A LEGALIDADE DE COMPUTAÇÕES SOBRE MEMÓRIA COMPARTILHADA

Ayru Leal de Oliveira Filho

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Edil Severiano Tavares Fernandes, Ph.D.

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Mario Roberto Folhadela Benevides, Ph.D.

Prof. Lúcia Maria de A. Drummond, D.Sc.

Prof. Noemi de La Rocque Rodriguez, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2000

OLIVEIRA FILHO, AYRU LEAL

A legalidade de Computações sobre Memória Compartilhada [Rio de Janeiro] 2000

X, 103 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2000)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Sistemas Operacionais

2 - Memória Compartilhada

I. COPPE/UFRJ II. Título (série)

A meus pais, Ayru e Therezinha.

A Maria, Pedro e Julia.

Agradecimentos

Inicialmente, gostaria de agradecer ao Prof. Valmir Carneiro Barbosa, meu orientador neste trabalho, pelas incontáveis manhãs de sexta-feira que dispensou na condução desta tese. Sem dúvida foi um privilégio trabalhar ao seu lado e desfrutar da sua experiência, conhecimento e amizade por estes anos.

Gostaria de agradecer também ao Prof. Claudio Luis Amorim por ter me acolhido como seu aluno num momento de difícil de troca de orientador no início deste trabalho e também pela orientação em uma das qualificações que resultaram nesta tese.

Ao Prof. Edil Severiano Tavares Fernandes por ter aceitado presidir a banca desta tese na ausência de meu orientador.

Ao CEPEL - Centro de Pesquisas de Energia Elétrica, em especial aos pesquisadores Dr. Nelson Martins, Dr. Luis A. S. Pilotto, Dr. Antônio Ricardo C. D. Carvalho e Raul B. Sollero, bem como a toda sua diretoria por terem viabilizado a conclusão deste trabalho.

Agradeço também ao CNPq pelo suporte financeiro no início deste trabalho.

A toda equipe do SAGE — Sistema Aberto de Gerenciamento de Energia pelo apoio e cobertura nas minhas ausências para me dedicar a este trabalho.

As minhas colegas da COPPE Clícia, Cristiana e Carla que, apesar do pouco convívio, sempre incentivaram e sempre se colocaram a disposição para qualquer ajuda no sentido de concluir esta tese.

A toda minha família, em especial a meus pais pelo apoio e incentivo constantes.

Em especial, quero agradecer a minha esposa, Maria, pelo constante apoio e incentivo, por suportar os inevitáveis momentos de estresse, por não suportar mais ver grafos de execuções e por fazer tudo isso valer a pena.

Aos meus filhos Pedro e Julia que, mesmo sem saberem, renovam minha vontade de novas conquistas a cada sorriso.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc)

A Legalidade de Computações sobre Memória Compartilhada

Ayru Leal de Oliveira Filho

Março/2000

Orientador: Valmir Carneiro Barbosa

Programa: Engenharia de Sistemas e Computação

Neste trabalho, investigamos a consistência de computações sobre memória compartilhada do ponto de vista do conceito de *legalidade*. A modelagem formal de tais sistemas é fundamental para o correto entendimento das propriedades de cada modelo de consistência. Como uma primeira contribuição, propomos um novo modelo de dois níveis (eventos e operações) que não apresenta características indesejáveis como dependência de arquitetura, uso de referência global de tempo e uso de conjuntos totalmente ordenados de eventos ou operações. Em ambos os níveis, uma execução é vista como um conjunto parcialmente ordenado. Definida desta forma, uma execução pode ser representada por um grafo E–OU e propriedades relativas a condições de consistência podem ser derivadas das estruturas que surgem na representação gráfica. Em particular, a existência de *b–knots* e estruturas semelhantes está fortemente ligada à legalidade da execução a este grafo associada. Como uma segunda contribuição, definimos novas estruturas em grafos E–OU, *c–subgrafos* e *k–componentes* e estabelecemos as relações entre estas estruturas e outras já conhecidas (knots, b–knots e ciclos).

Utilizando os resultados obtidos pela teoria de grafos, apresentamos uma nova definição de legalidade que, em contraste com a definição original sobre uma seqüência de operações, é estabelecida sobre uma ordem parcial de operações. Utilizamos esta nova definição na apresentação formal de vários modelos de consistência. Concluímos que o novo modelo formal para computações sobre memória compartilhada e a teoria de legalidade aqui introduzidos podem ser utilizados para a análise formal de virtualmente todos os modelos de consistência, bem como para a proposição de novos modelos, podendo levar a implementações mais eficientes, uma vez que restrições de ordem entre operações não são impostas além daquelas realmente necessárias a cada modelo.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

The Legality of Shared–Memory Computations

Ayru Leal de Oliveira Filho

March, 2000

Advisor: Valmir Carneiro Barbosa

Department: Computing and Systems Engineering

In this thesis we investigate the consistency of shared–memory computations from the standpoint of legality. Modeling such systems formally is a fundamental step towards the correct understanding of each model's properties. As the first main contribution of this thesis, we propose a new, two-level (events and operations) formal framework without the undesirable characteristics of architecture dependence, use of global time, or use of totally ordered sets of events or operations. In both levels an execution is represented by a partially ordered set, which gives rise to an AND-OR graph representation from whose structure several properties related to consistency conditions can be deduced. In particular, the existence of b-knots and similar structures is strongly related to the legality of the execution. A second important contribution of this thesis is the introduction of new AND-OR graph structures, such as c-subgraphs, k-components, and the relationship between such structures and knots, b-knots, and cycles.

Our graph-theoretic results are then used to establish a new definition of legality, which, in contrast with the original definition over a sequence of operations, is established over a partially ordered set of operations. We show how this new definition can be used to formally characterize several of the existing consistency models. We conclude that the new formal framework for computations over shared memory and the legality theory introduced in this thesis are suitable to the formal analysis of virtually all consistency models, as well as to the definition of new ones. They may also lead to more efficient implementations, since no ordering constraints other than the ones required by each specific model are imposed by the framework itself.

Sumário

Lista de Figuras

Capítulo 1

Introdução

1.1 Motivação

Na última década, os sistemas para processamento paralelo evoluiram de arquiteturas baseadas em memória compartilhada centralizada para sistemas suportados por um conjunto distribuído de módulos de memória [28]. O motivo básico desta mudança foi a chamada escalabilidade, ou seja, devido a contenções impostas por elementos centralizados, o custo de comunicação entre processadores e a memória tornou-se inviável para arquiteturas com grande número de processadores e uma memória centralizada. Os sistemas baseados em memórias distribuídas suportam, de maneira aceitável, um grande número de processadores, uma vez que o acesso à memória passa a ser executado também concorrentemente. Apesar disso, o acesso à memória distribuída continua sendo um ponto crítico para o desempenho de máquinas paralelas.

Uma questão básica que surge é de como programar tais sistemas com memórias distribuídas, uma vez que as operações sobre a memória compartilhada não mais podem serem consideradas pelo programador com se fossem executadas atomicamente em um elemento central do sistema. A solução mais direta é, sem dúvida, a troca de mensagens entre processos. Neste modelo de programação, primitivas de comu-

nicação e sincronização são inseridas no código da aplicação de modo a implementar a interação entre os processos componentes da aplicação. Construir um programa paralelo eficiente, em geral, é uma tarefa complexa e a necessidade de lidar com os aspectos específicos da comunicação e sincronização aumenta esta complexidade. Por esse motivo, existe a tendência de fornecer ao programador uma interface com o sistema de memória que seja semelhante a do modelo centralizado, ou seja, na qual o programador tem acesso a todo o espaço de endereçamento e a comunicação se dá através de escritas e leituras em localizações compartilhadas de memória.

Apesar destas dificuldades na programação de algoritmos paralelos não ser um "privilégio" do modelo de programação por troca de mensagens, muitos defendem que um modelo de programação baseado em memória compartilhada torna mais simples a tarefa de programar um sistema paralelo [2, 72, 1]. Desta forma, vários esforços na direção de se construir sistemas que, apesar de implementados sobre uma arquitetura de memória distribuída, ofereçam uma interface que seja, senão idêntica, mas pelo menos próxima de um sistema baseado em memória compartilhada.

Estes sistemas são atualmente denominados sistemas de *memória compartilhada distribuída* (ou DSM — *Distributed Shared Memory*). Uma das principais questões no projeto de um sistema DSM é a de como oferecer ao programador uma interface (no sentido de como ele percebe as modificações que ocorrem nas localizações compartilhadas) com o sistema de memória que seja próxima à interface de um sistema com memória centralizada e que possa ser implementada de maneira eficiente.

Esta interface com o sistema de memória exibida ao programador é definida pelo modelo de consistência (ou condições de consistência) implementado pelo software e/ou hardware da arquitetura paralela. Como os acessos a localizações compartilhadas passam a concorrentes, este modelo define conceitualmente como estes acessos se

relacionam do ponto de vista do programador. O modelo de consistência não define, necessariamente, como os acessos à memória irão ocorrer, apesar de alguns modelos serem definidos nestes termos (Capítulo 5). Desta forma, o modelo de consistência define, indiretamente, quais os resultados que podem ser considerados corretos para um determinado programa paralelo. Por outro lado, o modelo também influencia a implementação do hardware e/ou software que dá suporte à abstração de memória compartilhada sobre memórias distribuídas.

O grande objetivo é, portanto, definir um modelo de consistência conceitualmente simples do ponto de vista do programador e que permita uma implementação eficiente. Buscando este objetivo, diversos modelos de consistência [56, 30, 63, 42, 39, 15, 36] e protocolos para implementação destes foram propostos. Os modelos mais relevantes serão vistos em detalhe nos próximos capítulos.

Para se ter uma noção mais intuitiva do conceito de modelo de consistência, vamos apresentar informalmente o modelo de Consistência Seqüencial. Utilizaremos a seguinte notação para representarmos operações sobre a memória compartilhada. Uma operação de leitura de uma localização x e que retorna o valor v será representada por $r(x)v$. Analogamente, uma operação de escrita de um valor v em uma localização compartilhada x será representada por $w(x)v$. Na definição de Lamport [56] é exigido que todas as execuções de programas em uma máquina paralela apresentem os mesmos resultados que apresentariam se suas instruções fossem executadas em (alguma) seqüência, a qual mantém a ordem especificada pelos respectivos programas. Veja o exemplo na Figura 1.1. Se uma execução exibe algum dos três primeiros conjuntos de valores para as leituras das localizações compartilhadas x e y (i.e. $(0,0)$, $(0,1)$ ou $(1,1)$), então ela pode ser classificada como seqüencialmente consistente, pois é possível supor uma seqüência para as operações de $P1$ e $P2$ que resulta em um destes conjuntos de valores, sem alterar a ordem especificada nos

respectivos programas. O mesmo não é possível para o resultado $(1, 0)$.

Na Figura 1.1, o resultado $(1, 0)$ não é aceitável porque toda seqüência de operações que mantenha a ordem especificada nos programas fere a semântica usual para operações de leitura e escrita. Por exemplo, a seqüência

$$w(y)0 \rightarrow w(x)0 \rightarrow w(y)1 \rightarrow w(x)1 \rightarrow r(x)1 \rightarrow r(y)0$$

não é aceitável porque a última leitura na seqüência retorna um valor antigo, já sobreposto por outro na mesma seqüência.

P_1	P_2	(x, y)
$w(y)0$	$r(x)$	$(0, 0)$
$w(x)0$	$r(y)$	$(0, 1)$
$w(y)1$		$(1, 1)$
$w(x)1$		$(1, 0)$

Figura 1.1: Trecho de programa e possíveis resultados.

A noção de que seqüências de operações podem ser consideradas corretas para um determinado tipo de objeto foi formalizada por Herlihy e Wing [45], sendo denominada *legalidade*. A legalidade é baseada na definição do conjunto de possíveis estados que um tipo de objeto pode assumir e no conjunto de possíveis seqüências de operações válidas para esse tipo de objeto. Para a descrição de condições de consistência em memória compartilhada, será suficiente considerarmos como objetos apenas as localizações compartilhadas de memória. Para este tipo de objeto, as seqüências válidas de operações são todas aquelas onde as operações de leitura não retornam valores sobrepostos como no exemplo anterior. Portanto, podemos dizer que o resultado $(1, 0)$ não é aceitável porque não existe seqüência *legal* daquelas operações, que mantenha a ordem especificada nos programas e na qual as operações de leitura retornem 1 e 0, respectivamente, para as localizações x e y.

A exigência de legalidade está presente em praticamente todos (senão realmente todos) os modelos de consistência propostos, ainda que implicitamente ou apenas para um subconjunto de operações. Entretanto, todas estas abordagens utilizam o conceito original, o qual é estabelecido sobre uma ordem total de operações. Neste trabalho, introduziremos duas novas definições de legalidade, ambas estabelecidas sobre ordens parciais que representam uma execução. A motivação para tal abordagem está no fato de que, ao utilizarmos somente ordens parciais, evitamos impor restrições de ordem além das realmente necessárias para garantir a legalidade e, conseqüentemente os modelos de consistência que exigem legalidade. Desta forma, é possível que protocolos de consistência mais eficientes possam ser projetados se baseados nestas novas definições.

Para dar suporte a estas novas definições de legalidade e demais condições de consistência, definimos também uma nova formalização para computações sobre memória compartilhada. Esta formalização é baseada somente nos eventos próprios de sistemas distribuídos assíncronos, ordenados parcialmente em função da implementação (hardware e/ou software) do sistema de memória. Desta forma, ficamos livres dos detalhes de cada implementação e podemos estabelecer definições que serão válidas para uma grande variedade de arquiteturas.

As formalizações existentes não nos foram suficientes porque, em geral, não são genéricas o bastante se baseando, por exemplo, na existência de um relógio global ou em detalhes específicos de arquitetura ou ainda em ordens totais de eventos. No Capítulo 2 as principais propostas serão comparadas com nossa proposta que busca evitar tais particularizações.

1.2 Tabela de Notações

Para facilitar a referência às simbologias adotadas, esta seção apresenta uma tabela contendo todas as notações utilizadas neste trabalho.

Notações	
$w(x)v$	Escrita do valor v na localização compartilhada x.
$r(x)v$	Leitura do valor v da localização compartilhada x.
$s[op]$, $e[op]$	Eventos de início e término de uma operação, respectivamente.
$P(x)$	Conjunto de processadores para os quais uma atualização da localização x deva ser publicada.
$p_i[w(x)v]$	Evento de publicação do valor v no processador i.
$b[r(x)v]$	Evento de busca que ocorre em um processador pertencente a $P(x)$.
$F_i(x)$	processador (fonte) no qual ocorrem os eventos de busca da localização x para as leituras iniciadas no processador i.
$\prec$	Relação de ordem parcial entre eventos $s[op]$ e $e[op]$.
$\prec_p$	Relação de ordem parcial entre eventos internos ao sistema de memória p e b.
$\xrightarrow{mi}$	Relação "pode influenciar" entre uma escrita e uma leitura relativas a um mesmo valor e a uma mesma localização de memória.
$\xrightarrow{xo}$	Relação "ordem de execução" entre uma leitura e outra operação do mesmo processo.
$\xrightarrow{po}$	Relação "ordem de programa" definida pelo código do programa de cada processo.
$\xrightarrow{\sigma}$	Relação de ordem parcial entre operações e que representa uma execução. Definida como $[\xrightarrow{mi} \cup \xrightarrow{xo}]^+$.
$\xrightarrow{lc}$	Relação "restrição de legalidade" entre uma leitura e uma escrita relativas à mesma localização e a valores diferentes.
$\xrightarrow{\hat{\sigma}}$	Relação de ordem parcial entre operações e que representa uma execução estendida pela restrição de legalidade. Definida como $[\xrightarrow{mi} \cup \xrightarrow{xo} \cup \xrightarrow{lc}]^+$.
G, $\hat{G}$	Grafos induzidos por $\xrightarrow{\sigma}$ e $\xrightarrow{\hat{\sigma}}$, respectivamente.

Tabela 1.1: Tabela de Notações

1.3 Contribuições

As principais contribuições deste trabalho são brevemente descritas a seguir:

1. Um novo modelo formal para a representação de sistemas concorrentes baseados em memória compartilhada. Neste modelo não fazemos suposições sobre a arquitetura, sendo este genérico o bastante para ser capaz de representar um grande número de modelos de consistência;

2. Uma abordagem baseada em teoria de grafos para a expressão de condições de consistência de execuções concorrentes, com a introdução de duas novas estruturas de grafos, aqui denominadas *c-subgrafos* e *k–componentes*, e prova de importantes relações entre estas novas estruturas e outras já conhecidas da teoria de grafos (ciclos, knots e b-knots);

3. Uma teoria sobre legalidade estabelecia utilizando-se a modelagem de uma execução utilizando o formalismo proposto. Como resultado, duas novas definições para o conceito de legalidade são introduzidas;

4. Novas definições formais para as principais condições de consistência utilizando a modelagem aqui introduzida e o novo conceito de legalidade.

1.4 Organização da Tese

Após esta introdução, iniciamos no Capítulo 2 com a descrição do modelo básico proposto para sistemas concorrentes que utilizam memória compartilhada para a interação entre processos. Neste capítulo, também iremos relacionar nossa abordagem com os principais trabalhos já descritos na bibliografia e que estão próximos dos nossos objetivos. No Capítulo 3, alguns aspectos da teoria de grafos relacionados

com a expressão de condições de consistência serão apresentados. Neste capítulo, serão introduzidos novos conceitos e provadas algumas relações entre estruturas de grafos E–OU que servirão de suporte para os conceitos introduzidos nos capítulos seguintes. No Capítulo 4, a definição de *legalidade* é revista e introduzimos duas novas definições estabelecidas sobre a ordem parcial que representa execuções. O Capítulo 5 apresenta a definição formal de alguns dos principais modelos de consistência propostos, utilizando a modelagem desenvolvida. Seguimos com as conclusões e trabalhos futuros no Capítulo 6.

Capítulo 2

Modelagem de Sistemas em Memória Compartilhada

Neste capítulo, apresentamos o modelo formal básico para representar operações e execuções em memória compartilhada. Este modelo é baseado na representação dos eventos de uma computação distribuída assíncrona. Sobre este modelo serão definidas todas as demais formalizações apresentadas no decorrer deste trabalho.

Nosso interesse em definir um novo modelo é construir uma ferramenta para analisarmos condições de consistência de memória que seja geral o bastante para representar diversas arquiteturas paralelas. O modelo aqui proposto é composto por dois níveis sendo que o mais baixo deles, baseado nos evento de uma computação distribuída, tem o objetivo justamente de não expor os detalhes de arquiteturas específicas. Neste nível, o modelo se baseia em um conjunto parcialmente ordenado de eventos, sendo estes o início e o fim das operações de leitura e escrita de uma dada execução. Estes eventos se relacionam entre si através da ordem parcial de um modo totalmente dependente dos detalhes da arquitetura. Portanto, no restante deste trabalho, não consideraremos os detalhes de como este conjunto parcialmente ordenado é gerado.

Em um nível mais alto, consideraremos as operações de leitura e escrita sobre

localizações da memória compartilhada. Tais operações também serão ordenadas parcialmente, através de uma composição de relações derivadas das relações entre eventos. Na seção 2.3, relacionaremos o modelo aqui descrito com as principais abordagens já propostas.

Consideraremos um sistema multiprocessado no qual a única forma de comunicação é através da memória compartilhada. Assumiremos que cada processador executa apenas um processo (relativo à aplicação) durante uma execução paralela e, por simplicidade, que os valores iniciais de cada localização de memória resultam de operações de escrita emitidas pelos processos (ainda que artificiais). Cada processo envia pedidos de execução de operações sobre a memória compartilhada ao sistema de memória e recebe deste eventos de retorno com diferentes significados de acordo com a operação solicitada.

Ao longo deste trabalho, estaremos considerando como operações sobre a memória compartilhada apenas leituras e escritas sendo o único objeto compartilhado as localizações de memória.

2.1 Eventos de uma execução

Nesta seção, descreveremos os eventos que estaremos considerando para modelar uma execução. Também vamos discutir como estes eventos devem ser interpretados e as relações de precedência entre eles.

Uma operação de leitura sobre uma localização de memória compartilhada x que retorna o valor v é representada por $r(x)v$. Uma leitura pode ser decomposta nos eventos de início e término, representados por $s[r(x)v]$ e $e[r(x)v]$, respectivamente. O evento de início de uma leitura está associado à emissão, por parte do processo, de um pedido de execução daquela operação, o qual é enviado ao sistema de memória.

O evento de término, por sua vez, denota que o valor v foi recebido do sistema de memória e que nenhum evento pode alterá-lo, estando a operação concluída.

Da mesma forma, uma operação de escrita do valor v sobre uma localização de memória compartilhada x é representada por $w(x)v$. O evento $s[w(x)v]$ tem o mesmo significado atribuído a operações de leitura, ou seja, o envio de um pedido de execução da operação ao sistema de memória. Já o evento de término de escrita, $e[w(x)v]$, possui uma semântica própria. Como estamos admitindo que a implementação do sistema de memória pode ser distribuída, o recebimento por parte de um processos de um evento de término de uma escrita não significa, necessariamente, que todos os eventos internos ao sistema de memória conseqüentes de $s[w(x)v]$ já tenham sido concluídos. Portanto, a ocorrência de $e[w(x)v]$ indica apenas que o sistema de memória recebeu o pedido enviado a ele e que o processo de atualização da localização x foi iniciado. A Figura 2.1 ilustra a interação entre a aplicação e o sistema de memória, mostrando também o fluxo de eventos entre eles.

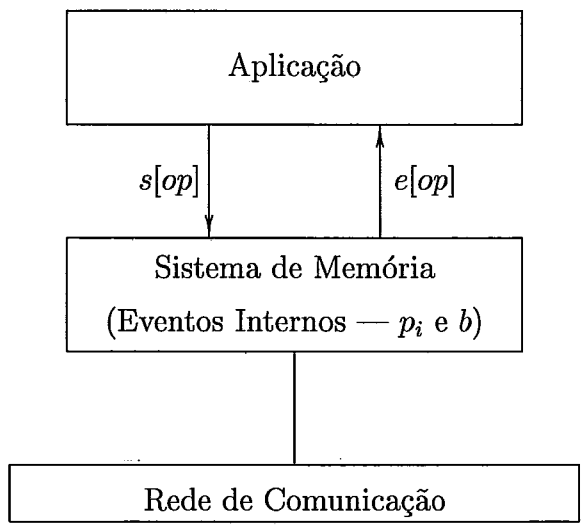


Figura 2.1: Módulos e eventos do modelo básico.

O conceito de término de uma operação (de escrita ou leitura) tem sido usado na literatura para definir características de modelos de consistência [30, 29, 39]

(veja Seção 2.3). Entretanto, o real significado de término de uma operação está fortemente associado à implementação do sistema de memória que se tem em mente. Isto conduziu a especificações imprecisas e voltadas para implementações específicas. Especificações formais mais rigorosas tratam deste problema modelando os efeitos de uma operação de escrita em cada elemento do sistema de memória (e.g. *caches* ou cópias locais de páginas de memória compartilhada) [37] ou ainda através da especificação de condições de consistência baseadas em como as operações são ordenadas, não descendo a detalhes de como elas são executadas pelo sistema de memória [15, 72, 67].

Nossa abordagem segue esta última linha, definindo condições de consistência baseadas em ordens parciais entre operações. Entretanto, para definir estas ordenações de maneira precisa, recorremos a uma modelagem em um nível mais baixo, onde tratamos das precedências entre eventos relacionados com as operações de leitura e escrita. Apesar da nossa proposta para modelagem de execuções e condições de consistência não necessitar da descrição dos eventos internos ao sistema de memória, a modelagem destes será útil para a análise de condições de consistência propostas baseadas na noção de término de operações. Portanto, na seção 2.1.1 apresentamos a descrição detalhada de alguns eventos internos e suas relações.

