3. Protocolos

A esta altura deve estar bem claro que o estudo de redes é o estudo de protocolos. Precisamos, então, de ferramentas para podermos discutir com clareza o assunto. Esta é uma área em desenvolvimento, mas existem soluções práticas disponíveis.

3.1. Verificação de Protocolos

Não é raro a situação em que dois sistemas que se comunicam sejam projetos de duas pessoas ou companias diferentes. Para que tudo funcione é necessário que as duas **implementações** do protocolo usado sejam corretas e que o protocolo em si seja correto. A engenharia de software nos fornece algumas ferramentas para garantirmos as implementações, mas vamos nos preocupar com os problemas que podem ocorrer no projeto do protocolo.

3.1.1. Impasse ("Deadlock")

O problema mais óbvio e comum em sistemas que se comunicam é o impasse onde cada espera que o outro faça algo e o outro também para esperando que algo aconteça. Um exemplo clássico é de dois caminhões se encontrando numa rua estreita com carros estacionados dos dois lados. Cada um para e espera que o outro recue. A simetria dos protocolos (cada um tem a mesma seqüência de ações) faz com que esse problema ocorra facilmente.

3.1.2. Atividade ("Liveness")

Quando um protocolo é desenvolvido informalmente (por exemplo, quando nem é descrito explicitamente mas apenas "surge" com a primeira implementação) pode acontecer de, sob certas circunstâncias, simplesmente cessarem todas as atividades. Programas que se comunicam não devem simplesmente encerrar sua execução sem que isso esteja previsto e a outra parte avisada.

3.1.3. Progresso

Não basta que haja atividade, mas sim que exista atividade útil. No caso em que duas pessoas se encontram na calçada e cada uma se desvia para dar passagem à outra e, vendo que continuam sem poder passar, voltam ao lado original (e assim por diante), está havendo muita atividade mas nenhum progresso. É preciso que informação seja transmitida e que o sistema não ande em círculos.

3.1.4. Completicidade

Um protocolo deve ter uma resposta adequada para cada mensagem que possa chegar. Um protocolo incompleto deixa de funcionar quando situações inesperadas, mas possíveis, ocorrem.

3.1.5. Terminação

O progresso de um sistema não garante necessariamente o progresso das atividades individuais. É preciso garantir que toda atividade que se inicia terminará em um tempo finito. Não é agradável a situação em uma lanchonete mal organizada em que uma pessoa fica sem atendimento enquanto outras vão passando na frente.

3.1.6. Estabilidade

Situações de erro ou de exceção ocorrem. É importante que o protocolo possa se recuperar automaticamente nesses casos. Qualquer protocolo que esteja sendo projetado ainda não está, é claro, em

funcionamento. Logo, todo protocolo deve cuidar de pelo menos uma exceção: a inicialização.

3.2. Linguagens de Especificação Formal

As linguagens de programação tradicionais são boas para se implementar protocolos, mas para descrever um protocolo as linguagens de especificação formal são mais adequadas. Normalmente elas são acompanhadas de ferramentas que permitem a verificação de diversos aspectos da especificação. Note que estas descrições não são executáveis.

Um método de desenvolver protocolos é o da **linguagem única**. O protocolo é descrito em uma linguagem de especificação formal (como Z, Lotos ou VDM) de uma maneira tão geral e simples que a probabilidade de existir algum erro seja mínima. Em seguida, o protocolo é descrito novamente na mesma linguagem, só que em maiores detalhes. Existem programas que podem comparar as duas descrições e apontar quaisquer discrepâncias. Quando os erros forem corrigidos, o processo pode ser repitido gerando-se uma terceira descrição ainda mais detalhado. Isto prossegue até que seja atingido um nível de detalhes satisfatório.

Em outras situações (no caso do SDL, por exemplo) são empregadas **duas linguagens**: a linguagem de especificação e a linguagem de critérios de validade. Comparando as duas descrições, um programa pode encontrar diversos tipos de erros.

3.3. Notações Gráficas

As notações gráficas são mais úteis para o comunicação entre pessoas e para a visualização das características dos protocolos. Todos os protocolos estudados no curso serão apresentados nas notações descritas a seguir.

