de objetos se mantém constante, independente das transições que são disparadas; 2) as seqüências de disparo de transições que não alteram a marcação da rede, ou seja, que fazem com que uma ficha, ou um conjunto delas, volte a um determinado lugar ou conjunto de lugares.

A partir da apresentação informal da rede de Petri apresentada no capítulo 2,

é possível perceber a associação da representação matricial (definida também no capítulo 2 por P, T, Pre, Post e M) com a representação gráfica (que utiliza circulos para representar os lugares, retângulos para as transições, arcos, etc) da rede de Petri. Um arco liga um lugar p a uma transição t se, e somente se, $Pre(p,t) \neq 0$. Um arco liga uma transição t a um lugar p se, e somente se, $Post(p,t) \neq 0$.

A partir dos elementos $a_{ij} = Pre(p_i, t_j)$ - que indicam o peso do arco -, ligando o lugar de entrada p_i à transição t_j, define-se a matriz de incidência anterior Pre de dimensão $n \times m$: o número de linhas é igual ao número de lugares e o número de colunas é igual ao número de transições. Da mesma forma, a matriz de incidência posterior $Post$ de dimensão $n \times m$ é definida a partir dos elementos $b_{ij} = Post(p_i, t_j)$.

Os valores não nulos das matrizes Pre e $Post$ são associados aos arcos do grafo como etiquetas. Se o valor é unitário, não é necessário etiquetar o arco correspondente no grafo. Da mesma forma, se nada é indicado no grafo, o valor correspondente na matriz é unitário.

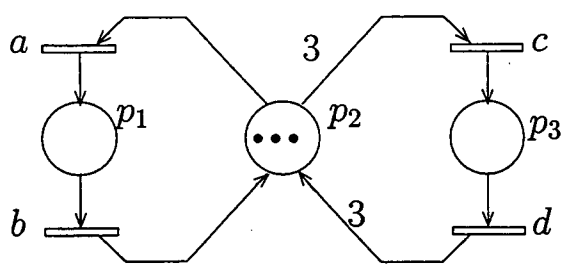


Figura A.2: Rede de Petri

A rede de Petri da figura A.2 representa a partilha de um conjunto de recursos (três), representado pelo lugar p_2, entre duas atividades, representadas pelos lugares p_1 e p_3. A atividade correspondente a p_1 necessita de apenas um recurso de cada vez (o peso do arco (p_2, a) vale 1, ou $Pre(p_2, a) = 1$). Já a atividade correspondente a p_3 necessita de todos os três recursos ao mesmo tempo. Observe também que há uma exclusão mútua entre p_1 e p_3: a partir do disparo de c, a transição a não pode mais disparar, e vice-versa. Entretanto, após o disparo de a, esta pode ainda disparar mais duas vezes. Assim, é possível haver três atividades p_1 sendo executadas ao mesmo tempo.

A notação matricial desta rede é dada por:

- $Pre = \begin{matrix} a & b & c & d \end{matrix}$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}$$

- $Post = \begin{matrix} a & b & c & d \end{matrix}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix} .$$

A partir de Pre e $Post$ defini-se a matriz de incidência C

$$C = Post - Pre \tag{A.1}$$

que fornece o *balanço* das fichas na rede quando do disparo das transições.

Neste exemplo, tem-se $C =$

$$\begin{bmatrix} a & b & c & d \\ 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix}.$$

Utiliza-se a notação $Pre(.,t)$ para a coluna da matriz Pre associada a uma transição t. A dimensão deste vetor é dada pelo número de lugares. Da mesma forma, definem-se os vetores coluna $Post(.,t)$ e $C(.,t)$ em relação às matrizes $Post$ e C, respectivamente.

A partir da definição dos vetores $Pre(.,t)$ e $Post(.,t)$ juntamente com a definição da marcação, apresentada no item 2.1.2, pode-se definir formalmente quando uma transição está sensibilizada. Uma transição t está sensibilizada ou habilitada *enabled* em inglês) se e somente se:

$$\forall p \in P, \quad M(p) \geq Pre(p,t). \tag{A.2}$$

Isto é, se o número de fichas em cada um dos lugares de entrada for maior (ou igual) que o peso do arco que liga este lugar à transição.

A equação acima pode ser escrita na forma

$$M \geq Pre(.,t)$$

onde o vetor coluna $Pre(.,t)$ é a coluna da matriz Pre referente à transição t e M, o vetor marcação inicial.

Desta forma, se t é sensibilizada por uma marcação M, uma nova marcação M' é obtida através do tiro ou disparo (*firing* em inglês) de t tal que:

$$\forall p \in P, \quad M'(p) = M(p) - Pre(p,t) + Post(p,t). \tag{A.3}$$

A nova marcação M' é dada pela equação

$$M' = M - Pre(.,t) + Post(.,t) = M + C(.,t). \tag{A.4}$$

O vetor coluna $Pre(.,t_i)$ referente à transição t_i da matriz Pre, pode ser escrito como $Pre[1]_i$, produto da matriz Pre pelo vetor coluna $[1]_i$ (vetor com todos os elementos nulos, exceto o da linha i).

A equação acima pode, então, ser reescrita como

$$M' = M - Pre[1]_i + Post[1]_i = M + C[1]_i. \tag{A.5}$$

Na rede de Petri da figura A.2, após o tiro de a a partir da marcação inicial M, obtém-se, utilizando-se a equação A.5, a seguinte marcação M':

$$\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}.$$

A exclusão mútua entre as transições a e c possibilita a introdução de uma maneira mais formal da noção de conflito. O conflito pode existir na estrutura do modelo, sem poder ser efetivado: é necessário que a marcação permita a sua ocorrência. Assim, pode-se caracterizar de forma diferenciada o conflito estrutural e o conflito efetivo.