Os eventos de início (s) e término (e) de operações estão relacionados por uma ordem parcial denotada por $\prec$, semelhante à relação usual "aconteceu-antes" definida em sistemas distribuídos [17, 55]. Conforme já mencionamos, a definição da ordem parcial $\prec$ é totalmente dependente da arquitetura e do protocolo de acesso à memória implementado. Note que os eventos relacionados por $\prec$ são externos ao sistema de memória e acontecem no escopo do processo em execução em determinado processador. Uma execução, portanto, pode ser representada como um conjunto parcialmente ordenado de todos os eventos de início e término de operações de leitura

e escritas.

Apesar de $\prec$ ser uma ordem parcial, quando restrita aos eventos de um mesmo processo, estes são totalmente ordenados. Apesar disso, é possível que estas seqüências não alternem entre os eventos de início e fim de cada operação. Isto significa que o modelo aqui apresentado permite que mais de uma operação de um mesmo processo estejam em andamento ao mesmo tempo e ainda que uma operação iniciada após outra, ambas relativas a um mesmo processo, termine (o evento de término acontece) antes da primeira, caracterizando uma execução fora da ordem de início das operações. Esta característica permite a modelagem de otimizações comuns em arquiteturas paralelas tais como a emissão de múltiplas instruções, o uso de leituras não bloqueantes e a execução especulativa (recentemente, o conjunto destas técnicas, entre outras, tem sido denominado *Instruction Level Parallelism – ILP* [68, 7]).

A Figura 2.2, a seguir, mostra uma execução representada pela ordem parcial entre eventos. As linhas tracejadas representam as linhas de tempo relativas a cada processador. Note que dois eventos ocorridos em processadores diferentes são incomparáveis, a menos que explicitamente relacionados, uma vez que não existe uma referência única de tempo. Repare também que os eventos de início das operações de leitura não têm valor associado. Isso se deve ao fato de apenas os eventos de término de leitura definirem o valor retornado por esta. Nesta figura, não estão representadas as relações devidas à transitividade.

2.1.1 Eventos Internos ao Sistema de Memória

Embora a modelagem das propriedades de execuções ao longo deste trabalho utilizem as relações entre operações e não entre eventos, para refletirmos neste modelo

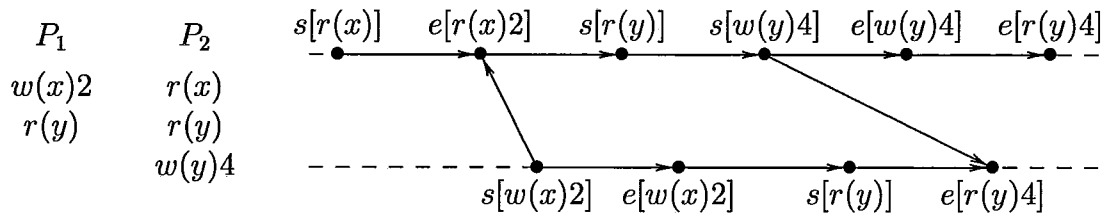


Figura 2.2: Exemplo de relacionamentos em $\prec$.

algumas condições de consistência conhecidas, será necessário expor uma modelagem mínima dos eventos internos ao sistema de memória. Também neste nível, optamos por um modelo genérico, onde não faremos suposições acerca da arquitetura e da implementação.

Seja $P(x)$ o conjunto de processadores para os quais uma atualização da localização x deva ser publicada (ou seja, aqueles processadores que possuírem cópia de x, sejam em cache ou na memória local). Se $w(x)v$ é uma operação de escrita do valor v na localização x iniciada em qualquer processador, então para cada elemento de $P(x)$, existe um evento $p_i[w(x)v]$ indicando a *publicação* de v no processador i. Note que $P(x)$ está fortemente ligado à implementação e à arquitetura do sistema.

De maneira análoga, a cada operação de leitura está associado um evento interno ao sistema de memória que denominaremos de *busca*. Dada uma localização de memória x, para cada operação de leitura de x, existe um evento $b[r(x)v]$ que ocorre em um processador pertencente a $P(x)$. A associação entre o processador onde uma leitura é iniciada e o processador pertencente a $P(x)$ para cada localização compartilhada também é determinada pela implementação. Vamos denotar por $F_i(x)$ (fonte) o processador no qual os eventos de busca da localização x ocorrem para as leituras iniciadas no processador i [1].

Para modelarmos a relação entre eventos internos e estes aos eventos externos

[1]Em geral, $F_i(x) = i$, mas, para sermos genéricos, consideraremos que $F_i(x) = j, j \in P(x)$. Veja Figura 2.3.

(s, e) ao sistema de memória, definimos, analogamente à $\prec$, outra relação de ordem parcial denotada por $\prec_p$. Note que $\prec_p$ ordena parcialmente todos os eventos do sistema de memória, sejam estes internos ou externos.

Baseados em $\prec_p$ podemos definir algumas relações entre eventos:

(a) $s[w(x)v] \prec_p p_i[w(x)v], \forall i \in P(x)$,

(b) $s[r(x)v] \prec_p b[r(x)v] \prec_p e[r(x)v]$

Ou seja, $\prec_p$ ordena o evento de início de uma escrita como anterior a todos os eventos p relativos a esta escrita. Da mesma forma, o evento de início de uma leitura precede seu respectivo evento de busca. $\prec_p$ ainda ordena os eventos de publicação e busca ocorridos em um mesmo processador. A Figura 2.3 ilustra as relações entre eventos internos e externos para uma localização compartilhada x onde $P(x) = \{1, 3, 4\}$ e $F_5(x) = 4$.

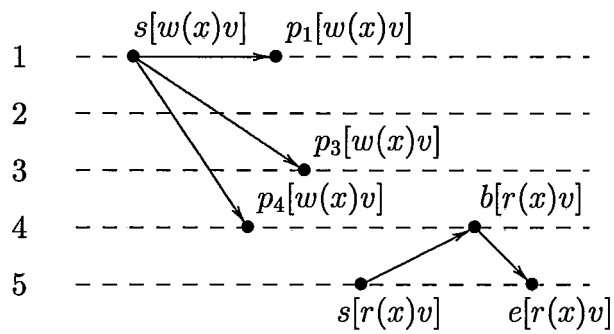


Figura 2.3: Exemplo de relacionamentos em $\prec_p$.

Informalmente, a ordem de ocorrência dos eventos de publicação e busca é que de fato irá definir as escritas que podem influenciar o retorno de determinada leitura. Esta noção será formalizada na seção 2.2.

Note que ambas as relações $\prec$ e $\prec_p$ são determinadas pela implementação e que $\prec_p$ inclui $\prec$. Enquanto $\prec$ será utilizada para definir as relações entre operações em outro nível de nossa formalização, $\prec_p$ será útil para restringir $\prec$ e, conseqüentemente as relações definidas sobre $\prec$, se desejarmos representar condições suficientes para que uma implementação satisfaça determinada condição de consistência.

2.2 Modelando Operações

Conforme mencionamos anteriormente, no modelo aqui proposto, uma execução pode ser representada em dois níveis distintos. Na seção anterior, vimos que uma execução pode ser representada através de uma ordem parcial dos eventos que a compõem. Nesta seção, vamos definir as relações entre operações que irão compor uma execução. As definições de condições de consistência apresentadas neste trabalho, em especial a condição de *legalidade*, serão definidas sobre estas relações entre operações. Desta forma, evitamos a utilização das precedências entre eventos, que são muito dependentes da arquitetura e das características de implementação.

Baseados na relação $\prec$, que ordena parcialmente os eventos de início e término de operações, podemos agora definir algumas relações sobre as operações de uma execução.

1. *Pode Influenciar* ($\xrightarrow{mi}$): Esta relação é definida somente entre escritas e leituras de uma mesma localização e de um mesmo valor. Ela está fortemente associada à relação "pode causalmente afetar" definida por Lamport [57], ou seja, $\xrightarrow{mi}$ indica quais as escritas que podem ter influenciado o retorno de uma determinada leitura. Desta forma, poderemos tratar a existência de múltiplas escritas concorrentes de um mesmo valor para uma mesma locali-

zação de memória. Portanto,

$$w(x)v \xrightarrow{mi} r(x)v \text{ se e somente se } s[w(x)v] \prec e[r(x)v].$$

2. *Ordem de Execução* ($\xrightarrow{xo}$): Esta relação ordena, também parcialmente, as operações iniciadas por um mesmo processo. Se op e op' são duas destas operações, sendo que op é uma operação de leitura ($r(x)v$), então tanto no caso de $op' = r(y)u$ ou $op' = w(y)u$ teremos:

$$op \xrightarrow{xo} op' \text{ se e somente se } e[op] \prec e[op'].$$

Neste ponto, é importante observar que, propositalmente, não estamos considerando o caso em que op e op' são, respectivamente, operações de escrita e leitura. Isto se deve ao fato de somente o evento de término de uma leitura ser bem definido, indicando que o valor lido foi retornado pelo sistema de memória e que a operação está completamente terminada, ou seja, todos os eventos internos ao sistema de memória já aconteceram. Entretanto, para uma operação de escrita, o evento de término, conforme estamos considerando neste trabalho, não tem relação com o efetivo término de todos os eventos causados pela emissão desta escrita (eventos internos de publicação).

A intuição que podemos ter relativa ao término de uma escrita vem, na verdade, dos eventos de término de leituras relacionadas a esta escrita através de $\xrightarrow{mi}$. Mais especificamente, o evento de término de uma leitura $r(x)v$ emitida por determinado processo indica que o evento de publicação de pelo menos uma das operações $w(x)v$ relacionadas à $r(x)v$ por $\xrightarrow{mi}$, já ocorreu em $F_i(x)$, indicando o término de $w(x)v$ especificamente para este processo.

Apesar de não ser definida sobre $\prec$ ou $\prec_p$, é importante formalizar também a noção de ordem de programa, uma vez que a maioria das condições de consistência

exigem que a ordem de programa seja respeitada pelo menos para um subconjunto das operações. Outras relações de interesse entre operações também serão definidas no decorrer deste trabalho, mas serão apresentadas oportunamente quando necessitarmos de novas relações para as definições de legalidade no Capítulo 4.

Ordem de Programa ($\xrightarrow{po}$): Da mesma forma que $\xrightarrow{xo}$, esta relação é definida apenas entre operações emitidas por um mesmo processo. Entretanto $\xrightarrow{po}$ ordena totalmente tais operações. Especificamente, para op e op' temos que $op \xrightarrow{po} op'$ se e somente se op precede op' no código do processo a que elas pertencem.

Podemos agora definir o que consideraremos como uma execução neste modelo. A definição de execução é dada sobre um conjunto de operações (leituras e escritas) parcialmente ordenadas pela fecho transitivo de todos os pares de operações relacionadas por $\xrightarrow{mi}$ ou $\xrightarrow{xo}$, ou seja, $[\xrightarrow{mi} \cup \xrightarrow{xo}]^+$. Denotaremos por Ω o conjunto de operações de uma execução e por $\xrightarrow{\sigma}$ a relação $[\xrightarrow{mi} \cup \xrightarrow{xo}]^+$. Portanto, representaremos uma execução pelo conjunto parcialmente ordenado $(\Omega, \xrightarrow{\sigma})$.

No decorrer deste trabalho, utilizaremos intensamente a representação de execuções através de grafos orientados. Seja $G = (N, E)$ um grafo orientado utilizado para representar uma execução. G tem como conjunto de vértices o conjunto de operações da execução e como arestas os pares de operações dados por $\xrightarrow{mi}$ e pela redução transitiva de $\xrightarrow{xo}$. Na Figura 2.4, cada grafo representa uma possível execução do respectivo trecho de programa. No primeiro grafo, nenhum par de operações é relacionado por $\xrightarrow{mi}$, uma vez que nenhuma leitura retorna o valor escrito pela única operação de escrita na execução. No segundo exemplo, apesar de $r(b) \xrightarrow{po} w(b)5$, a execução não exibe relação entre $r(b)5$ e $w(b)5$ por $\xrightarrow{xo}$. Supondo que $s[w(b)5] \prec e[r(b)]$ e que $r(b)$ retorna o valor 5, então $w(b)5 \xrightarrow{mi} r(b)5$.

Os exemplos da Figura 2.4 têm também o intuito de mostrar que a ordem entre

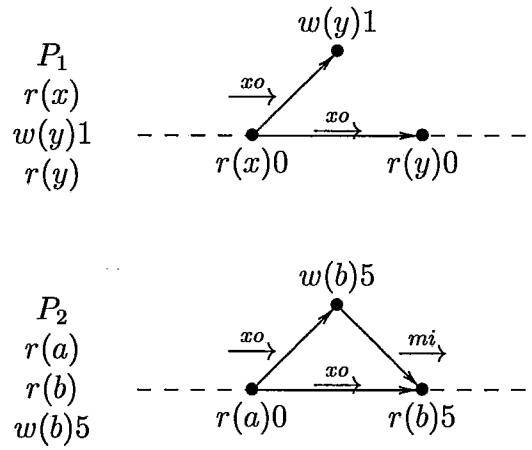


Figura 2.4: Execução representada por grafo orientado.

as operações que é exibida no código (ordem de programa) pode não ser respeitada dependendo das condições impostas pelo modelo de consistência de memória adotado. Conforme veremos no Capítulo 5, diferentes modelos impõem diferentes exigências quanto à preservação da ordem de programa.

Considere agora o exemplo da Figura 2.2. Naquela figura, são mostradas as precedências entre eventos de uma execução do respectivo programa. Na Figura 2.5, mostramos as relações entre operações conseqüentes das referidas precedências entre eventos. Note que a operação de escrita $w(x)2$ não precede $r(y)4$. Isto acontece porque, apesar de $e[w(x)2]$ preceder $e[r(y)4]$ em $\prec$, $\xrightarrow{xo}$ não é definida para pares escrita-leitura.

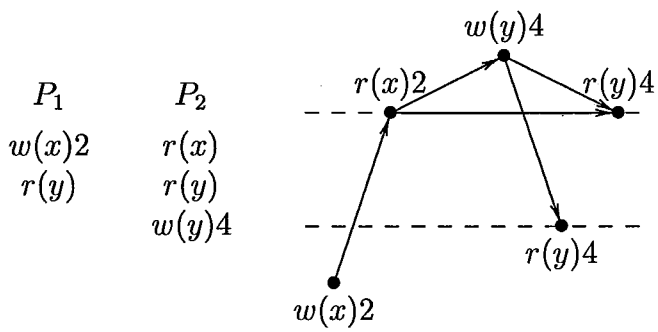


Figura 2.5: Execução representada por grafo orientado.

2.3 Trabalhos Relacionados

Muitos modelos formais para sistemas concorrentes baseados em memória compartilhada foram propostos, em geral como parte especificações de modelos e protocolos de consistência [15, 72, 37, 12, 67, 1, 5, 41, 45, 30]. Nesta seção, iremos comparar alguns destes trabalhos com o modelo aqui apresentado.

Um problema básico na definição formal de condições de consistência é a modelagem de operações não atômicas. Se admitimos que o sistema de memória permite a existência de múltiplas cópias de uma mesma localização de memória, então uma escrita deve ser tratada de modo não atômico. Dubois, Scheurich e Briggs [30] propuseram as noções "executada com relação a um processo", "executada" e "globalmente executada", que transcrevemos a seguir (com pequenas alterações de terminologia), para lidar com a não atomicidade das operações numa arquitetura de memória distribuída.

> [...]*uma leitura é considerada executada (terminada) com relação ao processador k no instante em que a emissão de uma escrita para a mesma localização pelo processador k não pode afetar o valor retornado pela leitura. Uma escrita é considerada executada com relação ao processador k, no instante em que uma leitura emitida pelo processador k para a mesma localização retorna o valor definido por esta escrita.*
>
> [...]*Uma escrita é considerada globalmente executada se está executada com relação a todos os processadores. Um leitura é considerada globalmente executada se está executada com relação a todos os processadores e se a escrita que gerou o valor lido também está globalmente executada.*

A principal deficiência destas definições é fato de serem baseadas em um tempo

global supostamente disponível em todos os processos, o que não é possível em um sistema distribuído assíncrono. O fato de não serem precisas, levou a interpretações erradas de seus significados gerando, inclusive, a revisão da definição do modelo de Consistência por Processador usada na definição da Consistência por Liberação (release consistency) [38]. O modelo por nós proposto dá suporte à representação de operações não atômicas de modo claro, através das relações entre eventos, sem se referenciar a um tempo global.

Attiya et al. [15, 32, 16] utilizam um modelo também baseado em eventos para a composição de operações. De forma semelhante a nossa abordagem, operações são caracterizadas por eventos de início e término (exceto operações de controle para as quais só um evento é definido). As condições de consistência são formalizadas considerando seqüências legais de subconjuntos de operações sobre as quais diversas restrições são impostas de acordo com a condição que se quer expressar. Em uma primeira abordagem [16], os autores assumem que as operações de cada processo são executadas em seqüência, na ordem de programa. Isto impede a modelagem de otimizações como as citadas na Seção 2.2. Além disso, esta abordagem também supõe a existência de um tempo global único que ordena todos os eventos e induz uma ordem parcial entre as operações. Em uma revisão desta formalização [15, 32], é permitida a execução não seqüencial das operações de um mesmo processo entretanto, a relação entre eventos, sobre a qual é definida a relação entre operações, continua sendo uma ordem total, o que contrasta com a nossa abordagem.

Mizuno e Raynal [72, 67, 71] modelam execuções como conjuntos de "histórias" (seqüências de operações de cada processo) parcialmente ordenadas através de uma relação entre escritas e leituras. Apesar de também definirmos uma execução como uma ordem parcial entre operações, as relações que definem tal ordem são bastante distintas. Primeiramente, as operações de um mesmo processo são parcialmente e

não totalmente ordenadas como nos referidos trabalhos. Em segundo lugar, não assumimos que todos os valores escritos em uma mesma localização são distintos. Acreditamos, portanto, que nossa abordagem é mais geral e que, considerando a modelagem de eventos, tem maior poder de expressão.

Os modelos utilizados por Gharachorloo [37] e Adve [1] são ambos derivados do formalismo introduzido por Collier [27]. Baseado em uma arquitetura abstrata onde para cada processador existe uma cópia integral da memória compartilhada, neste formalismo as operações são divididas em suboperações, de acordo com seu tipo. No caso de uma leitura, existem apenas as suboperações de início e aquela que indica a execução da leitura, ambas no contexto do mesmo processo. No caso de uma escrita, além da suboperação de início, para cada processo existe uma suboperação relativa à execução da escrita com relação aquele processo. Basicamente, uma execução é definida como uma ordem total sobre tais suboperações. Diversos modelos são então definidos através de restrições impostas a esta ordem total. Esta abordagem se adequa bem à modelagem da não atomicidade das escritas em um ambiente distribuído, servindo para a definição otimizada de condições de consistência. Em comparação com o modelo por nós definido, os eventos de início, término, publicação e busca associados a escritas e leituras são suficientes para também expressar a não atomicidade das escritas e leituras e também as relações entre suboperações sem a necessidade de assumir uma arquitetura abstrata. Além disso, como a relação entre eventos é por nós tratada como uma ordem parcial, esta é menos restrita que a relação de ordem total assumida entre suboperações.

De modo análogo ao modelo básico adotado por Mizuno e Raynal [67], Ahamad et al. [12, 8] considera que as operações de cada processo são totalmente ordenadas ("histórias") e define uma execução como o conjunto destas ordens totais. Diferentes condições de consistência são então definidas considerando serializações de

subconjuntos de operações. Diferentemente de Mizuno e Raynal, esta abordagem trata a existência de múltiplas escritas de mesmo valor para uma mesma localização. Cada possível combinação entre escrita e leitura é então considerada separadamente. Em contraste com esta abordagem, propomos a representação e tratamento em conjunto de todas as escritas que "podem influenciar" uma determinada leitura. No Capítulo 3, mostraremos como a modelagem de múltiplas escritas se relaciona com estruturas em grafos E–OU e, já no Capítulo 4, como uma nova teoria sobre legalidade pode ser estabelecida suportada pelos resultados obtidos na análise de tais grafos.

Gibbons, Merrit e Gharachorloo [40, 41] utilizaram o formalismo de autômatos de I/O [65, 64] para provar a equivalência do modelos de consistência por liberação e seqüencial para uma determinada classe de programas. Este formalismo é bastante geral e preciso, entretanto é criticado por produzir especificações complexas [15] e de difícil tradução para implementação [1].

Uma abordagem que difere bastante das anteriores foi proposta por Friggo [33]. Na formalização por ele adotada, os modelos de consistência são definidos sobre um grafo orientado acíclico (dag) onde cada vértice corresponde à execução de uma operação e as arestas são definidas por relações que dependem do modelo de programação adotada. Por exemplo, as arestas podem ser relativas à ordem de programa para operações de um mesmo processo e relativas a operações de sincronização para as operações entre processos. Frigo apresenta, então, várias especificações de condições de consistência (algumas delas detalhadas no Capítulo 5) baseadas em uma "função de observação", que, para cada leitura, indica qual vértice no dag foi o responsável pela escrita do valor lido. As diferentes condições de consistência são definidas impondo-se diferentes restrições a esta função.

A definição de execução, conforme apresentamos na Seção 2.2, pode ser representada por um grafo orientado, à semelhança do dag utilizado nesta última abordagem. Entretanto, o uso de tais funções de observação, acaba por encobrir alguns dos problemas por nós tratados detalhadamente, como a análise de execuções que exibem múltiplas escritas de um mesmo valor em uma mesma localização de memória.

Capítulo 3

Modelagem Grafo-Teórica

Neste capítulo, vamos apresentar alguns resultados em teoria de grafos que serão importantes ferramentas na análise de computações sobre memória compartilhada. Apesar das estruturas de grafos aqui introduzidas estarem diretamente relacionadas com as operações e relações entre estas definidas no Capítulo 2, os resultados aqui demonstrados são genéricos. Portanto, neste capítulo, consideraremos grafos orientados e grafos E–OU quaisquer. A relação entre tais grafos (e estruturas associadas) e condições de consistência atendidas ou não pelas execuções que eles representam serão apresentadas detalhadamente no Capítulo 4.

3.1 Grafos E–OU

Nesta seção, vamos apresentar o conceito de grafos E–OU e algumas estruturas associadas. Antes, porém, vamos introduzir a notação utilizada e o conceito de *knot* [46] em um grafo orientado.

Seja $H = (N, E)$ um grafo orientado. Para cada vértice $n_i \in N$, seja D_i e A_i, respectivamente, os conjuntos de sucessores e antecessores de n_i em H. Seja, também, $O_i \subseteq D_i$ o conjunto de sucessores imediatos de n_i e $I_i \subseteq A_i$ o conjunto de antecessores imediatos de n_i (aqueles que estão a uma aresta de n_i).

Definição 3.1 (Knot) *Seja $H = (N, E)$ um grafo orientado e K um subconjunto de N. K é dito um knot em H se e somente se, para todo vértice $n_i \in K$, $A_i = K$.*[1]

Como um exemplo, seja $H = (N, E)$ o grafo da Figura 3.1. Seja também $K \subseteq N$ o conjunto formado pelos vértices $\{2, 3, 6, 7, 10, 11\}$ destacados por círculos tracejados na referida figura. K é um knot em H, uma vez que, para todo elemento $n_i \in K$, os antecessores de n_i são os próprios elementos de K.

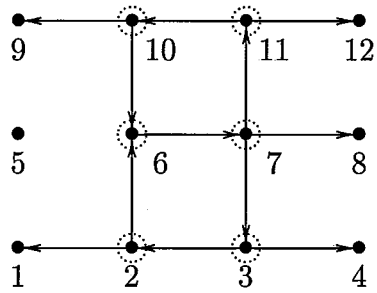


Figura 3.1: Knot em um grafo orientado.

