

Modelagem de Sistemas

Em muitas áreas de estudo, um fenômeno não é estudado diretamente, mas através de um **modelo**. Um modelo é uma **representação**, freqüentemente em termos matemáticos, do que parecem ser as **principais características do objeto ou sistema sob estudo**.

Exemplos:

- **Astronomia**, onde modelos do nascimento, morte e interação das estrelas permitem o estudo de teorias que consumiriam muito tempo e uma grande quantidade de matéria e energia;
- **Física nuclear**, onde as partículas atômicas e subatômicas radioativas sob estudo existem por períodos de tempo muito curtos;

- **Sociologia**, onde a manipulação direta de grupos de pessoas para estudos poderia causar problemas éticos;
- **Biologia**, onde modelos de sistemas biológicos requerem menos espaço, tempo e energia para desenvolver.

As principais **características** de muitos fenômenos físicos podem ser descritas **numericamente** e as **relações entre essas características** podem ser descritas por **equações ou desigualdades**.

Para utilizar o conceito de modelagem é necessário um **conhecimento tanto dos fenômenos modelados quanto das propriedades das técnicas de modelagem**.

Cálculo diferencial foi desenvolvido em resposta à necessidade de um meio de modelar propriedades que mudam continuamente, tais como posição, velocidade e aceleração em Física.

É possível modelar sistemas cada vez mais complexos e maiores representando-os por **um modelo matemático, convertendo o modelo em instruções para um computador e executando-as.**

Os computadores estão envolvidos em modelagem de duas formas: como uma **ferramenta computacional para modelagem** e como um **objeto a ser modelado.**

Sistemas computacionais são sistemas muito complexos e, normalmente, grandes, consistindo em vários **componentes separados, interativos.**

Cada **componente** pode ser um sistema, mas seu **comportamento** pode ser descrito, independentemente, por outros componentes do sistema, exceto no caso de bem definidas interações com outros componentes.

Cada componente tem o seu próprio estado de ser. O **estado** de um componente é uma **abstração** da informação relevante necessária para descrever suas ações. Normalmente, o estado de um componente depende da história passada do componente. Logo, o estado de um componente pode mudar com o tempo.

- Num modelo de fila de um banco, podem haver vários caixas e vários clientes. Os caixas podem estar desocupados, esperando por um cliente, ou ocupados, atendendo um cliente. Similarmente, os clientes podem estar desocupados, esperando por um caixa livre, ou podem estar ocupados, sendo atendidos por um caixa.
- Num modelo de um hospital, o estado de um paciente pode ser crítico, sério, razoável, bom ou excelente.

Os componentes de um sistema apresentam **concorrência** ou **paralelismo**. Atividades de um componente podem ocorrer simultaneamente com outras atividades de outros componentes. Num sistema computacional, temos, por exemplo, os dispositivos periféricos, que podem operar concorrentemente, sob controle do computador.

Uma vez que os componentes dos sistemas interagem, é necessário que haja **sincronização**. A transferência de informação de um componente para outro requer que as atividades dos componentes envolvidos sejam sincronizadas, enquanto a interação estiver acontecendo. Isto pode resultar em um componente ficar esperando pelo outro.

A **temporização** das ações de diferentes componentes pode ser muito complexa e as interações resultantes entre componentes podem ser difíceis de descrever.

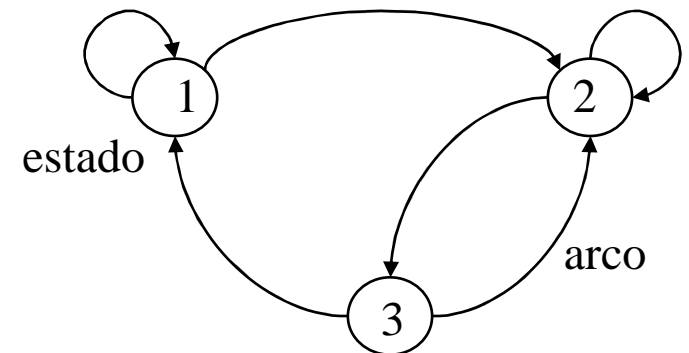
Redes de Petri

São uma ferramenta para **a modelagem e projeto de sistemas**, utilizando uma representação matemática do sistema, sendo uma extensão das **máquinas de estados finitos**.

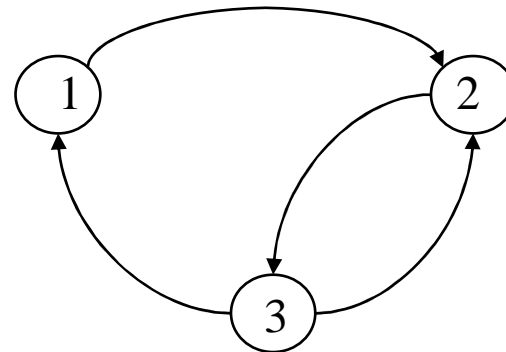
A análise da rede de Petri permite **avaliar a estrutura e o comportamento dinâmico do sistema modelado**. O resultado desta avaliação pode levar a melhorias ou mudanças no sistema.

Vamos considerar, inicialmente, o modelo de máquinas de estados finitos, representado pelo seguinte diagrama:

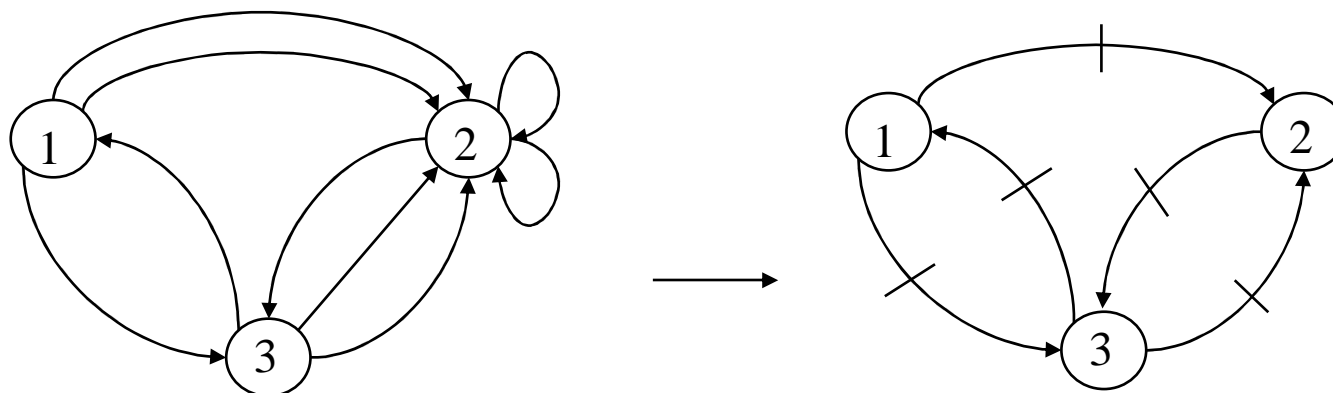
A passagem de um estado para outro, representada pelos arcos, é determinada pela ocorrência de certos **eventos**.



Um primeiro passo para modificar o modelo de máquinas de estados finitos é suprimir as transições que não forem importantes para a compreensão.



Em seguida, vamos condensar as condições que levam a uma mudança de estado numa única transição.



Uma rede de Petri **C** é composta por quatro partes:

- um conjunto de **lugares** $\mathbf{P} = \{p_1, p_2, \dots, p_n\}$;
- um conjunto de **transições** $\mathbf{T} = \{t_1, t_2, \dots, t_m\}$;
- uma função de **entrada** $\mathbf{I} : \mathbf{T} \rightarrow \mathbf{P}^\infty$;
- um conjunto de **saídas** $\mathbf{O} : \mathbf{T} \rightarrow \mathbf{P}^\infty$.

$$\mathbf{C} = (\mathbf{P}, \mathbf{T}, \mathbf{I}, \mathbf{O})$$

A função de entrada **I** mapeia uma transição t_j para uma coleção de lugares $\mathbf{I}(t_j)$, conhecida como **lugares de entrada de uma transição**.

A função de saída **O** mapeia uma transição t_j para uma coleção de saídas $\mathbf{O}(t_j)$, conhecida como **lugares de saída de uma transição**.

Considere a seguinte estrutura:

$$C = \{P, T, I, O\}$$

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$

$$T = \{t_1, t_2, t_3, t_4\}$$

$$I(t_1) = \{p_1\}$$

$$O(t_1) = \{p_2, p_3, p_5\}$$

$$I(t_2) = \{p_2, p_3, p_5\}$$

$$O(t_2) = \{p_5\}$$

$$I(t_3) = \{p_3\}$$

$$O(t_3) = \{p_4\}$$

$$I(t_4) = \{p_4\}$$

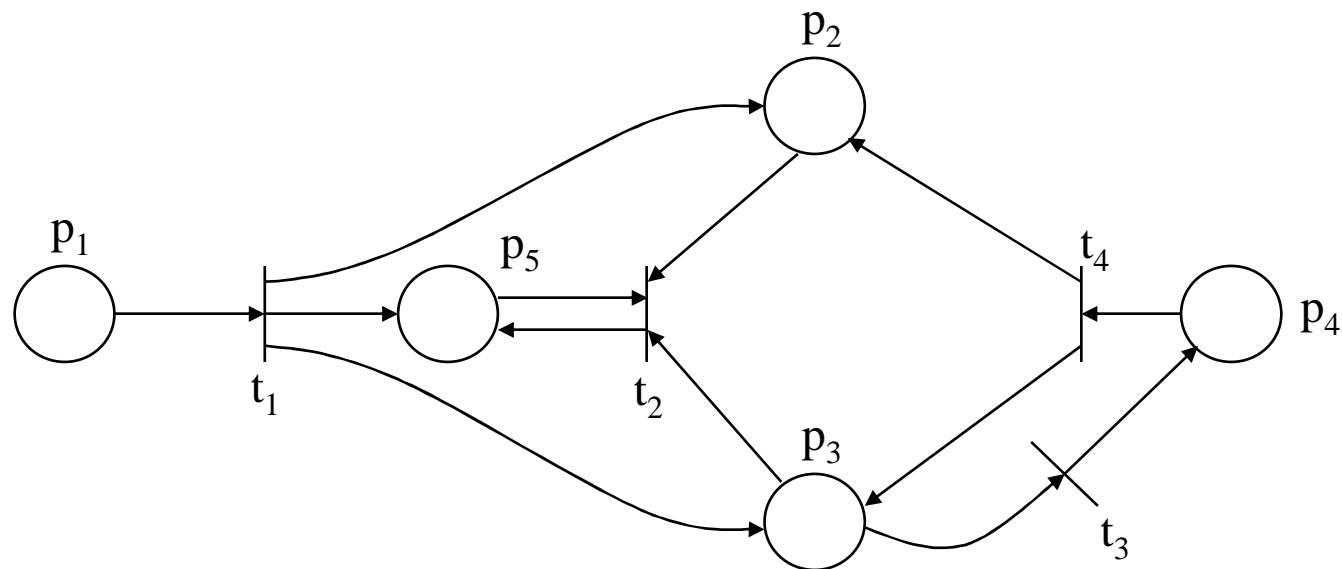
$$O(t_4) = \{p_2, p_3\}$$

O **grafo** da rede de Petri tem dois tipos de nós: um **círculo** representa um lugar e uma **barra** representa uma transição. **Arcos direcionados** conectam lugares e transições.