3.3.1. Máquinas de Estados Finitos

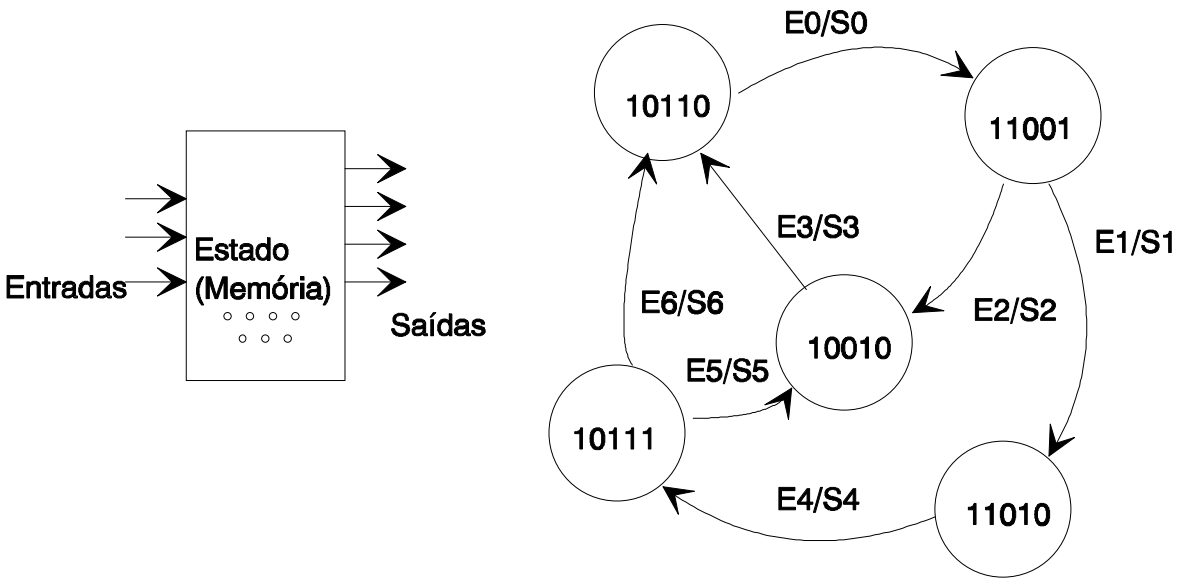


Figura 3.1 - Máquina de Estados Finitos

Todo sistema discreto (que é o caso de todos que estudaremos) pode ser descrito por uma máquina de estados finitos. Como mostra a figura 3.1, o sistema tem uma série de entradas, umas saídas e uma memória interna (que definine o estado). As saídas da máquina dependem não só das entradas, mas

também da história do sistema, conforme codificado em seu estado. Em nossa representação gráfica indicaremos cada estado pelo qual a máquina pode passar por um círculo com um nome que indica o conteúdo da memória naquele estado. Setas indicam as transições entre os estados. Elas recebem um nome da forma E/S, onde E é a entrada que provoca a transição e S é a saída apresentada pela máquina nesta situação. Numa representação alternativa as saídas podem ser associadas ao estado (e não à transição), mas para nosso curso esta primeira notação é mais interessante.

A cada novo bit de memória temos o dobro de estados possíveis. Rapidamente o número de estados fica muito grande para que o diagrama possa ser facilmente manipulado e compreendido. Uma opção é dividir a máquina em várias outras menores, mas nem sempre isto simplifica muito pois normalmente o estado dividido tem que ser comunicado entre as novas máquinas de estados finitos por meio de entradas e saídas adicionais.

3.3.2. Redes de Petri

As Redes de Petri são notações gráficas capazes de descrever sistemas nos quais ocorrem muitas atividades simultaneamente. A figura 3.2 mostra os elementos básicos de uma rede de Petri.

Os círculos representam processos enquanto que as barras grossas representam eventos. Uma ou mais bolas dentro de um círculo significa que o processo está ativo. Se todos os processo ligados à entrada do evento estiverem ativos, o evento "dispara" e ativa os processos que estiverem ligados às suas saídas e desativa os primeiros. Na parte esquerda da figura vemos dois processos ativos ligados às entradas do evento desenhado em preto. No caso do evento cinza, apenas um de seus processos de entrada está ativo e, portanto, ele não está apto para disparar. No lado direito da figura está a rede depois que o evento disparou - as bolas saíram dos processos de entrada e foram parar no processo de saída. Neste estado não há nenhum evento pronto para disparar.

