Derivação Formal de Estruturas Distribuídas

Osvaldo S. F. Carvalho
Professor Adjunto do Departamento de Ciência da Computação da UFMG
DCC/ICEX - Universidade Federal de Minas Gerais
Belo Horizonte - MG
Caixa Postal 702 - CEP: 30161-970
E-mail: vado@dcc.ufmg.br

Vladimir O. Di Iorio
Professor do Departamento de Informática da UFV
DPI - Universidade Federal de Viçosa
CEP:36570-000 Viçosa - MG
E-mail: vladimir@dcc.ufmg.br

Abstract

A distributed data structure has n components, one at each site in a distributed system. The correctness of a distributed data structure is often expressed by means of a predicate that should be kept invariant. As an example, a distributed data structure associated with a mutual exclusion problem has components that describe the actual state of each site – using the critical resource, or idle – and the predicate that expresses correctness imposes that no two sites can be using the critical resource at the same time. A formal technique for derivation of distributed data structures specified by an invariant is described, the correctness of which is defined by some (global) predicate. Using a network with only 2 nodes, we show that every predicate is equivalent to an inequality on a partially ordered domain suitably choosen.

Resumo

Uma estrutura de dados distribuída possui n componentes, uma em cada sítio de um sistema distribuído (nós de uma rede de computadores). A correção de uma estrutura de dados distribuída é freqüentemente expressada através de um predicado que deve ser mantido invariante. Por exemplo, uma estrutura associada com o problema da exclusão mútua pòssui componentes que descrevem o estado atual de cada sítio – usando ou não o recurso compartilhado – e o predicado que expressa correção exige que dois ou mais sítios não podem estar utilizando o recurso simultaneamente. Uma técnica formal para derivação de estruturas de dados distribuídas especificada por um invariante é descrita, cuja correção é definida por um predicado global. Utilizando uma rede com 2 nós, mostramos que qualquer predicado é equivalente a uma desigualdade sobre um domínio parcialmente ordenado adequadamente escolhido.

1 Introdução

Considere uma rede de computadores com n nós que se comunicam apenas através de mensagens, que não compartilham memória e que não possuem um relógio comum. Uma estrutura de dados distribuída é constituída por n componentes, cada um localizado em um dos nós da rede. A correção parcial de uma estrutura como essa pode ser expressa por um predicado que envolva o estado corrente de todos os componentes, que deve ser mantido invariante.

Como um exemplo, podemos citar o problema da *Exclusão Mútua*, que consiste de um recurso qualquer sendo compartilhado por todos os nós da rede. Cada componente da estrutura de dados distibuída descreve o estado atual do nó, que pode estar usando ou não o recurso compartilhado. O predicado que deve ser mantido invariante é o seguinte: dois nós não podem estar usando o recurso simultaneamente. Uma solução para esse problema é exibida em [Lamport, 1978], onde se introduziu o conceito de *relógios lógicos*. Outras soluções foram exibidas posteriormente em [Ricart e Agrawala, 1981, Carvalho e Roucairol, 1983, Maekawa, 1985, Barbara e Garcia-Molina, 1986].

Outro exemplo simples é o problema *Leitores e Escritores* ([Tanenbaum, 1992],p.58 e [Hansen, 1973],p.106). Nesse problema, tem-se um conjunto de dados que pode ser acessado concorrentemente pelos nós da rede, os quais podem executar uma operação de leitura ou de escrita (alteração). A estrutura distribuída mantém, em cada nó da rede, um componente que descreve o estado corrente, que pode ser *escrevendo*, *lendo*, ou *"nenhuma ação"*. Um algoritmo distribuído para tratar esse problema vai estar parcialmente correto se o seguinte predicado for mantido invariante: se um nó estiver no estado *escrevendo*, todos os outros devem estar no estado *"nenhuma ação"*.

Produzir um algoritmo para executar uma tarefa como as descritas acima é um empreendimento complexo, dadas as dificuldades envolvidas. Cada nó só pode executar uma *alteração segura* na sua componente da estrutura de dados distribuída. O conceito de alteração segura envolve a certeza de que, se um nó faz uma transição para um novo estado, as possíveis transições executadas pelos demais não irão violar a correção do predicado. A dificuldade reside em dois pontos principais:

- Não possuindo uma memória comum, um nó não pode consultar o estado corrente dos demais. A informação que ele possui sobre o estado dos outros nós é proveniente de mensagens recebidas. Toda mensagem que chega carrega uma informação sobre um momento no passado daquele nó que a enviou, assim não podemos afirmar que ela estará necessariamente atualizada.

- Não possuindo um relógio comum, não é possível sincronizar as transições executadas por dois ou mais nós da rede. Quando um nó vai realizar uma alteração no seu componente da estrutura distribuída, ele não tem subsídios para inferir o momento exato que algum outro nó da rede vai poder executar determinada transição.

Um algoritmo distribuído para resolver tarefas como as enumeradas deve possuir algumas propriedades indispensáveis, como por exemplo, nunca chegar à uma situação de *deadlock* (bloqueio). Essa propriedade é definida em [Shaw, 1974],p.203 e [Coffman e Denning, 1973],p.44. Outra exigência é garantir que todos os nós poderão atingir um estado desejado em tempo finito (*fairness*). Esse último conceito é formalizado de maneira elegante em [Chandy e Misra, 1984], através de *grafos de precedência*.

O trabalho desenvolvido em [Owick e Gries, 1976] mostra um método axiomático para provar propriedades como as descritas acima através da adição de variáveis auxiliares ao programa. Esse trabalho constitui uma proposta de extensão do conjunto de axiomas para correção parcial proposto em [Hoare, 1969].

Dado um predicado que envolve o estado dos componentes de uma rede de computadores, precisamos dispor também de meios que permitam verificar se o algoritmo utilizado mantém este predicado invariante. Para mostrar que um predicado **P** se mantém invariante, o método geral de prova, como descrito em [Keller, 1976], consiste em exibir um outro predicado mais forte que **P**, que seja indutivo. Essa tarefa, entretanto, envolve conhecimento de elementos específicos da implementação utilizada para resolver o problema, e nem sempre pode ser realizada mecanicamente.