Um arco direcionado de um lugar p_i para uma transição t_j define o lugar como sendo uma **entrada** da transição.

Um arco direcionado de uma transição t_j para um lugar p_i define o lugar como sendo uma **saída** da transição.

O grafo da rede de Petri definida pela estrutura anterior é:



Pode-se associar a cada arco um **peso**, que corresponde à sua multiplicidade.

Uma **marcação** μ é uma atribuição de **fichas** aos lugares de uma rede de Petri. Uma **ficha** é um conceito primitivo para redes de Petri, da mesma forma que lugares e transições.

O número e posição das fichas pode mudar durante a execução de uma rede de Petri. Dessa forma, as fichas são usadas para definir a execução de uma rede de Petri.

A marcação μ pode ser definida como sendo um vetor:

$$\mu = (\mu_1, \mu_2, \dots, \mu_n) \mid n = |P| \text{ e } \mu_i \in N, i=1, \dots, n$$

O número de fichas no lugar p_i é μ_i , $i = 1, \dots, n$. Assim sendo:

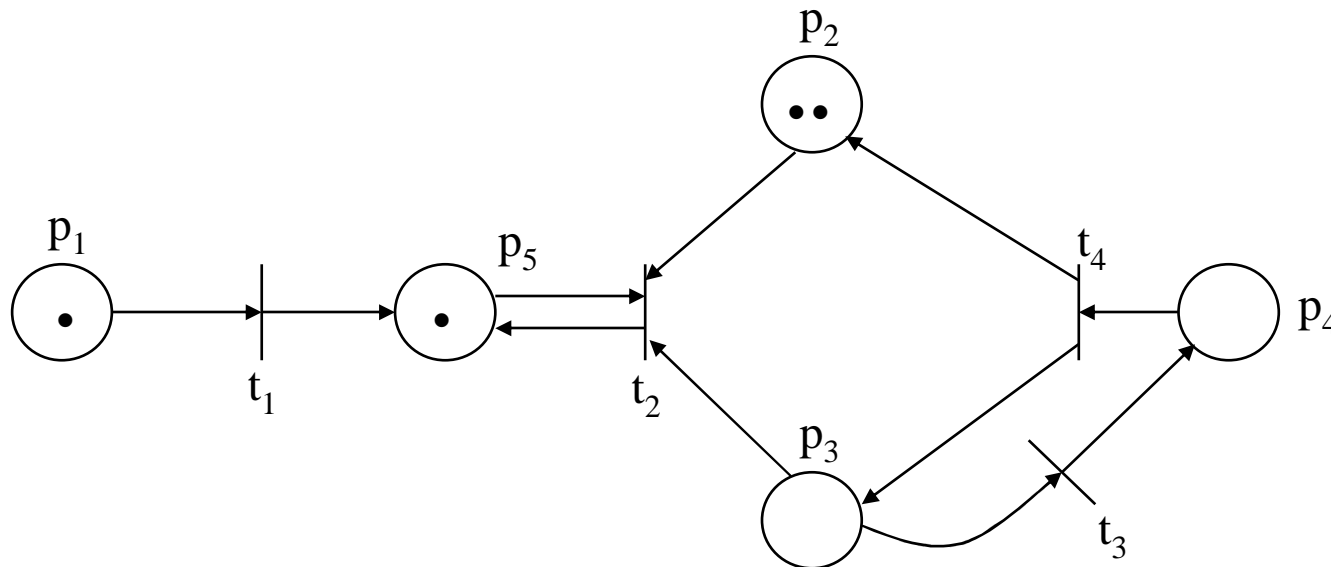
$$\mu(p_i) = \mu_i$$

Uma rede de Petri marcada pode ser definida como:

$$\mathbf{M} = (\mathbf{P}, \mathbf{T}, \mathbf{I}, \mathbf{O}, \boldsymbol{\mu})$$

Num grafo de rede de Petri, fichas são representadas por pontos dentro dos círculos. No exemplo abaixo, tem-se:

$$\boldsymbol{\mu} = (1, 2, 0, 0, 1)$$

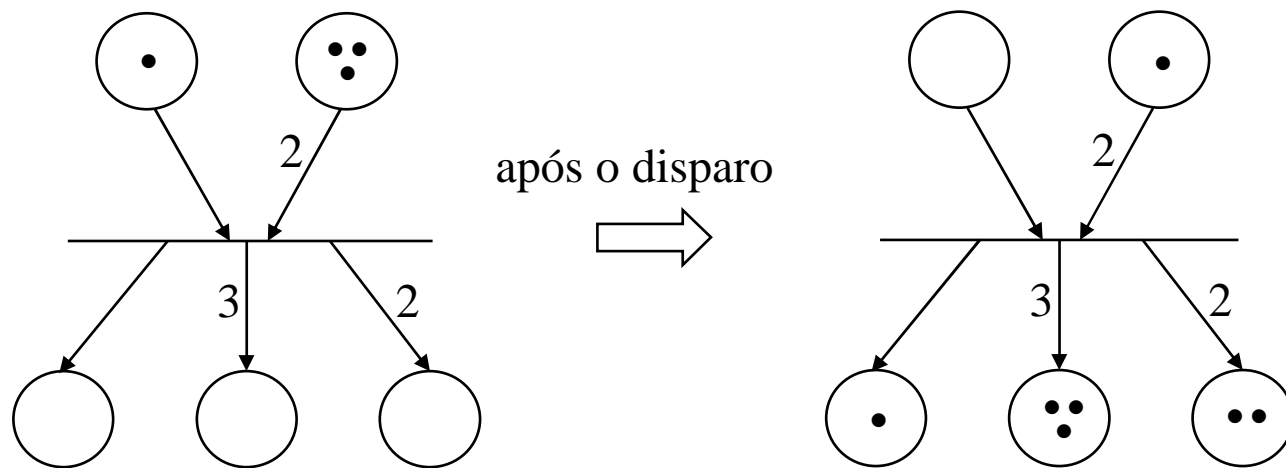


Uma rede de Petri executa através do **disparo de transições**. Uma transição dispara removendo fichas dos seus lugares de entrada e criando novas fichas, que são distribuídas nos seus lugares de saída.

Uma transição pode disparar se estiver **habilitada**. Uma transição está habilitada se cada um dos seus lugares de entrada tem, pelo menos, tantas fichas quanto arcos do lugar para a transição (**peso**).

O disparo de uma transição é formado por duas operações, instantâneas e indivisíveis:

- retira-se de cada lugar de entrada um número de fichas igual ao peso do arco;
- coloca-se em cada lugar de saída um número de fichas igual ao peso do arco.



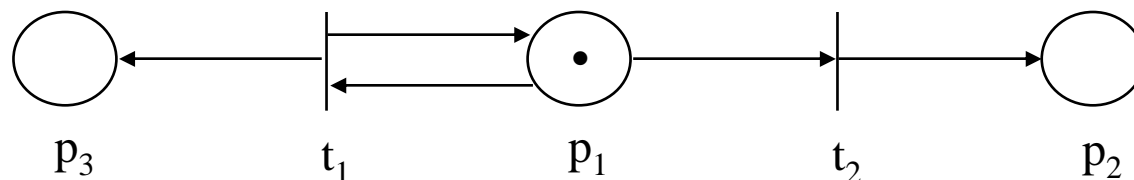
O disparo de uma transição muda a marcação μ da rede de Petri para uma nova marcação μ' .

O disparado de transições continua enquanto existir, pelo menos, uma transição habilitada. Quando não há transições habilitadas, a execução pára.

O **estado** de uma rede de Petri é definido por sua **marcação**. Dada uma rede de Petri $C = \{P, T, I, O\}$ e uma marcação inicial μ^0 , pode-se executar a rede de Petri pelo disparo sucessivo de transições.

Duas seqüências resultam da execução de uma rede de Petri: a **seqüência de marcações** $(\mu^0, \mu^1, \mu^2, \dots)$ e a **seqüência de transições** $(t_{j0}, t_{j1}, t_{j2}, \dots)$.

Para a rede de Petri abaixo e a marcação inicial $\mu^0 = (1,0,0)$, duas marcações são imediatamente **alcançáveis**: $(0,1,0)$ e $(1,0,1)$. Da primeira nenhuma marcação é alcançável. Da segunda pode-se chegar a $(0,1,1)$ e $(1,0,2)$.



Redes de Petri foram projetadas e são usadas principalmente para **modelagem**.

Muitos sistemas, especialmente aqueles com componentes independentes, podem ser modelados por uma rede de Petri. Exemplos: “hardware” de computadores, “software” de computadores, sistemas físicos, sistemas sociais, etc... .

As redes de Petri são usadas para modelar a ocorrência de diversos **eventos e atividades** em um sistema. Em particular, as redes de Petri podem **modelar o fluxo de informação ou outros recursos dentro de um sistema**.

Eventos são ações que acontecem num sistema. A ocorrência desses eventos é controlada pelo estado do sistema, descrito por um conjunto de **condições**.

Uma **condição** é um **predicado** ou **descrição lógica** do estado do sistema. Dessa forma, uma condição pode ser **verdadeira** ou **falsa**.

Uma vez que eventos são ações, eles podem ocorrer. Para que um evento ocorra, pode ser necessário que certas condições sejam verdadeiras. Estas são denominadas de **pré-condições** do evento.

A ocorrência de um evento pode tornar as pré-condições falsas e pode tornar outras condições verdadeiras, chamadas **pós-condições**.

Exemplo: Considere uma máquina que espera até que uma ordem surja e, então, prepara o pedido e o despacha.

As **condições** para o sistema são:

- a - A máquina está esperando;
- b - Uma ordem chega e está esperando ser atendida;
- c - A máquina está executando a ordem;
- d - A ordem é completada.

Os **eventos** seriam:

- 1 - Um ordem chega;
- 2 - A máquina começa a executar a ordem;
- 3 - A máquina termina de executar a ordem;
- 4 - A ordem é despachada.