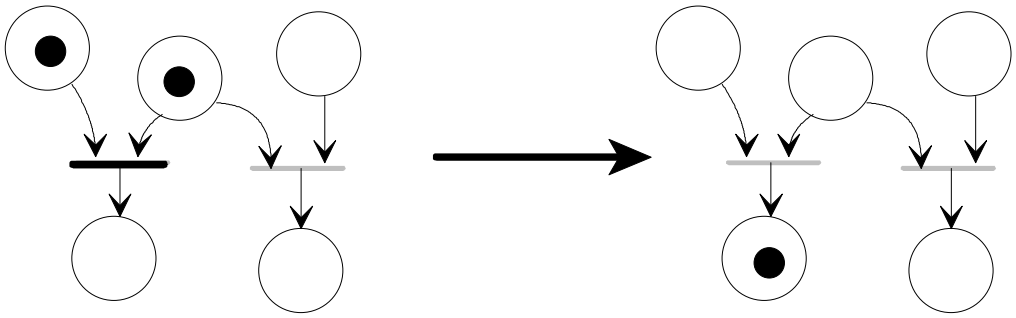


Figura 3.2 - Rede de Petri

3.3.3. Exemplos

Cada uma das notações gráficas apresentadas tem suas vantagens e desevantagens, como veremos, mas é sempre possível transformar uma em outra.

Na figura 3.3 vemos um sistema muito símples: um sinal de trânsito. Ele tem três lâmpadas - a vermelha (indicada por P, de Pare), a amarela (A, de Atenção) e a verde (L, de Livre). Cada uma pode estar acesa ou apagada, o que nos dá oito combinações possíveis. Destas, apenas as três indicadas pela máquina de estados no canto superior direito da figura são válidas. Não vamos nos preocupar em definir o que acontece se a máquina for parar em algum destes outros estados por algum motivo. Um estado em que as lâmpadas amarela e verde estivessem acesas ao mesmo tempo seria o estado AL, por exemplo. Precisamos definir qual é o estado inicial quando a máquina é ligada. Na figura o estado inicial é aquele em que só a luz vermelha está acesa, o estado P (Pare). A esta altura deve estar evidente que há algo muito estranho nesta máquina de estados finitos: as transições não têm nomes indicando entradas e saídas. Neste

caso, o próprio estado do sinal serve também como saída, mas e as entradas? Na realidade as transições ocorrem depois de um tempo predeterminado e não em função de algum evento exterior. Como isto está implícito no diagrama da figura 3.2, este deixou de ser uma notação completamente formal mas já é o suficiente para nossos propósitos.

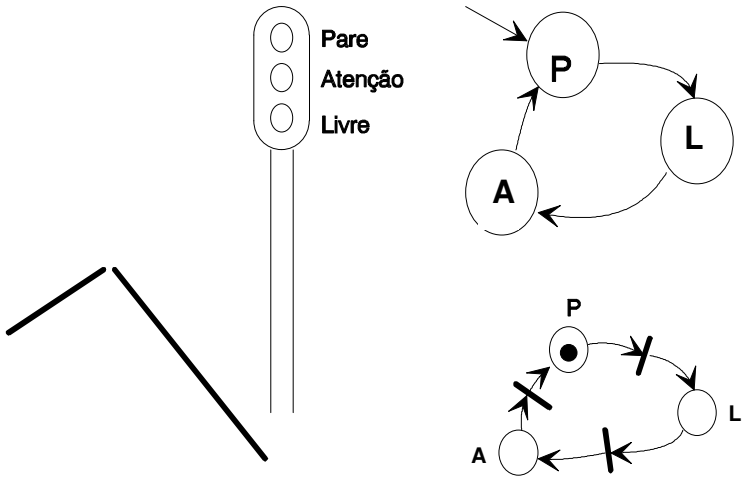


Figura 3.3 - Exemplo 1

A parte direita inferior da figura mostra a rede de Petri correspondente. Esta se encontra em seu estado inicial, P ativo e A e L inativos. O evento mais de cima é o único que está pronto e quando ele disparar P se tornará inativo e A ativo (a bolinha "pula" de P para A). Este evento corresponte à transição da máquina de estados do estado P para o estado A. As duas notações ficaram extremamente parecidas neste exemplo devido à simplicidade do sistema. Vamos complicar um pouco as coisas colocando um segundo sinal (figura 3.4).