Para caracterizarmos um grafo orientado como um grafo E–OU, necessitamos da definição de um modelo de precedência o qual define, para cada vértice n_i do grafo, como os seus antecessores imediatos são agrupados em subconjuntos de I_i. A definição 3.2 formaliza esta noção.

Definição 3.2 (Modelo de Precedência) *Seja $H = (N, E)$ um grafo orientado. Um modelo de precedência de H é uma coleção de $P_i^1, \ldots, P_i^{t_i}$, $t_i \geq 1$, de subconjuntos de I_i, para todo $n_i \in N$, tal que $P_i^1, \cup \ldots \cup, P_i^{t_i} = I_i$.*

No decorrer deste trabalho denominaremos de *conjuntos E de antecessores imediatos* de um vértice n_i a coleção de subconjuntos de I_i, denotados por $P_i^1, \ldots, P_i^{t_i}$.

[1] Quando mais apropriado, pode-se definir um knot como um subconjunto $K \subseteq N$ onde, para todo $n_i \in K$, $D_i = K$

A Figura 3.2, a seguir, apresenta um grafo no qual estão destacados por elipses tracejadas os conjuntos E de antecessores imediatos do vértice 6, ou seja, $P_6^1 = \{3, 4, 5\}$, $P_6^2 = \{2, 3\}$ e $P_6^3 = \{1\}$.

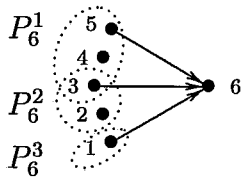


Figura 3.2: Conjuntos E de antecessores imediatos.

Um grafo orientado H será classificado como um grafo E–OU se a este estiver associado um modelo de precedência conforme a Definição 3.2. No restante deste trabalho,sempre que nos referirmos a um grafo E–OU estaremos considerando um grafo orientado com um modelo de consistência associado. Intuitivamente, os diferentes conjuntos E de antecessores imediatos definidos para cada vértice pelo modelo de precedência estão relacionados na prática com condições que devem ser atendidas simultaneamente. Por outro lado, estes conjuntos de antecessores imediatos se relacionam logicamente através de uma relação OU, indicando, na prática, que apenas um conjunto de condições deve ser atendido necessariamente.

Para tornar mais concretos os conceitos anteriores, vamos exemplificar como um modelo de precedência para um grafo que representa uma execução pode ser definido. Seja $G = (N, E)$ o grafo da Figura 3.3. Cada vértice $n_i \in N$ corresponde a uma operação e cada aresta aos pares de operações relacionados pela redução transitiva de $\xrightarrow{xo}$ e por $\xrightarrow{mi}$. Se n_i está associado a uma leitura, então existirão tantos conjuntos E de antecessores imediatos distintos de n_i quantos forem o número de escritas que "podem influenciar" esta leitura, no caso $r(x)v$. Cada conjunto E de antecessores imediatos conecta-se a n_i por no máximo uma aresta $\xrightarrow{xo}$ e por

exatamente uma aresta $\xrightarrow{mi}$. Na Figura 3.3, por simplicidade, não definimos a operação que precede n_i por $\xrightarrow{xo}$. O significado deste modelo de precedência deve ser entendido da seguinte maneira: $r(x)v$ pode ser influenciada por mais de uma escrita, portanto representamos em cada conjunto E de antecessores imediatos a possibilidade do valor lido ter se originado da primeira escrita **ou** da segunda escrita. Em ambos os casos, $r(x)v$ deve ser precedido por uma destas escritas **e** pela operação anterior a ela em $\xrightarrow{xo}$, se esta existir.

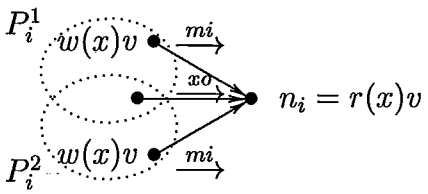


Figura 3.3: Modelo de precedência relativo a uma execução.

3.2 Estruturas em Grafos E–OU

Algumas estruturas conhecidas de grafos E–OU, e outras introduzidas neste trabalho, são apresentadas nesta seção, bem como alguns resultados relativos a propriedades e relacionamentos de tais estruturas.

Primeiramente, vamos considerar a definição de *b–subgrafo* e *b-knot* introduzidas por Barbosa e Benevides [18].

Definição 3.3 (b–subgrafo) *Seja $H = (N, E)$ um grafo E–OU e $H' = (N', E')$ um subgrafo de H. H' é dito um* b–subgrafo *de H se e somente se todo vértice $n_i \in N'$ tem no máximo t_i antecessores imediatos em H', dos quais exatamente um pertence a cada um dos conjuntos $P_i^1, \ldots, P_i^{t_i}$.*

Definição 3.4 (b-knot) *Um subconjunto $K \subseteq N$ é dito ser um* b-knot *em H se e somente se K é um knot em algum b–subgrafo de H.*

A Figura 3.4 mostra, na parte (a), um grafo E–OU que contém um b-knot e, na parte (b), um b–subgrafo correspondente que contém um knot. Considere que apenas o vértice 6 possui mais de um conjunto E de antecessores imediatos, no caso os conjuntos de vértices $\{2, 5\}$ e $\{5, 10\}$. O b–subgrafo na parte (b) da referida figura foi obtido escolhendo-se os vértices 2 e 10 como antecessores imediatos do vértice 6. Este b–subgrafo apresenta um knot formado pelo conjunto de vértices $\{2, 3, 6, 7, 10, 11\}$ (marcados com um círculo pontilhado na figura). Outro possível b–subgrafo incluiria apenas o vértice 5 como antecessor imediato do vértice 6, uma vez que este elemento pertence a ambos os conjuntos E de antecessores imediatos de 6. Entretanto, este último b–subgrafo não apresentaria knot. Como, pela Definição 3.4, basta que exista um b–subgrafo com knot para caracterizar um b-knot, temos que o grafo da parte (a) da Figura 3.4 possui um b-knot.

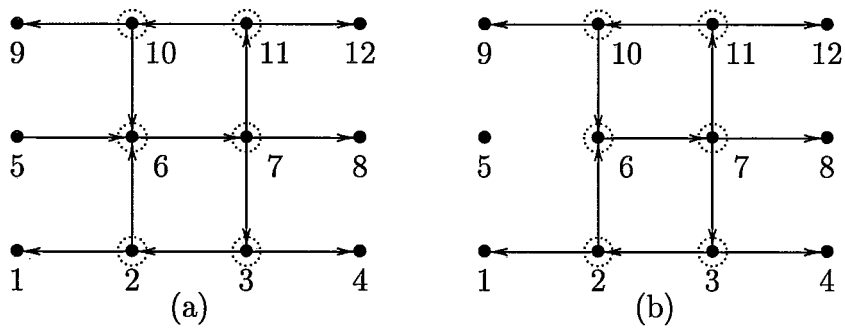


Figura 3.4: b-knot em grafo E–OU e b–subgrafo correspondente.

Intuitivamente, como um b–subgrafo inclui um representante de cada conjunto E de antecessores imediatos, estão nele representadas todas as possíveis combinações de escolhas de antecessores imediatos para cada vértice. De maneira análoga, podemos definir um outro subgrafo, denominado *c–subgrafo* onde, para cada vértice, seus antecessores imediatos são exatamente todos os elementos de apenas um dos

conjuntos E de antecessores imediatos do grafo E–OU original. Esta definição é formalizada a seguir.

Definição 3.5 (c–subgrafo) *Seja $H = (N, E)$ um grafo E–OU e $H' = (N', E')$ um subgrafo de H. H' é dito um* c–subgrafo *de H se e somente se todo vértice $n_i \in N'$ tem como seus antecessores imediatos em H' todos os elementos de exatamente um dos conjuntos $P_i^1, \ldots, P_i^{t_i}$.*

A Figura 3.5 ilustra um c–subgrafo associado ao grafo E–OU da Figura 3.4 no qual, para o vértice 6, foi escolhido o conjunto E de antecessores imediatos formado pelos vértices 2 e 5. Podemos observar que o c–subgrafo da Figura 3.5 possui um ciclo e que, mesmo se escolhêssemos o conjunto $\{2, 10\}$ como antecessores de 6, também teríamos um c–subgrafo com ciclo. Como veremos a seguir, a existência de knots em b–subgrafos está fortemente associado à existência de ciclos em c–subgrafos de um mesmo grafo E–OU.

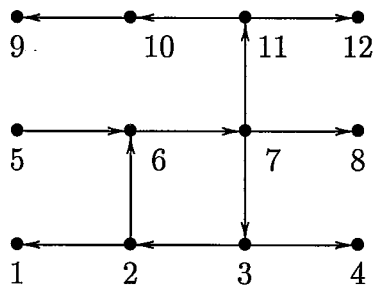


Figura 3.5: Exemplo de c–subgrafo.

O Teorema 3.1 relaciona a existência de b-knots em um grafo E–OU com a existência de ciclos em c–subgrafos deste grafo. Antes, porém, apresentamos o Lema 3.1 que auxilia a prova do referido Teorema.

Lema 3.1 *Se H não contém b-knots, então toda componente fortemente conexa de H tem pelo menos um vértice n_i tal que pelo menos um dos conjuntos $P_i^1, \ldots, P_i^{t_i}$ não intercepta a componente.*

Prova. O lema é trivial para componentes com um único vértice. Para componentes com cardinalidade maior que um, seja C uma componente fortemente conexa de H e seja L seu conjunto de vértices. Suponha que todo vértice $n_i \in L$ seja tal que todos os conjuntos $P_i^1, \ldots, P_i^{t_i}$ interceptam L. Vamos mostrar que H contém um b-knot. Para tal, primeiramente construímos um b–subgrafo H' de H. Este b–subgrafo tem como conjunto de vértices L no qual cada $n_i \in L$ tem exatamente um antecessor imediato em cada conjunto $P_i^1 \cap L, \ldots, P_i^{t_i} \cap L$. Note que, por suposição, esta construção é sempre possível. Note ainda que, porque C é fortemente conexa, H' não tem vértices fonte. Considere agora a seqüência de conjuntos $A_i^1, A_i^2, A_i^3, \ldots$, para algum vértice $n_i \in L$, onde A_i^1 é o conjunto de antecessores do vértice n_i em H' e, para $k > 1$, A_i^k é o conjunto de antecessores em H' de todos os vértices em A_i^{k-1}. A ausência de fontes em H' assegura que todos os conjuntos nesta seqüência contêm pelo menos um elemento. Além disso, porque L é finito, a seqüência tem um ponto fixo, digamos j, no qual $A_i^j = A_i^{j-1}$, o qual é, por definição, um knot em H' e, portanto, um b-knot em H. □

Podemos interpretar o Lema 3.1 da seguinte forma. Um b-knot pode ser definido também como uma componente fortemente conexa de um grafo E–OU onde, para cada vértice da componente, todos os conjuntos E de antecessores imediatos interceptam esta componente. Se um grafo E–OU não tem b-knots, isto significa que, em toda componente fortemente conexa, existe uma "saída" desta componente, ou seja, sempre existirá um b–subgrafo no qual, se iniciarmos uma travessia partindo de qualquer vértice da componente e que prossiga para um de seus antecessores

imediatos, sempre encontraremos um vértice que possui um antecessor imediato que não pertence à componente. A Figura 3.6 ilustra esta idéia. Nesta figura, duas componentes são destacadas, uma com um conjunto E de antecessores imediatos que intercepta a componente e outra representando a dita "saída" da componente.

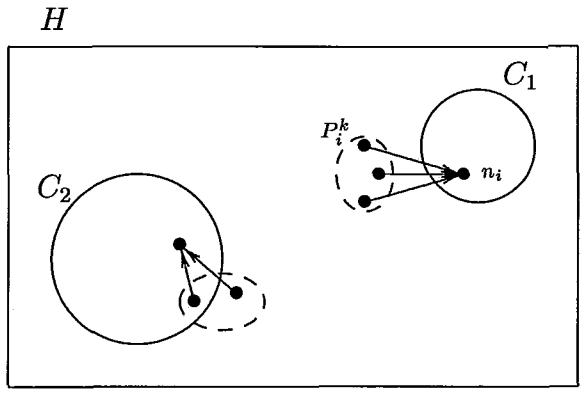


Figura 3.6: Componentes de um grafo E–OU.

Teorema 3.1 *H contém um b-knot se e somente se todo c–subgrafo gerador de H contém um ciclo orientado.*

Prova. Seja K um b-knot em H. Por definição, todo c–subgrafo gerador de H inclui K como parte de seu conjunto de vértices. Seja H' um c–subgrafo, e considere uma travessia de H' iniciada em qualquer vértice de K e que prossegue da seguinte forma. Quando em um vértice n_i, a travessia prossegue para outro vértice que é um antecessor imediato de n_i tanto em H' quanto no b–subgrafo de H onde K é um knot (note que tal antecessor sempre existe como conseqüência das próprias definições de b–subgrafo e c–subgrafo). Esta travessia é confinada em K e, porque K é finito, esta, em algum passo, retorna para um vértice já visitado, caracterizando, portanto, um ciclo orientado em H'.

Se H não contém b-knots, então é possível construirmos um c–subgrafo gerador acíclico de H. De modo a construir tal c–subgrafo, primeiramente vamos dividir H

em componentes fortemente conexas maximais $C_1, \ldots, C_m$. Se todas as componentes $C_1, \ldots, C_m$ são unitárias, então H é acíclico devido à maximalidade das componentes e, portanto, também são acíclicos cada um de seus c–subgrafos. Caso contrário, pelo Lema 3.1, e para $1 \leq k \leq m$, seja E_k o conjunto não vazio de vértices de C_k tal que, se $n_i \in E_k$, então pelo menos um dos conjuntos $P_i^1, \ldots, P_i^{t_i}$ não intercepta C_k. Se considerarmos cada componente $C_1, \ldots, C_m$ como um "super vértice" sendo que os únicos arcos que chegam ao "super vértice" C_k são aqueles vindos de todos os vértices de um dos conjuntos $P_i^1, \ldots, P_i^{t_i}$ que não intercepta C_k para $n_i \in E_k$, então obtemos um c–subgrafo acíclico sobre os "super vértices" (acíclico, novamente pela maximalidade das componentes fortemente conexas). Agora vamos diminuir o "super vértice" C_k removendo deste E_k, e recursivamente repetir todo o processo a partir da divisão em componentes fortemente conexas. A recursão termina quando estas componentes não podem mais ser formadas exceto se forem conjuntos unitários. Neste ponto, tem-se um c–subgrafo gerador acíclico de H. □

A Figura 3.7 ilustra o processo recursivo utilizado na prova do Teorema 3.1. Observe, inicialmente, os vértices 1 e 2 pertencentes à componente C. Vamos considerar que estes vértices possuem os conjuntos $\{a\}$ e $\{b\}$ como conjuntos E de antecessores imediatos de modo que $\{a\}$ e $\{b\}$ não interceptam a componente C. O primeiro passo da recursão para a formação de um c–subgrafo é a escolha destes conjuntos como os antecessores imediatos para os vértices 1 e 2, respectivamente. Num segundo passo, consideramos uma componente reduzida, que não contém os vértices 1 e 2. Repete-se, então, o mesmo processo a partir da identificação dos vértices que possuem conjuntos de antecessores imediatos que não interceptam a nova componente, no caso os vértices 3 e 4.

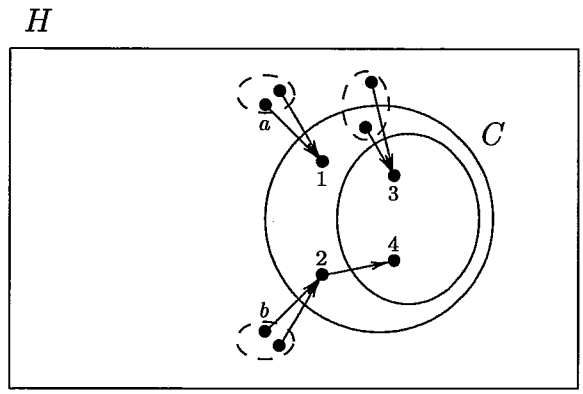


Figura 3.7: Componentes fortemente conexas maximais de um grafo E–OU.

3.2.1 C–subgrafos Consistentes

O Teorema 3.1 define a relação entre a existência de b-knots em um grafo E–OU e a existência de ciclos em todos os possíveis c–subgrafos geradores correspondentes. Entretanto, conforme veremos no Capítulo 4, estaremos interessados em apenas um subconjunto destes possíveis c–subgrafos. Nesta próxima seção, iremos caracterizar este subconjunto (de modo genérico) e estabelecer a relação entre ele e estruturas no grafo E–OU original. Também no Capítulo 4, identificaremos este conjunto de c–subgrafos como aquele que representam possíveis execuções.

Definição 3.6 (c–subgrafo Consistente) *Seja H um grafo E–OU e Φ o conjunto de todos os possíveis c–subgrafos geradores de H. Seja, ainda, $\phi \subseteq \Phi$ um subconjunto de c–subgrafos de interesse. $S \in \Phi$ é um c–subgrafo consistente se e somente se $S \in \phi$.*

Para prosseguirmos com a análise da relação entre os c–subgrafos em ϕ e estruturas no grafo original H, vamos identificar cada elemento de ϕ. Desta forma, seja $S_i, 1 \leq i \leq l$, um elemento de ϕ, onde l é o número de possíveis c–subgrafos geradores distintos de H. Vamos, também, denotar por P_i^k o conjunto E de antecessores imediatos do vértice n_i que participa do c–subgrafo S_k.

Dadas a definição de c–subgrafo consistente e as notações acima, podemos propor o Teorema 3.2 que, analogamente ao Teorema 3.1, estabelece as condições necessárias e suficientes para que todo c–subgrafo consistente com relação a um subconjunto de Φ contenha ciclos.

Teorema 3.2 *Todo c–subgrafo gerador de H consistente com relação a ϕ contém um ciclo orientado se e somente se, para todo elemento S_k de ϕ, existe uma componente fortemente conexa não unitária C de H tal que, para todo vértice $n_i \in C$, P_i^k intercepta C.*

Prova. A prova segue a mesma linha da prova do Teorema 3.1. Suponha que exista $S_k \in \phi$ tal que toda componente fortemente conexa não unitária de H contém pelo menos um vértice n_i para o qual P_i^k não intercepta a componente. Vamos mostrar que é possível construir um c–subgrafo gerador consistente com relação a ϕ acíclico de H. De modo a construirmos tal c–subgrafo, vamos primeiramente dividir H em componentes fortemente conexas maximais $C_1, \ldots, C_m$. Se todas as componentes $C_1, \ldots, C_m$ são unitárias, então H é acíclico devido à maximalidade das componentes e, portanto, também são acíclicos cada um de seus c–subgrafos. Caso contrário, para $1 \leq j \leq m$, seja E_j o conjunto não vazio de vértices de C_j tal que, se $n_i \in E_j$, então P_i^k não intercepta C_j. Se considerarmos cada componente $C_1, \ldots, C_m$ como um "super vértice" sendo que os únicos arcos que chegam ao "super vértice" C_j são aqueles vindos de todos os vértices de um dos conjuntos P_i^k que não interceptam C_j para $n_i \in E_j$, então obtemos um c–subgrafo acíclico sobre os "super vértices" (acíclico, novamente pela maximalidade das componentes fortemente conexas). Vamos, agora, diminuir o "super vértice" C_j removendo deste E_j, e recursivamente repetir todo o processo a partir da divisão em componentes fortemente conexas (considerando o mesmo elemento S_k). A recursão termina quando estas componen-

tes não podem mais ser formadas exceto se forem conjuntos unitários. Neste ponto, temos um c–subgrafo gerador acíclico de H. Este c–subgrafo é ainda consistente com relação a ϕ, uma vez que todas as arestas que conectam os "super vértices", em cada passo da recursão, são relativas ao mesmo elemento S_k de ϕ.

Por outro lado, seja S_k um elemento de ϕ e seja C uma componente fortemente conexa não unitária de H tal que, para todo vértice $n_i \in C$, P_i^k intercepta C. Por definição, todo c–subgrafo gerador consistente de H inclui C como parte de seu conjunto de vértices. Considere uma travessia de S_k iniciada em qualquer vértice de C e que prossegue da seguinte forma. Quando em um vértice n_i, a travessia prossegue para outro vértice que é um antecessor imediato de n_i tanto em S_k quanto em C. Note que tal antecessor sempre existe como conseqüência da própria definição de c–subgrafo e pelo fato de que P_i^k intercepta C. Esta travessia é confinada em C e, porque C é finito, esta, em algum passo, retorna para um vértice já visitado, caracterizando, portanto, um ciclo orientado em S_k. □

Apesar da definição de c–subgrafos consistentes ser bastante abstrata neste ponto, no próximo capítulo, iremos definir precisamente os elementos de um conjunto de c–subgrafos que estará associado às possíveis subexecuções quando representarmos uma execução por um grafo E–OU.

Capítulo 4

Legalidade

Neste capítulo, apresentamos a definição original de legalidade e introduzimos duas definições alternativas. A motivação para estas novas definições, conforme veremos nas seções seguintes, é o fato destas serem estabelecidas sobre a ordem parcial definida por uma dada execução. Em contraste, a definição original de legalidade é dada sobre uma seqüência de operações que, de alguma maneira, se relaciona com a execução propriamente dita.

Como ressaltamos no Capítulo 1, o conceito de legalidade está presente na grande maioria das definições de modelos de consistência. Por esse motivo, uma definição menos rígida deste conceito pode levar a implementações mais eficientes de um determinado modelo.

4.1 Legalidade Seqüencial

O conceito de legalidade foi primeiramente apresentado por Herlihy e Wing [45] com o intuito de formalizar quais as seqüências de operações que deveriam ser consideradas corretas para um determinado tipo de objeto. Esta definição original, dada sobre uma ordem total de operações, assume que cada objeto tem uma *especificação seqüencial* que estabelece o conjunto de operações válidas e o conjunto de

seqüências de operações consideradas corretas para o objeto. Baseado nesta especificação seqüencial de cada objeto, a definição de legalidade é apresentada a seguir. Devido à característica de ser especificada sobre uma seqüência de operações, nos referiremos a esta definição como *legalidade seqüencial* no restante deste trabalho.

Definição 4.1 (Legalidade Seqüencial) *Uma seqüência de operações* $\xrightarrow{\rho}$ *é legal se e somente se, para todo objeto* x*, a subseqüência de* $\xrightarrow{\rho}$ *composta apenas de operações sobre* x *está na especificação seqüencial de* x*.*

Na definição acima, note que, por simplicidade, usamos uma ordem total ($\xrightarrow{\rho}$) e não um conjunto totalmente ordenado para denotar uma seqüência de operações. Exceto quando necessário, o respectivo conjunto de operações ficará por ser inferido a partir do contexto. Utilizaremos esta simplificação notacional no restante deste texto.

Neste trabalho, consideraremos como objetos apenas localizações compartilhadas de memória e, como operações, apenas leituras e escritas. A especificação seqüencial deste tipo de objeto dita que as seqüências válidas são aquelas nas quais as operações de leitura não retornam valores sobrepostos, ou seja, a usual semântica para operações de leitura, que devem retornar o último valor escrito na seqüência.

De modo a sermos mais específicos, seja $\xrightarrow{\rho}$ uma seqüência de operações. Uma operação de leitura $r(x)v$ é dita ser *legal* se existe uma operação de escrita $w(x)v$ tal que $w(x)v \xrightarrow{\rho} r(x)v$ e não existe nenhuma outra operação de escrita $w(x)v'$ tal que $v' \neq v$ e $w(x)v \xrightarrow{\rho} w(x)v' \xrightarrow{\rho} r(x)v$. Baseados nesta noção de legalidade de uma operação de leitura, podemos tornar mais específica a definição de legalidade seqüencial, da seguinte forma.