O quadro abaixo resume os eventos e suas respectivas pré e pós-condições:

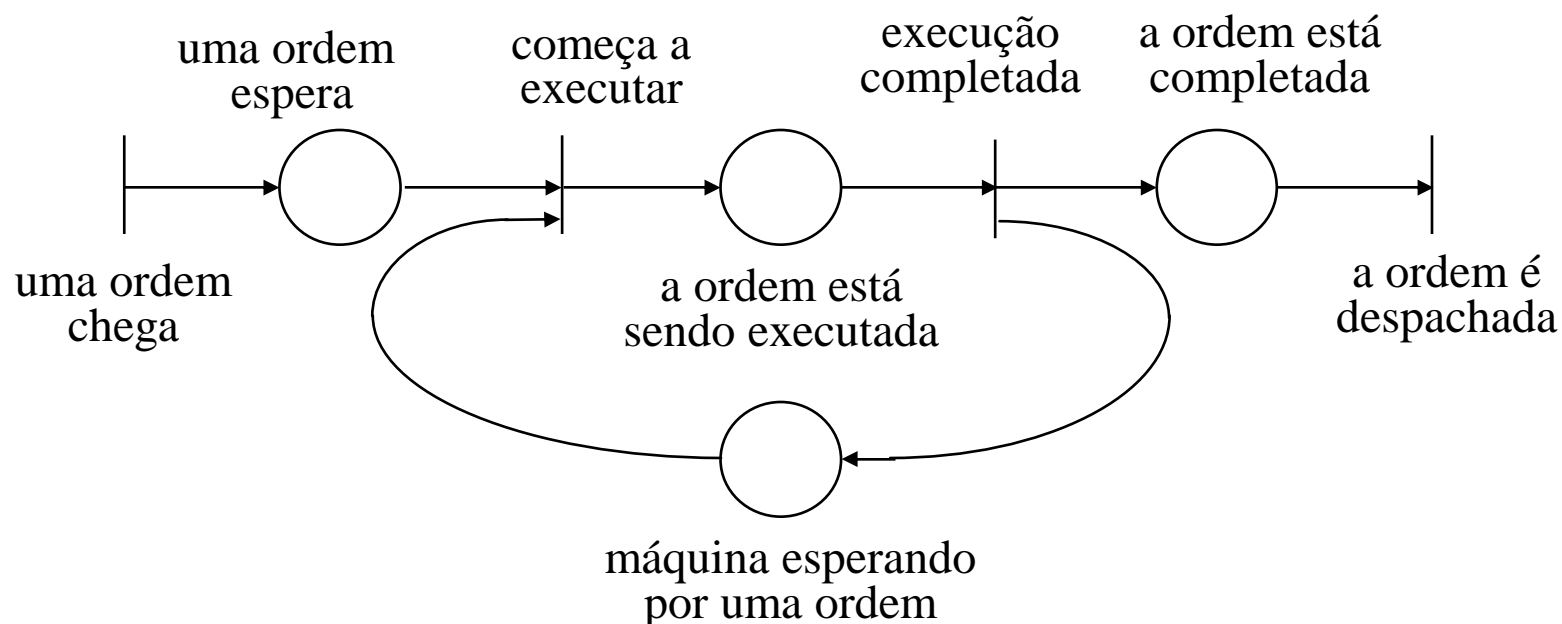
<i>evento</i>	<i>pré-condições</i>	<i>pós-condições</i>
1	nenhuma	b
2	a, b	c
3	c	d, a
4	d	nenhuma

Condições são modeladas por **lugares** numa rede de Petri, e **eventos** são modelados por **transições**.

As **entradas** de uma transição são as **pré-condições** do correspondente evento. As **saídas** são as **pós-condições**. A ocorrência de um evento corresponde ao disparo da transição correspondente.

Uma **condição verdadeira** é representada por uma **ficha** no lugar que corresponde à condição. Quando a transição dispara, ela remove as fichas que representam as pré-condições verdadeiras e cria novas fichas que representam a transformação das pós-condições correspondentes em verdadeiras.

A rede de Petri abaixo é um modelo da máquina citada anteriormente.



Exemplo: Considere um sistema computacional que processa programas a partir de um dispositivo de entrada e emite resultados em um dispositivo de saída.

As **condições** para o sistema são:

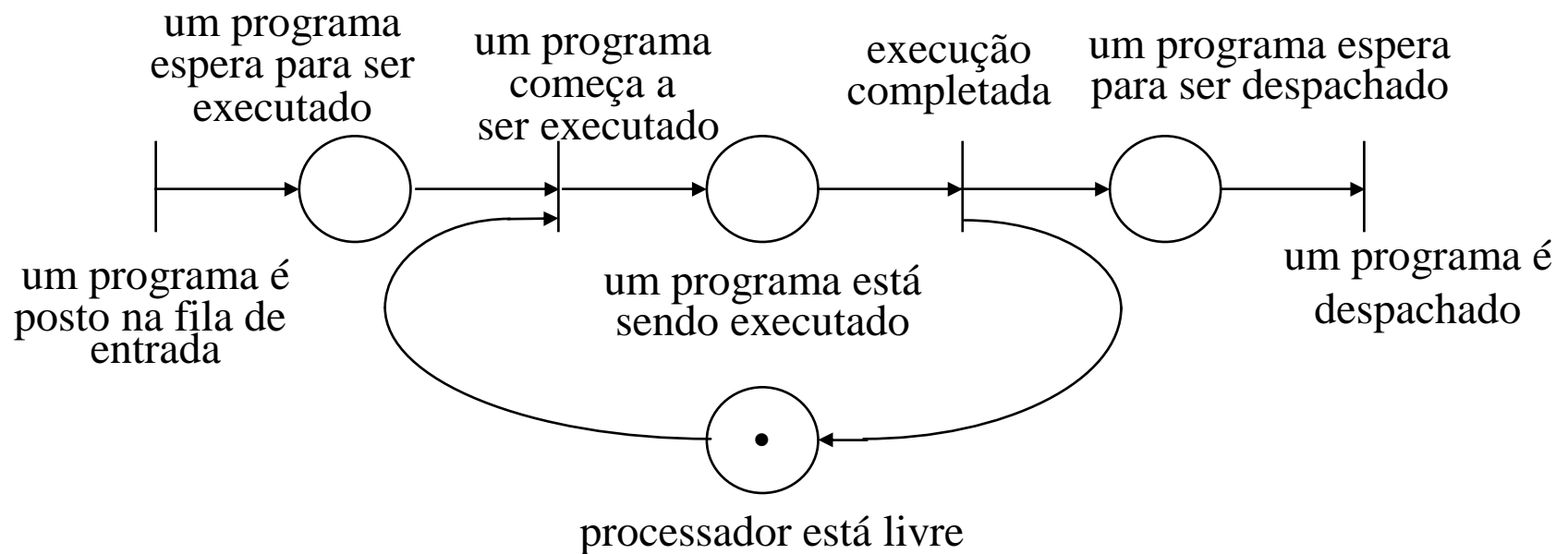
- a - Um programa esta esperando pelo processador;
- b - O processador esta livre;
- c - Um programa esta sendo processado;
- d - Um programa esta esperando para ser despachado.

Os **eventos** seriam:

- 1 - Um programa é posto no dispositivo de entrada;
- 2 - Um programa começa a ser executado;
- 3 - O processador terminou de processar um programa;
- 4 - Um programa é despachado.

O quadro abaixo resume os eventos e respectivas pré e pós-condições:

<i>evento</i>	<i>pré-condições</i>	<i>pós-condições</i>
1	nenhuma	a
2	a, b	c
3	c	d, b
4	d	nenhuma



No modelo de rede de Petri, dois eventos que são habilitados e não interagem podem ocorrer independentemente. A isto chamamos **paralelismo** ou **concorrência**.

Não há necessidade de sincronizar os eventos, a menos que seja requerido pelo sistema sendo modelado. Dessa forma, redes de Petri parecem ser ideais para modelar **sistemas de controle distribuído**, com múltiplos processos executando concorrentemente no tempo.

Não há medida de tempo ou fluxo de tempo numa rede de Petri, o que define uma característica **assíncrona**. A única propriedade importante do tempo, de um ponto de vista lógico, está em **definir uma ordem parcial da ocorrência de eventos**.

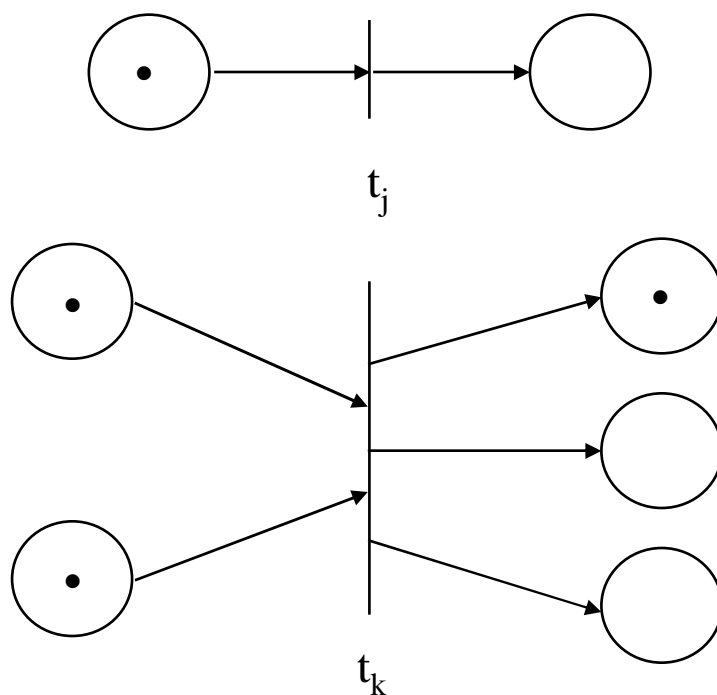
Os eventos levam um tempo variável para ocorrer e isto está refletido no modelo de rede de Petri por não depender da noção de tempo para controlar os eventos.

Uma execução da rede de Petri é vista como uma **seqüência de eventos discretos**. A ordem de ocorrência dos eventos é uma dentre várias permitidas pela estrutura, o que leva a um aparente **não-determinismo** na execução da rede de Petri.

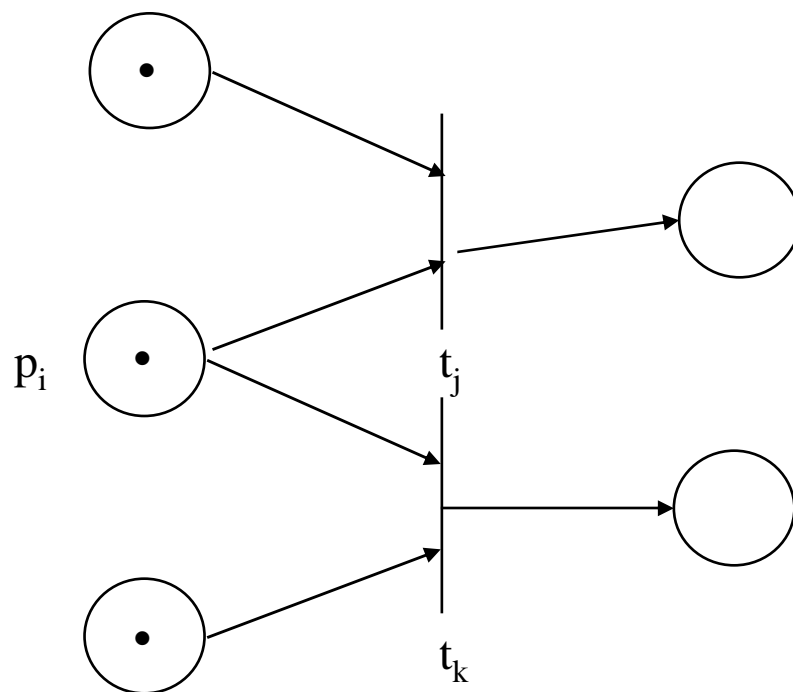
Se, em qualquer momento, mais de uma transição estiver habilitada, então qualquer uma dessas pode ser a próxima a ser disparada. A escolha de qual transição irá disparar é feita aleatoriamente.