1.1 Objetivos do Trabalho

Este trabalho procura determinar diretivas para tornar a construção de algoritmos distribuídos uma tarefa mais sistemática, propondo que a prova de correção venha como conseqüência da concepção do algoritmo. É descrita uma técnica formal para derivação de estruturas de dados distribuídas, especificadas a partir da definição de um predicado que se deseja manter invariante.

Inicialmente, em uma rede com apenas dois nós, é proposto um problema simples especificado através de um predicado sobre a estrutura de dados distribuída. As alterações que cada nó pode realizar sobre sua componente são classificadas como *seguras* ou *perigosas*, indicando se as mesmas podem levar o sistema de uma situação correta para uma situação de erro. Em seguida, discutimos o formato de protocolos que possibilitem a manutenção desse predicado.

As idéias são então generalizadas através de um mapeamento dos domínios relacionados ao problema em domínios parcialmente ordenados adequadamente escolhidos. O predicado original **P** é distribuído em duas componentes locais e uma componente global, com o objetivo de reduzir o tráfego de mensagens. Essas componentes se baseiam unicamente em um dos dois domínios relacionados a **P**. Com a distribuição proposta consegue-se então, a partir de dois domínios arbitrários E_1 e E_2, um único domínio parcialmente ordenado (por exemplo, $(2^{E_1}, \subseteq)$) e duas funções f_1 e f_2 que mapeiam o predicado original em uma desigualdade sobre esse domínio parcialmente ordenado. Os conjuntos representados nesse domínio são apenas aqueles cuja "perda de território" por parte de um nó permita ao outro nó um conjunto maior de possíveis transições.

Usando essa teoria, são derivados protocolos simples para resolver problemas especificados por um predicado sobre uma estrutura de dados distribuída, em uma rede com dois nós. Por exemplo, serão desenvolvidos algoritmos para resolver o problema da *Exclusão Mútua* e o *Problema das Engrenagens*, cuja formulação introduziremos no momento oportuno.

2 Especificando problemas através de predicados

Considere um problema envolvendo estruturas de dados distribuídas com as seguintes características:

- A rede de computadores possui apenas 2 nós que se comunicam unicamente através de mensagens, não compartilham memória e não possuem um relógio comum; estes nós serão designados *nó 1* e *nó 2*.
- Cada componente da estrutura é um inteiro com valores variando entre 0 e 9; usaremos e_1 para representar a componente do nó 1 e e_2 para representar a componente do nó 2.
- a estrutura de dados estará correta se, e somente se, o valor de e_1 for menor ou igual ao valor de e_2.

Para $i = 1, 2$, vamos chamar de E_i o domínio dos valores que a componente do nó i pode assumir. A estrutura de dados será designada por **E** e representada por um par ordenado onde

cada componente é um elemento de E_i. Podemos então especificar o problema em questão usando um predicado $\mathbf{P} : \mathbf{E} \rightarrow \{true, false\}$ que envolva o estado das componentes de cada nó. Para $i = 1, 2$:

$$E_i = (0, \ldots, 9), \quad e_i \in E_i, \quad \mathbf{E} = E_1 \times E_2, \quad \mathbf{P} \equiv e_1 \leq e_2.$$

2.1 Determinação de protocolos

No problema proposto, supondo que o predicado inicialmente é válido, devemos exibir um protocolo que o mantenha invariante. Cada nó poderá incrementar ou decrementar o valor de sua componente, ficando a cargo desse protocolo gerenciar as transições válidas (chamaremos de *transição* qualquer mudança de estado da componente de um nó).

Supondo um meio em que a transmissão das mensagens é sempre feita sem distorções ou perdas, o seguinte protocolo irá satisfazer os requisitos necessários:

1. Um único token circula entre os dois nós; para $i = 1, 2$, se o nó i envia o token, ele conterá o valor de e_i naquele exato momento.

2. O nó 1 só poderá realizar uma transição $(e_1 = v) \rightarrow (e_1 = v')$, onde $v' > v$, quando receber o token trazendo um valor de e_2 e este valor for maior ou igual a v'.

3. O nó 2 só poderá realizar uma transição $(e_2 = v) \rightarrow (e_2 = v')$, onde $v' < v$, quando receber o token trazendo um valor de e_1 e este valor for menor ou igual a v'.

Esse protocolo será designado *protocolo do token circulante*.

As transições podem ser classificadas como *seguras* ou *perigosas*. O primeiro tipo engloba as transições que nunca poderiam levar o sistema de um estado correto para um incorreto. Por exemplo, se o valor de e_1 é 5 e o sistema está em um estado que satisfaz o predicado $\mathbf{P}$, pode-se alterá-lo para 4 com a certeza de que não causaria uma violação de $\mathbf{P}$. E isso pode ser feito sem que haja comunicação entre os nós. O segundo tipo, por outro lado, engloba as transições que podem acarretar uma violação do predicado $\mathbf{P}$. Por exemplo, se $e_1 = 6$, só podemos fazer $e_1 = 7$ se tivermos a certeza de que a componente e_2 vai se manter em um valor maior ou igual a 7.

2.2 Distribuição de um predicado

O tráfego de mensagens pode ser reduzido consideravelmente se for utilizada uma técnica simples para "distribuição" do predicado $\mathbf{P}$. Para isso, vamos introduzir duas novas variáveis: c_1, no nó 1, e c_2, no nó 2. Vamos também substituir o predicado $\mathbf{P} \equiv e_1 \leq e_2$ pela conjunção dos predicados

$$L_1 \equiv e_1 \leq c_1, \quad L_2 \equiv c_2 \leq e_2, \quad G \equiv c_1 \leq c_2.$$

Os predicados L_1 e L_2 são chamados de *predicados locais* e G é chamado de *predicado global*.

Para mantermos a validade do predicado G, podemos utilizar novamente o protocolo do token circulante. Para o caso dos predicados locais, basta introduzir, sempre que um nó for executar uma transição, testes que garantam que as variáveis locais e_1 e e_2 não irão violar esses predicados.

Usando essa abordagem, o nó 1 não precisa necessariamente esperar a chegada de uma mensagem sempre que for incrementar o valor de e_1. Da mesma forma, podem ocorrer casos em que o nó 2 decrementa o valor de e_2 sem a troca de mensagens. O ganho vem da memorização do conhecimento adquirido, registrado nas variáveis c_1 e c_2. A figura 1 mostra um algoritmo que implementa essas idéias. A notação utilizada é a sugerida em [Hoare, 1978], usando os *comandos guardados* introduzidos em [Dijkstra, 1975]. Neste trabalho utilizaremos constantemente essa

```
TYPE State = 0..9;
  Knowledge = 0..9;
[ P1:: e1: State; hastoken: BOOLEAN; c1: Knowledge;
  e1 := 5; hastoken := TRUE; c1 := 5;
  *[ e1 > 0 → e1 := e1 - 1;
  | receive(token) → c1 := token.c; hastoken := TRUE;
  | TRUE → c1 := e1;
  | c1 > e1 → e1 := e1 + 1;
  | hastoken → token.c := c1; send(token); hastoken := FALSE;
  ]
]
||
[ P2:: e2: State; hastoken : BOOLEAN; c2: Knowledge;
  e2 := 5; hastoken := FALSE; c2 := 5;
  *[ e2 < 9 → e2 := e2 + 1;
  | receive(token) → c2 := token.c; hastoken := TRUE;
  | TRUE → c2 := e2;
  | c2 < e2 → e2 := e2 - 1;
  | hastoken → token.c := c2; send(token); hastoken := FALSE;
  ]
]
```

Figura 1: Algoritmo usando distribuição do predicado **P**.

notação nos algoritmos apresentados. Convém salientar, entretanto, que a comunicação não é feita de maneira síncrona como na linguagem CSP. Estaremos sempre considerando uma comunicação assíncrona ([Andrews e Schneider, 1983, Bal *et al.*, 1989]), sem estabelecer nenhuma suposição sobre a velocidade relativa das mensagens trocadas.

2.3 Domínios Parcialmente Ordenados

Nesta seção vamos produzir uma generalização das idéias discutidas até agora. Para isso, usaremos o conceito de domínios parcialmente ordenados, que são estruturas da forma $(\mathcal{D}, \preceq)$, onde $\mathcal{D}$ é um conjunto qualquer e $\preceq$ é uma relação de ordenação parcial em $\mathcal{D}$. Uma forma bastante utilizada para se representar um domínio como esse é o diagrama de Hasse ([Grimaldi, 1988]), como o exibido na figura 2.

A representação de uma relação em um conjunto qualquer através de um grafo direcionado é feita colocando-se os elementos desse conjunto como vértices do grafo, sendo que uma aresta liga o vértice x ao vértice y se, e somente se, o par ordenado (x, y) pertencer a essa relação. Sempre que uma relação induz a formação de um domínio parcialmente ordenado em um conjunto qualquer, essa relação tem que constituir uma ordenação parcial. Ou seja, tem que ser reflexiva, antissimétrica e transitiva. O diagrama de Hasse se vale do conhecimento prévio dessas propriedades para produzir uma representação mais econômica. Adota-se a convenção de que a direção das arestas do grafo é sempre de baixo para cima.

A generalização que iremos propor a seguir vai utilizar as estruturas de ordenação parcial que acabamos de apresentar. As variáveis e_1 e e_2 não mais estarão restritas a inteiros entre 0 e 9, mas poderão assumir valores de domínios arbitrários E_1 e E_2, os quais podem ser distintos. Suponha que tenhamos um domínio parcialmente ordenado $(\mathcal{D}, \preceq)$, e funções $f_i : E_i \rightarrow \mathcal{D}$ (para

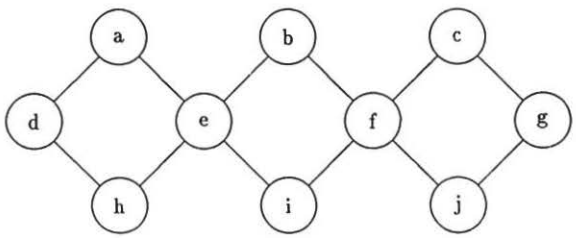

Figura 2: Domínio parcialmente ordenado.

$i = 1, 2$) que mapeiam os valores de e_1 e e_2, de seus respectivos domínios em $\mathcal{D}$.

É fácil ver que, se o predicado que desejamos manter invariante puder ser escrito como $\mathbf{P} \equiv f_1(e_1) \preceq f_2(e_2)$, todos os resultados discutidos até agora poderão ser usados, com pequenas adaptações. As transições seguras e perigosas podem ser facilmente identificadas:

- No nó 1, as transições do tipo $(e_1 = v) \rightarrow (e_1 = v')$, onde $f_1(v') \preceq f_1(v)$, são seguras, podendo ser executadas sem troca de mensagens. Todas as outras são perigosas e devem seguir algum protocolo.
- No nó 2, as transições do tipo $(e_2 = v) \rightarrow (e_2 = v')$, onde $f_2(v) \preceq f_2(v')$, são seguras e todas as outras são perigosas.

O protocolo do token circulante pode ser generalizado da seguinte forma, para $i = 1, 2$ e $v, v' \in E_i$:

1. Um único token circula entre os dois nós; se o nó i envia o token, ele conterá o valor de $f_i(e_i)$, avaliado no momento da transmissão.
2. O nó 1 só poderá realizar uma transição $(e_1 = v) \rightarrow (e_1 = v')$, onde $f_1(v') \not\preceq f_1(v)$, quando receber o token trazendo um valor u (representando $f_2(e_2)$) tal que $f_1(v') \preceq u$.
3. O nó 2 só poderá realizar uma transição $(e_2 = v) \rightarrow (e_2 = v')$, onde $f_2(v) \not\preceq f_2(v')$, quando receber o token trazendo um valor u (representando $f_1(e_1)$) tal que $u \preceq f_2(v')$.

A distribuição do protocolo $\mathbf{P}$ também pode ser feita de maneira bem simples. Os nós deverão manter também variáveis c_1 e c_2, e ambas assumirão valores do domínio parcialmente ordenado $(\mathcal{D}, \preceq)$. Pode-se obter uma economia no tráfego de mensagens ao substituirmos o predicado $\mathbf{P} \equiv f_1(e_1) \preceq f_2(e_2)$ pela conjunção dos predicados

$$L_1 \equiv f_1(e_1) \preceq c_1, \quad L_2 \equiv c_2 \preceq f_2(c_2), \quad G \equiv c_1 \preceq c_2.$$

2.4 Definição dos Mapeamentos

Na seção anterior, vimos como generalizar as idéias introduzidas no início do capítulo, usando domínios parcialmente ordenados. Vamos agora determinar de que forma poderemos implementar esses resultados.

Considere dois domínios arbitrários E_1 e E_2 para as variáveis locais e_1 e e_2 e um predicado $\mathbf{P} : E_1 \times E_2 \rightarrow \{true, false\}$. Vamos tentar identificar um domínio parcialmente ordenado $(\mathcal{D}, \preceq)$

e funções f_1 e f_2 que mapeiam os domínios E_1 e E_2 e o predicado $\mathbf{P}$ em uma desigualdade em $\mathcal{D}$.

Como proposta inicial, vamos tomar o conjunto 2^{E_1} dos subconjuntos de E_1, parcialmente ordenado pela relação de inclusão. Precisamos encontrar funções $f_1 : E_1 \rightarrow 2^{E_1}$ e $f_2 : E_2 \rightarrow 2^{E_1}$ tais que

$$f_1(e_1) \preceq f_2(e_2) \Leftrightarrow \mathbf{P}(e_1, e_2).$$

Para simplificar, vamos fazer $f_1(e_1) = \{e_1\}$. A função f_2 deve funcionar de maneira que

$$\{e_1\} \preceq f_2(e_2) \Leftrightarrow \mathbf{P}(e_1, e_2).$$

Nesse ponto, é conveniente introduzirmos a seguinte definição:

Definição 1 Dados dois conjuntos E_1 e E_2 e um predicado $\mathbf{P}$ sobre $E_1 \times E_2$, dizemos que dois subconjuntos $S_1 \subseteq E_1$ e $S_2 \subseteq E_2$ são $\mathbf{P}$-*compatíveis* se, e somente se, para $e_1 \in E_1$, $e_2 \in E_2$, temos

$$(e_1, e_2) \in S_1 \times S_2 \Rightarrow \mathbf{P}(e_1, e_2).$$

Ou seja, dois conjuntos S_1, S_2 são $\mathbf{P}$-compatíveis se, e somente se, cada par ordenado formado por um elemento de S_1 e um elemento de S_2 satisfizer ao predicado $\mathbf{P}$. □

Vamos definir também o seguinte mapeamento, para $i, j \in \{1, 2\}$, $i \neq j$:

$$\begin{array}{lcl} \mathbf{P}_{ji}(S_j) & = & \{e_i \in E_i \mid \forall\, e_j \in S_j, \mathbf{P}(e_i; e_j)\}, \text{ se } S_j \neq \emptyset_j, \\ \mathbf{P}_{ji}(\emptyset_j) & = & E_i \end{array} \quad (1)$$

onde $(e_i; e_j)$ denota (e_i, e_j) se $i < j$, e (e_j, e_i) se $j < i$. $\mathbf{P}_{ji}$ mapeia um subconjunto S_j de E_j em um subconjunto $\mathbf{P}$-compatível de E_i. Chamaremos o mapeamento $\mathbf{P}_{ji}(S_j)$ de *transformada* de S_j por $\mathbf{P}$.

Podemos adotar então o seguinte formato para f_2:

$$f_2(e_2) = \mathbf{P}_{21}(\{e_2\}).$$

Como $\{e_1\} \subseteq \mathbf{P}_{21}(\{e_2\}) \Leftrightarrow \mathbf{P}(e_1, e_2)$, chegamos ao resultado desejado. Ou seja, dados dois domínios arbitrários E_1 e E_2 e um predicado $\mathbf{P}$, encontramos um domínio parcialmente ordenado $(2^{E_1}, \preceq)$ e duas funções f_1 e f_2 que mapeiam o predicado original em uma desigualdade sobre esse domínio parcialmente ordenado.

Como vimos nas seções 2.1 e 2.2, podemos utilizar protocolos simples para manter invariante essa desigualdade. Além disso, ela pode ser decomposta em dois predicados locais e um predicado global, seguindo a idéia de economia no tráfego de mensagens que discutimos anteriormente. Assim teremos:

$$L_1 \equiv \{e_1\} \subseteq c_1, \quad L_2 \equiv c_2 \subseteq \mathbf{P}_{21}(\{e_2\}), \quad G \equiv c_1 \subseteq c_2.$$

Nesse caso, chamaremos as variáveis c_1 e c_2 de *territórios de possíveis estados* (ou simplesmente *territórios*) dos nós 1 e 2, respectivamente.

Vamos utilizar essas idéias em um exemplo prático. Considere o problema da *Exclusão Mútua* envolvendo dois nós. Nesse problema, temos

$$E_1 = E_2 = \{idle, busy\}$$

onde *idle* representa um estado em que o nó não está utilizando o recurso compartilhado, e *busy* representa o contrário. O predicado pode ser especificado como:

$$\mathbf{P} \equiv (e_1 = idle) \vee (e_2 = idle).$$

```
TYPE State = (idle, busy);
  Knowledge = SET OF State;
[ P1:: e1: State; c1: Knowledge; hastoken: BOOLEAN;
  e1 := idle; c1 := [idle]; hastoken := TRUE;
  *[[idle] ⊆ c1 → e1 := idle;
  | [busy] ⊆ c1 → e1 := busy;
  | e1 = idle → c1 := [idle];
  | e1 = busy → c1 := [busy];
  | receive(token) → c1 := token.c;
                hastoken := TRUE;
  | hastoken → token.c := c1;
                send(token); hastoken := FALSE;
  ]
]
||
[ P2:: e2: State; c2: Knowledge; hastoken: BOOLEAN;
  e2 := idle; c2 := [idle, busy]; hastoken := FALSE;
  *[c2 ⊆ [idle] → e2 := busy;
  | c2 ⊆ [idle,busy] → e2 := idle;
  | e2 = idle → c2 := [idle,busy];
  | e2 = busy → c2 := [idle];
  | receive(token) → c2 := token.c;
              hastoken := TRUE;
  | hastoken → token.c := c2;
              send(token); hastoken := FALSE;
  ]
]
```

Figura 3: Exclusão Mútua.

A figura 3 mostra um algoritmo distribuído que implementa uma solução para esse problema. Esse algoritmo utiliza um mapeamento do predicado original no domínio parcialmente ordenado $(2^{E_1}, \preceq)$. O protocolo utilizado é o do token circulante, sendo que a desigualdade $\{e_1\} \subseteq \mathbf{P}_{21}(\{e_2\})$ foi distribuída em predicados locais L_1 e L_2, e no predicado global G, da maneira que especificamos anteriormente.

No nó 1, todas as transições devem obedecer ao predicado local $L_1 \equiv \{e_1\} \subseteq c_1$. No nó 2, precisamos antes calcular

$$\begin{aligned} \mathbf{P}_{21}(\{idle\}) &= \{idle, busy\} \\ \mathbf{P}_{21}(\{busy\}) &= \{idle\} \end{aligned}$$

e descobrir os valores corretos que e_2 pode assumir de maneira a manter o predicado local $L_2 \equiv c_2 \subseteq \mathbf{P}_{21}(\{e_2\})$ invariante. Por exemplo, uma condição necessária para que e_2 possa assumir o valor *busy* é que $c_2 \subseteq \mathbf{P}_{21}(\{busy\}) = \{idle\}$.

Nesse ponto, algumas questões devem ser levantadas. Nós elegemos 2^{E_1} para ser o domínio parcialmente ordenado com que iríamos trabalhar. Se 2^{E_2} fosse um domínio diferente do primeiro, quais seriam as implicações de escolher este ou aquele? E se o tamanho dos domínios fosse muito discrepante? Outra questão importante é a seguinte: a estrutura de dados distribuída que utilizamos nessa solução é a mais eficiente?

Essas e outras dúvidas serão respondidas na seção seguinte.

3 Conexões de Galois

Considere o algoritmo exibido na figura 3. Observando o seguinte comando do processo P1:

$$\text{e1} = \text{busy} \rightarrow \text{c1} := [\text{busy}]$$

notamos que o nó 1 pode restringir seu território de possíveis transições de uma forma não inteligente. Ou seja, ele poderia, por exemplo, alterar o valor de c_1 de $\{idle, busy\}$ para $\{busy\}$. Mas, com isso, não estaria de forma alguma possibilitando ao nó 2 realizar uma expansão no seu território, uma vez que

$$\mathbf{P}_{12}(\{busy\}) = \{idle\} = \mathbf{P}_{12}(\{idle, busy\}).$$

Nesta seção estudaremos uma forma de representação para as estruturas de dados distribuídas que evite esse tipo de "desperdício". Ou seja, para $i, j \in \{1,2\}$, $i \neq j$, no nó i serão representados apenas os territórios cuja expansão só possa ser realizada se ocorrer uma restrição do território do nó j.

3.1 Conjuntos P-fechados

Sejam E_1 e E_2 dois conjuntos quaisquer e $\mathbf{P}$ um predicado sobre $E_1 \times E_2$. Para $i = 1, 2$, as transformadas de S_i por $\mathbf{P}$, onde $S_i \subseteq E_i$, podem ser utilizadas para expressar $\mathbf{P}$-compatibilidade da seguinte forma:

$$\begin{pmatrix} S_1 \neq \emptyset_1 \\ \wedge \\ S_2 \neq \emptyset_2 \end{pmatrix} \Rightarrow \begin{pmatrix} S_1 \times S_2 \subseteq \mathbf{P} \\ \Updownarrow \\ S_1 \subseteq \mathbf{P}_{21}(S_2) \\ \Updownarrow \\ S_2 \subseteq \mathbf{P}_{12}(S_1) \end{pmatrix} \tag{2}$$

A partir dessa equação, podemos concluir que a transformada por $\mathbf{P}$ de um subconjunto não vazio S_i de E_i é o maior subconjunto de E_j que é $\mathbf{P}$-compatível com S_i.

As duas propriedades seguintes caracterizam uma *Conexão de Galois* [Ore, 1944, Szasz, 1963] entre os reticulados 2^{E_1} e 2^{E_2}:

$$S_i \subseteq R_i \Rightarrow \mathbf{P}_{ij}(S_i) \supseteq \mathbf{P}_{ij}(R_i) \tag{3}$$

$$\mathbf{P}_{ji}(\mathbf{P}_{ij}(Si)) \supseteq S_i \tag{4}$$

Se S_i é um subconjunto de R_i, então a transformada de R_i por $\mathbf{P}$ é um subconjunto de S_i. Além disso, S_i será sempre um subconjunto da transformada dupla de $\mathbf{P}$ aplicada a ele mesmo – este importante conceito será utilizado logo a seguir, quando definirmos a noção de conjuntos $\mathbf{P}$-fechados.

Outras propriedades de interesse são:

$$S_i \supseteq R_i \Rightarrow \mathbf{P}_{ji}(\mathbf{P}_{ij}(S_i)) \supseteq \mathbf{P}_{ji}(\mathbf{P}_{ij}(R_i)) \tag{5}$$

$$\mathbf{P}_{ij}(S_i \cup R_i) = \mathbf{P}_{ij}(S_i) \cap \mathbf{P}_{ij}(R_i) \tag{6}$$

$$\mathbf{P}_{ij}(S_i \cap R_i) \supseteq \mathbf{P}_{ij}(S_i) \cup \mathbf{P}_{ij}(R_i) \tag{7}$$

Usando as equações 3 e 4, podemos chegar ao resultado seguinte:.

$$\mathbf{P}_{ji}(\mathbf{P}_{ij}(\mathbf{P}_{ji}(\mathbf{P}_{ij}(S_i)))) = \mathbf{P}_{ji}(\mathbf{P}_{ij}(S_i)) \tag{8}$$

As propriedades 4 e 8 caracterizam as transformadas duplas como *operações de fechamento* nos reticulados 2^{E_i}, $i = 1, 2$. Dado um subconjunto S_i de E_i, denotamos seu fechamento por $\mathbf{P}$ - dado por $\mathbf{P}_{ji}(\mathbf{P}_{ij}(S_i))$ - por $\lceil(S_i)\rceil_P$. Chegamos então a uma definição que será muito importante no nosso estudo:

Definição 2 Um subconjunto S_i de E_i é **P**-*fechado* se, e somente se, $S_i = \lceil(S_i)\rceil_P$. □

O teorema que se segue resume algumas propriedades úteis desses subconjuntos.

Teorema 1 (Ore, 1944)
Considere dois conjuntos quaisquer E_1 e E_2, e uma relação binária $\mathbf{P} \subseteq E_1 \times E_2$. Sejam $\mathbf{P}_{12}$ e $\mathbf{P}_{21}$ aplicações entre 2^{E_1} e 2^{E_2} determinadas por **P**, de acordo com as equações exibidas em 1. Então:

1. O mapeamento $\mathbf{P}_{ji}(\mathbf{P}_{ij}())$ é uma operação de fechamento em 2^{E_i}; os elementos **P**-fechados têm a forma $\mathbf{P}_{ji}(\mathbf{P}_{ij}(S_i))$ para algum $S_i \subseteq E_i$, e também a forma $\mathbf{P}_{ji}(S_j)$ para algum $S_j \subseteq E_j$.
2. Os subconjuntos **P**-fechados de E_i compõem um sub-reticulado de 2^{E_i}.
3. Existe um isomorfismo dual entre o reticulado de subconjuntos **P**-fechados de E_1 e o reticulado de subconjuntos **P**-fechados de E_2 (as correspondências são dadas pelas transformadas $\mathbf{P}_{ij}$ e $\mathbf{P}_{ji}$).

□

3.2 Utilização

Na seção 2.4, utilizamos o domínio $(2^{E_1}, \subseteq)$ e as funções $f_1(e_1) = \{e_1\}$, $f_2(e_2) = \mathbf{P}_{21}(\{e_2\})$ na distribuição do predicado **P**. Vamos nos valer dos resultados obtidos a partir do teorema 1 para podermos utilizar não mais o conjunto 2^{E_1}, mas apenas o conjunto dos subconjuntos **P**-fechados de E_1. Na realidade, esse teorema nos fornece um resultado ainda mais forte, mostrando que, se escolhêssemos o domínio dos subconjuntos **P**-fechados de E_2, o resultado seria equivalente.

Seguindo essas idéias, vamos produzir uma nova versão para o algoritmo que resolve o problema da exclusão mútua em uma rede com dois nós. Os domínios 2^{E_1} e 2^{E_2} são idênticos e o predicado que descreve o problema é simétrico, assim os subconjuntos **P**-fechados de E_1 são os mesmos de E_2. Podemos identificá-los da forma a seguir. Para $i, j \in \{1, 2\}$, $i \neq j$:

$$\begin{aligned} \{idle, busy\} &= \mathbf{P}_{ji}(\mathbf{P}_{ij}(\{idle, busy\})) \\ \{idle\} &= \mathbf{P}_{ji}(\mathbf{P}_{ij}(\{idle\})) \end{aligned}$$

A figura 4 mostra o domínio dos subconjuntos **P**-fechados de E_1 e E_2 e o isomorfismo dual entre eles.

A nova versão do algoritmo pode ser vista na figura 5. Usando transformações algébricas simples que aproveitam a natureza simétrica do problema, produzimos um único código para ser executado pelo nó i ($i = 1, 2$).

Observe que a variável que representa o território de possíveis estados do nó i ($i = 1, 2$) assume apenas valores dentro do conjunto de subconjuntos **P**-fechados de E_i. Ou seja, c_i assume apenas os valores $\{idle\}$ ou $\{idle, busy\}$. Como havíamos ressaltado anteriormente, se o território fosse, por exemplo, o valor $\{busy\}$, o nó i estaria restringindo suas possíveis transições sem com isso permitir ao nó j ($j \in \{1, 2\}$, $j \neq i$) qualquer expansão. Isso porque, se $c_i = \{busy\}$, então $e_i = busy$, uma vez que $e_i \in c_i$. Para satisfazer o predicado **P**, teríamos $e_j = idle$ e $c_j = \{idle\}$. A componente e_i do nó i não poderia passar para *idle*, embora as restrições impostas ao nó j

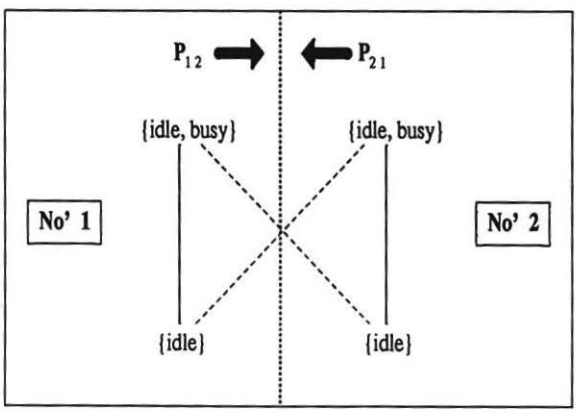

Figura 4: Subconjuntos **P**-fechados para o problema da exclusão mútua.

```
TYPE State = (idle, busy);
  Knowledge = SET OF State;
[ Pi:: e: State; c: Knowledge; hastoken: BOOLEAN;
  e := idle; c := [idle]; hastoken := (i = 1);
  *[TRUE → e := idle;
  | c = [idle,busy] → e := busy;
  | e = idle → c := [idle];
  | receive(token) → c := Pji(token.c);
                hastoken := TRUE;
  | hastoken → token.c := c;
                send(token); hastoken := FALSE;
  ]
]
```

Figura 5: Exclusão Mútua usando conjuntos **P**-fechados.

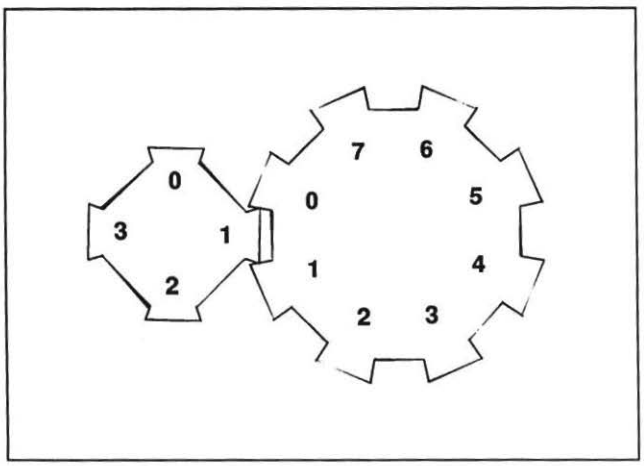

Figura 6: Problema das Engrenagens.

o permitam. A formalização da derivação de estruturas que contornam esse problema é um dos principais resultados obtidos.

Vamos estudar mais um exemplo da utilização dessa teoria, considerando um caso onde os domínios E_1 e E_2 são distintos. O problema que veremos será designado de *O Problema das Engrenagens*, onde as componentes dos nó 1 e 2 podem ser definidas como

$$E_1 = \{0,1,2,3\}, \quad E_2 = \{0,1,2,3,4,5,6,7\}$$

e a especificação pode ser estabelecida por um predicado da maneira a seguir:

$$\mathbf{P} \equiv (e_1 = e_2 \bmod 4) \vee (e_1 = (e_2 + 1) \bmod 4), \qquad e_1 \in E_1, e_2 \in E_2.$$

A figura 6 mostra como essa situação pode ser representada graficamente, sendo que o nó 1 está associado à primeira engrenagem (a menor) e o nó 2, à segunda. Inicialmente, o nó 1 possui um valor fixo representado por "1", enquanto o nó 2 pode assumir os valores "0" ou "1". Observe como a primeira engrenagem é numerada no sentido horário e a segunda, no sentido anti-horário. Quando as componentes tiverem seus valores alterados, as engrenagens irão, como era de se esperar, executar movimentos de rotação em sentidos contrários.

Na figura 7, pode-se ver o domínio parcialmente ordenado formado pelos subconjuntos **P**-fechados de 2^{E_1}. A função $\mathbf{P}_{12}$, aplicada a esses subconjuntos, leva aos seguintes resultados:

$$\begin{array}{ll} \mathbf{P}_{12}(\{\{0\}) = \{7,0,3,4\} & \mathbf{P}_{12}(\{\{1\}) = \{0,1,4,5\} \\ \mathbf{P}_{12}(\{\{2\}) = \{1,2,5,6\} & \mathbf{P}_{12}(\{\{3\}) = \{2,3,6,7\} \\ \mathbf{P}_{12}(\{\{0,1\}) = \{0,4\} & \mathbf{P}_{12}(\{\{1,2\}) = \{1,5\} \\ \mathbf{P}_{12}(\{\{2,3\}) = \{2,6\} & \mathbf{P}_{12}(\{\{0,3\}) = \{3,7\} \end{array}$$

Um algoritmo para resolver esse problema é exibido na figura 8. Algumas modificações foram introduzidas no modelo geral, de modo a simular uma situação real:

- o nó 1 possui uma variável local designada *key* cujo valor determina se a primeira engrenagem irá rodar no sentido horário (-1) ou anti-horário (1);

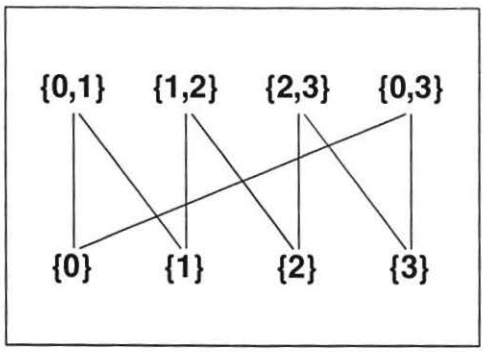

Figura 7: Subconjuntos **P**-fechados de 2^{E_1}, para o problema das engrenagens.

- o nó 1 é interpretado como o *comando*, enviando mensagens que o nó 2 irá interpretar e executar uma rotação no sentido correto (contrário);
- os valores das componentes de cada nó só podem ser alterados de 1 unidade de cada vez; por exemplo, embora $\mathbf{P}_{12}(\{0\}) = \{7, 0, 3, 4\}$, o nó 2 não poderá nunca alterar o valor de sua componente de 0 para 4.
- as funções *INC(x,n)* e *DEC(x,n)* retornam o valor de x incrementado ou decrementado de 1 unidade, no anel de n elementos.

4 Conclusões e Trabalhos Futuros

Utilizando as técnicas apresentadas, exibimos uma derivação formal de estruturas distribuídas para problemas simples como a exclusão mútua e a sincronização de engrenagens. Utilizando uma rede com dois nós, outros problemas estabelecidos através de um predicado sobre uma estrutura distribuída podem ser desenvolvidos de maneira análoga, como por exemplo, o problema dos leitores e escritores. A representação derivada apresenta sempre uma característica importante: uma restrição ao território de possíveis transições de um nó só ocorre com o intuito de permitir uma possível expansão para o território do outro nó.

A generalização natural da teoria é a utilização de uma rede com n nós. Esse estudo está sendo conduzido, com a utilização de um grafo acíclico cobrindo toda a rede. Outra aplicação da teoria é a derivação de protocolos simples para problemas envolvendo um meio de comunicação onde as mensagens podem ser perdidas ou duplicadas, mas a ordem de envio é preservada. Um dos resultados dessa aplicação compreende a derivação formal do protocolo do Alternating Bit ([Schwartz e Melliar-Smith, 1982]), e constitui um trabalho de dissertação de mestrado em fase de conclusão.

Referências

[Andrews e Schneider, 1983] Gregory R. Andrews e Fred B. Schneider. Concepts and Notations for Concurrent Programming. *Computing Surveys*, 15(1):3–43, March 1983.

```
TYPE State = {0,1,2,3};
  Knowledge = SET OF State;
  Rotation = {-1,0,1};
[ P1:: x1: State; c1: Knowledge; key: Rotation; hastoken: BOOLEAN;
  x1 := 0; c1 := {0};
  key := 0; hastoken := TRUE;
  *[
    chave=0 → chave := choose({-1,0,1});
    | key=-1, DEC(x1,4) in c1 → x1 := DEC(x1,4); key := 0;
    | key=1, INC(x1,4) in c1 → x1 := INC(x1,4); key := 0;
    | receive(token) → c1 := token.c; hastoken := TRUE;
    | hastoken, ((key=-1 and not (DEC(x1,4) in c1))
        or (key=1 and not (INC(x1,4) in c1))) →
            token.c := c1; token.k := key; send(token); hastoken := FALSE;
  ]
]
||
[ P2:: x2: State; c2: Knowledge; key: Rotation; hastoken: BOOLEAN;
  x2 := 0; c2 := {0,1};
  key := 0; hastoken := FALSE;
  *[
    key=-1, DEC(x2,8) in P12(c2) → x2 := DEC(x2,8);
            c2 := P21({x2}); key := 0;
    | key=1, INC(x2,8) in P12(c2) → x2 := INC(x2,8);
            c2 := P21({x2}); key := 0;
    | receive(token) → c2 := token.c; key := token.k;
            hastoken := TRUE;
    | key=0, hastoken → token.c := c2;
            send(token); hastoken := FALSE;
  ]
]
```

Figura 8: Um algoritmo para o problema das engrenagens.

[Bal *et al.*, 1989] Henri E. Bal, Jennifer G. Steiner, e Andrew S. Tanenbaum. Programming Languages for Distributed Computing Systems. *ACM Computing Surveys*, 21(3):261–322, September 1989.

[Barbara e Garcia-Molina, 1986] Daniel Barbara e Hector Garcia-Molina. Mutual exclusion in partitioned distributed systems. *Distributed Computing*, 1:119–132, 1986.

[Carvalho e Roucairol, 1983] Osvaldo S. F. Carvalho e Gerard Roucairol. On mutual exclusion in computer networks. *Communications of the ACM*, 26(2), February 1983.

[Chandy e Misra, 1984] K. M. Chandy e J. Misra. The Drinking Philosophers Problem. *ACM Transactios on Programming Languages and Systems*, 6(4):632–646, October 1984.

[Coffman e Denning, 1973] Edward G. Coffman e Peter J. Denning. *Operating Systems Theory.* Prentice-Hall, 1973.

[Dijkstra, 1975] E. W. Dijkstra. Guarded commands, nondeterminancy, and formal derivation of programs. *Communications of the ACM*, 8(18):453–457, August 1975.

[Grimaldi, 1988] Ralph P. Grimaldi. *Discrete and Combinatorial Mathematics.* Addison-Wesley, 1988.

[Hansen, 1973] Per Brinch Hansen. *Operating System Principles.* Prentice-Hall, 1973.

[Hoare, 1969] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 10(12):576–580, October 1969.

[Hoare, 1978] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), August 1978.

[Keller, 1976] R. M. Keller. "Formal Verification of Parallel Programs". *Communications of the ACM*, 19(7), July 1976.

[Lamport, 1978] Leslie Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System". *Communications of the ACM*, 21(7), July 1978.

[Maekawa, 1985] Mamoru Maekawa. "A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized Systems". *ACM Transactions on Computing Systems*, 3(2), May 1985.

[Ore, 1944] Oysten Ore. "Galois Connections". *Transactions of the American Mathematical Society*, 55:493–513, 1944.

[Owick e Gries, 1976] S. Owick e D. Gries. Verifying properties of parallel programs: an axiomatic approach. *Communications of the ACM*, 19(5), May 1976.

[Ricart e Agrawala, 1981] G. Ricart e A. K. Agrawala. "An Optimal Algorithm for Mutual Exclusion in Computer Networks". *Communications of the ACM*, 24(1), January 1981.

[Schwartz e Melliar-Smith, 1982] Richard L. Schwartz e P. Michael Melliar-Smith. "From State Machines to Temporal Logic: Specification Methods for Protocol Standars". *IEEE Transactions on Communications*, 30(12):2486–2496, December 1982.

[Shaw, 1974] Alan C. Shaw. *The Logical Design of Operating Systems.* Prentice-Hall, 1974.

[Szasz, 1963] G. Szasz. *Introduction to Lattice Theory.* Academic Press, 1963.

[Tanenbaum, 1992] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall, 1992.