Definição 4.2 (Legalidade Seqüencial revisada) *Uma seqüência de operações* $\xrightarrow{\rho}$ *é legal se e somente se todas as operações de leitura em* $\xrightarrow{\rho}$ *são legais.*

A Definição 4.2 e a noção de leituras legais sobre a qual esta é baseada é freqüentemente utilizada na definição de condições de consistência sobre conjuntos de operações de leituras e escritas (parcialmente ou totalmente ordenados) [15, 72, 37, 67, 1, 5, 45]. Na verdade, desconhecemos abordagens onde o conceito de legalidade é definido formalmente sobre um conjunto parcialmente ordenado conforme apresentaremos a seguir. Mesmo a abordagem de Mizuno [67] que, de maneira semelhante a nossa, define condições de consistência sobre uma ordem parcial de operações, a definição de legalidade é estabelecida sobre sub-seqüências totalmente ordenadas de operações nas quais todas as leituras devem ser legais.

Antes de nos aprofundarmos nos detalhes da definição de legalidade sobre conjuntos parcialmente ordenados de operações, é importante ressaltar que tal definição deve de alguma maneira estar ligada à definição usual de legalidade sobre seqüências de operações, de modo que a semântica de se evitar o retorno de valores "sobrepostos" seja mantida na nova definição. Para tal, nossa abordagem aqui será a de exigir que exista uma seqüência legal de operações que estenda, de uma maneira a ser definida precisamente nas próximas seções, o conjunto parcialmente ordenado que definiremos como legal.

Vale observar que a exigência de equivalência semântica distingue nossa abordagem daquela definida nos trabalhos de Mizuno [67] e Raynal [72] (Seção 4.4). Nestes trabalhos, uma execução é definida como legal se todas as leituras nela forem legais. Entretanto, para classificar leituras como legais dentro da ordem parcial que representa a execução, Mizuno e Raynal tomam apenas o subconjunto das operações relacionadas com cada leitura separadamente. Como conseqüência, uma execução

classificada por esta abordagem pode não ter uma extensão linear legal. A Figura 4.1 mostra uma execução onde todas as leituras são legais e que, portanto, seria classificada como legal por tal definição. Entretanto, conforme veremos na próxima seção, esta execução não pode ser estendida por uma seqüência legal de operações.

Relembrando as definições introduzidas no Capítulo 2, uma execução é representada por um conjunto parcialmente ordenado $(\Omega, \xrightarrow{\sigma})$, onde Ω é o conjunto de operações e $\xrightarrow{\sigma}$ é o fecho transitivo de todos os pares de operações relacionadas por $\xrightarrow{mi}$ ou $\xrightarrow{xo}$.

Neste capítulo, usaremos representar uma dada execução por um grafo orientado, o que possibilitará a aplicação dos resultados do Capítulo 3 nas definições e teoremas aqui introduzidos. Portanto, dada uma execução $(\Omega, \xrightarrow{\sigma})$, iremos representá-la por um grafo orientado $G(N, E)$ que tem como conjunto de vértices Ω e como arestas os pares de operações dados por $\xrightarrow{mi}$ e pela redução transitiva de $\xrightarrow{xo}$. A Figura 4.1 mostra um exemplo de tal representação gráfica, onde as arestas de um vértice relativo a uma escrita para um vértice relativo a uma leitura são aquelas devido à $\xrightarrow{mi}$ e as demais arestas são devido à $\xrightarrow{xo}$.

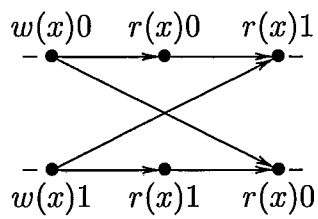


Figura 4.1: Execução representada por grafo orientado.

Nas próximas seções, apresentaremos duas novas definições para o conceito de legalidade.

4.2 Legalidade sob Escritas Únicas

Nossa primeira definição de legalidade sobre um conjunto parcialmente ordenado é baseada, por simplicidade, na suposição de que, para cada leitura na execução, existe exatamente uma operação de escrita que se relaciona com ela através da relação "pode influenciar", $\xrightarrow{mi}$. Esta simplificação não elimina a existência de múltiplas escritas de mesmo valor para uma mesma localização e nem a possibilidade de uma mesma escrita "poder influenciar" várias leituras, mas implica que os conjuntos compostos por pares escrita–leitura relacionadas por $\xrightarrow{mi}$ são disjuntos. Por esta razão, tais escritas podem ser identificadas e a execução pode ser tratada como se todo valor fosse unicamente identificado, isto é, como se nenhum valor fosse escrito na mesma localização de memória mais de uma vez. Como conseqüência, a semântica associada à relação $\xrightarrow{mi}$ passa a ser de "certamente influenciou" ao invés de "pode influenciar". Esta suposição tem sido usada em em algumas definições formais de condições de consistência [72, 71, 67, 66]. Entretanto, na seção 4.3, relaxaremos esta suposição e uma segunda definição de legalidade sobre conjuntos parcialmente ordenados será introduzida.

O primeiro passo necessário à definição de legalidade sobre uma execução é estender a ordem parcial $\xrightarrow{\sigma}$ que representa esta execução, através da adição relacionamentos entre pares de operações necessários em uma seqüência legal com o intuito de refletir as condições necessárias para a existência de uma extensão linear legal desta ordem parcial. Isto irá garantir a equivalência semântica com a definição de legalidade sobre ordens totais, conforme justificamos anteriormente. A seguinte definição formaliza esta noção através da relação $\xrightarrow{lc}$, denominada *restrição de legalidade.*

Definição 4.3 (Restrição de Legalidade sob escritas únicas) *Para $v \neq v'$, seja $r(x)v$ e $w(x)v'$ tais que $r(x)v$ não precede $w(x)v'$ em $\xrightarrow{\sigma}$. Então $r(x)v \xrightarrow{lc} w(x)v'$ se e somente se existe $r(x)v'$ tal que $w(x)v \xrightarrow{\sigma} r(x)v'$.*

A intuição neste conceito de restrição de legalidade é a seguinte. Se $r(x)v$, $w(x)v'$ e $r(x)v'$ estão relacionado conforme descrito na Definição 4.3, então, em qualquer extensão linear de $\xrightarrow{\sigma}$, $w(x)v$ deve preceder $w(x)v'$. Portanto, de modo a não violar a legalidade seqüencial, $r(x)v$ deve preceder $w(x)v'$. Observe que obrigatoriamente $w(x)v \xrightarrow{mi} r(x)v$ e $w(x)v' \xrightarrow{mi} r(x)v'$, uma vez que estamos supondo a existência de uma única operação de escrita para cada valor.

Note ainda que, na Definição 4.3, não consideramos a existência de uma operação de escrita $w(x)v'$ sucedendo $w(x)v$ em $\xrightarrow{\sigma}$ para relacionarmos $r(x)v$ e $w(x)v'$ por $\xrightarrow{lc}$. Isto se deve ao fato de que, se existe $w(x)v'$ tal que $w(x)v \xrightarrow{\sigma} w(x)v'$, então, ou existe também $r(x)v'$ tal que $w(x)v' \xrightarrow{mi} r(x)v'$ e, portanto, $w(x)v \xrightarrow{\sigma} r(x)v'$ ou podemos desconsiderar tal operação de escrita ($w(x)v'$), uma vez que esta não tem influência alguma sobre a execução [1].

Vamos, agora, estender a relação $\xrightarrow{\sigma}$ acrescentando os pares relacionados por $\xrightarrow{lc}$. A esta nova relação denominamos $\xrightarrow{\hat{\sigma}}$, sendo definida pelo fecho transitivo da união de $\xrightarrow{xo}$, $\xrightarrow{mi}$ e $\xrightarrow{lc}$ (ou seja, $\xrightarrow{\hat{\sigma}} = [\xrightarrow{xo} \cup \xrightarrow{mi} \cup \xrightarrow{lc}]^+$). A Figura 4.2 ilustra a inclusão de pares devido à restrição de legalidade ($\xrightarrow{lc}$), representados pelas arestas tracejadas, em uma dada execução. Em ambos os casos (partes (a) e (b)), existem as operações $w(x)v$ e $r(x)v'$ tal que $w(x)v \xrightarrow{\sigma} r(x)v'$. Observe, entretanto, que, na parte (b), o "caminho" entre $w(x)v$ e $r(x)v'$ envolve a própria operação de leitura que participa da restrição de legalidade adicionada.

[1] Como os modelos de consistência, em geral, exigem alguma forma de consistência com a ordem de programa, no Capítulo 5, introduziremos operações artificiais de leitura em execuções equivalentes a fim de possibilitar a análise também destas operações de escrita que não influenciam nenhuma leitura nas execuções propriamente ditas.

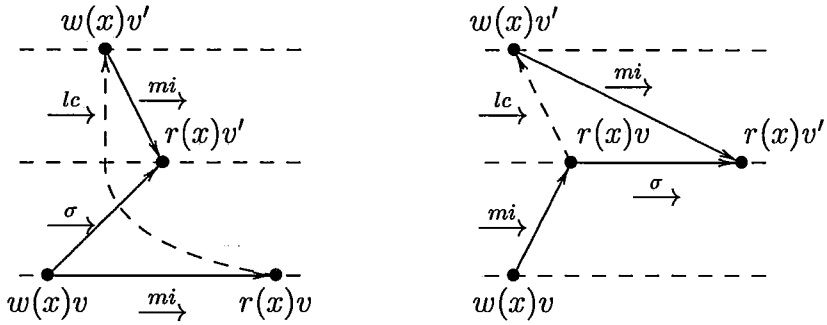


Figura 4.2: Inclusão dos pares relativos à restrição de legalidade.

Podemos, agora, estabelecer nossa primeira definição de legalidade baseada na ordem parcial $\xrightarrow{\hat{\sigma}}$ e considerando escritas unicamente identificadas.

Definição 4.4 (Legalidade sob escritas unicamente identificadas) $(\Omega, \xrightarrow{\sigma})$ *é legal se e somente se* $\xrightarrow{\hat{\sigma}}$ *é acíclica.*

Para ilustrar os conceitos envolvidos na Definição 4.4, seja a Figura 4.3 onde duas execuções são exibidas, uma legal, na parte (a) e outra que não é legal na parte (b). Nesta figura, os grafos correspondentes são mostrados com seus conjuntos de arestas expandidos pela adição das arestas que representam a restrição de legalidade (arestas tracejadas). As linhas horizontais tracejadas agrupam as operações de um mesmo processo, com exceção das escritas que foram destacadas para indicar que estas só possuem sucessores diretos através da relação $\xrightarrow{mi}$. Note, por exemplo, na parte (a), que $w(x)1 \xrightarrow{\sigma} r(x)0$ e, portanto, as leituras influenciadas por esta escrita devem preceder $w(x)0$ em $\xrightarrow{lc}$. Isto, entretanto, não gera ciclos no grafo expandido. Logo, a execução representada na parte (a) é legal. Já na parte (b), $w(x)0 \xrightarrow{\sigma} r(x)1$, portanto a leitura $r(x)0$ que não precede a operação $w(x)1$ em $\xrightarrow{\sigma}$ deve precedê-la na restrição de legalidade. De modo simétrico, $r(y)0$ deve preceder $w(y)1$ na restrição de legalidade. A execução representada por G, neste caso, não é legal porque existe um ciclo direcionado em $\xrightarrow{\hat{\sigma}}$ (este ciclo pode ser facilmente

visualizado no grafo expandido).

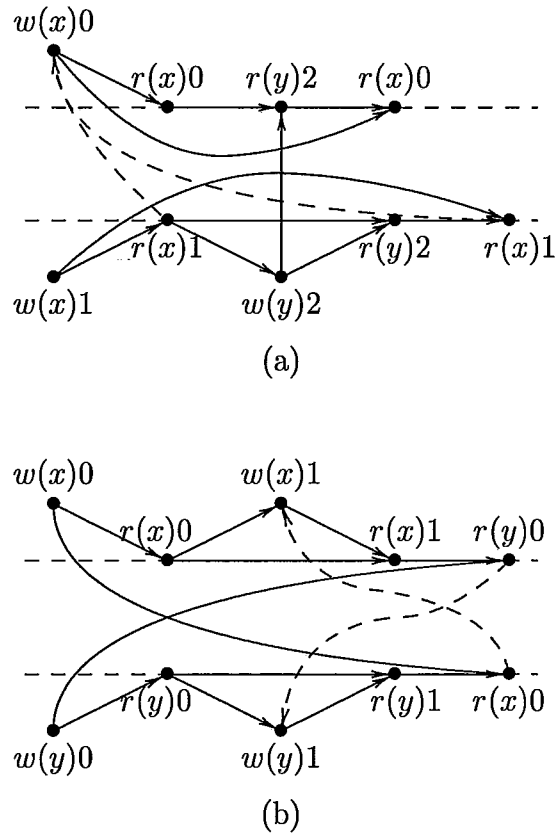


Figura 4.3: Uma execução legal e uma que não é legal.

O Teorema 4.1, apresentado a seguir, relaciona execuções que são legais pela Definição 4.4 com a legalidade seqüencial.

Teorema 4.1 $(\Omega, \xrightarrow{\sigma})$ *é legal se e somente se* $\xrightarrow{\sigma}$ *tem uma extensão linear legal.*

Prova. Se $\xrightarrow{\sigma}$ não tem extensão linear legal, então existem localizações x e y e valores v_x e v'_x, v_y, e v'_y, tais que $w(x)v_x \xrightarrow{\sigma} r(x)v'_x \xrightarrow{\sigma} r(y)v_y$ e $w(y)v_y \xrightarrow{\sigma} r(y)v'_y \xrightarrow{\sigma} r(x)v_x$. Neste caso, $\xrightarrow{lc}$ é tal que $r(x)v_x \xrightarrow{lc} w(x)v'_x$ e $r(y)v_y \xrightarrow{lc} w(y)v'_y$ e, portanto, $\xrightarrow{\hat{\sigma}}$ contém um ciclo orientado. Pela Definição 4.4, $(\Omega, \xrightarrow{\sigma})$ não é legal.

Por outro lado, se $\xrightarrow{\sigma}$ tem uma extensão linear legal $\xrightarrow{\rho}$, então, pela Definição 4.2, uma dada localização x produz um padrão alternante em $\xrightarrow{\rho}$: primeiro uma escrita para x aparece, em seguida as leituras daquele valor, então a escrita de outro valor seguida pelas leituras deste e assim por diante. Pela Definição 4.3, tal serialização $\xrightarrow{\rho}$ contém $\xrightarrow{lc}$, portanto $\xrightarrow{\rho}$ também é uma extensão linear de $\xrightarrow{\hat{\sigma}}$ que, portanto, tem que ser acíclica. Pela Definição 4.4, $(\Omega, \xrightarrow{\sigma})$ é legal. □

4.3 Legalidade sob Múltiplas Escritas

Nesta seção, eliminaremos a suposição de que todos os valores são unicamente identificados em uma execução. Nossa abordagem aqui será a de considerarmos todas as escritas que "podem influenciar" uma determinada leitura simultaneamente. Essa abordagem contrasta com outros modelos que também permitem múltiplas escritas, mas que procedem a análise considerando cada possibilidade separadamente. A seção 4.4 apresenta uma comparação mais detalhada destas abordagens.

De modo a generalizarmos a restrição de legalidade da Definição 4.3 para permitir múltiplas escritas, alguns conceitos precisam, primeiramente, ser definidos. Considere uma execução $(\Omega, \xrightarrow{\sigma})$. Vamos denominar a relação $\xrightarrow{\sigma'}$ de uma "redução para escritas únicas" de $\xrightarrow{\sigma}$ se $\xrightarrow{\sigma'}$ é definida da mesma forma que $\xrightarrow{\sigma}$, exceto pela troca de $\xrightarrow{mi}$ por um subconjunto de $\xrightarrow{mi}$ que inclui exatamente um par para cada leitura. Em outras palavras, os pares em $\xrightarrow{mi}$ que participam de $\xrightarrow{\sigma'}$ são aqueles que definem, para cada leitura, exatamente uma escrita que "pode influenciá-la". Note que, de acordo com nossa discussão da seção 4.2, tal subconjunto de $\xrightarrow{mi}$ pode ser considerado como um conjunto que caracteriza a existência de escritas únicas de um mesmo valor para uma mesma localização de memória. Cada redução para escritas únicas representa uma possível combinação entre escritas e leituras de um mesmo

valor em uma mesma localização, sendo semelhante a uma execução onde os valores são unicamente identificados.

Precisamos, agora, redefinir a restrição de legalidade considerando a existência de múltiplas escritas.

Definição 4.5 (Restrição de Legalidade sob múltiplas escritas) *Para $v \neq v'$, seja $r(x)v$ e $w(x)v'$ tais que $r(x)v$ não precede $w(x)v'$ em $\xrightarrow{\sigma}$. Então $r(x)v \xrightarrow{lc} w(x)v'$ se e somente se existe uma redução para escritas únicas de $\xrightarrow{\sigma}$, denominada $\xrightarrow{\sigma'}$, tal que $r(x)v \xrightarrow{lc'} w(x)v'$, onde $\xrightarrow{lc'}$ é a restrição de legalidade que resulta da aplicação da Definição 4.3 sobre $\xrightarrow{\sigma'}$ ao invés de sobre $\xrightarrow{\sigma}$.*

De acordo com a Definição 4.5, a relação restrição de legalidade $\xrightarrow{lc}$ para múltiplas escritas é a união das restrições de legalidade que resultam da aplicação da Definição 4.3 sobre todas as reduções para escritas únicas de $\xrightarrow{\sigma}$. Esta definição de $\xrightarrow{lc}$ pode ser considerada muito abstrata se pretendemos que ela seja útil para um uso prático eventual. Entretanto, como demonstraremos a seguir através da Proposição 4.1, é possível mostrar exatamente quais pares leitura-escrita resultam da Definição 4.5.

Proposição 4.1 *Para $v \neq v'$, seja $r(x)v$ e $w(x)v'$ tais que $r(x)v$ não precede $w(x)v'$ em $\xrightarrow{\sigma}$. Então $r(x)v \xrightarrow{lc} w(x)v'$ se e somente se existem $w(x)v$ e $r(x)v'$ tais que*

(a) $w(x)v \xrightarrow{mi} r(x)v$,

(b) $w(x)v \xrightarrow{\sigma} r(x)v'$, *e*

(c) $w(x)v' \xrightarrow{mi} r(x)v'$.

Prova. Se $r(x)v \xrightarrow{lc} w(x)v'$, então, pela Definição 4.5, existe $\xrightarrow{\sigma'}$, uma redução para escritas únicas de $\xrightarrow{\sigma}$, tal que $r(x)v \xrightarrow{lc'} w(x)v'$, onde $\xrightarrow{lc'}$ resulta da aplicação

da Definição 4.3 sobre $\xrightarrow{\sigma'}$. Pela Definição 4.3, segue que existe $r(x)v'$ tal que $w(x)v \xrightarrow{\sigma'} r(x)v'$, onde $w(x)v$ é a única escrita que precede $r(x)v$ por $\xrightarrow{mi}$ em $\xrightarrow{\sigma'}$, portanto implicando (a). De forma análoga, $w(x)v'$ precede $r(x)v'$ por $\xrightarrow{mi}$ em $\xrightarrow{\sigma'}$, implicando (c). O item (b) segue diretamente do fato que todo par em $\xrightarrow{\sigma'}$ é também um par em $\xrightarrow{\sigma}$.

Por outro lado, assuma que existe $w(x)v$ e $r(x)v'$ tais que (a), (b) e (c) sejam verdadeiros. Se $\xrightarrow{\sigma'}$ é a redução para escritas únicas de $\xrightarrow{\sigma}$ na qual $w(x)v$ precede $r(x)v$ por $\xrightarrow{mi}$ e $w(x)v'$ precede $r(x)v'$ por $\xrightarrow{mi}$, então, pela Definição 4.3, $r(x)v \xrightarrow{lc'} w(x)v'$, onde $\xrightarrow{lc'}$ resulta da aplicação daquela definição à $\xrightarrow{\sigma'}$. Pela Definição 4.5, segue que $r(x)v \xrightarrow{lc} w(x)v'$. □

A partir da Proposição 1, podemos descrever o conceito introduzido na Definição 4.5 de maneira mais intuitiva, da seguinte forma. Se $r(x)v$ se relaciona com as demais operações sobre a localização x conforme na Definição 4.5 e se pelo menos uma das operações de escrita que "podem influenciar" $r(x)v$ precede, em $\xrightarrow{\sigma}$, uma operação de leitura que também envolve a localização x e um diferente valor v', então $r(x)v$ deve também preceder as escritas que "podem influenciar" aquele $r(x)v'$, uma vez que pretendemos considerar todas as múltiplas escritas de um mesmo valor para uma mesma localização de memória simultaneamente. Isto é ilustrado na Figura 4.4, onde um fragmento de uma execução é mostrado, sendo as arestas tracejadas representantes de pares em $\xrightarrow{lc}$ e as arestas sólidas representantes dos pares em $\xrightarrow{\sigma}$. As operações sobre x, nesta figura, se relacionam com a Proposição 1 de forma que $v = 0$ e $v' = 1, 2$.

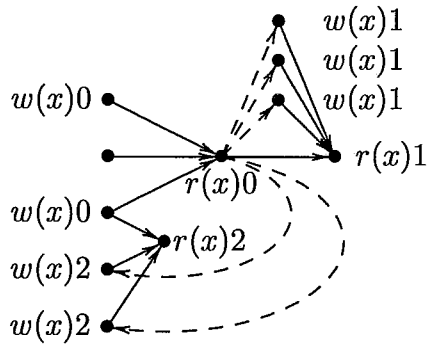


Figura 4.4: Pares devido à Restrição de Legalidade.

4.3.1 Execuções como Grafos Orientados e Grafos E–OU

O próximo passo para a nossa definição de legalidade sob múltiplas escritas é a definição de um novo grafo, denotado por $\hat{G}$, que representa uma dada execução estendida pela restrição de legalidade. Como definimos na Seção 4.1, se $(\Omega, \xrightarrow{\sigma})$ uma execução, esta pode ser representada pelo grafo $G(N, E)$ que tem como conjunto de vértices Ω e como arestas os pares de operações dados por $\xrightarrow{mi}$ e pela redução transitiva de $\xrightarrow{xo}$. Se, a este conjunto de arestas, adicionarmos os pares em $\xrightarrow{lc}$, então o grafo resultante é $\hat{G}$. Note que o grafo $\hat{G}$ já foi utilizado informalmente nas nossas ilustrações das definições de restrição de legalidade e da própria definição de legalidade quando consideramos a existência de escritas únicas (Figuras 4.2 e 4.3). Note que, pela semântica atribuída aos elementos de $\hat{G}$, este grafo deve ser considerado como um grafo E–OU, cujos grupos E de antecessores imediatos são caracterizados a seguir.

Em $\hat{G}$, os grupos E de antecessores imediatos de um vértice n_i são determinados de acordo com o tipo de operação que n_i representa. Recordando as notações introduzidas no Capítulo 3, para cada vértice n_i de $\hat{G}$, $P_i^1, \ldots, P_i^{t_i}$ representam os conjuntos E de antecessores imediatos de n_i. Se n_i é um vértice relativo a uma leitura, então cada conjunto em $P_i^1, \ldots, P_i^{t_i}$ tem cardinalidade 1 ou 2, conectando-se a n_i

por no máximo uma aresta $\xrightarrow{xo}$ e exatamente uma aresta $\xrightarrow{mi}$, sendo que existirão tantos conjuntos E distintos de antecessores imediatos quantas forem as escritas que "podem influenciar" esta leitura. Note que pode existir um número maior destes conjuntos E, uma vez que mais de um deles podem ser idênticos. Os grupos E de antecessores imediatos, quando n_i é um vértice relativo a uma escrita, são determinados de maneira semelhante à Definição 4.4 relativa à restrição de legalidade para múltiplas escritas (seção 4.5), conforme descrevemos a seguir.