Em situações na vida real nas quais várias coisas ocorrem simultaneamente, a ordem aparente da ocorrência dos eventos não é única: qualquer conjunto de seqüências de eventos pode ocorrer.

Concorrência ocorre quando duas transições habilitadas não afetam uma a outra, de nenhuma forma, e as possíveis seqüências de eventos incluem algumas em que uma transição ocorre primeiro, e outras em que a outra ocorre primeiro.



Conflito ocorre quando havendo transições habilitadas, somente uma pode ser disparada, uma vez que o disparo de uma remove a ficha na entrada compartilhada, desabilitando a outra transição.



Modelagem de Hardware

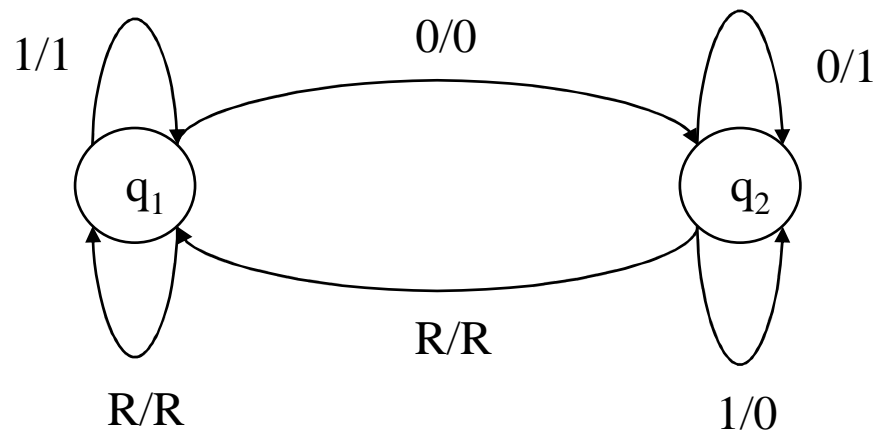
O hardware pode ser dividido em vários níveis e as redes de Petri podem modelar cada um desses níveis:

- em um nível, computadores são construídos de **memórias e portas**;
- em um nível mais alto, os principais componentes são **unidades funcionais e registradores**;
- em um nível ainda mais alto, os componentes de uma **rede** podem ser **sistemas computacionais** inteiros.

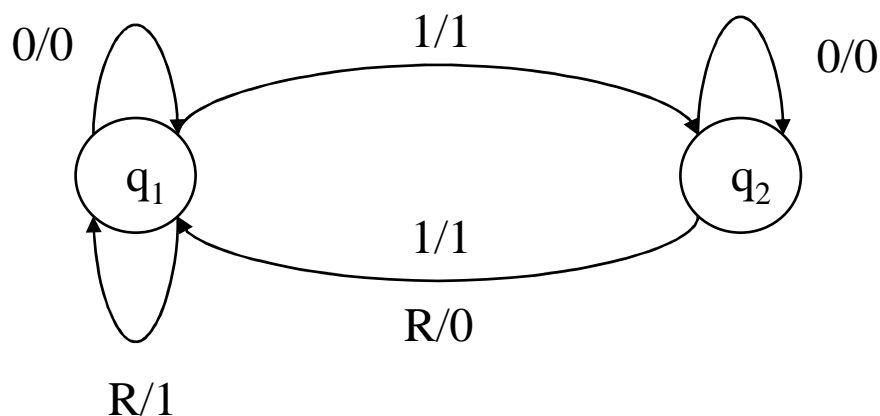
Máquinas de estados finitos são comumente representadas por **diagrama de estados**, em que os estados são representados por **círculos** (nós do grafo) e as transições por **arcos**.

Um arco do estado q_i para o estado q_j , rotulado a/b , significa que no estado q_i , com entrada a , a máquina vai mudar para o estado q_j , com saída b .

Exemplo: Conversor serial de um número binário em seu complemento a 2, em que o LSB é introduzido primeiro. O alfabeto de entrada e o de saída consistem de três símbolos: 0, 1 e R. A máquina começa no estado q_1 . O símbolo de *reset* (R) indica o fim (ou começo) de um número e inicializa a máquina para o seu estado inicial. A saída da máquina para o símbolo de *reset* é simplesmente um eco do símbolo R.



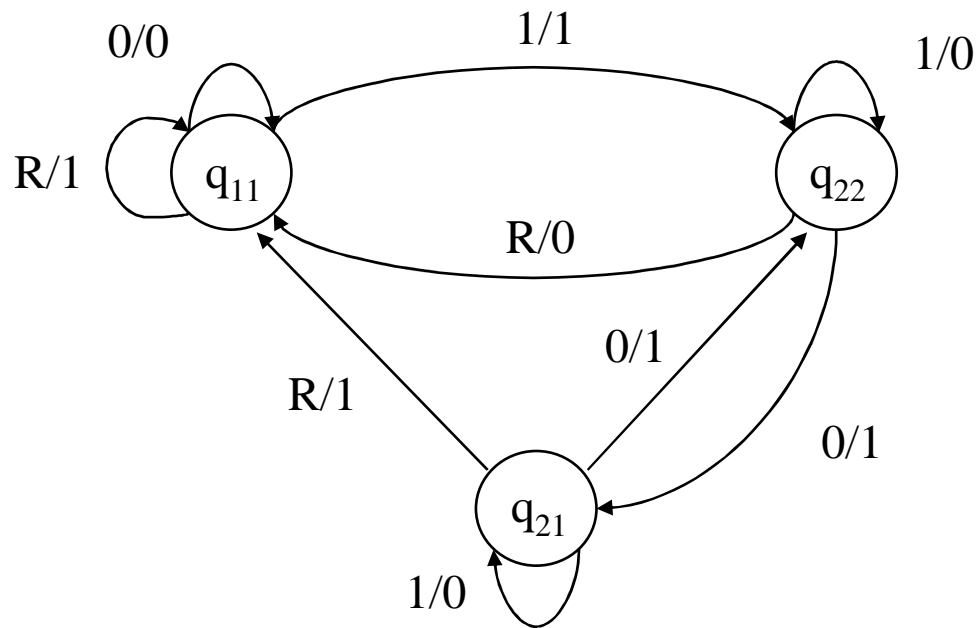
Exemplo: Uma máquina de estados que computa serialmente a paridade de um número binário, utilizando as mesma entradas que o exemplo anterior. A saída simplesmente copia a entrada até que o símbolo de entrada é um *reset*. A máquina começa no estado q_1 . A saída copia a entrada até que o símbolo de entrada seja um *reset* (\mathbf{R}). A saída para um *reset* é 0 para um número com paridade ímpar e 1 para um número com paridade par.



Vamos representar **cada estado da máquina de estados por um lugar na rede de Petri**. O estado atual é, então, marcado por uma ficha e todos os demais lugares estão vazios. A interação com o meio externo é feita através das **entradas e saídas**, que também serão representadas por **lugares**.

Para cada par de **estado e entrada**, definimos uma **transição** cujos **lugares de entrada** são os lugares correspondentes ao estado e à entrada, e cujos **lugares de saída** são os lugares correspondentes ao **próximo estado e saída**.

Pode-se construir uma **máquina composta** que compute o complemento a 2 de um número e sua paridade. Numa máquina de estados, esta operação requer um **estado composto** com componentes de ambas as sub-máquinas.



Para uma máquina em rede de Petri, a composição é feita **sobrepondo-se os lugares de saída da primeira rede com os lugares de entrada da segunda.**

Se duplicarmos as fichas de entrada, alimentando ambas as sub-máquinas da rede de Petri, teremos uma composição paralela, permitindo que as sub-máquinas executem simultaneamente.

Pipeline

Sistemas computacionais são construídos de muitos componentes e muitos projetos tentam aumentar a capacidade computacional através da execução paralela de funções. Isto torna as redes de Petri particularmente apropriadas para representar estes sistemas.

Um **pipeline** é composto por um número de estágios, que podem estar em execução **simultaneamente**. Quando o estágio k termina, ele passa o resultado para o estágio $k+1$ e olha para o estágio $k-1$ para nova tarefa.

Se cada estágio leva t unidades de tempo e há n estágios, então a operação completa para um operando levará nt unidades de tempo. No entanto, mantendo-se o **pipeline** suprido com novos operandos, ele poderá fornecer resultados à razão de um a cada t unidades de tempo.

Exemplo: Considere a soma de dois números em ponto flutuante. Os passos seriam:

- extrair os expoentes dos dois números;
- comparar os expoentes;
- deslocar a menor fração para igualar os expoentes;
- somar as frações;
- normalizar;
- considerar *overflow* ou *underflow* do expoente e montar o expoente e a fração do resultado.

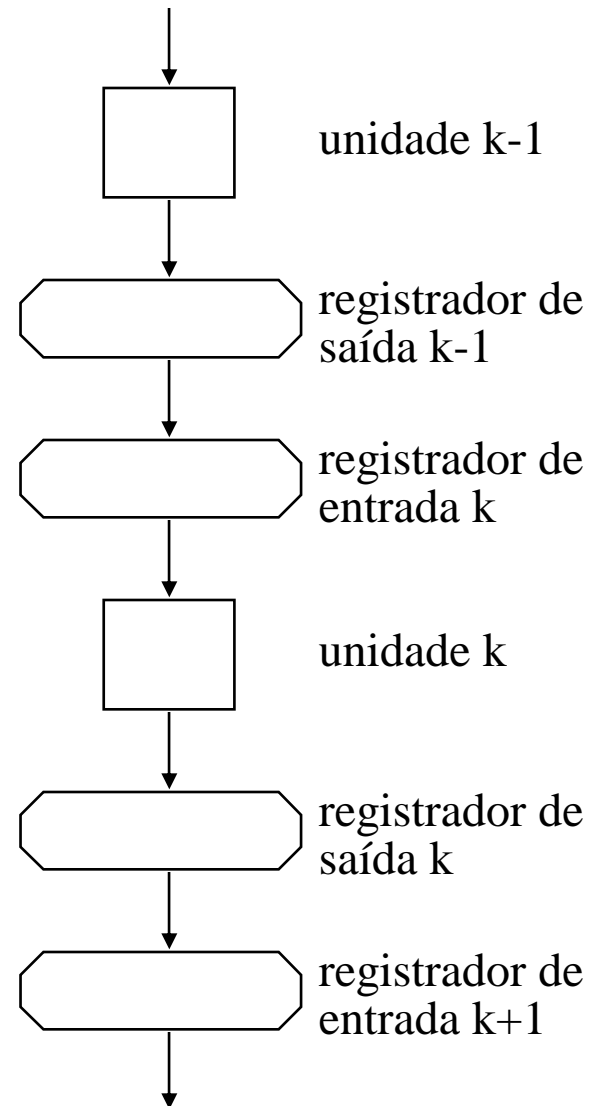
Cada um desses passos pode ser implementado por uma unidade funcional, com um operando próprio sendo passado de unidade para unidade até a operação estar completa.