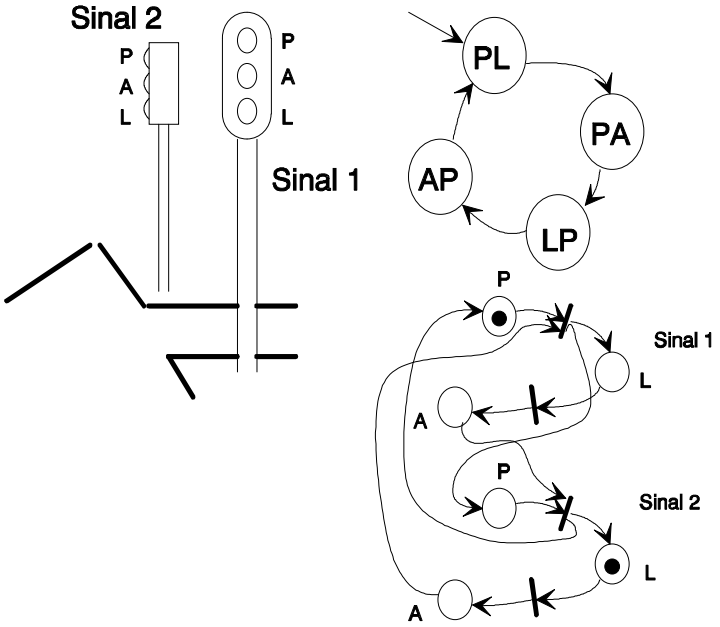


Figura 3.4 - Continuação do Exemplo 1

Agora temos mais três lâmpadas - teoricamente 64 estados possíveis. Como mostra a figura de cima, apenas quatro destes estados nos interessam. Indicaremos o estado por nomes do tipo PL, que significa lâmpada vermelha (P) do sinal 1 e verde (L) do sinal 2. A máquina de estados libera alternadamente o trânsito numa rua e depois na outra. Apesar de muito parecida à primeria vista, esta M.E.F. difere fundamentalmente da apresentada na figura 3.3. Já a rede de Petri tem, essencialmente, duas cópias da anterior interligadas para mostrar a dependência dos dois sinais. Ela ficou, no entanto, muito mais complexa do que a MEF. Apareceram uma série de setas que ligam uma rede de Petri à outra para manter os sinais sempre sincronizados. Isto ocorreu pois os dois sinais são **fortemente acoplados**, formando um sistema cujo comportamente global é bastante simples. Os outros exemplos, e praticamente todos os sistemas estudados no curso, são fracamente acoplados de modo que é muito difícil descrever seu comportamente por uma máquina de estados.

O problema do sistema como está proposto até agora é que os pedestres não têm nenhuma chance. Que tal acrescentarmos um estado PP para eles poderem atravessar uma rua sem se preocupar com carros que querem virar? A máquina de estados finitos ficaria assim: PL-->PA-->PP-->LP-->AP--> volta para PL. Novamente, neste caso a rede de Petri fica um pouco mais complicada. E se quisermos manter a simetria e fazer: PL-->PA-->PP-->LP-->AP--->PP--> volta para PL? Isto não é possível, pois não podemos ter dois estados PP. Se aumentarmos o estado com uma variável booleana podemos ter: 0PL-->0PA-->0PP-->1LP-->1AP-->1PP--> volta para 0PL. Por que 0PA e não 1PA? Na realidade, tanto faz. O que importa é distinguir 0PP de 1PP. Com esta complicação adicional o número de estados possíveis é agora 128, dos quais apenas 6 interessam. Tudo para que os pedestres tenham duas chances de atravessar a rua por ciclo da máquina.