Seja $\xrightarrow{\sigma'}$ uma redução para escritas únicas de $\xrightarrow{\sigma}$. Quando a Definição 4.4 é aplicada sobre $\xrightarrow{\sigma'}$, se n_i é um vértice relativo a uma escrita, as leituras que antecedem n_i através de arestas $\xrightarrow{lc}$ irão compor um grupo E de antecessores imediatos, juntamente com a leitura que se conecta a n_i através de uma aresta $\xrightarrow{xo}$, se esta existir. Desta forma, o número t_i destes grupos E depende de quantos grupos de antecessores podem ser produzidos, via Definição4.4, a partir de reduções de $\xrightarrow{\sigma}$ para escritas únicas. A Figura 4.5 ilustra estes grupos em fragmentos de $\hat{G}$.

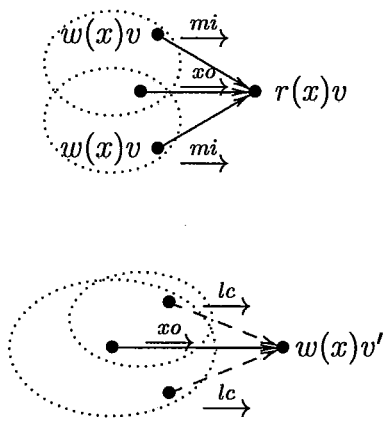


Figura 4.5: Grupos de antecessores e sucessores imediatos.

As elipses pontilhadas na Figura 4.5 indicam os diferentes conjuntos E de antecessores imediatos para um vértice relativo a uma leitura e para um vértice relativo a uma escrita. Apesar de não estar explícito na figura, ressaltamos que os antecessores imediatos de um vértice relativo a uma escrita só podem ser vértices relativos

a leituras, isto porque tanto em $\xrightarrow{xo}$ quanto em $\xrightarrow{lc}$, as operações que antecedem diretamente uma escrita são sempre leituras.

Antes de apresentarmos a definição de legalidade sob múltiplas escritas, necessitamos de relacionar os resultados da teoria de grafos E–OU obtidos no capítulo 3 com as características do grafo $\hat{G}$, associando as possíveis reduções para escritas únicas com c–subgrafos consistentes.

4.3.2 C–subgrafos Consistentes

Conforme definimos na Seção 3.2.1 do Capítulo 3, um c–subgrafo é dito consistente se ele pertence a um subconjunto dos possíveis c–subgrafos de um determinado grafo E–OU. Vamos, nesta seção, caracterizar o subconjunto dos possíveis c–subgrafos de $\hat{G}$ de interesse na definição de legalidade.

Seja $\hat{G}$ o grafo associado a uma execução $(\Omega, \xrightarrow{\sigma})$ e seja ϕ o subconjunto dos possíveis c–subgrafos de $\hat{G}$ em que estamos interessados. Um subgrafo $\hat{G}''$ de $\hat{G}$ é um c–subgrafo pertence a ϕ se existe uma redução para escritas únicas $\xrightarrow{\sigma'}$ (composta por $\xrightarrow{mi'}$ e $\xrightarrow{xo'}$) de $\xrightarrow{\sigma}$ tal que os pares relacionados por $\xrightarrow{mi'}$ são exatamente aqueles conectados pelas arestas relativas a $\xrightarrow{mi}$ escolhidas na formação de $\hat{G}''$ e tal que, quando aplicada a restrição de legalidade sobre $\xrightarrow{\sigma'}$, os pares relacionados por $\xrightarrow{lc'}$ são exatamente aqueles conectados pelas arestas relativas a $\xrightarrow{lc}$ escolhidas na formação de $\hat{G}''$.

Portanto, ϕ é o conjunto de c–subgrafos consistentes com possíveis reduções para escritas únicas de $(\Omega, \xrightarrow{\sigma})$. A Figura 4.6 mostra uma execução representada em forma de grafo E–OU. Note que este grafo não possui b-knot, o que, pelo Teorema 3.1, indica que existe um c–subgrafo gerador acíclico deste grafo, conforme verificamos na figura 4.7. Entretanto, os c–subgrafos da referida figura não são consistentes com

reduções para escritas únicas e todos aqueles c–subgrafos consistentes, mostrados na Figura 4.8, apresentam ciclo. Isto indica, conforme formalizaremos a seguir, que não é possível estender a execução correspondente para uma seqüência legal equivalente.

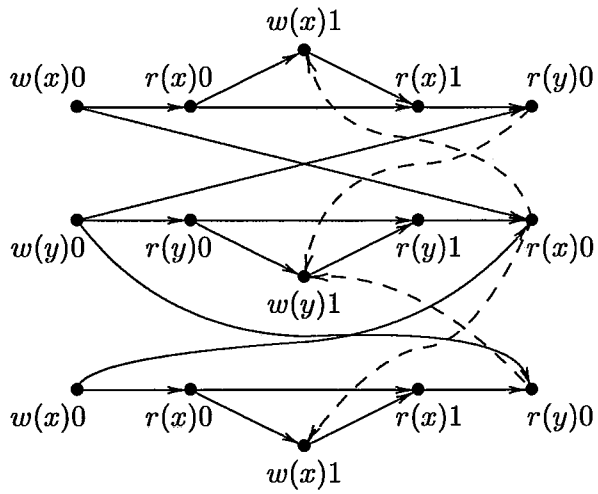


Figura 4.6: Grafo $\hat{G}$ contendo k-componente.

Podemos, agora, apresentar uma versão mais especializada do Teorema 3.2, utilizando a definição de reduções para escritas únicas. Observe que, pela própria definição de c–subgrafos consistentes, para cada um destes c–subgrafos, existe uma redução para escritas únicas correspondente e vice-versa. Vamos identificar cada redução para escritas únicas e, conseqüentemente, cada c–subgrafo consistente. Da mesma forma que na Seção 3.2.1, vamos denotar por P_i^k o conjunto E de antecessores imediatos do vértice n_i que participa do c–subgrafo k correspondente à redução para escritas únicas $\xrightarrow{\sigma'_k}$. Antes, porém, considere a seguinte definição de *k–componentes.*

Definição 4.6 (K–componente) *Seja C uma componente fortemente conexa não unitária de* $\hat{G}$. *C é uma* k–componente *se e somente se, dada uma redução para escritas únicas* $\xrightarrow{\sigma'_k}$, *para todo vértice* $n_i \in C$, P_i^k *intercepta C.*

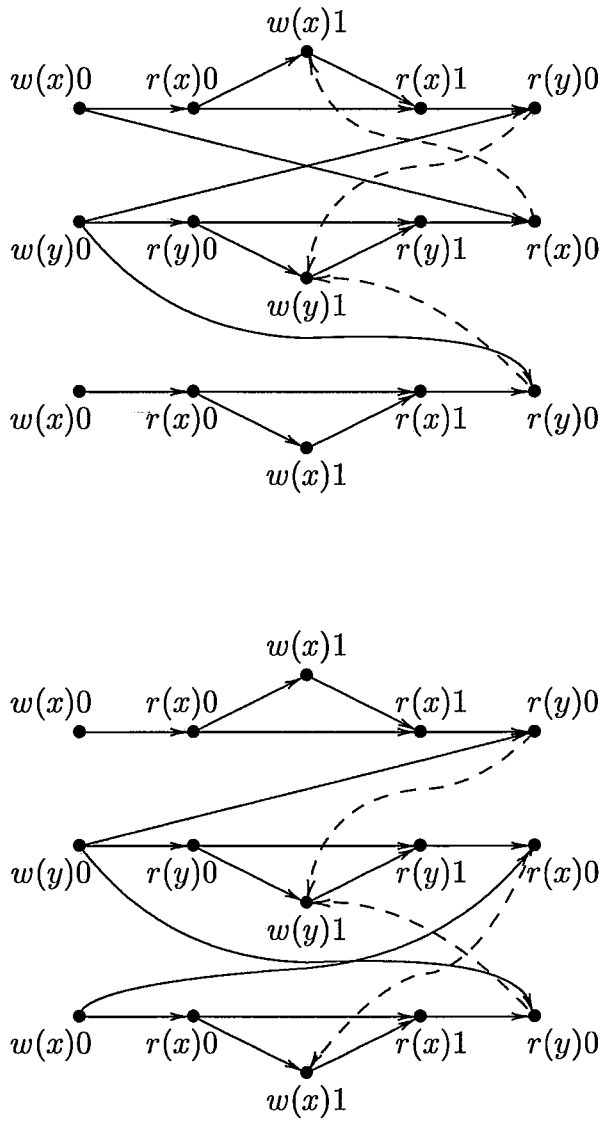


Figura 4.7: c–subgrafos consistentes de $\hat{G}$.

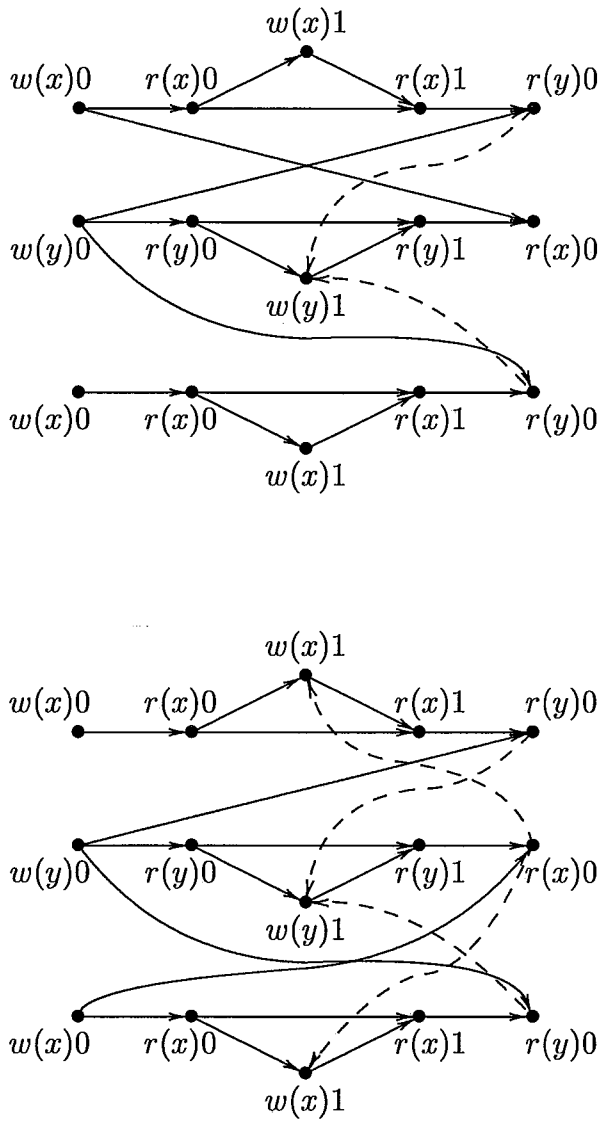


Figura 4.8: c–subgrafos inconsistentes de $\hat{G}$.

Teorema 4.2 *Todo c–subgrafo gerador consistente de $\hat{G}$ contém um ciclo orientado se e somente se para toda redução para escritas únicas $\xrightarrow{\sigma'_k}$, existe uma k–componente em $\hat{G}$.*

Prova. A prova é trivial, uma vez que o conjunto de reduções para escritas únicas estendidas pela restrição de legalidade define um subconjunto de c–subgrafos consistentes conforme a Definição 3.6 e, portanto, podemos usar o resultado do Teorema 3.2. □

Como $\hat{G}$ é, por definição, um grafo E–OU, podemos utilizar os resultados acima e aqueles da Seção 3.1 para definirmos a legalidade quando múltiplas escritas são permitidas.

Definição 4.7 (Legalidade com múltiplas escritas) $(\Omega, \xrightarrow{\sigma})$ *é legal se e somente se, para toda redução para escritas únicas $\xrightarrow{\sigma'_k}$, $\hat{G}$ não contém k–componentes.*

Esta definição pode ser vista como complementar à Definição 4.4 na presença de múltiplas escritas. Da mesma forma, o Teorema 4.3, apresentado a seguir, pode ser visto como complementar ao Teorema 4.1. A intuição neste teorema é que a legalidade de uma execução conforme estabelecida pela Definição 4.7 não diverge da usual noção de legalidade seqüencial.

Teorema 4.3 $(\Omega, \xrightarrow{\sigma})$ *é legal se e somente se existe uma redução para escritas únicas de $\xrightarrow{\sigma}$ que tem uma extensão linear legal.*

Prova. A prova é baseada na observação de que, para todo c–subgrafo gerador consistente de $\hat{G}$, existe uma relação correspondente que é uma redução para escritas únicas de $\xrightarrow{\sigma}$ e vice-versa. Tal relação é o fecho transitivo de todos os pares em $\xrightarrow{xo}$ ou no conjunto de pares $\xrightarrow{mi}$ que aparecem no c–subgrafo. O c–subgrafo, portanto,

está sujeito aos resultados da Seção 4.2. Se $(\Omega, \xrightarrow{\sigma})$ é legal, então pela Definição 4.7 $\hat{G}$ não contém k–componentes. Pelo Teorema 4.2, pelo menos um c–subgrafo gerador consistente de $\hat{G}$ é acíclico. Portanto, a correspondente redução para escritas únicas de $\xrightarrow{\sigma}$ tem, pelo Teorema 4.1, sob a Definição 4.4, uma extensão linear legal.

Por outro lado, se $(\Omega, \xrightarrow{\sigma})$ não é legal, então segue da Definição 4.7 que $\hat{G}$ contém uma k–componente, e pelo Teorema 4.2 que todos os c–subgrafos geradores consistentes de $\hat{G}$ contêm ciclos orientados. Novamente, pela Definição 4.4 e pelo Teorema 4.1, nenhuma redução para escritas únicas de $\xrightarrow{\sigma}$ tem uma extensão linear legal. □

Portanto, temos agora uma nova definição de legalidade, dada sobre uma ordem parcial de operações que representa uma execução e que admite a existência de múltiplas escritas de um mesmo valor em uma mesma localização compartilhada. Tal definição será utilizada na apresentação de condições de consistência no Capítulo 5.

4.4 Trabalhos Relacionados

Até o momento da escrita deste trabalho, tínhamos conhecimento apenas dos trabalhos de Mizuno [67] e Raynal [72] cuja abordagem no tratamento da legalidade se aproxima daquela descrita neste capítulo. Estes dois trabalhos são complementares e envolvem o mesmo formalismo no que diz respeito à definição de legalidade.

Neste ponto cabe observar que a definição de legalidade aqui introduzida está bastante relacionada com a condição para "histórias seqüencializáveis" definida em [67]. Entretanto, importantes diferenças existem entre aquela condição e a definição por nós introduzida. Por exemplo, em [67], a execução está sujeita a uma restrição na ordem de escritas (Write Order — WO) ou a uma restrição na ordem

de objetos (Object Order — OO). Além disso, as "arestas de exclusivas" ("exclusive edges"), apesar de definidas de maneira similar a nossa restrição de legalidade, não contemplam o caso no qual uma escrita precede uma leitura de outro valor, como conseqüência da restrição de ordenação de escritas. Portanto, a abordagem aqui apresentada é mais geral, além de propiciar uma extensão natural para lidar com a existência de múltiplas escritas de um mesmo valor, como vimos na seção 4.3. Este caso não é tratado nas referências acima citadas.

Repare agora a Figura 4.3. Esta execução foi apresentada como um exemplo de execução não legal pela Definição 4.4. Entretanto, se analisarmos as precedências do exemplo utilizando a definição de Mizuno e Raynal, concluiremos que esta execução seria classificada como legal. Este exemplo, portanto, realça as diferenças entre as duas abordagens.

Capítulo 5

Condições de Consistência

Neste capítulo, utilizaremos a teoria da legalidade desenvolvida nos capítulos anteriores para expressar algumas das principais condições (modelos) de consistência propostas na literatura. De um modo geral, a caracterização de cada condição de consistência seguirá a mesma estrutura, na qual, dada uma execução, identifica-se o subconjunto de interesse das operações da execução original, sendo que deve existir uma execução legal que retorne os mesmos valores da execução original e que apresente um conjunto particular de relações entre estas operações, característico de cada condição de consistência.

A grande dificuldade de se expressar as várias condições de consistência existentes em um formalismo único está justamente no fato de que cada modelo é expresso originalmente com a utilização de formalismos próprios ou mesmos de maneira informal. Procuramos, neste capítulo, seguir a intuição básica que deu origem a cada condição e, portanto, não consideraremos os detalhes específicos devidos à formalização própria de cada definição.

Antes, porém, considere as definições a seguir relativas a execuções equivalentes e parcialmente equivalentes, que serão úteis na apresentação dos diversos modelos de consistência. Estas definições têm o intuito de formalizar a noção de duas execuções

nas quais as operações de leitura comuns retornam os mesmos valores.

Definição 5.1 (Execução Equivalente) *Seja* $(\Omega, \xrightarrow{\sigma})$ *uma execução.* $(\Omega', \xrightarrow{\sigma'})$ *é dita ser equivalente à* $(\Omega, \xrightarrow{\sigma})$ *se e somente se as seguintes condições são satisfeitas:*

(i) $\Omega \subseteq \Omega'$;

(ii) $\Omega' \setminus \Omega$ *contém apenas operações de leitura;* [1]

(iii) Toda leitura em $\Omega \cap \Omega'$ *retorna o mesmo valor em ambas as execuções;*

Definição 5.2 (Execução Parcialmente Equivalente) *Seja* $(\Omega, \xrightarrow{\sigma})$ *uma execução.* $(\Omega', \xrightarrow{\sigma'})$ *é dita ser parcialmente equivalente à* $(\Omega, \xrightarrow{\sigma})$ *se e somente se as seguintes condições são satisfeitas:*

(i) $\Omega \cap \Omega' \neq \emptyset$;

(ii) $\Omega' \setminus (\Omega \cap \Omega')$ *contém apenas operações de leitura;*

(iii) Toda leitura em $\Omega \cap \Omega'$ *retorna o mesmo valor em ambas as execuções;*

Definição 5.3 (Subexecução) *Seja* $(\Omega', \xrightarrow{\sigma'})$ *uma execução parcialmente equivalente à* $(\Omega, \xrightarrow{\sigma})$. $(\Omega', \xrightarrow{\sigma'})$ *é dita ser uma subexecução de* $(\Omega, \xrightarrow{\sigma})$ *se e somente se* $\Omega' \subseteq \Omega$ *e* $\xrightarrow{\sigma} \subseteq \xrightarrow{\sigma'}$.

Dadas estas definições básicas, passamos a apresentar formalmente algumas condições de consistência propostas. A relação das condições de consistência a seguir não representa uma lista exaustiva de todas aquelas existentes. Procuramos apresentar aquelas de maior relevância, tanto do ponto de vista teórico como de implementação, visando ilustrar a adequação da teoria proposta neste trabalho para a especificação de condições de consistência.

[1] $\setminus$ denota diferença de conjuntos.

5.1 Consistência Seqüencial

Em uma máquina monoprocessada, os resultados obtidos com a execução de um programa qualquer são sempre consistentes com a execução seqüencial (seguindo a ordem estabelecida pelo código do programa) de cada instrução deste programa, mesmo que esta ordem não seja a que realmente ocorreu durante a execução. Em uma máquina multiprocessada, a maneira mais intuitiva de se admitir a ordem de execução das operações é considerar que cada processador se comporta como uma máquina monoprocessada e que os resultados obtidos são consistentes com um entrelaçamento das seqüências de operações executadas por cada processador. Esta intuição foi formalizada por Lamport [56] conforme transcrito abaixo, definindo o modelo de Consistência Seqüencial.

Definição 5.4 (Consistência Seqüencial por Lamport) *Um multiprocessador é seqüencialmente consistente se o resultado de qualquer execução é o mesmo daquele obtido se as operações de todos os processadores fossem executadas em alguma ordem seqüencial e se as operações de cada processador aparecem nesta seqüência na ordem especificada pelo seu programa.*

O modelo de Consistência Seqüencial, na prática, é o modelo que impõe mais restrições com relação aos possíveis resultados de computações paralelas sobre memória compartilhada. Outro modelo bastante restritivo é conhecido por *linearizabilidade* [45]. Este último modelo acrescenta às restrições da Consistência Seqüencial a condição de que a ordem de tempo-real entre as operações deve ser respeitada nas seqüências equivalentes admitidas. Esta restrição torna inviável, em termos de desempenho, a implementação de tal modelo.

Para definirmos Consistência Seqüencial, vamos utilizar o conceito de legalidade conforme a Definição 4.7. Pelo Teorema 4.3, temos que uma execução legal possui uma extensão linear legal. Entretanto, isso não é suficiente para classificarmos a execução como seqüencialmente consistente, uma vez que a ordem das operações de um mesmo processo nesta seqüência pode não ser consistente com a ordem de programa. A Definição 5.5, apresentada a seguir, contempla também esta condição.

Definição 5.5 (Consistência Seqüencial) $(\Omega, \xrightarrow{\sigma})$ *é seqüencialmente consistente se e somente se existe uma execução legal equivalente* $(\Omega', \xrightarrow{\sigma'})$ *tal que* $\xrightarrow{\sigma'}$ *é uma ordem total consistente com* $\xrightarrow{po}$ *quando restrita a um único processo.*

Note que a condição de equivalência entre execuções apresentada na introdução deste capítulo estende o conjunto de operações Ω através da inclusão de leituras "artificiais" com o intuito de assegurar que a relação $\xrightarrow{\sigma'}$ resultante é uma ordem total, dado um processo, consistente com $\xrightarrow{po}$. Claramente, em tal ordem total, operações de escrita alternam com grupos de leituras, dos quais a que imediatamente segue uma operação de escrita $w(x)v$ é uma operação de leitura $r(x)v$. Esta leitura pode ou não ser uma das "artificialmente" incluídas, dependendo se uma leitura apropriada já existe ou não em Ω.

Para ilustrar esta técnica, seja o exemplo da Figura 1.1. Vamos supor que aquele programa deu origem à execução ilustrada na parte (a) da Figura 5.1. Podemos estender esta execução com a adição de leituras artificiais de modo a obter uma ordem total entre as operações de cada processo. Esta extensão é ilustrada na parte (b) da Figura 5.1.

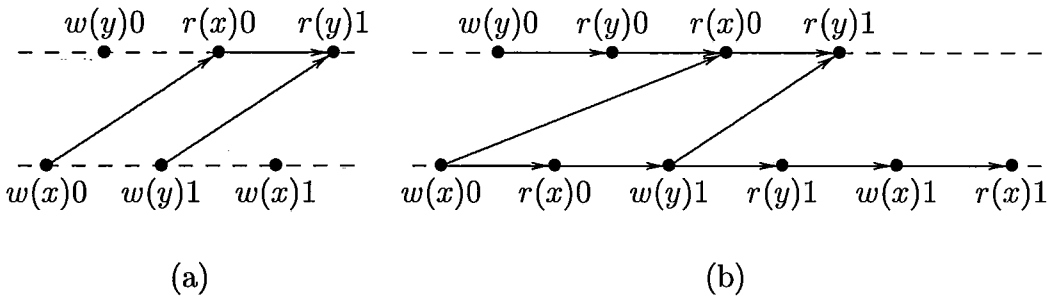


Figura 5.1: Execução e execução expandida equivalente.