Podemos considerar, inicialmente, o controle **síncrono** do **pipeline**: o tempo permitido para cada passo é um tempo constante **t**. Isto pode segurar desnecessariamente o processamento, uma vez que o tempo necessário pode variar de unidade para unidade, inclusive dentro da própria unidade, dependendo do operando.

Um **pipeline assíncrono** pode acelerar o processo, em média, sinalizando quando cada estágio está completo e pronto para passar o resultado adiante e receber um novo operando. Deve haver um lugar para armazenar as entradas e saídas, enquanto estiverem sendo usadas ou geradas, o que implica no uso de registradores.

O controle para o estágio k precisa verificar as seguintes condições:

- registrador de entrada cheio;
- registrador de entrada vazio;
- registrador de saída cheio;
- registrador de saída vazio;
- unidade ocupada;
- unidade livre;
- cópia sendo realizada.



Múltiplas unidades funcionais

O CDC6600 tem dez unidades funcionais:

- uma unidade para desvios condicionais;
- uma unidade para operações lógicas;
- uma unidade para deslocamentos;
- uma unidade para soma em ponto flutuante;
- uma unidade para soma em ponto fixo;
- duas unidades para multiplicação;
- uma unidade para divisão;
- duas unidades para incremento.

Múltiplos registros são fornecidos para guardar as entradas e saídas das unidades funcionais.

Uma possível sequência de instruções no CDC6600 seria:

- multiplicar **x1** por **x1**, dando **x0**;
- multiplicar **x3** por **x1**, dando **x3**;
- somar **x2** a **x4**, dando **x4**;
- somar **x0** a **x3**, dando **x3**;
- dividir **x0** por **x4**, dando **x6**.

Esse tipo de paralelismo deve ser controlado de forma que o resultado obtido com e sem paralelismo seja o mesmo. Dessa forma, para duas operações **a** e **b**, tal que **a** precede **b** na precedência linear do programa:

- **b** pode começar a ser executada antes de **a** se e somente se **b** não precisa dos resultados de **a** como entradas;
- os resultados de **b** não mudam nem as entradas nem os resultados de **a**.

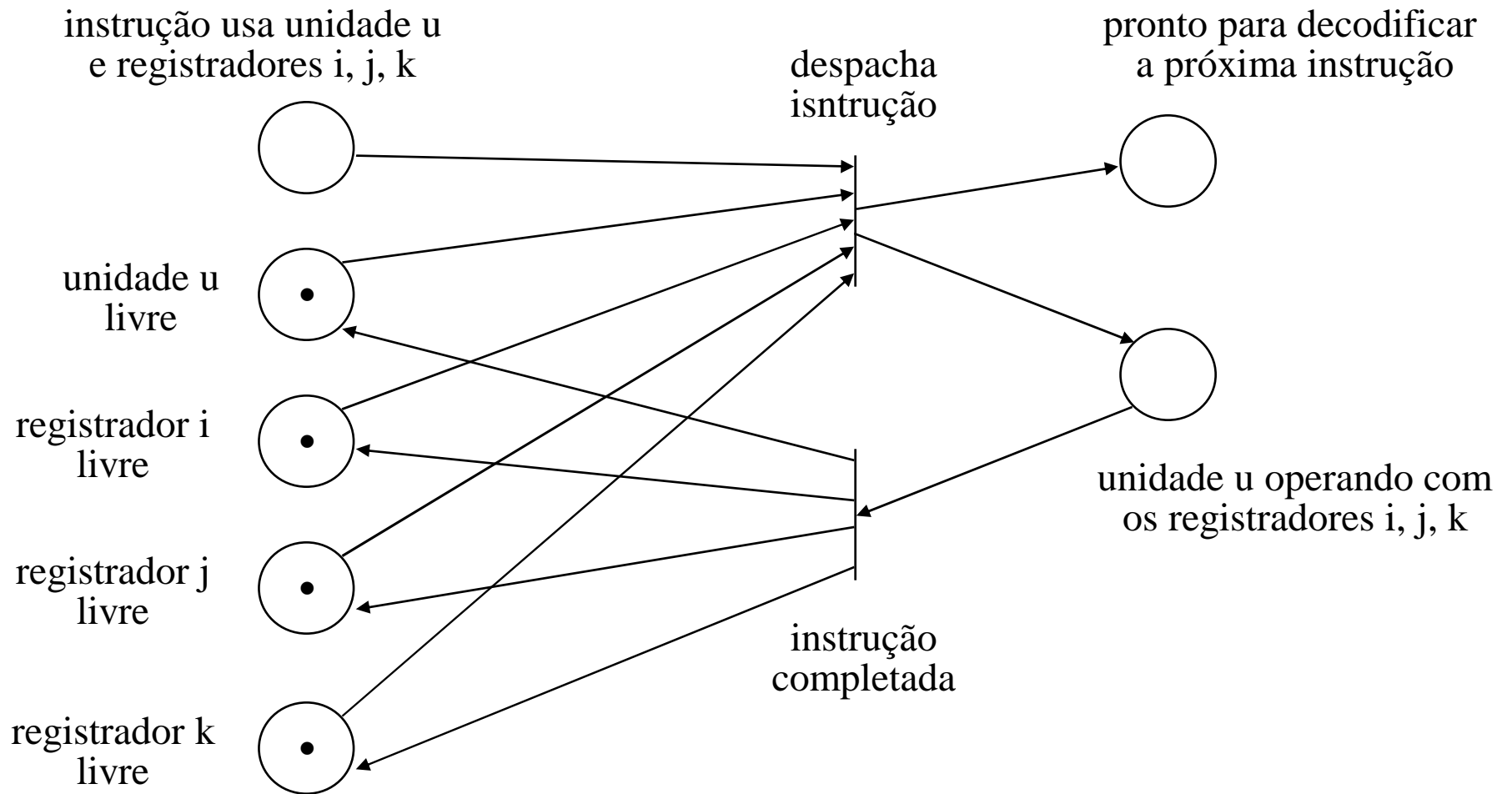
Uma **tabela de reservas** é um método de aplicar estas restrições na construção da unidade de controle que vai despachar instruções para as unidades funcionais:

- uma instrução para a unidade funcional **u**, usando registradores **i**, **j** e **k** pode ser despachada somente se todos os quatro componentes não estão reservados; quando a instrução é despachada, todos os quatro componentes se tornam reservados.

O modelo em rede de Petri do esquema acima consiste em alocar **um lugar para cada unidade funcional e cada registrador**: se a unidade funcional ou o registrador está **livre**, uma **ficha** é posta no respectivo lugar.

Múltiplas unidades funcionais idênticas podem ser indicadas por **múltiplas fichas nos lugares**.

Parte de uma rede de Petri usada para modelar a execução de uma instrução usando a unidade **u** e os registradores **i**, **j** e **k**:



Software

A modelagem de software tem se concentrado na análise, especificação e descrição de programas sequenciais. Pouco tem sido feito no que diz respeito a sistemas compostos por processos concorrentes. É dentro deste contexto que as redes de Petri se mostram adequadas para a modelagem do software.

A representação de um sistema composto por processos concorrentes parte da combinação de redes de Petri que representem processos simples.

Um processo simples é descrito por um programa, que representa dois aspectos separados do processo: **computação** e **controle**.

Computação trata das operações lógicas e aritméticas, entrada e saída, e manipulação de memória.

Controle trata da ordem com que as operações são realizadas.

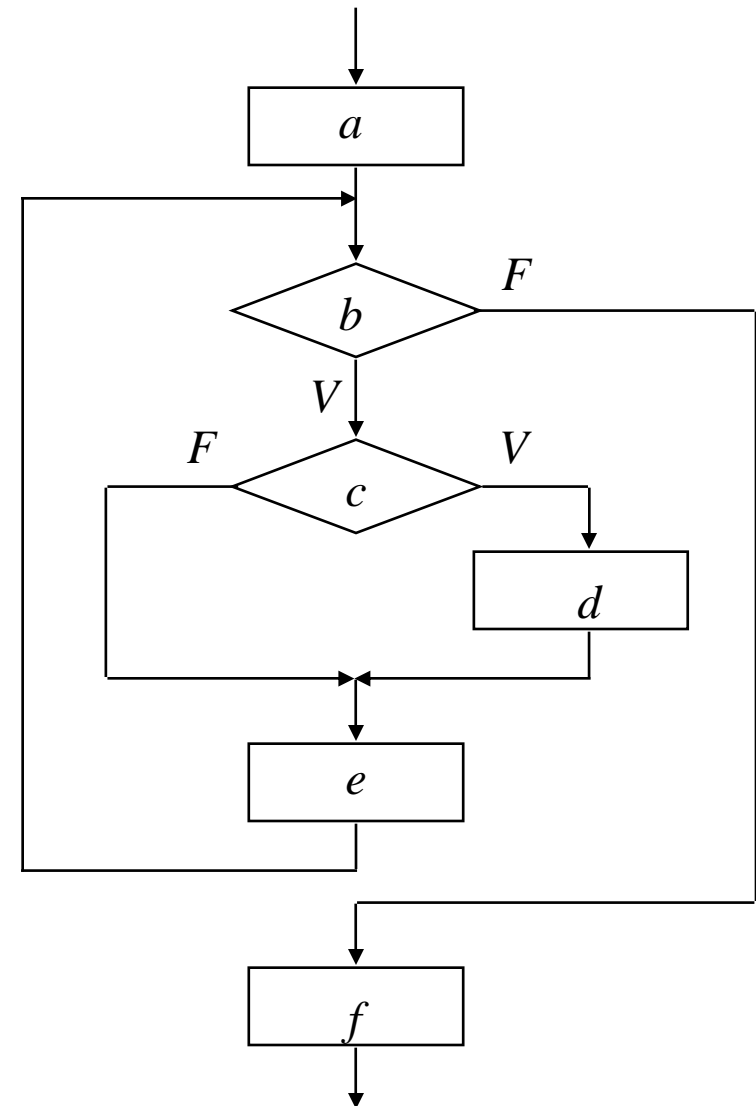
As redes de Petri são adequadas para modelar o sequenciamento das instruções e o fluxo de informação e computação, mas não os valores atuais da informação, propriamente ditos.