Conflito estrutural: duas transições t_1 e t_2 estão em conflito estrutural se e somente se elas têm ao menos um lugar de entrada em comum:

$$\exists p \in P, \quad Pre(p,t_1)Pre(p,t_2) \neq 0. \tag{A.6}$$

Conflito efetivo: duas transições t_1 e t_2 estão em conflito efetivo para uma marcação

M se e somente se ambas estão em conflito estrutural e estão sensibilizadas:

$$M \geq Pre(.,t_1) \text{ e } M \geq Pre(.,t_2). \tag{A.7}$$

No exemplo, pode-se mostrar que as transições a e c estão em conflito estrutural visto que $Pre(p_2,a)Pre(p_2,c) \neq 0$. O conflito efetivo entre as duas transições acontecerá somente quando o número de fichas no lugar p_2 for maior ou igual a três. Caso o número de fichas em p_2 seja menor do que três, somente o disparo de a é possível, não existindo portanto o conflito efetivo.

Em relação ao paralelismo entre as transições, de maneira análoga ao definido para o conflito, é possível caracterizar a existência de 2 tipos de paralelismo: o paralelismo estrutural e o paralelismo efetivo.

Paralelismo estrutural: duas transições t_1 e t_2 são paralelas estruturalmente se não possuem nenhum lugar de entrada em comum:

$$\forall p \in P \;\; Pre(p,t_1)Pre(p,t_2) = 0 \tag{A.8}$$

que equivale a verificar se $Pre(.,t_1)^T \times Pre(.,t_2) = 0$, onde $\times$ é o produto escalar de dois vetores.

Paralelismo efetivo: Duas transições t_1 e t_2 são paralelas para uma marcação dada M se e somente se são paralelas estruturalmente e:

$$M \geq Pre(.,t_1) \text{ e } M \geq Pre(.,t_2). \tag{A.9}$$

Na figura A.2 as transições b e d são paralelas estruturalmente.

$$Pre(.,b) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ e } Pre(.,d) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

tem-se $Pre(.,b)^T \times Pre(.,d) = 0$.

Na mesma estrutura da rede da figura A.2, mas agora com a marcação

$$M' = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

as transições b e d são efetivamente paralelas.

Apêndice B

O exemplo da célula de usinagem

Neste apêndice serão apresentados os arquivos que definem a especificação da rede de Petri para o exemplo da célula de usinagem, apresentada na seção 2.2 (página 14). Serão apresentados também os arquivos gerados automaticamente que definem o sistema especialista que controla a simulação da célula de usinagem. A seguir é apresentado o arquivos com a especificação da rede utilizando a sintaxe da linguagem de entrada definida na ferramenta.

```
;;;;;;;;;; Inicio do Arquivo usinagem.dat ;;;;;;;;;;;;;;;;

;; Exemplo da célula de usinagem
;; Arquivo de especificação da rede

  Class:=
    Pallet: (posicao );
    Peca: (job );
    Maquina: (posicao );

  Places:=
    peca_estoque: <Peca>;
    pallet_estoque: <Pallet>;
    mesa: <Pallet, Peca>;
    maq_livre: <Maquina>;
    usinando: <Pallet, Peca, Maquina>;

  Objects:=
    Pallet.pal_1: (location pallet_estoque) (posicao 1) ;
    Pallet.pal_2: (location pallet_estoque) (posicao 2) ;
    Pallet.pal_3: (location pallet_estoque) (posicao 3) ;
    Pallet.pal_4: (location pallet_estoque) (posicao 4) ;
    Maquina.M1: (posicao 2) (location maq_livre) ;
    Maquina.M2: (posicao 3) (location maq_livre) ;
    Peca.P1: (job (M1 M2 M1)) (location peca_estoque) ;
```

```
    Peca.P2: (job (M2 M1))   (location peca_estoque) ;

  Variables:=
    Pallet: pal;
    Peca: pec;
    Maquina: maq;

  Structure:=
    Entrada (user): (pallet_estoque <pal>, peca_estoque <pec>)
                                          -> (mesa <pal, pec>);
    Rotacao (user): (mesa <pal, pec>) -> (mesa <pal, pec>);
    Inicio  (user): (mesa <pal, pec>, maq_livre <maq>)
                        -> (usinando <pal, pec, maq>);
    Fim     (user): (usinando <pal, pec, maq>)
                    -> (mesa <pal, pec>, maq_livre <maq>);
    Saida   (user): (mesa <pal, pec>)
                  -> (peca_estoque <pal>, peca_estoque <pec>);

  Conditions:=
    Entrada:
      true: ver_entrada(pec.job, pal.posicao);
    Rotacao:
      true: ver_rotacao(pal.posicao);
    Inicio:
      true: ver_inicio(pal.posicao, maq.posicao, maq.ident, pec.job);
    Saida:
      true: ver_saida(pec.job, pal.posicao);

  Actions:=
    Rotacao:
      functions:
        pal.posicao = exec_rotacao(pal.posicao);
    Inicio:
      functions:
        pec.job = exec_inicio(pec.job);

    file: usinagem.lsp;

  Endnet
;;;;;;;;;; Final do Arquivo usinagem.dat ;;;;;;;;;;;;;;;;
```

Na descrição da rede que modela a célula de usinagem são definidas quatro funções para verificação dos atributos dos objetos que devem ser disparados pela transição. Além disto, os atributos *pal.posicao* e it pec.job são modificados, respectivamente pelas transições *Rotação* e *Início* através das funções *exec_rotacao* e *exec_inicio*. Como descrito na especificação estas funções são implementadas no arquivo *usinagem.lsp*, o qual é listado a seguir.

```
;;;;;;;;;; Inicio do Arquivo usinagem.lsp ;;;;;;;;;;;;;;;;

;; Arquivo com a implementação das funcões
```

```
;; de verificação das condições
;; e aplicação das ações das transições
;; para o exemplo da celula de usinagem

(defun ver_entrada (pec.job pal.posicao)
   (cond ((and (not (null pec.job))
               (= pal.posicao 1)) t)))

(defun ver_rotacao(pal.posicao)
   (cond ((or (= pal.posicao 1)
              (= pal.posicao 2)
              (= pal.posicao 3)
              (= pal.posicao 4)) t)))

(defun ver_inicio (pal.posicao maq.posicao maq.ident pec.job)
   (cond ((and (equal pal.posicao maq.posicao)
               (equal maq.ident (car pec.job))) t)))

(defun ver_saida(pec.job pal.posicao)
   (cond ((and (null pec.job)
               (= pal.posicao 4)) t)))

(defun exec_rotacao (pal.posicao)
   (case pal.posicao
            (1 (return-from exec_rotacao 2))
            (2 (return-from exec_rotacao 3))
            (3 (return-from exec_rotacao 4))
            (4 (return-from exec_rotacao 1))))

(defun exec_inicio (pec.job)
   (return-from exec_inicio (cdr pec.job)))

;;;;;;;;;; Final do Arquivo usinagem.lsp ;;;;;;;;;;;;;;;;
```

Para esta especificação é gerado a seguinte base de fatos. No início do arquivo são definidos as informações que utilizam o formalismo de lógica (no caso, a representação do tempo, o nome do arquivo que implementa as funções e as variáveis externas). A seguir é definida a estrutura de quadros mostrada na figura 5.2 e a partir de então as informações da rede são armazenadas.

```
;;;;;;;;;; Inicio do Arquivo usinagem.f ;;;;;;;;;;;;;;;;
;; Arquivo de fatos para o
;; exemplo da célula de usinagem

((logic (now 1)
        (file usinagem.lsp)
        (extern (nil)))

 (frame
   (rdp root)
   (class root)
```

```
   (transition root)

   (composicao rdp
     (componente_1 )
     (componente_2 )
     (componente_3 ))
   (lugar rdp
     (interface place if-modify))

;; ------> Transitions Functions <-------

   (entrada transition
       (certain (ver_entrada (pec job) (pal posicao)))
       (uncert  nil)
       (lug_entr (pallet_estoque peca_estoque))
       (var_entr ((pal) (pec))))

   (rotacao transition
       (certain (ver_rotacao (pal posicao)))
       (uncert  nil)
       (lug_entr (mesa))
       (var_entr ((pal pec))))

   (inicio transition
       (certain (ver_inicio (pal posicao) (maq posicao) (maq ident)
                    (pec job)))
       (uncert  nil)
       (lug_entr (mesa maq_livre))
       (var_entr ((pal pec) (maq))))

   (fim transition
       (certain nil)
       (uncert  nil)
       (lug_entr (usinando))
       (var_entr ((pal pec maq))))

   (saida transition
       (certain (ver_saida (pec job) (pal posicao)))
       (uncert  nil)
       (lug_entr (mesa))
       (var_entr ((pal pec))))

;; ------> Class <-------

   (pallet class
       (variables (pal))
       (posicao nil))

   (peca class
       (variables (pec))
       (job nil))

   (maquina class
       (variables (maq))
       (posicao nil))
```

```
;; ------> Objects <-------

   (pal_1 pallet
       (ident pal_1)
       (arrived 0)
       (location pallet_estoque)
       (posicao 1))

   (pal_2 pallet
       (ident pal_2)
       (arrived 0)
       (location pallet_estoque)
       (posicao 2))

   (pal_3 pallet
       (ident pal_3)
       (arrived 0)
       (location pallet_estoque)
       (posicao 3))

   (pal_4 pallet
       (ident pal_4)
       (arrived 0)
       (location pallet_estoque)
       (posicao 4))

   (m1 maquina
       (ident m1)
       (arrived 0)
       (posicao 2)
       (location maq_livre))

   (m2 maquina
       (ident m2)
       (arrived 0)
       (posicao 3)
       (location maq_livre))

   (p1 peca
       (ident p1)
       (arrived 0)
       (job (m1 m2 m1))
       (location peca_estoque))

   (p2 peca
       (ident p2)
       (arrived 0)
       (job (m2 m1))
       (location peca_estoque))

;; ------> Places <-------

   (peca_estoque lugar
       (identificacao peca_estoque)
       (tipo_objeto (peca))
       (transicoes_anteriores (saida saida))
```

```
      (transicoes_posteriores (content_entrada))
      (interface )
      (content_unc )
      (content (p2 p1))
      (content_entrada ))

   (pallet_estoque lugar
      (identificacao pallet_estoque)
      (tipo_objeto (pallet))
      (transicoes_anteriores nil)
      (transicoes_posteriores ( content_entrada))
      (interface )
      (content_unc )
      (content (pal_4 pal_3 pal_2 pal_1))
      (content_entrada ))

   (mesa lugar
      (identificacao mesa)
      (tipo_objeto (pallet peca))
      (transicoes_anteriores (entrada rotacao fim))
      (transicoes_posteriores (content_rotacao content_inicio content_saida))
      (interface )
      (content_unc )
      (content nil)
      (content_rotacao )
      (content_inicio )
      (content_saida ))

   (maq_livre lugar
      (identificacao maq_livre)
      (tipo_objeto (maquina))
      (transicoes_anteriores (fim))
      (transicoes_posteriores (content_inicio))
      (interface )
      (content_unc )
      (content (m2 m1))
      (content_inicio ))

   (usinando lugar
      (identificacao usinando)
      (tipo_objeto (pallet peca maquina))
      (transicoes_anteriores (inicio))
      (transicoes_posteriores (content_fim))
      (interface )
      (content_unc )
      (content nil)
      (content_fim ))
))
;;;;;;;;;; Final do Arquivo usinagem.f ;;;;;;;;;;;;;;;;
```