5.1.1 Condições para Consistência Seqüencial

A Definição 5.5 apresenta uma formalização para a Consistência Seqüencial baseada na modelagem de execuções. Nesta seção, vamos abordar o problema do ponto de vista oposto, ou seja, a formalização das condições impostas para um sistema ser seqüencialmente consistente e a verificação da corretude destas condições sobre o modelo de execução adotado.

No trabalho original onde o modelo de consistência seqüencial foi descrito, Lamport [56] também estabeleceu condições para que uma implementação produzisse somente execuções seqüencialmente consistentes.

Condição 5.1 (Consistência Seqüencial. Condições de Lamport)

(a) Cada processo emite os pedidos de execução de operações sobre a memória compartilhada na ordem especificada pelo seu programa;

(b) Pedidos de acesso à memória compartilhada emitidos por todos os processos para um mesmo módulo de memória são servidos a partir de uma fila FIFO. Emitir um pedido de acesso à memória corresponde a incluir este pedido nesta fila.

Ao estabelecer estas condições, Lamport considerou a existência de uma memória

compartilhada dividida em módulos únicos nos sistema. Entretanto, para arquiteturas com caches ou com localizações de memória replicadas em mais de um módulo, estas condições não são suficientes e, portanto, não podem ser aplicadas.

Scheurich e Dubois [73] propuseram condições mais gerais que suportam sistemas com cache e replicação. A diferença básica destas especificações está na suposição de atomicidade das escritas. Nas condições propostas em [73] e descritas a seguir, as escritas não são consideradas atômicas, em contraste com a suposição nas condições de Lamport.

Condição 5.2 (Condições de Scheurich e Dubois)

(a) Cada processo emite os pedidos de execução de operações sobre a memória compartilhada na ordem especificada pelo seu programa;

(b) Depois que uma operação de escrita é emitida, o processador espera até que esta termine antes de emitir a próxima operação;

(c) Depois que uma operação de leitura é emitida, o processador espera até que esta termine e que a escrita do valor que está sendo retornado termine antes de emitir a próxima operação;

(d) Escritas para uma mesma localização são serializadas na mesma ordem com relação a todos os processos.

Vamos formalizar as condições acima com base no modelo introduzido no Capítulo 2. Primeiramente, precisamos formalizar a noção de término de uma operação. Seja op uma operação iniciada em um processador qualquer. Se $op = r(x)v$, a ocorrência do evento de término desta operação carrega a mesma semântica adotada nas condições acima, ou seja, na ocorrência de $e[r(x)v]$, o evento de busca $b[r(x)v]$

já ocorreu e o valor de retorno foi definido. Caso $op = w(x)v$, conforme definido no Capítulo 2, a ocorrência do evento $e[w(x)v]$ não significa o término desta operação. Portanto, é necessário relacionar a ocorrência de todos os eventos $p_i[w(x)v]$ com $s[w(x)v]$ e $e[w(x)v]$, a fim de capturar o conceito expresso na Condição 5.2.

Seja $(\Omega, \xrightarrow{\sigma})$ uma execução e op, $r(x)v$ e $w(x)v$ operações pertencentes a Ω. As condições de Scheurich e Dubois descritas anteriormente podem ser modeladas da seguinte forma, utilizando-se as relações $\prec$ e $\prec_p$ entre eventos:

Condição 5.3 (Condição de Scheurich e Dubois revisada)

(a) *Se* $op \xrightarrow{po} op'$, *então* $s[op] \prec s[op']$;

(b) *Se* $w(x)v \xrightarrow{po} op$, *então* $p_i[w(x)v] \prec_p e[w(x)v] \prec_p s[op], \forall i \in P(x)$;

(c) *Se* $w(x)v \xrightarrow{mi} r(x)v \xrightarrow{po} op$, *então* $p_i[w(x)v] \prec_p e[r(x)v] \prec_p s[op], \forall i \in P(x)$;

(d) *Se* $p_i[w(x)v] \prec_p p_i[w(x)v']$, *então* $p_j[w(x)v] \prec_p p_j[w(x)v'], \forall j \in P(x)$.

A seguir, vamos mostrar que a Condição 5.3 é suficiente para que um sistema gere somente execuções seqüencialmente consistentes conforme a Definição 5.5. Para tal, seja $(\Omega, \xrightarrow{\sigma})$ uma execução qualquer produzida por uma determinada implementação que atende à Condição 5.3. A Proposição 5.1, enunciada a seguir, estabelece que $(\Omega, \xrightarrow{\sigma})$ é seqüencialmente consistente.

Proposição 5.1 *Se a Condição 5.3 é satisfeita, então* $(\Omega, \xrightarrow{\sigma})$ *é seqüencialmente consistente conforme a Definição 5.5.*

Prova. Seja $(\Omega', \xrightarrow{\sigma'})$ uma execução equivalente à $(\Omega, \xrightarrow{\sigma})$ conforme a Definição 5.1 e $\hat{G}'$ o grafo estendido pela restrição de legalidade derivado de $\xrightarrow{\sigma'}$. Sejam também $\prec'$ e $\prec'_p$ as relações entre eventos desta execução equivalente. Vamos supor

que a todo par de eventos relativos a operações de Ω e relacionados por $\prec$ ou $\prec_p$ corresponda um par de eventos relativos às mesmas operações de Ω' e relacionados da mesma forma por $\prec'$ ou $\prec'_p$. Isto é sempre possível uma vez que, pela Condição 5.3, $\xrightarrow{xo}$ nunca contraria $\xrightarrow{po}$ e, portanto, existe uma execução equivalente que mantém as relações originais entre eventos e que produz uma ordem de execução ($\xrightarrow{xo'}$) consistente com $\xrightarrow{po}$. Além disso, toda operação de leitura *op* artificialmente introduzida para formar Ω' é tal que nenhum evento ϵ relativo a outra operação no mesmo processo é tal que $s[op] \prec' \epsilon \prec' e[op]$. Portanto, a Condição 5.3 também é válida para $(\Omega', \xrightarrow{\sigma'})$. Vamos mostrar que $(\Omega', \xrightarrow{\sigma'})$ é legal e, portanto, $(\Omega, \xrightarrow{\sigma})$ é seqüencialmente consistente.

Pelos Teoremas 4.2 e 4.3, $(\Omega', \xrightarrow{\sigma'})$ é legal se e somente se existe um c–subgrafo acíclico consistente de $\hat{G}'$. Vamos supor que todos os c–subgrafos consistente de $\hat{G}'$ contenham ciclos. Seja $\hat{G}''$ um destes c–subgrafos e $\xrightarrow{\sigma''}$ a redução para escritas únicas associada a $\hat{G}''$. Usando o mesmo raciocínio da prova do Teorema 4.1, se $\hat{G}''$ possui ciclo, então existem localizações x e y e valores v_x e v'_x, v_y, e v'_y, tais que

(a) $w(x)v_x \xrightarrow{\sigma''} r(x)v'_x \xrightarrow{\sigma''} r(y)v_y$ e

(b) $w(y)v_y \xrightarrow{\sigma''} r(y)v'_y \xrightarrow{\sigma''} r(x)v_x$.

Para simplificar a notação, vamos supor, sem perda da generalidade, que $v_x = 0$, $v'_x = 1$, $v_y = 2$ e $v'_y = 3$. Portanto,

(a) $w(x)0 \xrightarrow{\sigma''} r(x)1 \xrightarrow{\sigma''} r(y)2$ e

(b) $w(y)2 \xrightarrow{\sigma''} r(y)3 \xrightarrow{\sigma''} r(x)0$.

Vamos mostrar que $s[r(x)0] \prec'_p e[r(x)1]$ e $s[r(y)2] \prec'_p e[r(y)3]$ caracterizando um ciclo em $\prec'_p$. Inicialmente, podemos concluir que existe $r'(x)0 \in \Omega'$ tal

que $w(x)0 \xrightarrow{mi''} r'(x)0 \xrightarrow{\sigma''} r(x)1$. Se $b[r'(x)0]$ ocorre no processador i, então $p_i[w(x)0] \prec_p b[r'(x)0]$. Como $b[r'(x)0] \prec_p e[r'(x)0]$, pelos itens (b) e (c) da Condição 5.3, $e[r'(x)0] \prec_p s[r(x)1]$ e $p_k[w(x)0] \prec_p s[r(x)1], \forall k \in P(x)$. Como $s[r(x)1] \prec_p b[r(x)1]$, supondo que $b[r(x)1]$ ocorre no processador j, $p_j[w(x)1] \prec_p b[r(x)1]$ e, portanto $p_j[w(x)0] \prec_p p_j[w(x)1] \prec_p b[r(x)1]$. Pelo item (d) da Condição 5.3, $p_k[w(x)0] \prec_p p[w(x)1], \forall k \in P(x)$. Supondo agora que $r(x)0$ ocorre no processador i, em $F_i(x)$, $p_i[w(x)0] \prec'_p b[r(x)0] \prec'_p p_i[w(x)1]$ e, portanto, $s[r(x)0] \prec'_p p_i[w(x)1]$. Analogamente, $s[r(y)2] \prec'_p p_i[w(y)3]$.

Pelo item (c) da Condição 5.3, $p_i[w(x)1] \prec'_p e[r(x)1]$ e, portanto, $s[r(x)0] \prec'_p e[r(x)1]$. De modo simétrico, $s[r(y)2] \prec'_p e[r(y)3]$, caracterizando um ciclo em $\prec'_p$. Como $\prec'_p$ é uma ordem parcial, temos uma contradição. □

5.2 Consistência Causal

O modelo de Consistência Seqüencial é o único dentre os apresentados neste capítulo que implica na existência de uma seqüência de operações legal e consistente do ponto de vista de todos os processos. O modelo de Consistência Causal, definido originalmente por Ahamad et al. [9, 10, 50, 11, 12, 70, 71, 72], é o modelo mais próximo da Consistência Seqüencial e que, basicamente, relaxa esta condição.

A intuição neste modelo é que, para uma grande classe de aplicações, não é necessária a existência de uma única seqüência de operações consistente com todas as precedências derivadas da ordem de processo e das relações entre escritas e leituras. Ao contrário, bastaria que as relações de causalidade derivadas desta ordens fossem preservadas do ponto de vista de cada processo, separadamente. Formalmente temos a seguinte definição:

Definição 5.6 (Consistência Causal) $(\Omega, \xrightarrow{\sigma})$ *é causalmente consistente se e*

somente se existe uma execução $(\Omega', \xrightarrow{\sigma'})$ *equivalente a* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(i) $\xrightarrow{\sigma'}$*, quando restrita às operações de um único processo é uma ordem total consistente com* $\xrightarrow{po}$*;*

(ii) Para cada processo P_i *existe uma subexecução* $(\Omega'', \xrightarrow{\sigma''})$ *de* $(\Omega', \xrightarrow{\sigma'})$ *legal e tal que* Ω'' *contém apenas as operações relativas a* P_i *e as operações de escrita dos demais processos.*

O item (*i*) da Definição 5.6 estabelece que os resultados obtidos na execução real sejam os mesmos obtidos por uma execução que mantivesse a ordem de programa, de maneira análoga à definição de Consistência Seqüencial. Entretanto, no caso da Consistência Causal, esta execução coerente com a ordem de programa não é necessariamente legal. O que o item (*ii*) define é, de certa forma, uma legalidade do ponto de vista de cada processo, quando apenas um subconjunto das operações é considerado, no caso o subconjunto das operações do processo em questão e as operações de escrita dos demais (Ω''). Como conseqüência, para cada processo, existe uma serialização legal das operações de Ω'' que é consistente com a ordem de programa e com a ordem (causal) entre operações de escrita de processos diferentes. A Figura 5.2 ilustra uma execução causalmente consistente e que não atende às restrições da Consistência Seqüencial.

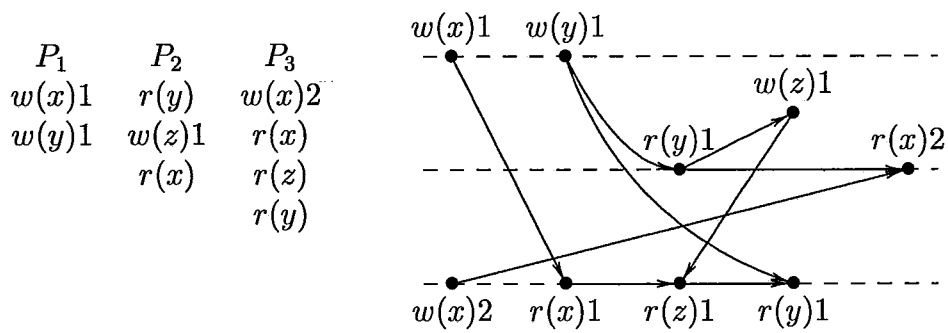


Figura 5.2: Execução causalmente consistente.

A execução apresentada na Figura 5.2 é causalmente consistente porque existe uma execução equivalente (Figura 5.3) tal que todas as subexecuções relativa a cada processo são legais. A Figura 5.3 mostra também a subexecução para o processo P_3, onde a aresta tracejada corresponde ao único par relacionado pela restrição de legalidade (arestas tracejadas) nesta subexecução [2]. É fácil verificar que as subexecuções para os demais processos também são legais.

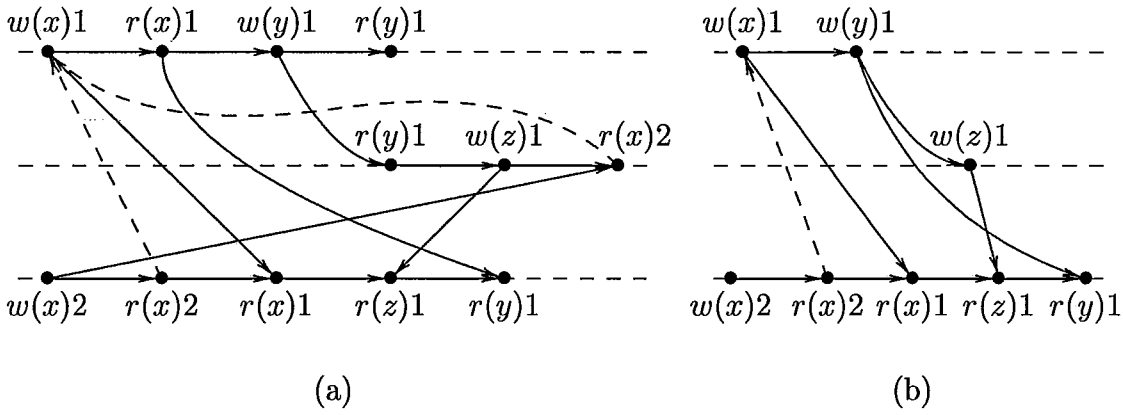


Figura 5.3: Execução e subexecução equivalentes.

5.3 Coerência

Esta condição de consistência é derivada dos protocolos de consistência de cache e estabelece que cada localização de memória deve passar por uma única seqüência de estados do ponto de vista de qualquer dos processos envolvidos. Esta noção pode ser formalizada utilizando-se a definição de legalidade e o modelo introduzido no Capítulo 2 da seguinte forma.

Definição 5.7 (Coerência) $(\Omega, \xrightarrow{\sigma})$ *é uma execução coerente se e somente se para cada localização compartilhada* x *existe uma execução* $(\Omega', \xrightarrow{\sigma'})$ *legal e parcial-*

[2]Na parte (a) da Figura 5.3. aparecem também os arcos devido à restrição de legalidade. Note que esta execução não é legal, uma vez que o grafo expandido contém ciclo.

mente equivalente à $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(i) Ω' *contém apenas as operações sobre a localização* x*;*

(ii) $\xrightarrow{\sigma'}$*, quando restrita às operações de um único processo é uma ordem total consistente com* $\xrightarrow{po}$*.*

De maneira semelhante à definição de Consistência Causal, onde um certo tipo de legalidade por processo era definida, a Coerência exige uma legalidade por localização. Esta característica, recentemente, motivou Frigo [33] a definir este modelo com o nome de "Consistência por Localização", apesar deste nome ter sido utilizado anteriormente para denominar o modelo proposto por Gao e Sarkar [34] (Seção 5.8). A coerência é, inclusive, defendida por Frigo como o modelo mais fraco de consistência "razoável" de se adotar em uma arquitetura paralela.

Um sistema que mantém as localizações de memória coerentes não implementa necessariamente um sistema seqüencialmente consistente ou causalmente consistente. A Figura 5.4 apresenta uma execução coerente mas que não pode ser classificada como seqüencialmente consistente ou causalmente consistente. O motivo desta diferença está no poder de expressão da condição de coerência que somente trata das relações entre operações envolvendo uma mesma localização compartilhada.

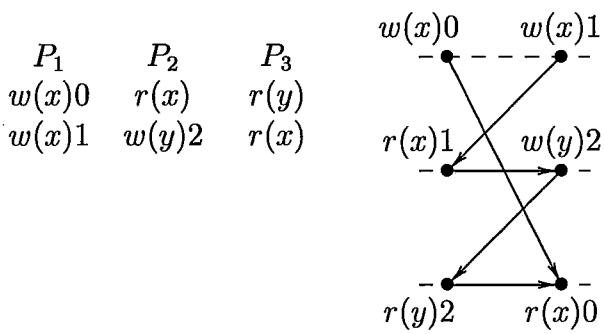


Figura 5.4: Exemplo de execução Coerente.

A Figura 5.5, a seguir, mostra uma execução parcialmente equivalente àquela exibida na Figura 5.4 relativa à localização x, já estendida pela restrição de legalidade e consistente com a ordem de programa. É fácil verificar que esta execução é legal, uma vez que só existe uma operação de escrita para cada valor lido e o grafo correspondente é acíclico. A subexecução para localização y é trivialmente legal, uma vez que apenas um único valor é envolvido em todas as operações sobre esta localização.

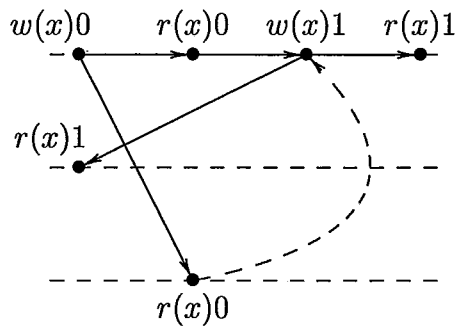


Figura 5.5: Execução legal parcialmente equivalente relativa à x.

5.4 Consistência por Processador

A intuição básica neste modelo, e compartilhada pelo modelo PRAM [63], é que apenas as operações iniciadas por um mesmo processo precisam ter garantias de que terão seus efeitos ocorridos na mesma ordem em todos os processos. Sendo assim, leituras de processos distintos podem retornar valores compatíveis com diferentes ordens para escritas iniciadas em processos também distintos.

A definição original deste modelo de consistência foi estabelecida de maneira informal por Goodman [42, 43], baseada na intuição descrita anteriormente. Segundo esta definição, uma execução é Consistente por Processador se

[...]*os resultados são os mesmos obtidos se as operações de cada processo aparecem na ordem seqüencial especificada por seu programa.*

Na interpretação mais comum na literatura, esta definição implica que, para cada processo, deve ser possível ordenar totalmente, de maneira legal e consistente com a ordem de programa de todos os processos, as operações de uma dada execução. Como as operações de leitura são estritamente locais, para cada processo, tal ordenação deve incluir as operações deste processo e as operações de escrita dos demais. A diferença conceitual básica entre este modelo e a Consistência Causal (Seção 5.2) está no fato de que a Consistência por Processador não exige que a ordem causal entre operações (escritas) de processos distintos seja mantida nas referidas ordenações para cada processo. Também é comum considerar que a definição original só tem sentido se a memória é mantida coerente (Seção 5.3). Temos, portanto, uma forma de legalidade por processo em conjunto com uma legalidade por localização.

Esta noção pode ser formalizada utilizando o modelo exposto nos capítulos anteriores da seguinte forma:

Definição 5.8 (Consistência por Processador) $(\Omega, \xrightarrow{\sigma})$ *é consistente por processador se e somente se as seguintes condições são satisfeitas:*

(i) $(\Omega, \xrightarrow{\sigma})$ *é coerente;*

(ii) Para cada processo P_i*, existe uma execução* $(\Omega', \xrightarrow{\sigma'})$ *legal e parcialmente equivalente à* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(ii-a) $\Omega \cap \Omega'$ *contém apenas as operações relativas a* P_i *e as operações de escrita dos demais processos;*

(ii-b) $\xrightarrow{\sigma'}$*, quando restrita a um único processo, é uma ordem total consistente com* $\xrightarrow{po}$*;*

(iii) A legalidade de cada execução $(\Omega', \xrightarrow{\sigma'})$ *deve ser consistente com a condição de coerência do item (i).*

A condição de que $(\Omega, \xrightarrow{\sigma})$ seja coerente, expressa no item *(i)*, não pode ser derivada da definição informal de Goodman. Entretanto, conforme observado por Ahamad [8], a classificação de Goodman para a Consistência por Processador como mais forte que a coerência e os exemplos por ele apresentados, implicam na necessidade desta condição.

O item *(iii)* da Definição 5.8 também é necessário pelo seguinte fato. Se $(\Omega, \xrightarrow{\sigma})$ é coerente, existe pelo menos uma serialização para cada localização, o que define uma ordem para as escritas relativas a cada localização. Entretanto, o item *(ii)* também implica em uma ordem entre as escritas para cada execução envolvendo as operações em $\Omega \cap \Omega'$, devido à legalidade. Portanto, deve existir uma ordem entre as escritas de cada localização que satisfaça simultaneamente as condições *(i)* e *(iii)*. A Figura 5.6 ilustra uma execução onde as condições *(i)* e *(ii)* são atendidas separadamente, mas que não é considerada consistente por processador.

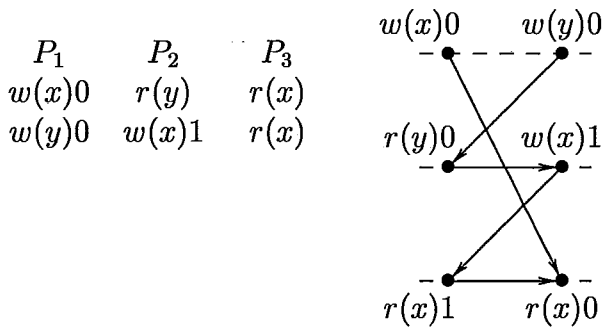
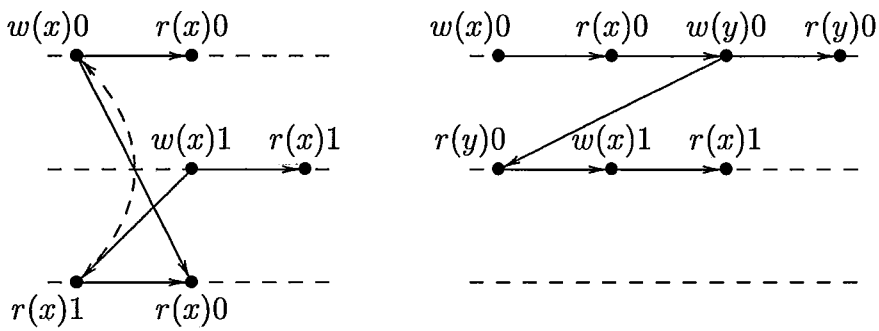


Figura 5.6: Execução coerente mas não consistente por processador.

A execução da Figura 5.6 é coerente porque existem execuções legais parcialmente equivalentes relativas às localizações x e y. A Figura 5.7 mostra, na parte (a), a execução parcialmente equivalente relativa à localização x (a execução relativa à localização y é trivial). A parte (b) da Figura 5.7 exibe uma execução parcialmente equivalente relativa ao processo P_2. Note que as relações entre as operações sobre a

localização x não são consistentes entre si. Nas possíveis serializações da execução na parte (a) da referida figura, as operações sobre x envolvendo o valor 1 devem preceder as operações sobre x envolvendo o valor 0, enquanto que nas possíveis serializações da execução na parte (b) tais operações devem ser ordenadas de maneira oposta.