Um modelo de um sistema, por sua própria natureza, é uma **abstração** do sistema modelado, ignorando detalhes específicos tanto quanto possível.

Fluxogramas representam o fluxo de controle de um programa. Todo programa sequencial pode ser representado por um fluxograma.

Considere o programa abaixo e o fluxograma correspondente.

```
início  
ler(y1);  
ler(y2);  
y3:=1;  
enquanto y1>0 faça  
  início  
    se ímpar (y1)  
      então início  
        y3:=y3*y2;  
        y1:=y1-1;  
      fim;  
    y2:=y2*y2;  
    y1:=y1-2;  
  fim;  
escreve(y3);  
fim;
```



Uma interpretação das ações do fluxograma consiste em:

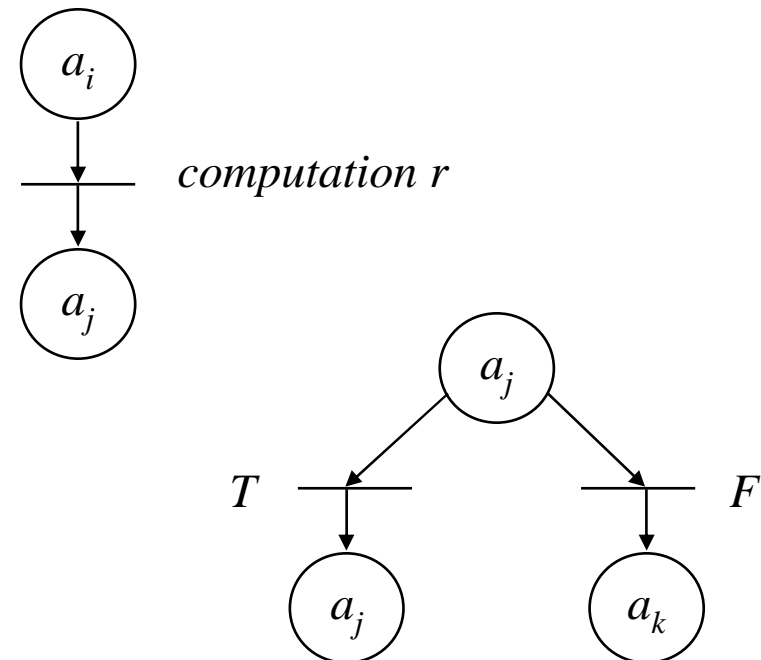
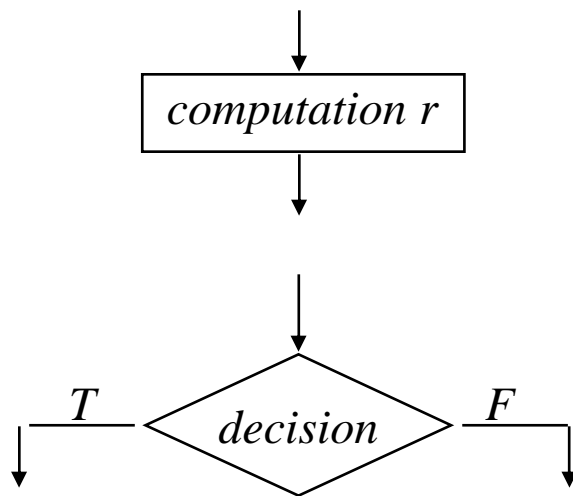
<i>ações</i>	<i>interpretação</i>
<i>a</i>	$input(y1); input(y2); y3 := 1;$
<i>b</i>	$y1 > 0?$
<i>c</i>	$odd(y1)?$
<i>d</i>	$y3 := y3 * y2; y1 := y1 - 1;$
<i>e</i>	$y2 := y2 * y2; y1 := y1 / 2;$
<i>f</i>	$output(y3);$

Um fluxograma é composto de **nós** e de **arcos** entre eles. Os nós são de dois tipos: **decisões**, representadas pelos losangos, e **computações**, representadas por retângulos.

No modelo de **rede de Petri**, as **transições** modelam **ações**, enquanto que no modelo de **fluxograma** os **nós** modelam **ações**.

Assim sendo, a forma apropriada de traduzir um fluxograma em uma rede de Petri consiste em:

- substituir os nós do primeiro por transições no segundo (observando que há dois tipos de nós);
- substituir cada um dos arcos do primeiro por exatamente um lugar no segundo.



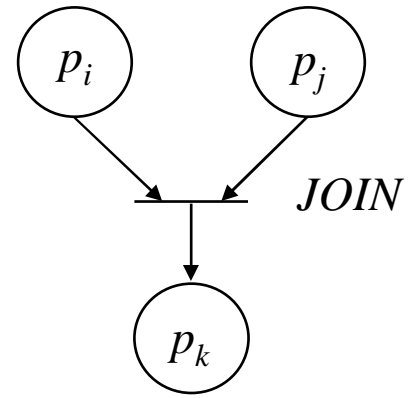
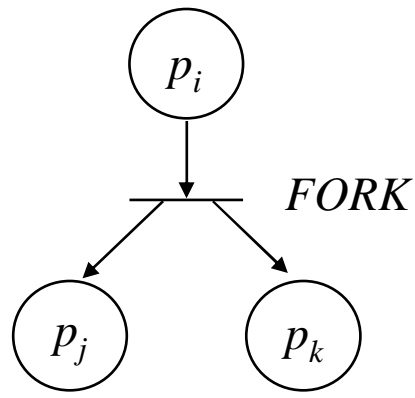
(aqui entra o slide 45)

Considere o caso de dois processos concorrentes. Numa primeira proposta, cada processo pode ser representado por uma rede de Petri.

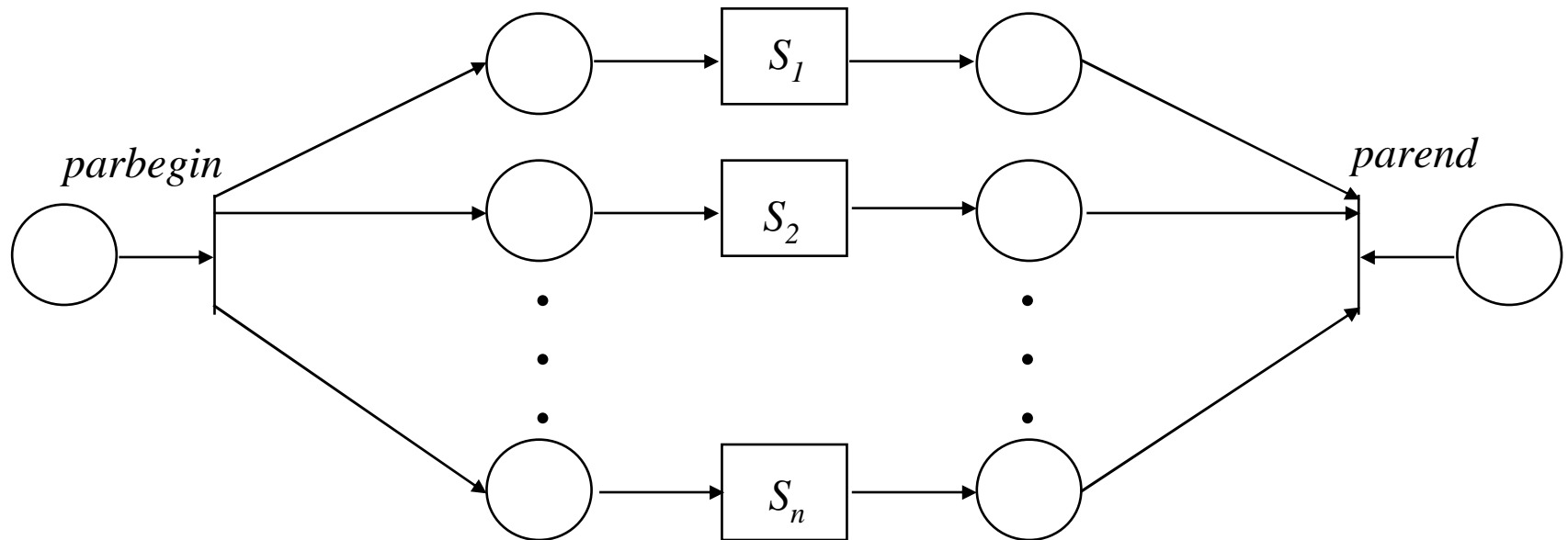
A rede de Petri composta, que é simplesmente a união das redes de Petri para cada um dos processos, pode representar a execução concorrente dos dois processos.

A marcação inicial da rede de Petri composta tem duas fichas, cada uma no lugar representando o contador de programa inicial de um processo.

Paralelismo pode ser introduzido em um processo através das operações **FORK** e **JOIN**.



Outra proposta para introduzir paralelismo é através das estruturas de controle **parbegin** e **parend**.



Paralelismo é introduzido na solução de um problema somente se os processos componentes podem cooperar na solução do problema. Tal cooperação requer o compartilhamento de informação e recursos entre os processos.

Uma variedade de problemas tem sido proposta para ilustrar os tipos de problemas que podem surgir entre processos cooperantes, dentre eles o problema do **produtor/consumidor**, o problema dos **filósofos** e o problema dos **leitores/escritores**.

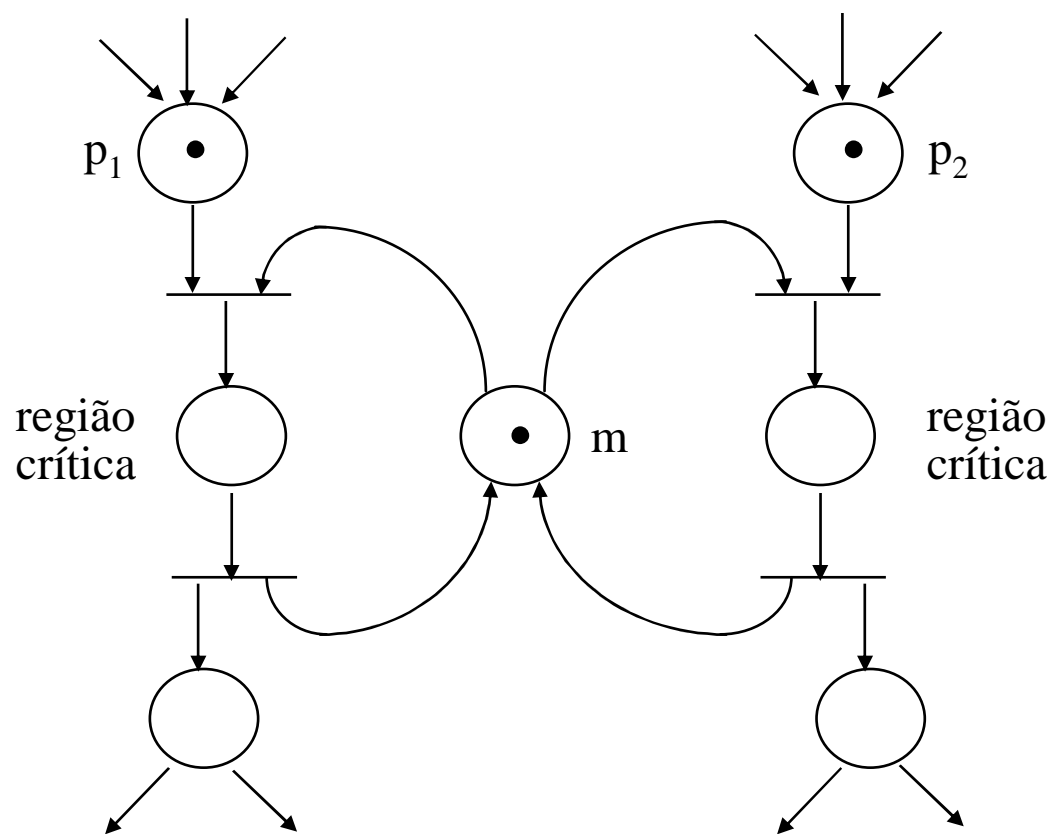
Embora as redes de Petri sejam um esquema de modelagem, e não um mecanismo de sincronização, elas devem ser capazes de modelar mecanismos de sincronização que resolvam esses problemas.