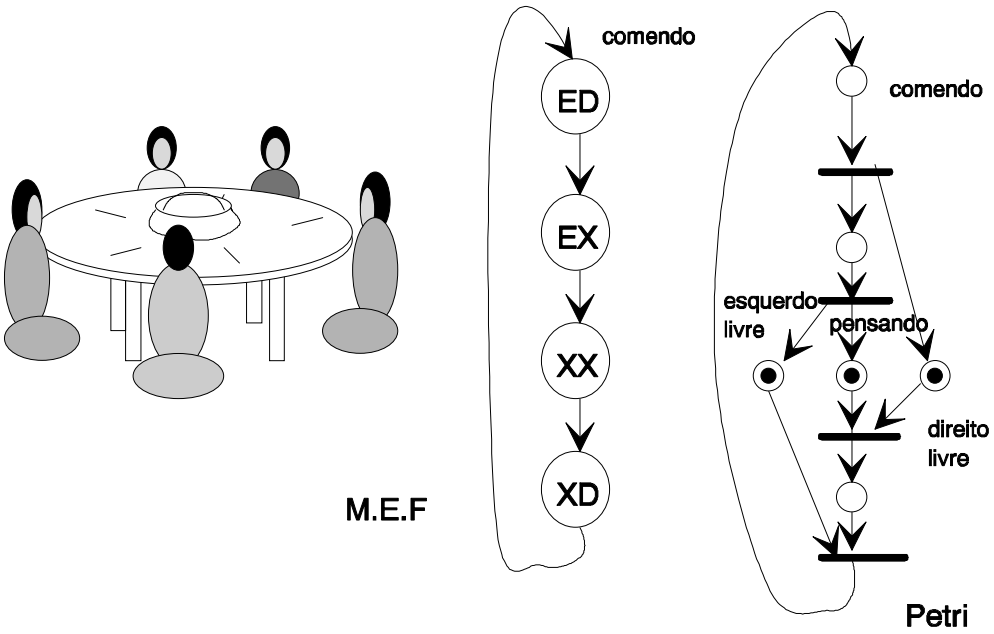


Figura 3.5 - Problema dos Filósofos Famintos

A figura 3.5 mostra um problema proposto para testar a expressividade de notações para descrever sistemas paralelos. Em volta de uma mesa sentam-se um certo número de filósofos (cinco, na figura) que nada fazem na vida senão pensar e comer. Bem no centro da mesa existe um prato com arroz, e distribuídos entre cada dois folósofos existem palitos. Para poder comer, um folósofo precisa pegar dois palitos, tanto o da sua esquerda como o da sua direita. Infelizmente isto significa que seus vizinhos não

podem comer ao mesmo tempo, já que dependem destes palitos também (não vamos nos preocupar com o problema de higiene neste exemplo). A figura mostra a máquina de estados finitos e a rede de Petri que descrevem o comportamento de apenas um filósofo. Na MEF estamos indicando a posse do palito esquerdo por E e do direito por D. Usamos X para indicar uma mão vazia. A figura indica que o filósofo pega (ou tenta pegar até conseguir) o palito direito, depois o esquerdo, come um pouco, devolve o palito direito, em seguida devolve o esquerdo, pensa um tanto e inicia tudo outra vez.

Se tentarmos descrever um sistema com dois filósofos, teremos um diagrama com oito estados. Para mostrar os cinco filósofos precisaríamos da ordem de 77 estados! Tal diagrama seria muito difícil de ser desenhado e de ser entendido. Em contraste, a figura 3.6 mostra que a rede de Petri do nosso sistema é relativamente símples. Este diagrama mostra várias coisas sobre o nosso sistema, das quais a mais importante é que se, da situação indicada na figura, todos os cinco filósofos pegarem os palitos à sua direita teremos uma situação de impasse. Isto é muito pouco provável, mas não podemos nos contentar com o projeto de um sistema que "quase sempre" funciona.

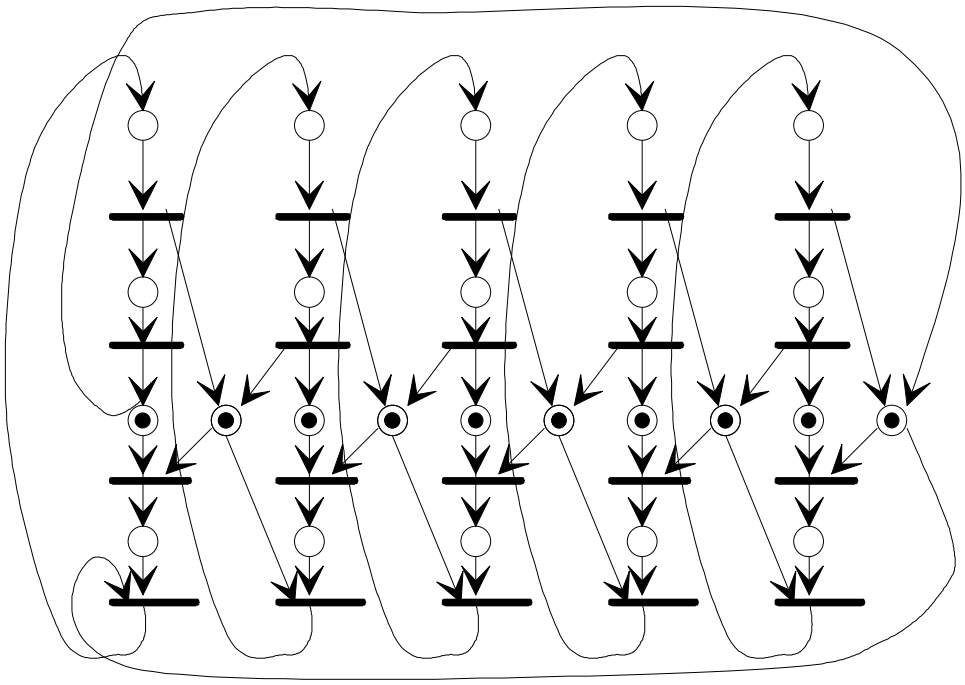


Figura 3.6 - Rede de Petri dos Cinco Filósofos

Na figura 3.7 modificamos o comportamento dos filósofos para que ou peguem os dois palitos e passem diretamente de pensar para comer, ou não pegam nenhum. Como isso é possível na prática se isto vai ser implementado em alguma linguagem de programação em que as coisas acontecem uma após a outra? Existem mecanismos como semáforos e regiões críticas que podem ser usados, mas não entraremos em detalhes aqui. Uma solução bastante óbvia é a seguinte: pega o palito direito e se não for possível pegar o da esquerda devolve o da direita e volta para o inicio. Isto realmente resolve o problema de impasse (deadlock) mas cria a possibilidade do "livelock", onde deixa de haver progresso. Nesta solução é possível que todos os filósofos fiquem pegando e soltando seus palitos direitos eternamente sem que nada de útil ocorra. Outra solução é quebrar a simetria: se um dos filósofos for diferente e pegar primeiro seu palito esquerdo desaparece a possibilidade de impasse. Finalmente, a figura 3.7 indica que agora os filósofos soltam os dois palitos simultaneamente. Isto não é necessário para que o sistema

funcione. A solução apresentada não garante terminação; isto é: é possível que um filósofo decida comer mas encontre seus palitos com o vizinho. Quando o vizinho devolver o palito, quem garante que é o primeiro filósofo quem vai pegá-lo agora e não o próprio vizinho? Nada impede que um filósofo mais rápido fique sempre passando na frente dos outros. Assim, enquanto o sistema como um todo está funcionando, é possível que partes dele fiquem paradas.

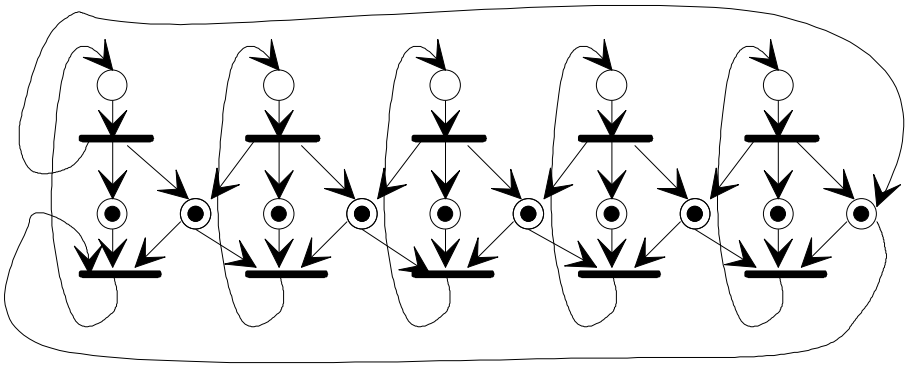


Figura 3.7 - Alteração da Rede de Petri dos Filósofos

O último exemplo é de um sistema de navegação em rios com comportas. A figura 3.8 mostra a rede de Petri proposta para o sistema. Existem dois navios esperando para passar do nível mais baixo (esquerdo) para o nível mais alto (direito), um navio esperando para fazer o percurso contrário e ainda outro já subindo.

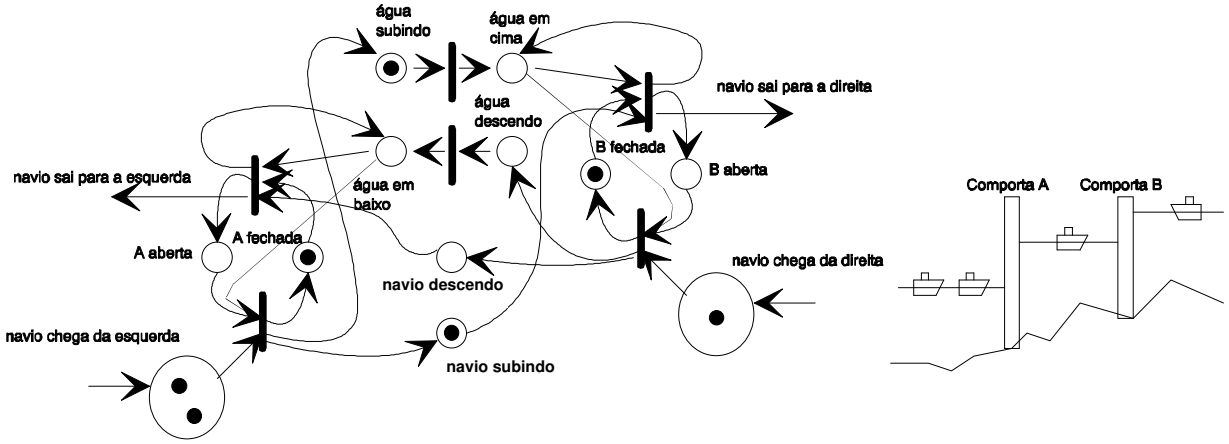


Figura 3.8 - Sistema de Navegação em Rio com Desnível

Na situação mostrada, o evento mais de cima é o único que está apto a disparar. Isto indica que o nível da água está no máximo. Neste instante, o evento da direita fica apto a disparar e a porta B se abre e o navio sai para a direita. Em seguida, o evento logo abaixo dispara e um navio entra na comporta, a porta B se fecha e a água começa a descer. Chegando no nível mais baixo, a porta A se abre e o navio sai pela esquerda. Um dos navios que está esperando à esquerda pode agora entrar na comporta e subir para a direita como o primerio. Assim, esse sistema permite exatamente um navio de cada vez passar alternadamente de um nível do rio para o outro.

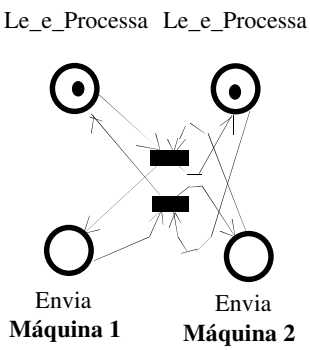
Se continuarmos nossa simulação veremos que este protocolo é incompleto, pois quando o segundo navio tiver subido, o sistema não abaixa a água para pegar o último que está esperando já que não tem

ninguém esperando para descer. É claro que assim como navios vão saindo para fora do sistema outros podem chegar pelas setas de baixo, mas não é possível fazer um projeto contando que os navios virão alternadamente da esquerda e da direita. Não vamos mostrar a solução completa pois é bastante complexa, mas a idéia básica de modelamento ou descrição de protocolos por notações gráficas está contida neste exemplo.

Note a semelhança entre navios que chegam "de algum" lugar e depois desaparecem do outro lado com a estrutura de uma camada do modelo de referência OSI. Na camada chegam pacotes de dados tanto da camada inferior como da superior que, depois de processados, atravessam para o outro nível. Normalmente estudaremos protocolos mais complexos que processam os pacotes transportados de alguma maneira e podem trabalhar com vários pacotes ao mesmo tempo. Os conceitos básicos, no entanto, são iguais aos dos exemplos vistos aqui.

Exercícios

1. Duas máquinas trocam informações usando o programa indicado ao lado, como mostrado pela Rede de Petri da figura. O que há de errado aqui e como pode ser consertado? (Apenas explique o que deve ser feito - não é necessário escrever um programa)

```
program Papo;
uses Comm;
begin
        while true do
        begin
                Le_e_Processa;
                Envia;
        end;
end.
```

Rede de Petri