O arquivo com o conjunto de regras que realiza a resolução de conflito das fichas, denominado *usinagem_con.r*, é apresentado a seguir. Todas as transições especificadas utilizam como estratégia de resolução de conflito a escolha pelo usuário.

```
;;;;;;;;;; Inicio do Arquivo usinagem_con.r ;;;;;;;;;;;;;;;;
;; Arquivo de Resolução de Conflitos das Fichas
;; Exemplo da Célula de Usinagem

(entrada
    ((frame (pallet_estoque (content x1)) (peca_estoque (content x2)))
     (include (z . (ask-instance 'user 'entrada '(pallet_estoque peca_estoque) 'content_entrada))))
    ())

(rotacao
    ((frame (mesa (content x1)))
     (include (z . (ask-instance 'user 'rotacao '(mesa) 'content_rotacao))))
    ())

(inicio
    ((frame (mesa (content x1)) (maq_livre (content x2)))
     (include (z . (ask-instance 'user 'inicio '(mesa maq_livre) 'content_inicio))))
    ())

(fim
    ((frame (usinando (content x1)))
     (include (z . (ask-instance 'user'fim '(usinando) 'content_fim))))
    ())

(saida
    ((frame (mesa (content x1)))
     (include (z . (ask-instance 'user 'saida '(mesa) 'content_saida))))
    ())
;;;;;;;;;; Final do Arquivo usinagem_con.r ;;;;;;;;;;;;;;;;
```

Para finalizar é apresentado o arquivo com o conjunto de regras que realiza o disparo normal da transição. Neste arquivo pode-se visualizar os diferentes formatos das regras, devido a manipulação das relações dinâmicas e a aplicação das funções que executam as ações da transição.

```
;;;;;;;;;; Inicio do Arquivo usinagem.r ;;;;;;;;;;;;;;;;
;; Arquivo de Regras de Disparo Normal das Transições
;; Exemplo da Célula de Usinagem

(entrada
  ((frame
     (pallet_estoque (content_entrada ?-pal))
     (peca_estoque (content_entrada ?-pec)))
   (include (z1 . (gensym "-LLC-"))))
  ((frame
     (pallet_estoque (interface (remove ?-pal content_entrada)))
     (peca_estoque (interface (remove ?-pec content_entrada)))
     (z1 composicao (componente_1 ?-pal) (componente_2 ?-pec))
     (mesa (interface (push z1))))))

(rotacao
  ((frame
```

```
     (mesa (content_rotacao ?-mesa))
     (?-mesa composicao (componente_1 ?-pal) (componente_2 ?-pec))
     (?-pal ?-y1)
     (?-pal (posicao ?-p2))   )
   (include (z1 . (exec_rotacao '?-p2)))
   (include (z2 . (gensym "-LLC-"))))
  ((frame
     (off (?-mesa))
     (mesa (interface (remove ?-mesa content_rotacao)))
     (?-pal ?-y1 (posicao z1))
     (z2 composicao (componente_1 ?-pal) (componente_2 ?-pec))
     (mesa (interface (push z2))))))

(inicio
  ((frame
     (mesa (content_inicio ?-mesa))
     (?-mesa composicao (componente_1 ?-pal) (componente_2 ?-pec))
     (maq_livre (content_inicio ?-maq))
     (?-pec ?-y1)
     (?-pec (job ?-p2))   )
   (include (z1 . (exec_inicio '?-p2)))
   (include (z2 . (gensym "-LLC-")))
   (include (z3 . (gensym "-LLC-"))))
  ((frame
     (off (?-mesa))
     (mesa (interface (remove ?-mesa content_inicio)))
     (maq_livre (interface (remove ?-maq content_inicio)))
     (?-pec ?-y1 (job z1))
     (z2 composicao (componente_1 ?-pal) (componente_2 ?-pec)
                    (componente_3 ?-maq))
     (usinando (interface (push z2))))))

(fim
  ((frame
     (usinando (content_fim ?-usinando))
     (?-usinando composicao (componente_1 ?-pal) (componente_2 ?-pec)
                            (componente_3 ?-maq)))
   (include (z1 . (gensym "-LLC-"))))
  ((frame
     (off (?-usinando))
     (usinando (interface (remove ?-usinando content_fim)))
     (z1 composicao (componente_1 ?-pal) (componente_2 ?-pec))
     (mesa (interface (push z1)))
     (maq_livre (interface (push ?-maq))))))

(saida
  ((frame
     (mesa (content_saida ?-mesa))
     (?-mesa composicao (componente_1 ?-pal) (componente_2 ?-pec)))
   (include (z1 . (gensym "-LLC-"))))
  ((frame
     (off (?-mesa))
     (mesa (interface (remove ?-mesa content_saida)))
     (pallet_estoque (interface (push ?-pal)))
     (peca_estoque (interface (push ?-pec))))))
;;;;;;;;;; Final do Arquivo usinagem.r ;;;;;;;;;;;;;;;;
```

Apêndice C

O exemplo do sistema de AGVs

Neste apêndice serão apresentados os arquivos que definem a especificação da rede de Petri para o exemplo do sistema de AGVs, apresentado na seção 2.3.4. Serão apresentados também os arquivos gerados automaticamente que definem o sistema especialista que realiza a simulação do sistema de AGVs. A seguir é apresentado o arquivos com a especificação da rede utilizando a sintaxe da linguagem de entrada definida na ferramenta.

```
;;;;;;;;;; Inicio do Arquivo agv.dat ;;;;;;;;;;;;;;;
;; Arquivo de especificação do
;; Exemplo do sistema de AGVs

  Class:=
    veiculo: (trajeto ) (proxima_s);
    sessao: (status );

  Places:=
    MS:  <veiculo, sessao>;
    S:   <sessao>;
    C:   <veiculo, sessao>;
    MOS: <veiculo>;
    LB:  <veiculo>;

  Objects:=
    veiculo.v1: (location C) (proxima_s s3) (trajeto (s1 s3 s4 s2));
    veiculo.v2: (location C) (proxima_s s2) (trajeto (s4 s2 s3 s1));
    sessao.s1: (status t) (location C) ;
    sessao.s2: (status t) (location S) ;
    sessao.s3: (status t) (location S) ;
    sessao.s4: (status t) (location C) ;

  Variables:=
    sessao: s, ns;
```

```
    veiculo: v;

  Structure:=
    trans_a (user): (MS <v,s>) -> (C <v,s>);
    trans_b (user): (C <v,s>, S <ns>)  -> (MS <v,ns>, S <s>);
    trans_f (user): (MS <v,s>) -> (MOS <v>, S <s>);
    trans_d (user): (MOS <v>) ;

  Extern:=
    trans_a: sensor_a = 0;

  Conditions:=
    trans_a:
      true: true_trans_a(v.time, sensor_a)
      uncertain: unc_trans_a(v.time);
    trans_b:
      true: true_trans_b(v.proxima_s, ns.ident);
    trans_f:
      true: true_trans_f(v.time)
      uncertain: unc_trans_f(v.time);
    trans_d:
      true: true_trans_d(v.time)
      uncertain: unc_trans_d(v.time);

  Actions:=
    trans_b:
        funtions:
          v.proxima_s = exec_troca(v.trajeto, v.proxima_s, ns.ident);

  File: agv.lsp;

Endnet
;;;;;;;;;; Final do Arquivo agv.dat ;;;;;;;;;;;;;;;;
```

Na descrição da rede que modela o sistema de AGVs são definidas quatro funções para verificar as condições de disparo preciso da transição e três funções que verificam as condições para o pseudo-disparo das transições. Além disto, o atributo *v.proxima_s* é modificado pelas transição *trans_b* através da função *exec_troca*. Como descrito na especificação estas funções são implementadas no arquivo *agv.lsp*, o qual é listado a seguir.

```
;;;;;;;;;; Inicio do Arquivo agv.lsp ;;;;;;;;;;;;;;;;

;; Arquivo com a implementação das funções
;; de verificação das condições
;; e aplicação das açõoes das transições
;; para o exemplo do sistema de AGVs

(defun true_trans_a(v.time sensor_a)
   (cond ((and (= sensor_a 1)
               (<= v.time 10)) t)))
```

```
(defun unc_trans_a(v.time)
   (cond ((> v.time 10)) t))))

(defun true_trans_b(v.proxima_s ns.ident)
   (cond ((equal v.proxima_s ns.ident))))

defun true_trans_f(v.time)
   (return-from true_trans_f nil))

(defun unc_trans_f(v.time)
   (cond ((> v.time 5)) t)

defun true_trans_d(v.time)
   (return-from true_trans_d nil))

(defun unc_trans_d(v.time)
   (cond ((> v.time 5)) t)

(defun exec_troca(v.trajeto v.proxima_s ns.ident)
  (dotimes (cont (length v.trajeto))
    (cond ((equal  (nth cont v.trajeto) ns.ident)
           (return-from exec_troca (nth (+ cont 1) v.trajeto))))))

;;;;;;;;;; Final do Arquivo agv.lsp ;;;;;;;;;;;;;;;;
```

A partir da especificação apresentada é gerada a base de fatos mostrada a seguir. No início do arquivo são definidos as informações que utilizam o formalismo de lógica (no caso, a representação do tempo, o nome do arquivo que implementa as funções e as variáveis externas). A seguir é definida a estrutura de quadros, e a partir de então as informações da rede especificada são armazenadas.

```
;;;;;;;;;; Inicio do Arquivo agv.f ;;;;;;;;;;;;;;;;
;; Arquivo de fatos para o
;; exemplo do sistema de AGVs

((logic (now 1)
        (file agv.lsp)
        (extern (sensor_a ))
        (sensor_a 0))

 (frame
   (rdp root)
   (class root)
   (transition root)

   (composicao rdp
     (componente_1 )
     (componente_2 )
     (componente_3 ))
   (lugar rdp
```

```
    (interface place if-modify))

;; ------> Initials Compositions <-------

   (-llc-1001 composicao (arrived 0) (componente_1 v1)
       (componente_2 s1))

   (-llc-1002 composicao (arrived 0) (componente_1 v2)
       (componente_2 s4))

;; ------> Transitions Functions <-------

   (trans_a transition
       (sensor_a 0)
       (certain (true_trans_a (v time) sensor_a))
       (uncert  (unc_trans_a (v time)))
       (lug_entr (ms))
       (var_entr ((v s))))

   (trans_b transition
       (certain (true_trans_b (v proxima_s) (ns ident)))
       (uncert  nil)
       (lug_entr (c s))
       (var_entr ((v s) (ns))))

   (trans_f transition
       (certain (true_trans_f (v time)))
       (uncert  (unc_trans_f (v time)))
       (lug_entr (ms))
       (var_entr ((v s))))

   (trans_d transition
       (certain (true_trans_d (v time)))
       (uncert  (unc_trans_d (v time)))
       (lug_entr (mos))
       (var_entr ((v))))

;; ------> Class <-------

   (veiculo class
       (variables (v))
       (trajeto nil)
       (proxima_s nil))

   (sessao class
       (variables (s ns))
       (status nil))

;; ------> Objects <-------

   (v1 veiculo
       (ident v1)
       (arrived 0)
       (location c)
       (proxima_s s3)
       (trajeto (s1 s3 s4 s2)))
```

```
   (v2 veiculo
       (ident v2)
       (arrived 0)
       (location c)
       (proxima_s s2)
       (trajeto (s4 s2 s3 s1)))

   (s1 sessao
       (ident s1)
       (arrived 0)
       (status t)
       (location c))

   (s2 sessao
       (ident s2)
       (arrived 0)
       (status t)
       (location s))

   (s3 sessao
       (ident s3)
       (arrived 0)
       (status t)
       (location s))

   (s4 sessao
       (ident s4)
       (arrived 0)
       (status t)
       (location c))

;; ------> Places <-------

   (ms lugar
       (identificacao ms)
       (tipo_objeto (veiculo sessao))
       (transicoes_anteriores (trans_b))
       (transicoes_posteriores (content_trans_a content_trans_f))
       (interface )
       (content_unc )
       (content nil)
       (content_trans_a )
       (content_trans_f ))

   (s lugar
       (identificacao s)
       (tipo_objeto (sessao))
       (transicoes_anteriores (trans_b trans_f))
       (transicoes_posteriores (content_trans_b))
       (interface )
       (content_unc )
       (content (s3 s2))
       (content_trans_b ))

   (c lugar
```

```
      (identificacao c)
      (tipo_objeto (veiculo sessao))
      (transicoes_anteriores (trans_a))
      (transicoes_posteriores (content_trans_b))
      (interface )
      (content_unc )
      (content (-llc-1002 -llc-1001))
      (content_trans_b ))

   (mos lugar
      (identificacao mos)
      (tipo_objeto (veiculo))
      (transicoes_anteriores (trans_f))
      (transicoes_posteriores (content_trans_d))
      (interface )
      (content_unc )
      (content nil)
      (content_trans_d ))

   (lb lugar
      (identificacao lb)
      (tipo_objeto (veiculo))
      (transicoes_anteriores nil)
      (transicoes_posteriores ())
      (interface )
      (content_unc )
      (content nil))
))
;;;;;;;;;; Final do Arquivo agv.f ;;;;;;;;;;;;;;;;
```

O arquivo com o conjunto de regras que realiza a resolução de conflito das fichas, denominado *agv_con.r*, é apresentado a seguir. Todas as transições especificadas utilizam como estratégia de resolução de conflito a escolha pelo usuário.

```
;;;;;;;;;; Inicio do Arquivo agv_con.r ;;;;;;;;;;;;;;;;
;; Arquivo de Resolução de Conflitos das Fichas
;; Exemplo do sistema de AGVs

(trans_a
    ((frame (ms (content x1)))
     (include (z . (ask-instance 'user 'trans_a '(ms) 'content_trans_a))))
    ())

(trans_b
    ((frame (c (content x1))(s (content x2)))
     (include (z . (ask-instance 'user 'trans_b '(c s) 'content_trans_b))))
    ())

(trans_f
    ((frame (ms (content x1)))
     (include (z . (ask-instance 'user 'trans_f '(ms) 'content_trans_f))))
    ())
```

```
(trans_d
    ((frame (mos (content x1)))
     (include (z . (ask-instance 'user 'trans_d '(mos) 'content_trans_d))))
    ())
;;;;;;;;;; Final do Arquivo agv_con.r ;;;;;;;;;;;;;;;;
```

A seguir é apresentado o arquivo com o conjunto de regras que realiza o disparo normal da transição. Os diferentes formatos das regras, devido a manipulação das relações dinâmicas e a aplicação das funções que executam as ações da transição, podem ser vistas neste arquivo.

```
;;;;;;;;;; Inicio do Arquivo agv.r ;;;;;;;;;;;;;;;;
;; Arquivo de Regras de Disparo Normal das Transições
;; Exemplo do Sistema de AGVs

  (trans_a
    ((frame
       (ms (content_trans_a ?-ms))
       (?-ms composicao (componente_1 ?-v) (componente_2 ?-s)))
     (include (z1 . (gensym "-LLC-"))))
    ((frame
       (off (?-ms))
       (ms (interface (remove ?-ms content_trans_a)))
       (z1 composicao (componente_1 ?-v) (componente_2 ?-s))
       (c (interface (push z1))))))

  (trans_b
    ((frame
       (c (content_trans_b ?-c))
       (?-c composicao (componente_1 ?-v) (componente_2 ?-s))
       (s (content_trans_b ?-ns))
       (?-v ?-y1)
       (?-v (trajeto ?-p2))
       (?-v (proxima_s ?-p3))
       (?-ns (ident ?-p4))   )
     (include (z1 . (exec_troca '?-p2 '?-p3 '?-p4)))
     (include (z2 . (gensym "-LLC-"))))
    ((frame
       (off (?-c))
       (c (interface (remove ?-c content_trans_b)))
       (s (interface (remove ?-ns content_trans_b)))
       (?-v ?-y1 (proxima_s z1))
       (z2 composicao (componente_1 ?-v) (componente_2 ?-ns))
       (ms (interface (push z2)))
       (s (interface (push ?-s))))))

  (trans_f
    ((frame
       (ms (content_trans_f ?-ms))
       (?-ms composicao (componente_1 ?-v) (componente_2 ?-s)))
     (include (z1 . (gensym "-LLC-"))))
```

```
    ((frame
       (off (?-ms))
       (ms (interface (remove ?-ms content_trans_f)))
       (mos (interface (push ?-v)))
       (s (interface (push ?-s))))))

  (trans_d
    ((frame
       (mos (content_trans_d ?-v)))
     (include (z1 . (gensym "-LLC-"))))
    ((frame
       (mos (interface (remove ?-v content_trans_d))))))

;;;;;;;;;; Final do Arquivo agv.r ;;;;;;;;;;;;;;;;
```

Como pode ser visto no arquivo de especificação, algumas transições desta sistema, definem condições para o disparo pseudo disparo. A seguir é apresentado o arquivo de define o conjunto de regras responsável pela criação da imprecisão.

```
;;;;;;;;;; Inicio do Arquivo agv_pre.r ;;;;;;;;;;;;;;;;
;; Arquivo de Regras de Criação da Imprecisão
;; Exemplo do Sistema de AGVs

(trans_a
    ((frame (ms (content x1)))
     (include (z . (ask-precision 'user 'trans_a '(ms)))))
    ())

(trans_f
    ((frame (ms (content x1)))
     (include (z . (ask-precision 'user 'trans_f '(ms)))))
    ())

(trans_d
    ((frame (mos (content x1)))
     (include (z . (ask-precision 'user 'trans_d '(mos)))))
    ())
;;;;;;;;;; Final do Arquivo agv_pre.r ;;;;;;;;;;;;;;;;
```

Finalmente, é apresentado o arquivo que contém o conjunto de regras responsável pela propagação da imprecisão. Como colocado anteriormente, estas regras colocam os objetos nos lugares de saída da transição, sem entretanto retirá-los dos lugares de entrada. Este conjunto de regras está definido no arquivo *agv_unc.r*, o qual é apresentado a seguir.

```
;;;;;;;;;; Inicio do Arquivo agv_unc.r ;;;;;;;;;;;;;;;;
;; Arquivo de Regras de Propagação da Imprecisão
;; Exemplo do Sistema de AGVs
```

```
        (trans_a
          ((frame (ms (content ?-ms)))
           (include (z1 . (propagar '?-ms))))
          ((frame (c (content_unc z1)))))

        (trans_f
          ((frame (ms (content_unc ?-ms)))
           (include (z1 . (propagar '?-ms)))
           (include (z2 . (car z1)))
           (include (z3 . (cadr z1))))
          ((frame (mos (content_unc z2))
                  (s (content_unc z3)))))

        (trans_d
          ((frame (mos (content ?-mos)))
           (include (z1 . (propagar '?-mos))))
          ((frame (lb (content_unc z1)))))

;;;;;;;;;; Final do Arquivo agv_unc.r ;;;;;;;;;;;;;;;;
```

Referências Bibliográficas

[BB94] A.J. Bugarin and S. Barro. Fuzzy reasoning supported by Petri nets. *IEEE Transaction on Fuzzy Systems*, 2(2):135–150, may 1994.

[Bit95] G. Bittencourt. Um ambiente para ensino e desenvolvimento de sistemas especialistas. In *III Workshop sobre Educação em Informática/IV Congresso Íbero-Americano de Educação Superior em Computação*, 1995. 29 de julho a 4 de agosto, Canela, RS.

[Bit96] G. Bittencourt. *Inteligência Artificial: Ferramentas e Teorias.* Editora da UNICAMP, Campinas, SP, 1996.

[BM93] G. Bittencourt and M. Marengoni. A customizable tool for the generation of production-based systems. In G. Rzevski, J. Pastor, and R.A. Adey, editors, *Eighth International Conference on Applications of Artificial Intelligence in Engineering (AIENG'93)*, pages 337–352. Computational Mechanics Publications, Elsevier Applied Science, 1993. held in Toulouse, France, 29 June - 1 July, 1993.

[Can90] E. Cantú. *Uma Abordagem para a Representação, Simulação e Implementação de Sistemas Baseada na Rede de Petri a Objetos.* Dissertação de Mestrado, UFSC, 1990.

[Car98] J. Cardoso. Time fuzzy petri nets. In J. Cardoso and H. camargo, editors, *Fuzziness in Petri nets, Physica-verlag*, 1998.

[CKC90] S.-M. Chen, J.-S. Ke, and J.-F. Chang. Knowledge representation using fuzzy petri nets. *IEEE Trans. on Knowledge and Data Engeneering*, 2(3):311–19, 1990.

[CM85] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley Publishing Company, Reading, MA, 1985.

[CS93] T. Cao and A. C. Sanderson. Variable reasoning and analysis about uncertainty with Fuzzy Petri nets. In M. Ajmone Marsan, editor, *Lecture Notes in Computer Science, Application and Theory of Petri nets 1993*, volume 691, pages 126–145. Springer-Verlag, 1993.

[CV97] J. Cardoso and R. Valette. *Redes de Petri.* Editora da UFSC, Florianópolis, SC, 1997.

[CVD91] J. Cardoso, R. Valette, and D. Dubois. Petri nets with uncertain markings. In G. Rozenberg, editor, *Lecture Notes in Computer Science, Advances in Petri nets 1990*, volume 483, pages 64–78. Springer Verlag, 1991.

[DP88a] D. Dubois and H. Prade. An introdution to possibilistic and fuzzy logics. In *Non-Standard logics for automated reasoning.* Plenum Press, 1988.

[DP88b] D. Dubois and H. Prade. *Possibility Theory - An Approach to the Computerized Processing of Uncertainty.* Academic Press, 1988.

[GAG91] M.L. Garg, S.I. Ahson, and P.V. Gupta. A Fuzzy Petri net for knowledge representation qand reasoning. *Information Processing Letters*, 39:165–171, 1991.

[Gen86] H.J. Genrich. Predicate Transition nets. In *Lecture Notes in Computer Science 254*, pages 207–247. Springer Verlag, 1986.

[Gev87] W.B. Gevarter. The nature and evaluation of commercial expert systems building tools. *IEEE Computer*, pages 24–41, May, 1987.

[GG95] L. Gomes and A. Steiger Garcao. Programmable controller design based on a synchronized Colored Petri net model and integrating fuzzy reasoning. In *Lecture Notes in Computer Science 935*, pages 218–237. Springer Verlag, 1995.

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.

[JC96] D. Dubois J. Cardoso, R. Valette. Fuzzy petri net: an overview. In *13^{th} IFAC World Congress, San Francisco, EUA*, 30 june-5 july 1996.

[Joh75] S. C. Johnson. Yacc: Yet another compiler-compiler. Technical report, Bell Laboratories, Murray Hill, New Jersey, October 1975. Technical Report No 32.

[Les75] M. E. Lesk. Lex - a lexical analyzer generator. Technical report, Bell Laboratories, Murray Hill, New Jersey, October 1975. Technical Report No 39.

[LG94] H-P. Lipp and R. Günther. A study of Fuzzy Petri nets concepts. In *Proc. European Congress on Fuzzy and Intelligent Technologies*, pages 1461–1468, Aachen, Germany, 1994.

[Loo88] C. G. Looney. Fuzzy petri nets for rule-based decision making. *IEEE Trans. on Systems Man and Cybernetics*, 18(1):178–183, Jan-Feb 1988.

[Maz90] C. A. Maziero. *Um Ambiente para a Análise e Simulação de Sistemas Modelados por Redes de Petri.* Dissertação de Mestrado, UFSC, 1990.

[Min75] M. Minsky. A framework to represent knowledge. In *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, 1975.

[Nil71] N.J. Nilsson. *Problem Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[NL87] B. Nebel and K. Von Luck. Issues of integration and balancing in hybrid knowledge representation systems. In K. Morik, editor, *Proceedings of the 11th German Workshop on Artificial Intelligence*, pages 115–123, 1987. September-October.

[Pet62] C. A. Petri. *Kommunikation mit Automaten.* PhD Thesis, Univ. Bonn, 1962.

[PG94] W. Pedriycz and F. Gomide. A generalized Fuzzy Petri net model. *IEEE Transaction on Fuzzy Systems*, 2(4):295–301, nov 1994.

[Pos43] E. Post. Formal reductions of the general combinatorial problem. *American Journal of Mathematics*, 65:197–268, 1943.

[Rab95] R. A. Rabuske. *Inteligência Artificial.* Editora da UFSC, Florianópolis, SC, 1995.

[Ric83] E. Rich. *Artificial Intelligence.* McGraw-Hill Book Company, 1983.

[RV89] D. Dubois R. Valette, J. Cardoso. Monitoring manufacturing systems by means of petri nets with imprecise markings. In *Proceedings of the IEEE International Symposium on Intelligent Control 1989, Albany, New York-USA*, pages 233–8, 25-26 September 1989.

[SA75] R.C. Schank and R.P. Abelson. Scripts, plans and knowledge. In *Proceedings of IJCAI 4*, pages 151–157, 1975.

[SA77] R.C. Schank and R. Abelson. *Scripts, Plans, Goals and Understanding.* Laurence Ealbaum Associates Inc., Milesdale, N.J., 1977.

[SB85] C. Sibertin-Blanc. High-level Petri nets with data structures. In *European Workshop on Application and Theory of Petri nets*, pages 141–170, Helsinki, Finland, June 1985.

[SG93] H. Scarpelli and F. Gomide. Fuzzy reasoning and high level fuzzy Petri nets. In *Proc. First European Congress on Fuzzy and Intelligent Technologies*, pages 600–605, Aachen, Germany, Sep 7-10, 1993.

[SG94] H. Scarpelli and F. Gomide. High-level Fuzzy Petri nets and backward reasoning. In *Proc. IPMU*, pages 1275–1280, Paris, France, July 1994.

[SJ84] G.L. Steele Jr. *Common LISP, the Language.* Digital Press, Burlington, 1984.

[THT87] D.S. Touretzky, J.F. Horty, and R.H. Thomason. A clash of intuitions : The current state of nonmonotonic multiple inheritance systems. In *Proceedings of IJCAI 10*, pages 476–482, 1987.

[Val88] R. Valette. Les reseaux de petri. Rapport LAAS/CNRS, 1988.

[Wat86] D.A. Waterman. *A Guide to Expert Systems.* Addison-Wesley Publishing Company, Reading, MA, 1986.

[Win84] P.H. Winston. *Artificial Intelligence (2nd Edition).* Addison-Wesley Publishing Company, Reading, MA, 1984.

[Zad78] L.A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.