(a) (b)

Figura 5.7: Execuções parcialmente equivalentes relativas a x e P_2.

5.4.1 Definição de Gharachorloo

Baseado na mesma intuição, Gharachorloo [39, 38] estabeleceu as seguintes regras suficientes para que uma implementação produza execuções consistentes por processador.

Condição 5.4 (Condições para a Consistência por Processador)

(a) Antes que uma operação de leitura possa ser iniciada, todas as operações de leitura anteriores (na ordem de programa) devem ter sido executadas;

(b) Antes que uma operação de escrita possa ser iniciada, todas as operações de escrita e leituras anteriores (na ordem de programa) devem ter sido executadas.

Alguns aspectos da Condição 5.4 devem ser ressaltados. Primeiramente, as condições são dadas com base no formalismo de Dubois e Scheurich [73, 30] (Seção 2.3)

que, conforme observado pelo próprio autor [37], trata-se de um formalismo pouco preciso, e, por isso, não revela todas as relações entre operações conseqüentes desta condição. Em segundo lugar, deve-se considerar que as regras de implementação correspondem a uma condição suficiente e não necessariamente implicam em uma implementação eficiente da intuição que as motivou. A restrição original relativa à manutenção da ordem das operações de um mesmo processo é atendida pelo item *(b)*. Ao exigir a espera de todos os eventos das escritas anteriores terminem antes do início de uma nova escrita, esta regra assegura a ordem de programa entre tais operações do ponto de vista de todos os processos. Além disso, como tais escritas não mais precisam ser atômicas, esta regra exige apenas que leituras anteriores a uma escrita tenham sido executadas e não "globalmente executadas" como na Condição 5.2. Já o item *(a)*, na verdade, relaxa a condição original de manutenção da ordem de programa entre todas as operações de um mesmo processo. Este enfraquecimento da ordem de programa se deve à observação de que a ordem entre operações sobre localizações diferentes pode ser relaxada sem comprometer a semântica do modelo [76, 1, 37].

Uma tradução para a Condição 5.4 foi proposta por Ahamad et al. [8, 54]. Nesta proposta as regras de atraso de operações são traduzidas para uma relação de ordem parcial denominada pelos autores de "semi-causalidade". O modelo seria então definido de maneira análoga à Definição 5.8 (definição original de Consistência por Processador), exceto que as execuções parcialmente equivalentes relativas a cada processo deveriam ser consistentes com esta "semi-causalidade". Entretanto, não existe prova que tal definição é realmente equivalente às regras propostas por Gharachorloo.

5.5 Consistência Fraca (*Weak Ordering*)

O modelo de Consistência Fraca (ou ordenação fraca) foi proposto por Dubois, Scheurich e Briggs [30]. Este trabalho foi o primeiro a explorar a classificação de operações como acessos a dados ou operações de sincronização. Baseado no fato que a maioria dos programas concorrentes são escritos utilizando-se instruções de sincronização para coordenar o acesso a localizações compartilhadas, o modelo de Consistência Fraca propõe que a ordenação entre operações seja garantida apenas entre pares que envolvam pelo menos uma operação de sincronização.

A definição original é baseada na noção operações "executadas"e "globalmente executadas" [73, 30], termos estes definidos na Seção 2.3. Também utilizaremos, na condição apresentada a seguir, o termo "acesso a dados" como acessos a localizações compartilhadas que não são utilizadas para sincronização.

Condição 5.5 (Consistência Fraca)

(a) Acessos a localizações de sincronização são seqüencialmente consistentes;

(b) Nenhum acesso a uma localização de sincronização é emitido por um processador antes que todos os seus acessos a dados anteriores tenham sido "executados globalmente";

(c) Nenhum acesso a dados é emitido por um processador antes que todos os seus acessos a localizações de sincronização tenham sido "executados globalmente".

Estas regras levam a uma implementação de uma condição de consistência claramente mais fraca que a Consistência Seqüencial, uma vez que permite que operações entre as regiões definidas pelas operações de sincronização sejam executadas em paralelo. A intuição é que, garantindo a ordenação somente nos pontos de sincronização, resultados corretos (i.e. seqüencialmente consistentes) são gerados para a maioria dos programas.

A seguir, apresentamos nossa interpretação para a Condição 5.5, expressa nos termos da modelagem aqui proposta. Para identificar os tipos de acesso, seja y uma localização utilizada para sincronização e x uma localização compartilhada qualquer. Vamos denotar por $w_s(y)$ e $r_s(y)$ operações de escrita (e.g. liberação de um *lock*) e leitura (e.g. aquisição de um *lock*) relativa a sincronizações, respectivamente. Caso o tipo (leitura ou escrita) não seja relevante, denotaremos tais operações simplesmente por op_s e op.

Definição 5.9 (Ordenação Fraca) $(\Omega, \xrightarrow{\sigma})$ *é fracamente consistente se e somente se as seguintes condições são satisfeitas:*

(i) $(\Omega, \xrightarrow{\sigma})$ *é coerente;*

(ii) Existe uma execução $(\Omega', \xrightarrow{\sigma'})$ *legal e parcialmente equivalente à* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(ii-a) $\Omega' \cap \Omega$ *inclui apenas operações de sincronização;*

(ii-b) $\xrightarrow{\sigma'}$ *é consistente com* $\xrightarrow{po}$ *quando restrita a um processo;*

(iii) Para cada processo P_i*, existe uma execução* $(\Omega_i, \xrightarrow{\sigma_i})$ *legal e equivalente à* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(iii-a) $\xrightarrow{\sigma_i}$ *é consistente com* $\xrightarrow{po}$ *quando restrita a* P_i*;*

(iii-b) $\xrightarrow{\sigma_i}$ *é consistente com* $\xrightarrow{po}$ *para todo par de operações que envolve pelo menos uma operação de sincronização;*

(iv) As condições de legalidade dos itens (ii) e (iii) devem ser consistentes entre si e com a condição de coerência do item (i).

Vamos utilizar a Figura 5.6 para ilustrar uma execução que não atende Definição 5.9. Para tal, sejam $w(y)0$ e $r(y)0$ operações de sincronização. Pode-se verificar que não é possível exibir uma execução relativa ao processo P_3 que atenda aos itens (*ii*) e (*iii*) da Definição 5.9. Entretanto, se tais operações não fossem classificadas como

de sincronização, a execução da Figura 5.6 poderia ser classificada como fracamente ordenada.

A observação do exemplo anterior leva à questão de como classificar as operações de maneira a se produzir execuções sempre corretas (i.e. seqüencialmente consistentes). Adve [1, 5, 4, 6] e Gharacholoo [37, 3, 39] propuseram diversos modelos de programação (*DRF0–1 — Data Race Free, PL1–2–3 — Properly Labeled 1, 2 e 3*) visando a classificação adequada das operações de sincronização. Nestas abordagens, um modelo de consistência é visto como um contrato entre o programa e a implementação do sistema de memória, isto é, o sistema de memória sempre irá gerar execuções seqüencialmente consistentes desde que o programa classifique corretamente suas operações, de acordo com as regras de cada modelo. Neste trabalho, não detalharemos tal abordagem, uma vez que o principal interesse está na modelagem das condições de consistência com base nas execuções ocorridas.

5.6 Consistência Híbrida

Attiya e Friedman [15, 32, 69, 16] propuseram uma generalização para os modelos que se baseiam na distinção entre tipos de operação. Nesta abordagem, operações são classificadas como fortes ou fracas e diferentes imposições sobre a ordem relativa entre tais operações são definidas de acordo com o modelo de consistência que se deseja representar. Baseado nesta noção, um modelo de consistência denominado *híbrido* é definido.

A formalização do modelo de Consistência Híbrida é bastante semelhante à formalização da Consistência Fraca (Seção 5.5), exceto que as operações são agora classificadas como fortes ou fracas. Desta forma, temos a seguinte definição:

Definição 5.10 (Consistência Híbrida) $(\Omega, \xrightarrow{\sigma})$ *é hibridamente consistente se e somente se as seguintes condições são satisfeitas:*

(i) $(\Omega, \xrightarrow{\sigma})$ *é coerente;*

(ii) Existe uma execução $(\Omega', \xrightarrow{\sigma'})$ *legal e parcialmente equivalente à* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(ii-a) $\Omega' \cap \Omega$ *inclui apenas operações fortes;*

(ii-b) $\xrightarrow{\sigma'}$ *é consistente com* $\xrightarrow{po}$ *quando restrita a um processo;*

(iii) Para cada processo P_i*, existe uma execução* $(\Omega_i, \xrightarrow{\sigma_i})$ *legal e equivalente à* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(iii-a) $\xrightarrow{\sigma_i}$ *é consistente com* $\xrightarrow{po}$ *quando restrita a* P_i*;*

(iii-b) $\xrightarrow{\sigma_i}$ *é consistente com* $\xrightarrow{po}$ *para todo par de operações que envolve pelo menos uma operação forte;*

(iv) As condições de legalidade dos itens (ii) e (iii) devem ser consistentes entre si e com a condição de coerência do item (i).

A principal diferença entre a Consistência Fraca e a Consistência Híbrida está no fato de que, neste último modelo, qualquer operação pode ser classificada como forte ou fraca. Desta forma, se todas as operações são classificadas como fortes, tem-se um modelo equivalente à Consistência Seqüencial. Por outro lado, se todas as operações são classificadas como fracas, tem-se um modelo próximo à Consistência PRAM [63] (Seção 5.4). Se apenas as operações de sincronização são classificadas como fortes, tem-se um modelo equivalente à consistência fraca.

5.7 Consistência por Liberação (*Release Consistency*)

O modelo de Consistência por Liberação foi originalmente definido por Gharachorloo et al. [39] como uma caracterização do modelo de memória implementado pela

arquitetura DASH [44, 60, 59, 58] e se baseia em uma classificação mais fina das operações sobre a memória compartilhada. Da mesma forma que no modelo de Consistência Fraca (Seção 5.5), as operações são classificada em acessos de sincronização e acessos a dados. Entretanto, neste modelo, os acessos de sincronização são ainda divididos em operações de aquisição e liberação (de um lock). As regras descritas na Condição 5.6 foram estabelecidas pelos autores para que uma implementação seja consistente por liberação

Condição 5.6 (Consistência por Liberação)

(a) A memória é mantida coerente;

(b) Acessos de sincronização são seqüencialmente consistentes ou consistentes por processador;

(c) Nenhum acesso comum *de leitura ou escrita é emitido por um processador antes que todos os seus acessos de leitura de localizações de sincronização (aquisições) tenham sido* executados*;*

(d) Nenhum acesso de escrita em uma localização de sincronização (liberação) é emitido por um processador antes que todos os seus acessos comuns *tenham sido* executados.

As regras estabelecidas na Condição 5.6 são semelhantes àquelas definidas para o modelo de Consistência Fraca, apenas com uma especialização maior na classificação das operações. Intuitivamente, o que este modelo estabelece é que as alterações efetuadas sobre localizações compartilhadas por um determinado processo sejam "divulgadas" aos demais processos somente nos momentos de liberação de locks. Na definição a seguir, formulada nos termos da formalização desenvolvida neste trabalho, relacionaremos as operações de leitura e escrita sobre uma localizações utilizadas para sincronização como operações de aquisição e de liberação, respectivamente.

Definição 5.11 (Consistência por Liberação) $(\Omega, \xrightarrow{\sigma})$ *é consistente por liberação se e somente se as seguintes condições são satisfeitas:*

(i) $(\Omega, \xrightarrow{\sigma})$ *é coerente;*

(ii) Existe uma execução $(\Omega', \xrightarrow{\sigma'})$ *legal e parcialmente equivalente à* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(ii-a) $\Omega' \cap \Omega$ *inclui apenas operações de sincronização;*

(ii-b) $\xrightarrow{\sigma'}$ *é consistente com* $\xrightarrow{po}$ *quando restrita a um processo;*

(iii) Para cada processo P_i*, existe uma execução* $(\Omega_i, \xrightarrow{\sigma_i})$ *legal e equivalente à* $(\Omega, \xrightarrow{\sigma})$ *tal que:*

(iii-a) $\xrightarrow{\sigma_i}$ *é consistente com* $\xrightarrow{po}$ *quando restrita a* P_i*;*

(iii-b) $\xrightarrow{\sigma_i}$ *é consistente com* $\xrightarrow{po}$ *para todo par de operações do tipo* $r_s(x)v \xrightarrow{po} op$ *ou* $op \xrightarrow{po} w_s(x)v$

(iv) As condições de legalidade dos itens (ii) e (iii) devem ser consistentes entre si e com a condição de coerência do item (i).

5.8 Consistência DAG

O modelo de Consistência *dag* (Directed Acyclic Graph) foi proposto por Blumofe et al. [33, 26, 25, 49] como parte do sistema paralelo *Cilk* [49, 24]. Duas definições diferentes foram apresentadas. Ambas são baseadas na existência de uma ordem parcial dada, definida por um *dag*, sobre a qual diferentes condições são impostas dependendo do modelo que se que expressar.

Para se ter uma noção mais intuitiva de como um *dag* é criado, podemos imaginar que esta ordem parcial é a composição da ordem de programa para cada processo com uma ordem devido às sincronizações presentes no programa paralelo. Entretanto, esta interpretação é apenas uma dentre as possíveis, sendo as definições dadas considerando um *dag* genérico. De modo a relacionar o conceito de um *dag* com os

termos utilizados até aqui, vamos considerar uma ordem parcial $\xrightarrow{\tau}$, que representa um dado *dag*, tal que as precedências (ou subconjuntos delas) definidas por $\xrightarrow{\tau}$ são as relações que devem ser respeitadas pelas execuções.

A primeira definição proposta para a Consistência *dag* é apresentada a seguir, com pequenas alterações para se adequar à terminologia utilizada neste trabalho.

Definição 5.12 (Consistência *dag*) $(\Omega, \xrightarrow{\sigma})$ *é dag consistente se e somente se, dada uma ordem parcial* $\xrightarrow{\tau}$, *existe uma execução equivalente* $(\Omega', \xrightarrow{\sigma'})$ *tal que:*

(i) $\xrightarrow{\tau} \subseteq \xrightarrow{\sigma'}$;

(ii) Se $op = r(x)v$, *então existe* $op' = w(x)v$ *tal que* $\neg(op \xrightarrow{\tau} op')$;

(iii) Se $op = r(x)v$, $op' = w(x)v'$ *e* $op'' = w(x)v''$ *são tais que* $op'' \xrightarrow{\tau} op' \xrightarrow{\tau} op$, *então* $v \neq v''$.

Podemos interpretar a Definição 5.12 da seguinte maneira. Para $(\Omega, \xrightarrow{\sigma})$ ser *dag* consistente, deve existir uma execução equivalente e consistente com o *dag* dado, na qual cada leitura é seqüencialmente legal (Definição 4.1). Ou seja, para cada leitura, existe uma serialização consistente com o *dag* na qual esta leitura é legal.

Este modelo é extremamente fraco e permite execuções pouco razoáveis, conforme admitido pelos próprios autores [33]. A Figura 5.8 mostra um dag permitido pela definição.

A característica ilustrada na Figura 5.8 foi denominada de "não confinamento do não determinismo" por Frigo [33]. O que o autor quer dizer é que, apesar do valor a ser retornado pela primeira operação de leitura ser não determinístico, uma vez definido seu retorno e não havendo outras escritas que "possam influenciar" as leituras da mesma localização relacionadas à frente no dag, estas deveriam retornar o

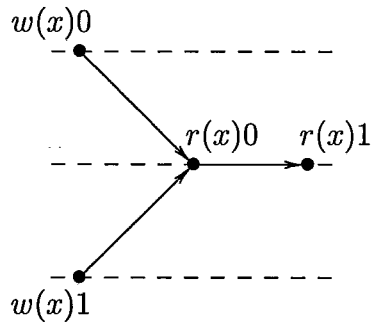


Figura 5.8: Exemplo de um *dag* permitido pela Definição 5.12.

mesmo valor desta primeira leitura. Se analisarmos as precedências do *dag* da Figura 5.8 nos termos da teoria de legalidade descrita anteriormente, veremos que, apesar de cada leitura ser "seqüencialmente legal" (no sentido de existir uma serialização do *dag* onde ela é legal), não existe execução consistente com o *dag* que seja legal (Definição 4.7).

Para contornar esta anomalia, uma nova definição para a Consistência *dag* foi apresentada por Blumofe et al. [25]. Esta nova definição é idêntica à Definição 5.12 exceto pelo item (*iii*), modificado para:

(iii) Se $op = r(x)v$, $op' = r(x)v'$ e $op'' = w(x)v''$ são tais que $op'' \xrightarrow{\tau} op' \xrightarrow{\tau} op$ e $v' \neq v''$, então $v \neq v''$.

Isto elimina a anomalia mostrada na Figura 5.8 entretanto, Frigo [33] demonstra que este modelo não pode ser construído. O que isto evidencia, é que, o algoritmo proposto para a implementação do modelo *dag*, na verdade implementa um modelo diferente, impondo restrições mais fortes do que aquelas previstas no modelo.

5.8.1 Consistência por Localização

Este modelo foi proposto por Gao e Sarkar [36, 35, 34] e tem como principal argumento a não exigência da condição de coerência para as execuções. O modelo de Consistência por Localização também se baseia na classificação dos acessos a localizações compartilhadas entre acessos de sincronização e acessos a dados. Entretanto, diferentemente da maioria dos demais modelos, tal classificação também é utilizada para relaxar a exigência da coerência.

A Definição 5.7 (coerência) implica na existência de uma ordem total para cada localização compartilhada. Neste modelo, apenas uma ordem parcial é exigida para as operações sobre uma mesma localização. Esta ordem parcial é definida pela composição da ordem de programa com relações definidas por operações de sincronização que o modelo suporta. Desta forma, para cada leitura, existe um conjunto de possíveis valores corretos que podem ser retornados, aqueles valores mais recentes segundo a ordem parcial definida.

Recentemente, concluiu-se que a primeira definição de Consistência *dag* e o modelo de Consistência por Localização de Gao e Sarkar são equivalentes [33, 35]. Considerando-se que a ordem parcial definida na Consistência por Localização define um *dag*, o conjunto de valores corretos para cada localização contém exatamente os mesmos valores que podem ser retornados por cada leitura na Definição 5.12.

5.9 Relacionamentos entre Modelos

Uma técnica comum adotada para a comparação entre modelos de consistência consiste na classificação de tais modelos como mais fortes ou mais fracos, baseando-se nas execuções permitidas por cada modelo. Neste trabalho, não nos dedicamos à

comparação detalhada entre modelos entretanto, dadas as definições acima, algumas relações podem ser concluídas.

O modelo de Consistência Seqüencial é geralmente classificado como o mais forte de todos, uma vez que a linearizabilidade, apesar de mais restritiva (Seção 5.1), não tem aplicação prática. Os modelos mais fracos, em muitos casos, são incomparáveis segundo esta classificação, uma vez que seguem linhas distintas de otimização das restrições impostas pela Consistência Seqüencial.

Numa primeira linha de relaxamento destas restrições, podemos considerar os modelos que exigem a preservação das relações de ordem somente entre as operações de um mesmo processo ou entre aquelas relativas a uma mesma localização compartilhada. Nesta linha estão a Consistência Causal que relaxa a condição da existência de uma única serialização envolvendo todas as operações da execução, a Consistência por Processador que relaxa a Consistência Causal por não exigir a a manutenção da ordem causal entre escritas nas serializações de cada processo, e finalmente, a Coerência que exige apenas a serialização das operações relativas a cada localização.

Em uma segunda linha, temos os modelos que fazem uso da classificação das operações para reduzir as restrições sobre as execuções. Nesta linha estão a Consistência Fraca e a Consistência por Liberação. Esta última relaxa as condições da Consistência Fraca através do maior refinamento da classificação das operações, considerando que as operações de aquisição e liberação são devidamente classificadas como tal. O modelo de Consistência Híbrida, que segue esta mesma linha, não pode ser comparado sem que se estabeleça regras de classificação para as operações.

Uma terceira linha seria a adotada pelo modelo *dag* e suas variações, onde incluimos o modelo de Consistência por Localização. Neste caso, classifica-se a primeira

definição da Consistência *dag* como mais fraca que a segunda pois esta última restringe execuções como a apresentada na Figura 5.8. Esta segunda definição é classificada pelo próprio autor [33] como mais fraca que a coerência.

A Figura 5.9 exibe uma relação entre os modelos apresentados nesta seção. As setas partem de um modelo mais forte para um mais fraco. Note que por esta classificação, alguns modelos são incomparáveis, o que significa que existem execuções permitidas por um modelo mas não no outro e vice-versa. A seguinte notação foi utilizada na Figura 5.9: CS – Consistência Seqüencial, CC – Consistência Causal, CP – Consistência por Processador, WO – Consistência Fraca (*Weak Ordering*), RC – Consistência por Liberação (*Release Consistency*), CH – Consistência Híbrida, Dag1 – Consistência *dag* original, Dag2/CL – Consistência *dag* modificada/Consistência por Localização.

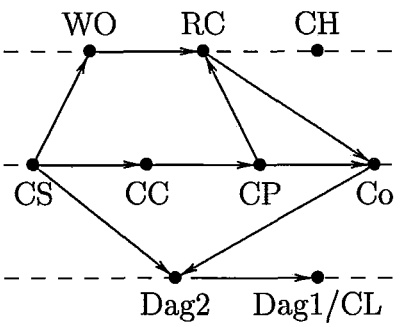


Figura 5.9: Relacionamento entre modelos de consistência.

5.10 Memória Compartilhada Distribuída por Sofware

Os modelos apresentados nas seções anteriores foram, em geral, projetados para implementação em hardware, como parte do sistema de memória de um multiprocessador. Entretanto, tais modelos também foram implementados em software, de

modo a suportar um espaço de endereçamento único em arquiteturas distribuídas, como redes de estações de trabalho. As implementações em software têm que lidar com a alta latência de comunicação via rede e o custo de execução de um protocolo de consistência que implementa o modelo adotado. Por este motivo, tais implementações buscam diminuir ao máximo o número de mensagens trocadas pelo protocolo de consistência, bem como sobrepor computação e comunicação como forma de otimizar a execução destes protocolos. Nesta seção, apresentaremos brevemente algumas propostas de protocolos de consistência. Vale ressaltar que, em geral, as implementações não são fieis às especificações do modelo de consistência no qual são baseadas. A formalização das reais condições de consistência implementadas pelos diversos protocolos propostos é um importante trabalho as ser desenvolvido, mas foge aos objetivos desta tese.

O sistema Ivy [61, 62] foi a primeira implementação em software da Consistência Seqüencial. Da mesma forma que a maioria das implementações em software, o sistema Ivy utiliza uma página da memória virtual como unidade de compartilhamento. A grande granularidade (alguns kbytes) é útil para amenizar os altos custos de comunicação mas cria o problema do *falso compartilhamento*. Este problema ocorre porque uma página abriga diversas localizações compartilhadas, as quais são tratadas pelo protocolo como uma única localização.

O sistema Munin [19, 20, 21] implementa um protocolo baseado nas especificações do modelo de Consistência por Liberação. Para aliviar o problema do falso compartilhamento, este sistema permite que múltiplas escritas ocorram concorrentemente para localizações em cópias de uma mesma página. A atualização das cópias das páginas alteradas são atrasadas até a próxima liberação de *lock*, quando as alterações são combinadas.