Assuma que vários processos compartilham uma variável, registro, arquivo ou outro tipo de dado. Esse dado compartilhado pode ser usado de diferentes formas pelos processos, mas basicamente se resumem a leitura ou escrita. Um problema surge quando dois processos tentam atualizar o dado compartilhado.

Exclusão mútua consiste em definir um código de entrada e de saída, de forma que somente um processo de cada vez tenha acesso ao dado compartilhado.

O código que tem acesso ao dado compartilhado e necessita proteção contra interferência de outros processos é chamado **região crítica**.

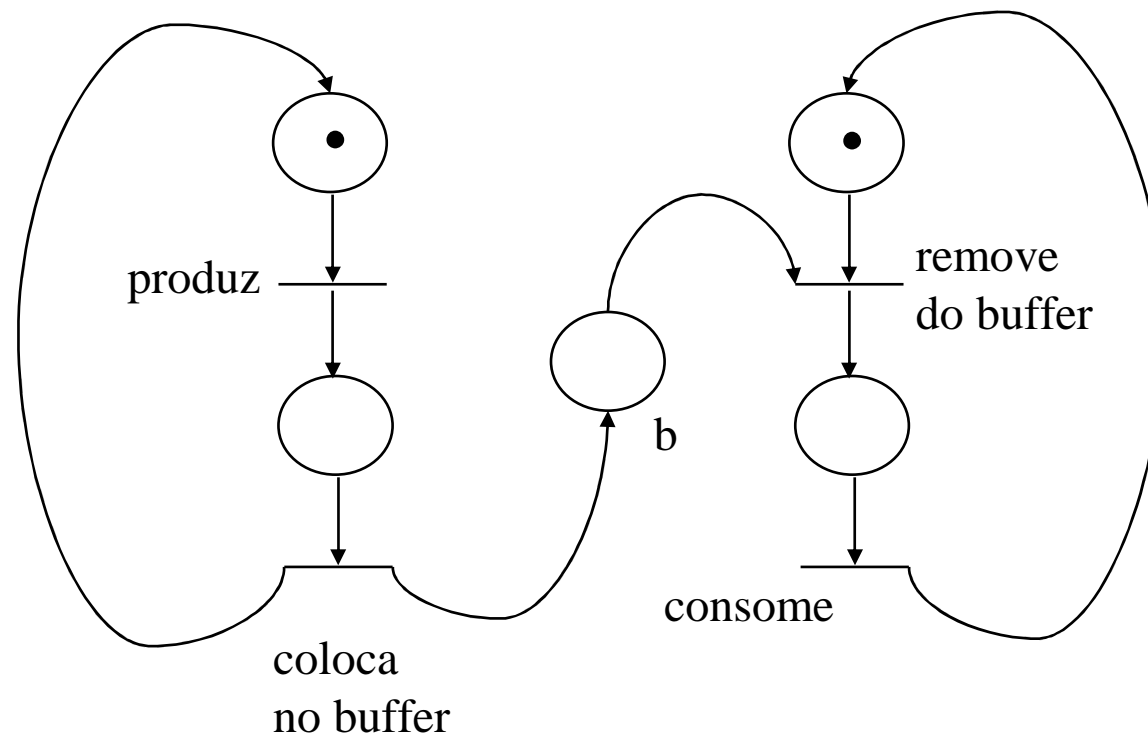
Na rede de Petri equivalente, o lugar m representa a permissão para entrar na região crítica. Para que um processo entre na região crítica, ele deve ter uma ficha em p_1 ou p_2 , sinalizando que ele deseja entrar na região crítica, e deve haver uma ficha em m , sinalizando permissão para entrar.



Se ambos os processos desejam entrar simultaneamente, então as transições t_1 e t_2 estarão em conflito e somente uma delas pode ser disparada. Disparando t_1 irá desabilitar a transição t_2 , fazendo com que o processo p_2 espere até que o primeiro processo saia de sua região crítica e ponha uma ficha de volta no lugar m .

O problema do **produtor/consumidor** também envolve dados compartilhados, que neste caso consiste de um meio de armazenamento temporário (**buffer**). O produtor cria objetos que são postos no *buffer*; o consumidor espera até que um objeto tenha sido posto no *buffer*, o remove e o consome.

Um lugar b vai representar o *buffer*; cada ficha representa um item que foi produzido, mas ainda não foi consumido.



Uma variante desse problema é o dos múltiplos produtores e múltiplos consumidores. Os itens produzidos são colocados num buffer comum. O lugar inicial do produtor vai conter tantas fichas quantos produtores. O lugar inicial do consumidor vai conter tantas fichas quantos consumidores.

Outra variante do problema do consumidor/produtor consiste do **buffer de tamanho limitado**. Neste caso, o buffer entre produtor/consumidor possui somente n posições.

Assim sendo, o produtor nem sempre poderá produzir tão rapidamente quanto deseja, podendo ter que esperar caso o consumidor seja lento e o buffer esteja cheio.

A solução consiste em representar o buffer limitado por dois lugares: b , representando o número de itens produzidos, mas ainda não consumidos; e b' , representando o número de posições vazias no buffer.

Dessa forma, se b' contiver zero fichas, o buffer está cheio e o produtor não poderá produzir nenhum item a mais.

Análise de uma Rede de Petri

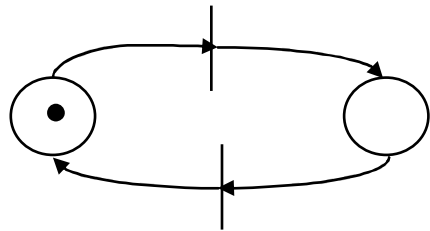
São definidas as seguintes propriedades básicas:

- Rede de Petri **limitada e segura**;
- Rede de Petri **própria (reinicializável)**;
- Rede de Petri **viva**.

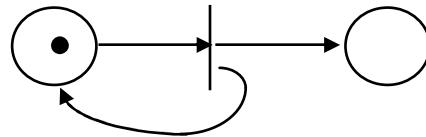
A análise destas três propriedades é feita através da construção da árvore das marcações alcançáveis.

O número de marcações alcançáveis pode ser infinito. Logo, usa-se um algoritmo de construção de uma árvore reduzida, que sempre é finita. Entretanto, pode haver o esgotamento de recursos computacionais.

Uma rede de Petri é **limitada** se o número de fichas em todos os seus lugares é finito, para todas as marcações alcançáveis. Esta rede é **segura** caso este limite seja 1, ou seja o número de fichas em todos os lugares nunca excede 1.



limitada e segura

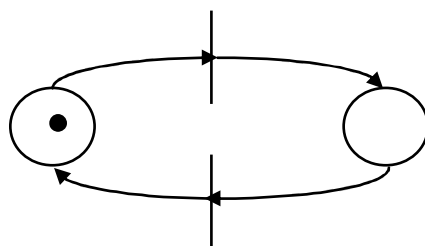


ilimitada

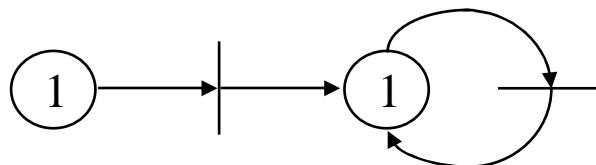
Na modelagem de hardware, se um lugar é seguro, o número de fichas nele será 0 ou 1. Logo, ele poderá ser implementado por um flip-flop.

Uma rede de Petri é **própria** e **reinicializável** se, a partir de qualquer marcação alcançável da marcação inicial, existe uma sequência de transições sucessivamente sensibilizadas que leva de volta à marcação inicial.

Exemplo:



própria

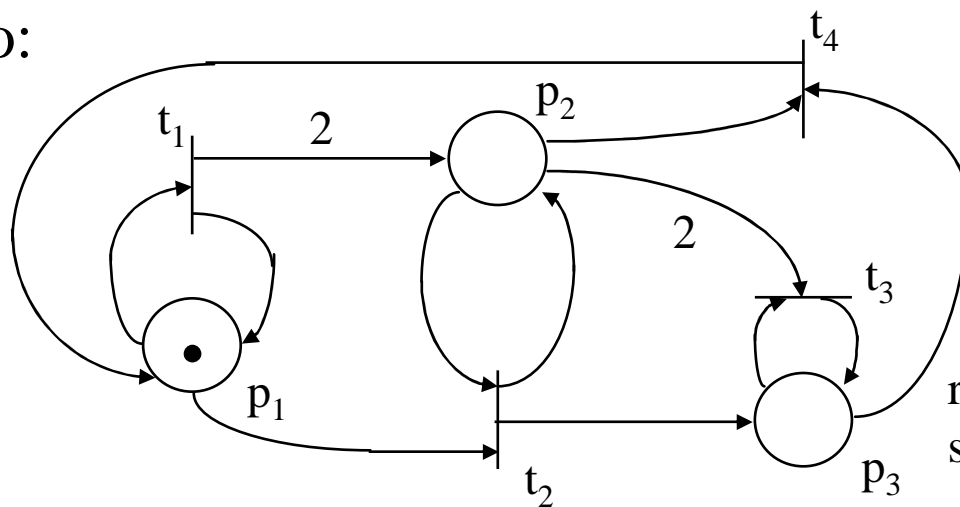


não própria

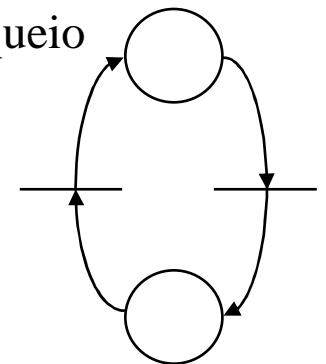
Uma rede de Petri é **viva** se e somente se, para qualquer marcação alcançável a partir da inicial e para qualquer transição, existe uma sequência de transições sensibilizadas a partir desta marcação que inclui aquela transição. O exemplo anterior também mostra uma rede **viva e não viva**.