O protocolo implementado pelo sistema Munin é freqüentemente denominado como *eager release consistency*, em contraste com o protocolo *Lazy Release Consistency* [51, 53] implementado no sistema Treadmarks [14, 53]. No primeiro, as alterações sobre páginas compartilhadas realizadas por um processo são propagadas para atualização das cópias no momento da próxima liberação de *lock*. Em outra abordagem, o protocolo *Lazy* não realiza tais propagações nas liberações mas, sim, aguarda até que uma aquisição seja solicitada para, como parte da mensagem de passagem do *lock*, enviar as alterações realizadas pelo processo que detinha o *lock*, somente para o processo que realiza a aquisição. Este mecanismo reduz o número de mensagens necessárias ao protocolo de consistência, levando a melhor desempenho [52, 31]. Este protocolo *Lazy*, apesar de baseado nas especificações do modelo de Consistência por Liberação, implementa um modelo mais fraco que este. Entretanto, não existe formalização conhecida do modelo exato que este protocolo implementa.

O sistema Midway [22, 23] propõe uma diferente implementação também baseada na Consistência por Liberação. Da mesma forma que o protocolo *Lazy Release Consistency*, o protocolo de consistência do Midway implementa um modelo mais fraco que a Consistência por Liberação, denominado consistência por Entrada (*Entry Consistency*). Este modelo requer que localizações compartilhadas sejam associadas a localizações de sincronização. Assim como no protocolo *Lazy Release Consistency*, as atualizações só ocorrem quando um processo executa uma aquisição. Entretanto, somente os dados associados à localização de sincronização envolvida na aquisição são atualizados. A exploração da associação entre uma localização de sincronização e as localizações por esta protegidas diminui a comunicação necessária devido ao protocolo de consistência em comparação com a abordagem *lazy*. Entretanto, essa também é a maior desvantagem da Consistência por Entrada, uma vez que o programador é obrigado a fornecer as informações necessárias relativas às associações entre

sincronizações e localizações compartilhadas. Como esta condição torna muito complexa a programação sobre este modelo, foram propostos mecanismos automáticos para garantir a consistência por liberação, ou por processador, para aquelas localizações não associadas a uma localização de sincronização [23].

Seguindo a idéia de automatizar a associação entre localizações de sincronização e localizações "comuns", Iftode et al. [47, 48] propuseram a Consistência por Escopo. A idéia básica neste modelo é a utilização do conceito de *escopos de consistência* para se estabelecer implicitamente a associação entre localizações "comuns" e as de sincronização. Um escopo de consistência é composto por trechos de código onde as operações iniciadas têm a garantia de serem executadas. Pode-se imaginar um escopo como todas as regiões críticas protegidas por uma mesma localização de sincronização. O intervalo durante o qual um escopo está "aberto" (e.g. o trecho protegido por uma região crítica) é denominado uma sessão. Qualquer escrita feita dentro de uma sessão tem a garantia de ser executada do ponto de vista dos processos que iniciam uma sessão do mesmo escopo. Este modelo explora o fato de que, na maioria dos casos, os escopos podem ser derivados das sincronizações já existentes em aplicações que respeitam as regras de programação para a Consistência por Liberação.

Buscando também eliminar a necessidade de associação explícita entre localizações de sincronização e localizações "comuns", também foi proposto o protocolo AEC (*Affinity Entry consistency*) [74, 13]. Neste protocolo, um processo que inicia uma região crítica protegida por uma determinada localização de sincronização, tem a garantia de ter atualizadas somente as localizações associadas à esta localização de sincronização (*lock*), sendo estas associações feitas de modo automático. Entretanto, diferentemente de outras implementações baseadas na Consistência por entrada [22, 47], o protocolo AEC busca melhorar seu desempenho utilizando uma técnica

de previsão de aquisição de *locks* (LAP) [74, 75]. Esta técnica tenta prever qual o processo fará a próxima aquisição de um *lock* no momento da liberação deste *lock*, adiantando a computação e transmissão das modificações das páginas envolvidas e executando estas funções do protocolo em paralelo com computações das aplicação.

Capítulo 6

Conclusões

6.1 Resumo da Tese

Neste trabalho, discutimos a respeito da modelagem de sistemas paralelos baseados em memória compartilhada e sobre condições de consistência desta memória necessárias para a efetiva utilização de tais sistemas.

Para tal, apresentamos um novo modelo formal para a descrição de sistemas concorrentes onde a comunicação entre processos se dá exclusivamente pela escrita e leitura em localizações compartilhadas de memória. Algumas formalizações para sistemas com tais características têm sido propostas. Entretanto, estas abordagens ou se baseiam na disponibilidade de um tempo global, comum a todos os processos, ou modelam execuções sobre ordens totais de eventos ou suboperações, características estas desnecessárias à modelagem de sistemas concorrentes.

O modelo por nós proposto se baseia nos eventos inerentes a um sistema distribuído assíncrono e é geral o bastante para representar diversas arquiteturas. Sobre este modelo, execuções são representadas como conjuntos parcialmente ordenados de eventos que definem relações entre operações em um segundo nível mais alto da nossa formalização. A modelagem de operações e relações entre estas é então utilizada para descrever condições de consistência.

A representação de uma execução através de um conjunto parcialmente ordenado permite a utilização da teoria de grafos para a análise de características desta execução. Observamos que, dada as características desta ordem parcial, o grafo por ela induzido deve ser interpretado como um grafo E–OU. Inspirados nos conceitos de b–subgrafos e b-knots, novas estruturas sobre grafos E–OU foram definidas e algumas relações entre estas foram provadas. Concluímos que a existência de componentes fortemente conexas específicas implica na existência de ciclos em c–subgrafos, fato este que está fortemente ligado à possibilidade de serializar uma execução ou um subconjunto de operações desta.

Baseado na teoria de grafos, redefinimos o conceito de legalidade, tanto para o caso onde consideramos a existência de uma única escrita para cada valor na execução, quanto para o caso em que permitimos a existência de múltiplas escritas. A legalidade é uma condição de consistência básica a qual faz parte, mesmo que implicitamente, da especificação da grande maioria das condições de consistência hoje utilizadas. Ambas as definições de legalidade por nós introduzidas são estabelecidas sobre uma ordem parcial de operações que define uma execução, em contraste com a definição original dada sobre uma seqüência de operações.

A teoria de legalidade aqui introduzida mostrou-se adequada à especificação de diversas condições de consistência. Apresentamos versões baseadas na teoria de legalidade para modelos de consistência bastante difundidos como a Consistência Seqüencial, Consistência por Processador, Consistência por Liberação e Consistência *dag*, entre outros, mostrando que a modelagem proposta é bastante flexível e abrangente.

6.2 Desenvolvimentos Futuros

O passo mais importante a ser dado é, sem dúvida, quantificar, em um sistema real, a melhoria de desempenho que a teoria aqui desenvolvida pode representar em uma implementação de um sistema de memória compartilhada distribuída. A intuição é que, como a modelagem aqui proposta é baseada apenas em ordens parciais, tanto no nível de relações entre eventos, quanto no nível de relações entre operações, não estaríamos impondo restrições de ordem além das necessárias a cada modelo de consistência. Desta forma, uma implementação, tanto em hardware quanto em software, estaria livre de imposições de ordem desnecessárias ao modelo implementado.

A constatação que a não existência de b–knots implica na existência de um c–subgrafo acíclico que pode ser consistente ou inconsistente levanta a questão da existência de uma propriedade estrutural semelhante aos b–knots que esteja ligada à existência ou não de ciclos somente em c–subgrafos consistentes. A solução apresentada pelos Teoremas 3.2 e 4.2 é satisfatória, porém uma caracterização menos explícita poderia levar a algoritmos mais eficientes para verificação da legalidade de execuções. A caracterização de tal estrutura é colocada aqui como um problema em aberto, sendo um importante passo para extensões deste trabalho.

Por outro lado, não temos conhecimento de um trabalho que descreva todos os modelos de consistência em uma única formalização. Este é um trabalho bastante árduo dado a diversidade de formalizações existentes e ao grande número de propostas, muitas delas descritas informalmente ou através dos protocolos que as implementam. Entretanto, uma organização completa de todas estas propostas seria de extrema importância para o entendimento mais profundo do relacionamento entre elas.

Como vimos no Capítulo 5, existem protocolos projetados visando um deter-

minado modelo que na verdade implementam restrições mais fortes do que aquelas previstas no modelo. Somente a formalização destes algoritmos pode revelar as reais condições implementadas. O trabalho desenvolvido nesta Tese pode ser utilizado também como base para tal formalização.

Referências Bibliográficas

[1] S. V. Adve. *Designing Memory Consistency Models for Shared-Memory Multiprocessors.* PhD thesis, University of Wisconsin-Madison, 1993.

[2] S. V. Adve and K. Gharachorloo. "Shared Memory Consistency Models: A Tutorial", *IEEE Computer*, v. 29, n. 12, pp. 66–76, December 1996.

[3] S. V. Adve, K. Gharachorloo, A. Gupta, J. L. Hennessy, and M. D. Hill. *Sufficient System Requirements for Supporting PLpc Memory Model.* Technical Report TR #1200, Computer Sciences Department, University of Wisconsin-Madison, December 1993.

[4] S. V. Adve and M. D. Hill. *Sufficient Conditions for Implementing the Data-Race-Free-1 Memory Model.* Technical Report TR #1107, Computer Sciences Department, University of Wisconsin-Madison, September 1992.

[5] S. V. Adve and M. D. Hill. "A Unified Formalization of Four Shared-Memory Models", *IEEE Trans. on Parallel and Distributed Systems*, v. 4, n. 6, pp. 613–624, June 1993.

[6] S. V. Adve, M. D. Hill, B. P. Miller, and R. H. B. Netzer. "Detecting Data Races on Weak Memory Systems", In: *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pp. 234–243, May 1991.

[7] S. V. Adve, V. S. Pai, and P. Ranganathan. "Recent Advances in Memory Consistency Models for Hardware Shared-Memory Systems", *Proceedings of the IEEE, special issue on distributed shared-memory*, v. 7, n. 3, pp. 445–455, March 1999.

[8] M. Ahamad, R. A. Bazzi, R. John, P. Kohli, and G. Neiger. "The Power of Processor Consistency (Extended Abstract)", In: *Proc. of the 5th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'93)*, pp. 251–260, June 1993.

[9] M. Ahamad, J. E. Burns, P. W. Hutto, and G. Neiger. "Causal Memory", In: *Proc. of the 5th Int'l Workshop on Distributed Algorithms (WDAG'91)*, number 579 in Lecture Notes in Computer Science, pp. 9–30. Springer-Verlag, October 1991.

[10] M. Ahamad, P. W. Hutto, and R. John. "Implementing and Programming Causal Distributed Shared Memory", In: *Proc. of the 11th Int'l Conf. on Distributed Computing Systems (ICDCS-11)*, pp. 274–281, May 1991.

[11] M. Ahamad, R. John, P. Kohli, and G. Nieger. "Causal Memory Meets the Consistency and Performance Needs of Distributed Applications!", In: *Proc. of the 6th ACM SIGOPS European Workshop*, pp. 271–281, September 1994.

[12] M. Ahamad, G. Neiger, P. Kohli, J. E. Burns, and P. W. Hutto. "Causal Memory: Definitions, Implementation and Programming", *Distributed Computing*, v. 9, n. 2, pp. 37–49, 1995.

[13] C.L. Amorim, C.B. Seidel, and R. Bianchini. *The Affinity Entry Consistency Protocol.* Technical Report ES-388/96, COPPE/UFRJ, Rio de Janeiro, Brazil, May 1996.

[14] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. "TreadMarks: Shared Memory Computing on Networks of Workstations", *IEEE Computer*, v. 29, n. 2, pp. 18–28, February 1996.

[15] H. Attiya, S. Chaudhuri, R. Friedman, and J. Welch. "Shared Memory Consistency Conditions for Non-Sequential Execution: Definitions and Programming Strategies", *SIAM J. on Computing*, v. 27, n. 1, pp. 65–89, February 1998.

[16] H. Attiya and R. Friedman. "A Correctness Condition for High Performance Multiprocessors", In: *Proc. of th 24th ACM Annual Symp. on the Theory of Computing*, pp. 679–690, 1992.

[17] V. C. Barbosa. *An Introduction to Distributed Algorithms.* Cambridge, MA, The MIT Press, 1996.

[18] V. C. Barbosa and M. R. F. Benevides. *A graph-theoretic characterization of AND-OR deadlocks.* Technical Report ES-472/98, COPPE/UFRJ, Rio de Janeiro, Brazil, July 1998.

[19] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. *Munin: Shared Memory for Distributed Memory Multiprocessors.* Technical Report COMP TR89-91, Dept. of Computer Science, Rice University, April 1989.

[20] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. "Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence", In: *Proc. of the Second ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP'90)*, pp. 168–177, March 1990.

[21] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. "Munin: Distributed Shared Memory Using Multi-Protocol Release Consistency", In: A. I. Karshmer and

J. Nehmer, editors, *Operating Systems of the 90s and Beyond*, number 563 in Lecture Notes in Computer Science, pp. 56–60. Springer-Verlag, July 1991.

[22] B. N. Bershad and M. J. Zekauskas. *Midway: Shared Memory Parallel Programming with Entry Consistency for Distributed Memory Multiprocessors.* Technical Report CMU-CS-91-170, School of Computer Science, Carnegie-Mellon University, September 1991.

[23] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. "The Midway Distributed Shared Memory System", In: *Proc. of the 38th IEEE Int'l Computer Conf. (COMPCON Spring'93)*, pp. 528–537, February 1993.

[24] R. D. Blumofe. *Executing Multithreaded Programs efficiently.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1995.

[25] R. D. Blumofe, M. Frigo, C. F. Joerg, C. E. Leiserson, and K. H. Randall. "An Analysis of Dag-Consistent Distributed Shared-Memory Algorithms", In: *Proc. of the 8th ACM Symp. on Parallel Algorithms and Architectures (SPAA'96)*, pp. 297–308, June 1996.

[26] R. D. Blumofe, M. Frigo, C. F. Joerg, C. E. Leiserson, and K. H. Randall. "DAG-Consistent Distributed Shared Memory", In: *Proc. of the 10th Int'l Parallel Processing Symp. (IPPS'96)*, pp. 132–141, April 1996.

[27] W. W. Collier. *Reasoning about Parallel Architectures.* Englewood Cliffs, NJ, Prentice-Hall, 1992.

[28] J. Dongarra, H. Meuer, and E. Strohmaier. *TOP500 Supercomputer Sites.* Technical Report UT-CS-97-365, University of Tennessee, June 1997.

[29] M. Dubois and C. Scheurich. "Correct Memory Operation of cache-based Multiprocessors", In: *Proceedings of the 14th Annual International Symposium on Computer Architecture*, pp. 234–243, June 1987.

[30] M. Dubois, C. Scheurich, and F. A. Briggs. "Memory Access Buffering in Multiprocessors", In: *Proceedings of the 13th Annual International Symposium on Computer Architecture*, June 1986.

[31] S. Dwarkadas, P. Keleher, A. L. Cox, and W. Zwaenepoel. "Evaluation of Release Consistent Software Distributed Shared Memory on Emerging Network Technology", In: *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pp. 144–155, May 1993.

[32] R. Friedman. *Consistency Conditions for Distibuted Shared-Memory.* PhD thesis, Israel Institute of Technology, 1994.

[33] M. Frigo. *The Weakest Reasonable Memory.* Master's thesis, Department of Electrical Engineering and Computer Science, MIT, 1998.

[34] G. R. Gao and V. Sarkar. "Location Consistency: Stepping Beyond the Memory Coherence Barrier", In: *Proc. of the 1995 Int. Conf. on Parallel Processing (ICPP'95)*, volume II, pp. 73–76, August 1995.

[35] G. R. Gao and V. Sarkar. "On the Importance of an end-to-end View of Memory Consistency in Future Computer Systems", In: *Proc. of the 1997 Int. Symp. on High Performance Computing*, pp. 173–184, November 1997.

[36] G. R. Gao and V. Sarkar. *Location Consistency — a new Memory Model and Cache Consistency Protocol.* Technical Report CAPSL Technical Memo 16, University of Delaware, February 1998.

[37] K. Gharachorloo. *Memory Consistency Models for Shared-Memory Multiprocessors.* PhD thesis, Stanford University, 1995.

[38] K. Gharachorloo, A. Gupta, and J. L. Hennessy. *Revision to "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors".* Technical Report CSL-TR-93-568, Computer Systems Laboratory, Stanford University, April 1993.

[39] K. Gharachorloo, D. E. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. L. Hennessy. "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors", In: *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 15–26, May 1990.

[40] P. B. Gibbons and M. Merritt. "Specifying Noblocking Shared Memories", In: *Proc. of the 4th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'92)*, pp. 306–315, July 1992.

[41] P. B. Gibbons, M. Merritt, and K. Gharachorloo. "Proving Sequential Consistency of High-Performance Shared Memories (Extended Abstract)", In: *Proc. of the 3th ACM Annual Symp. on Parallel Algorithms and Architectures (SPAA'91)*, pp. 292–303, July 1991.

[42] J. R. Goodman. *Cache Consistency and Sequential Consistency.* Technical Report 61, IEEE Scalable Coherence Interface Working Group, March 1989.

[43] J. R. Goodman. *Cache Consistency and Sequential Consistency.* Technical Report 61, Computer Sciences Department, University of Wisconsin-Madison, February 1991.

[44] S. Gupta. "Stanford DASH Multiprocessor: the Hardware and Software", In: *Proc. of Parallel Architectures and Languages Europe (PARLE'92)*, pp. 802–805, June 1992.

[45] M. P. Herlihy and J. M. Wing. "Linearizability: A Correctness Condition for Concurrent Objects", *ACM Trans. on Programming Languages and Systems*, v. 12, n. 3, pp. 463–492, July 1990.

[46] R. C. Holt. "Some Deadlock Properties of Computer Systems", *ACM Computing Survyes*, v. 4, pp. 179–196, 1972.

[47] L. Iftode, C. Dubnicki, E. W. Felten, and K. Li. "Improving Release-Consistent Shared Virtual Memory using Automatic Update", In: *Proc. of the 2nd IEEE Symp. on High-Performance Computer Architecture (HPCA-2)*, pp. 251–260, February 1996.

[48] L. Iftode, J. P. Singh, and K. Li. "Understanding Application Performance on Shared Virtual Memory Systems", In: *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pp. 122–133, May 1996.

[49] C. F. Joerg. *The Cilk System for Parallel Multithreaded Computing.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1996.

[50] R. John and M. Ahamad. *Causal Memory: Implementation, Programming Support and Experiences.* Technical Report GIT-CC-93-10, College of Computing, Georgia Institute of Technology, 1993.

[51] P. Keleher. *Lazy Release Consistency for Distributed Shared Memory.* PhD thesis, Department of Computer Science, Rice University, December 1994.

[52] P. Keleher, A. L. Cox, and W. Zwaenepoel. "Lazy Release Consistency for Software Distributed Shared Memory", In: *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 13–21, May 1992.

[53] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. "TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems", In: *Proc. of the Winter 1994 USENIX Conference*, pp. 115–131, January 1994.

[54] P. Kohli, G. Neiger, and M. Ahamad. "A Characterization of Scalable Shared Memories", In: *Proc. of the 1993 Int'l Conf. on Parallel Processing (ICPP'93)*, pp. 145–153, August 1993.

[55] L. Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM*, v. 21, n. 7, pp. 558–565, July 1978.

[56] L. Lamport. "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs", *IEEE Trans. on Computers*, v. C-28, n. 9, pp. 690–691, September 1979.

[57] L. Lamport. "On interprocess communication—Part I: Basic formalism", *Distributed Computing*, v. 1, n. 3, pp. 77–85, 1986.

[58] D. E. Lenoski. *The Design and Analysis of DASH: A Scalable Directory-Based Multiprocessor.* PhD thesis, Stanford University, 1991.

[59] D. E. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. L. Hennessy. "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor", In: *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 148–159, May 1990.

[60] D. E. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, J. L. Hennessy, M. Horowitz, and M. Lam. *Design of the Stanford DASH multiprocessor.* Technical

Report CSL-TR-89-403, Computer Systems Laboratory, Stanford University, December 1989.

[61] K. Li. *Shared Virtual Memory on Loosely Coupled Multiprocessors.* PhD thesis, Department of Computer Science, Yale University, September 1986.

[62] K. Li. "IVY: A Shared Virtual Memory System for Parallel Computing", In: *Proc. of the 1988 Int'l Conf. on Parallel Processing (ICPP'88)*, volume II, pp. 94–101, August 1988.

[63] R. J. Lipton and J. S. Sandberg. *PRAM: A Scalable Shared Memory.* Technical Report CS-TR-180-88, Dept. of Computer Science, Princeton University, September 1988.

[64] N. A. Lynch. *Distributed Algorithms.* San Francisco, CA, Morgan Kaufmann Publ, 1996.

[65] N. A. Lynch and M. R. Tuttle. *An Intruduction to Input/Output Automata.* Technical Report MIT/LCS/TM-373, MIT Laboratory for Computer Science, November 1988.

[66] J. Misra. "Axioms for Memory Access in Asynchronous Hardware Systems", *ACM Trans. on Programming Languages and Systems*, v. 8, n. 1, pp. 142–153, January 1986.

[67] M. Mizuno, M. Raynal, and J. Z. Zhou. Sequential consistency in distributed systems: Theory and implementation. In: F. Mattern K.P. Birman and A. Schiper, editors, *Lecture Notes in Computer Science*, pp. 224–241. Springer Verlag, 1995.

[68] S. V. Pai, P. Ranganathan, H. Abdel-Shafi, and S. Adve. "The Impact of Exploiting Instruction-Level Parallelism on Shared-Memory Multiprocessors ",

IEEE Transactions on Computers, special issue on caches, v. 8, n. 2, pp. 218–226, February 1999.

[69] M. Raynal and M. Mizuno. "How to find his way in the jungle of consistency criteria for distributed shared memories (or how to escape from Minos' labyrinth)", In: *Proc. of the 4th IEEE CS Workshop on Future Trends of Distributed Computing Systems*, pp. 340–346, 1993.

[70] M. Raynal, M. Mizuno, and M. L. Neilsen. *Causality Oriented Shared Memory for Distributed Systems.* Technical Report RR-1680, INRIA, France, May 1992.

[71] M. Raynal and A. Schiper. "From Causal Consistency to Sequential Consistency in Shared Memory Systems", In: *Proc. of the 15th Int'l Conf. on Foundation of Software Technology and Theoretical Computer Science*, number 1026 in Lecture Notes in Computer Science, pp. 180–194, December 1995.

[72] M. Raynal and A. Schiper. "A Suite of Formal Definitions for Consistency Criteria in Shared Memories", In: *Proc. of the ISCA Int. Conf. on Parallel and Distributed Computing Systems (ICPDS'96)*, pp. 125–130, September 1996.

[73] C. Scheurich and M. Dubois. "Correct Memory Operation of Cache-Based Multiprocessors", In: *Proc. of the 14th Annual Int. Symp. on Computer Architecture (ISCA'87)*, pp. 234–243, June 1987.

[74] C. B. Seidel, R. Bianchini, and C. L. Amorim. "The Affinity Entry Consistency Protocol", In: *Proceedings of the 1997 International Conference on Parallel Processing*, pp. 65–78, August 1997.

[75] C. B. Seidel, R. Bianchini, and C.L. Amorim. "Técnicas para Previsão Dinâmica do Próximo *Acquirer* em *Software DSMs*", In: *Anais do Simpósio Brasileiro de*

Arquitetura de Computadores e Processamento de Alto Desempenho, pp. 203–212, Agosto 1996.

[76] D. Shasha and M. Snir. "Efficient and correct Execution of Parallel Programs that Share Memory", *IEEE Trans. on Programming Languages and Systems*, v. 10, n. 2, pp. 282–312, April 1988.