Uma rede de Petri possui **bloqueio fatal (deadlock)** quando uma dada sequência de disparos, a partir da marcação inicial, conduz a uma situação em que não há qualquer transição sensibilizada.

Exemplo:



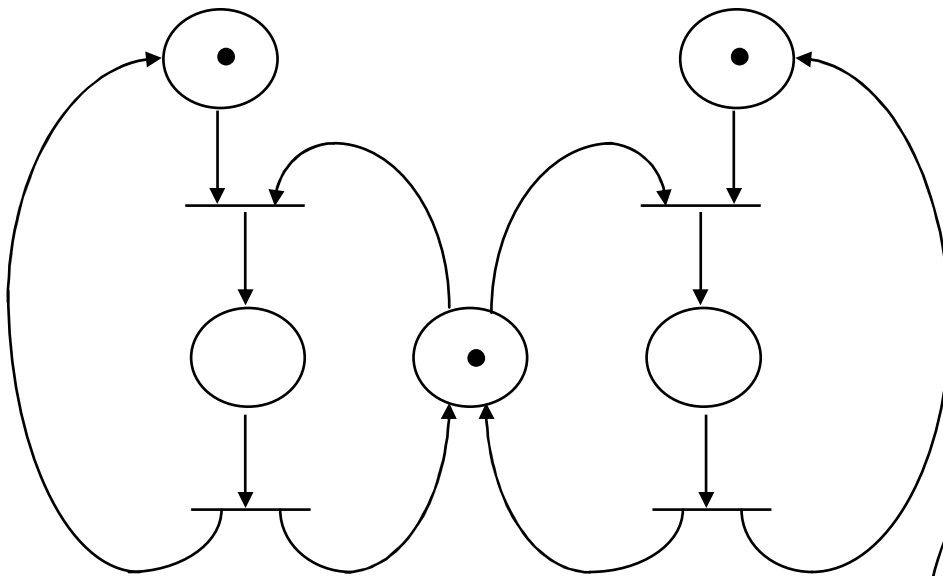
rede sem bloqueio



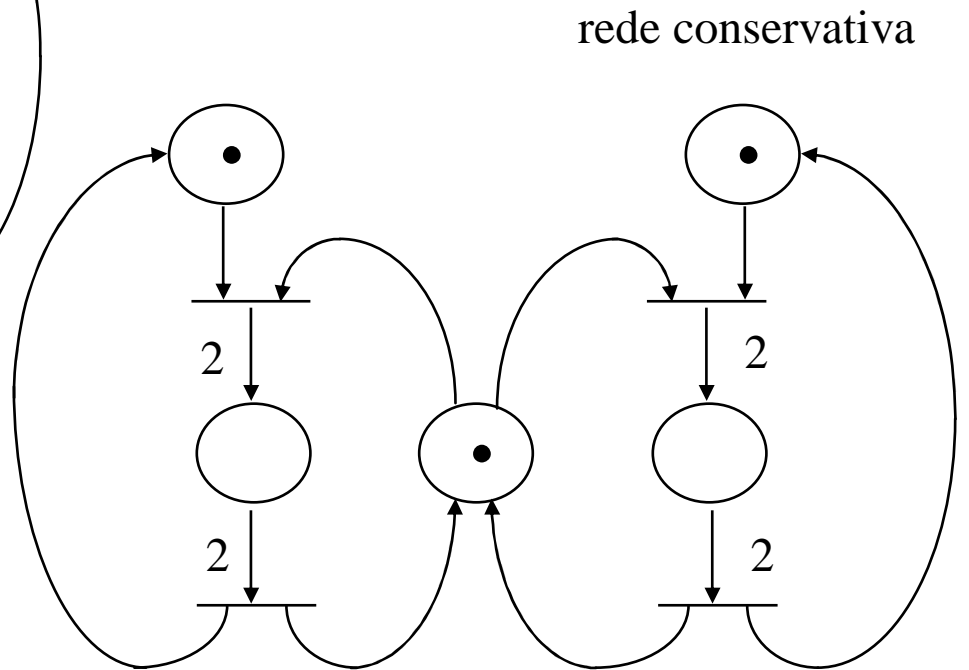
rede com bloqueio
sequência t_1, t_2, t_3

Uma rede de Petri é **conservativa** quando ela não cria e nem perde fichas ao longo de suas marcações acessíveis.

Exemplo:

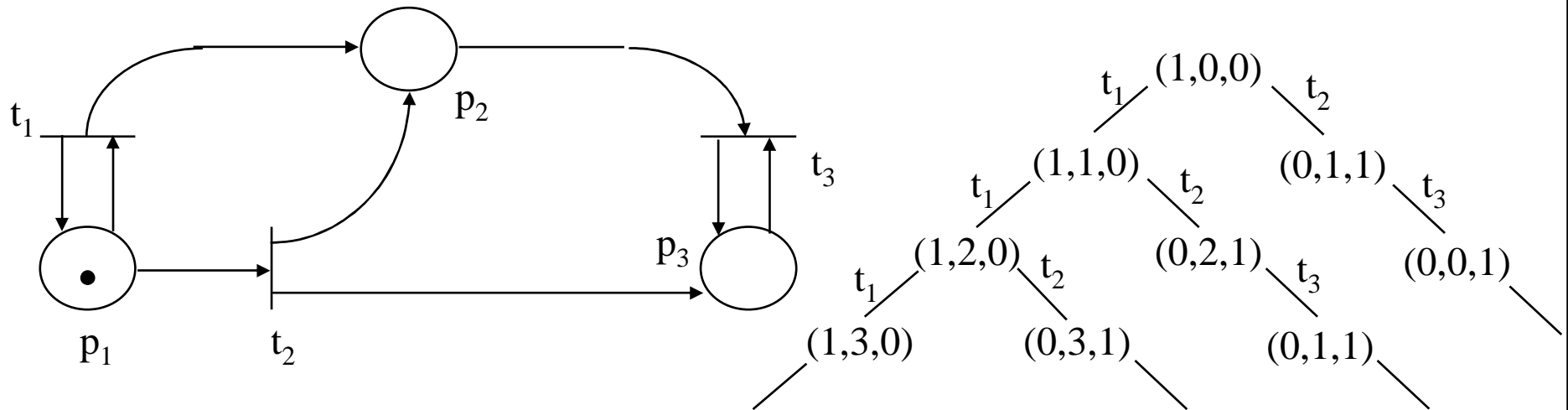


rede não conservativa



rede conservativa

Árvore de Alcançabilidade



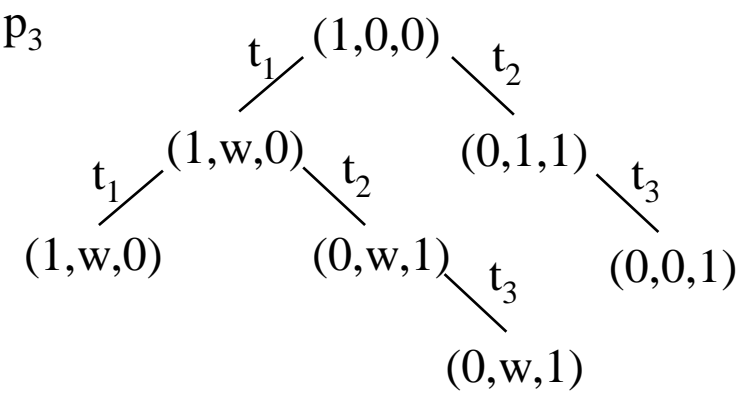
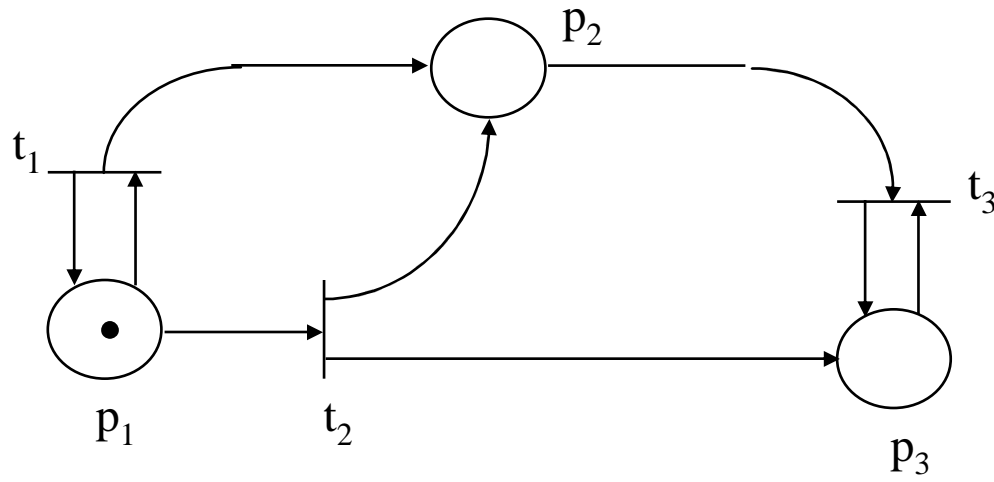
Os nós da árvore de alcançabilidade são classificados como: **terminal**, **fronteira**, **duplicado**, **interior**.

Nó **fronteira** é todo aquele que ainda não foi processado, vindo a ser convertido em terminal, duplicado ou interior. Nó **duplicado** é todo aquele que corresponde a marcação já ocorrida. Nó **terminal** é todo aquele que não habilita transições.

Algoritmo para gerar a árvore de alcançabilidade reduzida:

- Se $\exists y \mid \text{classe}(y) \neq \text{fronteira} \wedge \mu[x] = \mu[y]$,
então **classe(x) = duplicado**;
- Se nenhuma transição for habilitada por $\mu[x]$,
então **classe(x) = terminal**;
- $\forall t_j \in T$ habilitada por $\mu[x]$ que cria um novo nó z , temos:
se $\mu[x]_i = w \quad \Rightarrow \quad \mu[z]_i = w$;
se $\exists y$ no caminho da raiz a x com $\mu[y] < \mu[z]$, habilitando a
mesma transição t_j e ainda $\mu[y]_i < \mu[z]_i \Rightarrow \mu[z]_i = w$;
caso contrário, $\mu[z]_i = \{ \mu[x]_i \text{ com disparo de } t_j \}$.
- O algoritmo pára após todos os nós terem sido classificados
como **terminal, duplicado** ou **interior**.

Exemplo da aplicação do algoritmo de geração da árvore de alcançabilidade:



Outro exemplo:

