Fernando Náufel do Amaral

RETOOL: UMA LÓGICA DE AÇÕES PARA SISTEMAS DE TRANSIÇÃO TEMPORIZADOS

Dissertação apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para a obtenção do título de Mestre em Ciências em Informática.

Orientador: Prof. Edward Hermann Haeusler

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 9 de maio de 2000

A Bel, Bruno e Vitor, por me mostrarem o caminho e me acompanharem na viagem

Agradecimentos

Ao Prof. Edward Hermann Haeusler, pelas aulas, pelo exemplo, pelo estímulo, pela pressão, pela companhia, pela paciência e pela confiança.

Aos Profs. Luiz Carlos Guedes, Armando Haeberer e Sérgio Carvalho (*in memoriam*), pelo primeiro contato com a pesquisa acadêmica.

Ao DI da PUC-Rio, pelos recursos oferecidos.

À CAPES, pelo parco suporte financeiro.

Aos Profs. José Fiadeiro e Tom Maibaum, pelas discussões sobre o trabalho.

Ao Prof. Paulo Veloso, pelas aulas de Lógica e pela participação na defesa da proposta desta dissertação.

Ao Prof. Mario Benevides, por acompanhar o desenvolvimento desta dissertação.

Ao Prof. Luiz Carlos Pereira, pelas aulas de Filosofia.

Ao Prof. J.L. Rangel, pelas aulas, pelo apoio e pelo bom humor.

Aos meus pais, por minha formação.

A Bel e aos meninos, por mais do que eu seria capaz de enumerar.

A Deus, por tudo.

Imaginar uma linguagem significa imaginar uma forma de vida.

L. Wittgenstein

Resumo

Esta dissertação define RETOOL, uma lógica de ações dotada de um operador para denotar condições necessárias e pós-condições das ações de um sistema de transição temporizado (uma extensão do formalismo de sistemas de transição cujo objetivo é modelar sistemas computacionais reativos/concorrentes de tempo real). Uma semântica para RETOOL é apresentada e comparada com propostas anteriores. Uma axiomatização adequada é fornecida, com provas detalhadas de correção e completude fraca.

Abstract

This dissertation defines RETOOL, an action logic featuring an operator to denote necessary conditions and postconditions of actions in a timed transition system (an extension of the formalism of transition systems meant to model real-time reactive/concurrent computational systems). A semantics for RETOOL is presented and compared to previous proposals. An adequate axiomatization is given, along with detailed correctness and weak completeness proofs.

Conteúdo

Capítulo 1

Introdução

O termo "concorrência" diz respeito a sistemas de agentes múltiplos que interagem uns com os outros. Sistemas computacionais concorrentes têm sido o alvo de estudo de diversos pesquisadores nas últimas décadas. As noções de *sistemas reativos* e *sistemas de tempo real* aparecem frequentemente neste estudo.

A dificuldade de projetar sistemas desta natureza usando métodos convencionais de engenharia de *software* faz com esta seja uma área rica em oportunidades para aplicação de métodos formais de desenvolvimento de sistemas. Diversos modelos matemáticos têm sido propostos para descrever os fenômenos que ocorrem na computação concorrente/reativa/de tempo real. Ferramentas baseadas nestes modelos matemáticos têm sido propostas para especificar e verificar *software*.

Entre os formalismos usados para a especificação e verificação de sistemas, destacam-se algumas lógicas especiais, cujas características as tornam adequadas para raciocinar sobre modelos matemáticos de computação concorrente. Entre estas lógicas, destacam-se as variedades de Lógica Dinâmica, Lógica de Ações e Lógica Temporal.

O objetivo da presente dissertação é apresentar uma nova lógica de ações para

raciocinar sobre um destes modelos matemáticos: o modelo de sistemas de transição temporizados ([11]).

O capítulo 2 apresenta uma classificação dos sistemas computacionais, esclarecendo o relacionamento entre reatividade e concorrência e destacando os sistemas reativos/concorrentes de tempo real, que nossa lógica pretende modelar. Uma abstração inicial é proposta, para definir as características de sistemas reativos/concorrentes relevantes para este trabalho.

O capítulo 3 introduz o modelo matemático dos *sistemas de transição*, explicando como podemos usá-los para representar o comportamento de processos ou sistemas de processos. Como é o caso com muitos dos modelos formais de concorrência, o modelo dos sistemas de transição é estendido para incluir características de sistemas de tempo real, dando origem aos sistemas de transição temporizados.

O capítulo 4 define a lógica de ações RETOOL (REal-Time Object-Oriented Logic – uma referência ao objetivo maior de raciocinar sobre sistemas de tempo real orientados a objeto), cujas características a tornam adequada para lidar com modelos de sistemas reativos de tempo real, incluindo informações como condições necessárias, pós-condições e restrições de tempo para a execução de ações. Definida uma semântica formal para RETOOL, em termos de sistemas de transição temporizados, é apresentada uma axiomatização adequada, com provas detalhadas de correção e completude fraca.

Por fim, o capítulo 5 traça conclusões e relaciona trabalhos futuros.

Capítulo 2

Sistemas Computacionais

Este capítulo apresenta algumas considerações informais sobre sistemas computacionais e discute critérios que podem ser utilizados para classificar tais sistemas segundo sua natureza. Uma visão abstrata de sistemas reativos é construída para servir de base aos formalismos apresentados nos capítulos subsequentes da dissertação.

2.1 Tipos de Sistemas Computacionais

2.1.1 Sistemas Transformacionais

A visão "clássica" de um sistema computacional é a de um processo que é invocado pelo usuário, recebe um conjunto bem definido de dados, executa determinadas operações sobre eles e retorna ao usuário os resultados desejados. Pode-se dizer que um sistema deste tipo *transforma* os dados de entrada nos dados de saída.

Por exemplo, programas que lêem um arquivo de entrada, processam seu conteúdo e geram um arquivo de saída ao fim de sua execução se enquadram nesta categoria.

Programas executados na modalidade "batch", em geral, são exemplos de sistemas transformacionais.

Para que um sistema seja considerado transformacional, basta que seu comportamento possa ser modelado matematicamente como uma função dos naturais para os naturais: os dados recebidos pelo sistema durante sua execução formam uma sequência finita; independentemente dos tipos dos dados envolvidos, esta sequência pode ser traduzida para um número natural com a utilização de uma codificação apropriada. Se a execução do sistema terminar, a sequência de seus resultados será finita, podendo ser igualmente codificada para um número natural.

Alternativamente, um sistema transformacional pode ser visto como uma função que mapeia estados iniciais para estados finais, onde, por "estado", pode-se entender um conjunto de valores de dados (os valores das variáveis manipuladas pelo sistema). No caso não-determinístico, onde um sistema pode possuir mais de um estado inicial e/ou mais de um estado final, o comportamento do sistema pode ser visto como uma *relação* entre estados iniciais e estados finais.

Técnicas para o estudo do comportamento de sistemas transformacionais envolvem o uso de formalismos desenvolvidos e estudados desde a década de 1960, como Lógica de Hoare [12], por exemplo. As questões de maior interesse relativas a um sistema transformacional, que estas técnicas tencionam tratar, são sua *correção parcial* (se a execução do sistema termina, os resultados são os esperados em função dos dados de entrada?) e sua *correção total* (a execução termina e os resultados são os esperados em função dos dados de entrada?).

2.1.2 Sistemas Reativos

Sistemas que não se prestam à caracterização transformacional são denominados *reativos*. Um sistema reativo, embora possua estados iniciais bem definidos, geral-

mente é projetado para que sua execução jamais termine, não fazendo sentido, então, falar em estados finais. O comportamento de um sistema deste tipo consiste em respostas – reações – a eventos e requisições do seu ambiente (composto pela combinação de diversos elementos, entre os quais o usuário, o sistema operacional, outros processos com os quais o sistema pode interagir durante sua execução etc.).

Alguns exemplos comuns de sistemas reativos são programas de controle de processos, sistemas operacionais, protocolos de redes e sistemas de reserva de passagens aéreas.

Não havendo a noção de estado final, as técnicas usadas para formalizar o comportamento de sistemas transformacionais geralmente não se aplicam a sistemas reativos. Nestes sistemas, as perguntas que devem ser tratadas são, por exemplo:

- Toda requisição feita pelo usuário (ou por algum outro elemento com o qual o sistema interage) é atendida em algum momento?
- Toda reação observável do sistema corresponde a uma requisição feita anteriormente?
- O sistema está sempre pronto a atender uma requisição qualquer, ainda que não imediatamente?

Para raciocinar sobre questões deste tipo desenvolveram-se os formalismos apresentados nos próximos capítulos desta dissertação.

2.1.3 Sistemas de Tempo Real

Um sistema – seja ele transformacional ou reativo – pode ser projetado para satisfazer restrições sobre o tempo decorrido entre uma entrada e uma saída (ou entre

uma requisição e o seu atendimento) durante sua execução. Neste caso, o sistema é denominado *de tempo real*.

Exemplos típicos de sistemas de tempo real são programas de controle de usinas nucleares, programas de controle de vôo e sistemas de monitoramento de pacientes em hospitais.

Alguns sistemas de tempo real são executados em situações onde a não-satisfação das restrições de tempo pode levar a sérios prejuízos de ordem material, ou mesmo a perdas de vidas humanas. A complexidade do ambiente com o qual um sistema reativo precisa interagir, combinada com as restrições de tempo que o sistema deve satisfazer, torna a verificação destes sistemas crucial e de elevada dificuldade.

Os formalismos para o estudo de sistemas de tempo real são relativamente recentes, e consistem geralmente de versões modificadas de formalismos desenvolvidos para o estudo de sistemas reativos e/ou concorrentes sem restrições de tempo. Alguns deles também serão abordados nos próximos capítulos desta dissertação.

2.1.4 Sistemas Concorrentes e Distribuídos

Sistemas concorrentes são aqueles que consistem de diversos componentes, denominados *processos*, cuja execução conjunta caracteriza o comportamento do sistema como um todo. Os processos podem ser executados alternadamente, compartilhando a memória e os recursos de um sistema operacional multitarefa em uma única CPU. Um escalonador se responsabiliza por alocar o tempo de CPU para os processos. Este esquema de execução é denominado *multiprogramação*.

Alternativamente, os processos podem ser executados simultaneamente em CPUs diferentes (possivelmente situadas em locais distantes uns dos outros), trocando mensagens entre si. Este esquema de execução é denominado *multiprocessamento*.

No caso geral (uma CPU ou múltiplas CPUs), o sistema é considerado *concorrente*; no caso especial envolvendo múltiplas CPUs, o sistema é dito *distribuído*.

2.1.5 Reatividade × Concorrência/Paralelismo

Na literatura, é comum o uso indiscriminado dos termos "reatividade" e "concorrência" para denotar a característica principal de um único tipo de sistema. A identificação da classe de sistemas reativos com a classe de sistemas concorrentes pode ser justificada através dos seguintes argumentos:

- Na formalização de um sistema reativo, o ambiente com o qual o sistema interage durante sua execução deve ser levado em consideração. Mesmo que o sistema reativo consista de um único processo, o ambiente constitui um elemento adicional, e sua inclusão no modelo caracteriza uma situação de concorrência.
- Um sistema concorrente é composto de processos que interagem entre si e com a plataforma de execução subjacente. Sendo assim, cada um destes processos, separadamente, é um sistema reativo. No estudo de sistemas computacionais, é altamente desejável adotar um ponto de vista *composicional*, segundo o qual os componentes de um sistema são encarados como sendo da mesma natureza que o sistema como um todo. Assim, como os processos de um sistema concorrente possuem um caráter reativo, o sistema como um todo também deve ser considerado reativo.

2.2 Uma Abstração para Sistemas Concorrentes

A característica principal de sistemas reativos/concorrentes que desejamos capturar na abstração que vamos apresentar (ver [3]) é o *paralelismo*: a simultaneidade da

execução de dois ou mais processos. No entanto, somos obrigados a levar em consideração o próprio conceito de concorrência que adotamos acima, segundo o qual a execução intercalada de processos na CPU única de um ambiente de multiprogramação, embora não seja verdadeiramente paralela, também pode ser considerada como concorrente.

Nesta seção, veremos como as duas modalidades de concorrência (multiprogramação e multiprocessamento) são tratadas nesta abstração. Na verdade, um sistema reativo executando em um ambiente multiprogramado será modelado exatamente da mesma forma que um sistema composto pelos mesmos processos executando em múltiplos processadores. Ou seja, esta abstração não faz distinção entre o paralelismo verdadeiro de múltiplas CPUs e o paralelismo simulado pela execução alternada de processos em uma única CPU.[1]

2.2.1 Sequências de Execução

Um processo de um sistema reativo/concorrente é, basicamente, um programa sequencial; i.e., uma sequência de instruções executadas uma de cada vez, em uma ordem bem definida. Como existem outros componentes no sistema, porém, a execução sequencial de um processo pode ser afetada pela execução de outro processo: se o ambiente for multiprogramado, um processo pode ser interrompido para que a CPU seja cedida a outro processo; se o ambiente for multiprocessado, processos executando simultaneamente podem alterar valores de variáveis compartilhadas, ou trocar mensagens entre si.

Por exemplo, sejam P_a e P_b dois processos de um sistema reativo/concorrente. Digamos que o processo P_a seja um programa simples, consistindo apenas das duas

[1]Desde que questões de tempo real não estejam envolvidas, caso em que a diferença entre multiprogramação e multiprocessamento se torna relevante.

instruções i_{a1} e i_{a2}. O processo P_b também é composto de duas instruções: i_{b1} e i_{b2}.

Analisemos o caso em que P_a e P_b são executados em um ambiente multitarefa. O sistema reativo/concorrente resultante desta composição é denotado por $P_a \;|||\; P_b$ (ver [13]). É fácil ver que, se não houver restrições adicionais sobre as execuções, as instruções de P_a e P_b só poderão ser executadas pela CPU única nas seguintes ordens:

- $i_{a1}, i_{a2}, i_{b1}, i_{b2}$
- $i_{a1}, i_{b1}, i_{a2}, i_{b2}$
- $i_{a1}, i_{b1}, i_{b2}, i_{a2}$
- $i_{b1}, i_{a1}, i_{b2}, i_{a2}$
- $i_{b1}, i_{a1}, i_{a2}, i_{b2}$
- $i_{b1}, i_{b2}, i_{a1}, i_{a2}$

Estas seis alternativas são as possíveis *sequências de execução* do sistema $P_a \;|||\; P_b$. O conjunto de sequências de execução representa os comportamentos *potenciais* do sistema, uma vez que não é possível prever, a princípio, qual das sequências ocorrerá em uma execução específica.

Passemos ao caso em que os processos P_a e P_b são executados simultaneamente em processadores diferentes, situação denotada por $P_a \;||\; P_b$ (ver [13]). Aqui, dada uma instrução qualquer i_a do processo P_a e uma instrução qualquer i_b do processo P_b, duas situações são possíveis:

1. A instrução i_a tem sua execução iniciada e terminada antes do início da execução da instrução i_b;
2. Em algum momento, as instruções i_a e i_b estão sendo executadas simultaneamente.

A situação 1 corresponde ao trecho de sequência de execução i_a, i_b. Se desejarmos representar o comportamento de um sistema reativo/concorrente apenas por sequências de execução, a que trecho de sequência corresponderia a situação 2?

A resposta envolve uma suposição adicional: a de que o efeito da execução simultânea das duas instruções i_a e i_b é o mesmo que o efeito da sequência i_a, i_b e que o efeito da sequência i_b, i_a.

Se esta suposição for verdadeira, a situação 2 acima corresponderá aos trechos de sequência de execução i_a, i_b e i_b, i_a. Mas esta suposição é razoável no que diz respeito a sistemas computacionais?

Se as instruções i_a e i_b não afetam nenhum objeto compartilhado pelos processos P_a e P_b, como variáveis compartilhadas ou recursos do sistema operacional, a execução de uma das instruções não afetará de forma alguma a execução da outra, e o efeito final será o mesmo, independentemente da ordem de execução ou da simultaneidade. A suposição será verdadeira.

Se, por outro lado, as instruções i_a e i_b tentam acessar um objeto compartilhado, situações indesejáveis podem ocorrer, como mostra o seguinte exemplo:

Seja i_a a instrução `x := x + 1` e seja i_b a instrução `x := x - 1`. Suponhamos que o valor inicial da variável compartilhada `x` seja 0. Claramente, ao final de qualquer uma das duas sequências de execução i_a, i_b e i_b, i_a, o valor de `x` permanecerá 0.

No entanto, cada uma das duas instruções pode precisar ser traduzida para duas ou mais instruções de máquina para ser executada pela CPU em questão. Digamos que a instrução `x := x + 1` seja traduzida para

```
a1.  LOAD AC, X

a2.  ADD AC, 1
```

```
a3.  STORE X, AC
```

e a instrução `x := x - 1` para

```
b1.  LOAD AC, X

b2.  SUB AC, 1

b3.  STORE X, AC
```

Digamos, ainda, que, durante a execução simultânea dos dois conjuntos de instruções em duas CPUs diferentes, as instruções sejam executadas na seguinte ordem:

```
a1, a2, b1, b2, b3, a3
```

Neste caso, o valor de `x` ao final da sequência será 1, e não 0. A diferença advém do fato de que cada instrução original foi decomposta em três instruções de máquina, e a intercalação se deu, na verdade, entre estas instruções mais simples. Para que isto não ocorra, devemos estipular que as sequências de execução contenham apenas instruções *atômicas* – aquelas que não podem ser decompostas em instruções mais simples ou que, caso possam ser decompostas em sequências de instruções mais simples, estas devem ser tais que sua execução não possa ser interrompida ou intercalada com nenhuma outra instrução.

No exemplo, se a instrução `x := x + 1` fosse traduzida para `inc x` e a instrução `x := x - 1` para `dec x`, e se `inc x` e `dec x` fossem executadas atomicamente pelos processadores em questão, restaria determinar qual seria o efeito sobre a variável `x` da execução simultânea destas duas instruções atômicas. Devido à própria natureza digital dos processadores e das células de memória, a tentativa de acesso simultâneo à variável `x` seria resolvida pelo hardware de modo a satisfazer nossa suposição: se as instruções forem atômicas, o efeito será o de uma execução intercalada. Algo análogo ocorre com instruções que tentam acessar dispositivos de entrada e saída ou canais

de comunicação entre processos.

Em vista do que foi discutido até este ponto, portanto, os comportamentos potenciais de um sistema concorrente/reativo (seja em um ambiente multiprocessado, seja em um ambiente multiprogramado) são representados pelo conjunto de sequências de execução formadas pelas intercalações das instruções atômicas dos processos que compõem o sistema.

2.2.2 *Fairness*

Na realidade, porém, nem todas as intercalações de instruções atômicas podem ser aceitas como comportamentos possíveis de um sistema reativo/concorrente. Algumas não correspondem ao que ocorre na prática.

Em um sistema multiprocessado, nenhum processador é tão lento a ponto de não executar nenhuma instrução enquanto infinitas instruções são executadas pelos outros processadores. Analogamente, em um sistema multiprogramado, o escalonador jamais mantém esperando indefinidamente um processo que esteja pronto para ser executado.

Isto nos leva a exigir que, para todo processo de um sistema reativo, qualquer que seja um dado ponto de uma sequência de execução, sempre haja um ponto posterior onde uma instrução do referido processo seja executada. Esta propriedade é comumente denominada *fairness*.[2]

Assim, estipulamos que o comportamento potencial de um sistema reativo é representado pelo conjunto das sequências de execução de instruções atômicas que satisfazem a propriedade de *fairness*.

[2]Esta é, obviamente, uma conceituação informal de *fairness*. Uma definição mais precisa será fornecida no próximo capítulo, onde, aliás, tipos diferentes de *fairness* serão considerados.

2.2.3 Propriedades Interessantes de Sequências de Execução

Dizer que um sistema reativo satisfaz uma dada propriedade equivale a dizer que todas as sequências de execução que representam seu comportamento e que obedecem à condição de *fairness* satisfazem tal propriedade. Geralmente, as propriedades interessantes de sistemas reativos são divididas em dois tipos:

- Propriedades de *safety*: aquelas que estipulam que uma dada situação deve ocorrer em *todos os pontos* da sequência de execução. O termo "safety" se deve à idéia de que o sistema deve sempre estar em um estado seguro.
- Propriedades de *liveness*: aquelas que estipulam que uma dada situação deve ocorrer em *algum ponto* (presente ou futuro) da sequência de execução. O termo "liveness" se deve à idéia de que o sistema deve mostrar "algum sinal de vida".

Geralmente, no estudo do comportamento de um sistema reativo, há um interesse conjunto nos dois tipos de propriedades, pois um sistema pode facilmente ser seguro sem satisfazer propriedades de *liveness*, ou exibir *liveness* sem ser seguro.

Um exemplo comum de propriedade de *safety* é a *exclusão mútua*: determinados grupos de instruções não podem ser intercalados em uma sequência de execução, o que equivale a dizer que um grupo de instruções deve sempre ser executado completamente antes que outro grupo seja executado.

Um exemplo direto de propriedade de *liveness* é exatamente a propriedade de *fairness* mencionada na subseção anterior: para cada processo, em algum ponto presente ou futuro da sequência de execução do sistema, uma instrução do processo deve ser executada. Outro exemplo de propriedade de *liveness* é a de que cada requisição feita ao sistema deve ser atendida em algum momento futuro.

2.2.4 Sequências de Estados

Alternativamente, podemos pensar nos *estados* da memória durante a execução de um processo. Estipulamos que o estado do processo em um dado instante consiste dos valores das variáveis manipuladas pelo processo, mais o valor do ponteiro de instrução, i.e., a posição da próxima instrução a ser executada pelo processo.

Quando vários processos compõem um sistema, introduzimos o conceito de *estado global* do sistema: em um dado instante, o estado global do sistema é composto pelos estados correntes dos processos que o compõem.

Quando raciocinamos em termos de estados, as instruções são vistas como operadores de mudança de estado. A cada sequência de execução corresponde uma sequência de estados, e vice-versa.

Por exemplo: suponhamos que as execuções dos processos P_a e P_b se iniciem, respectivamente, nos estados s_{a1} e s_{b1} (chamados de *estados iniciais*), e que as instruções dos processos provoquem as seguintes mudanças de estado:

- i_{a1} muda o processo P_a do estado s_{a1} para o estado s_{a2}.
- i_{a2} muda o processo P_a do estado s_{a2} para o estado s_{a3}.
- i_{b1} muda o processo P_b do estado s_{b1} para o estado s_{b2}.
- i_{b2} muda o processo P_b do estado s_{b2} para o estado s_{b3}.

Compondo o estado do sistema a partir dos estados dos processos, temos que o estado global inicial pode ser descrito por (s_{a1}, s_{b1}). A sequência de execução

$$i_{a1}, i_{b1}, i_{b2}, i_{a2}$$

corresponde, então, à sequência de estados globais

$$(s_{a1}, s_{b1}), (s_{a2}, s_{b1}), (s_{a2}, s_{b2}), (s_{a2}, s_{b3}), (s_{a3}, s_{b3})$$

Os modelos formais de sistemas reativos apresentados no próximo capítulo se baseiam nos conceitos de estado global e de sequências de estados, mas a correspondência com sequências de execução, ilustrada no exemplo, é direta.

Capítulo 3

Modelos Formais de Sistemas Reativos

Uma vez abstraídas as características de sistemas reativos que desejamos estudar, o passo seguinte consiste na definição dos objetos formais que servirão de modelos para os sistemas em questão. Este capítulo apresenta uma seleção (de forma alguma exaustiva) de modelos formais, variando do abstrato para o concreto.

3.1 Modelos Concretos × Modelos Abstratos

Embora o comportamento de um sistema reativo seja uma noção abstrata, o sistema em si pode ser representado por objetos dotados de uma existência concreta, como o código-fonte dos programas, por exemplo. Dentre as diversas maneiras de representar um sistema reativo, convencionou-se considerar como *modelos concretos* aquelas representações que estão mais próximas destes objetos concretos, e como *modelos abstratos* aquelas representações que se distanciam deles. Obviamente, esta não é uma classificação precisa ou rigorosa. Para nossos propósitos, basta ter em mente

que os modelos concretos e os modelos abstratos carregam a mesma informação sobre o sistema reativo que eles representam; é apenas a forma desta informação que pode, no caso de um modelo concreto, facilitar a exibição do modelo como um objeto finito para um usuário, ou, no caso de um modelo abstrato, facilitar o raciocínio automático sobre o comportamento do sistema através do uso de uma linguagem lógica, por exemplo.

Para fins de especificação ou verificação formal, geralmente, em primeiro lugar, é dada ao sistema uma representação concreta, facilmente construída a partir do texto dos programas e/ou diagramas fornecidos pelos projetistas do sistema; em seguida, este modelo concreto é traduzido para um modelo abstrato, sobre o qual serão aplicadas diretamente as técnicas lógicas para o estudo das propriedades do sistema.

Este capítulo se inicia com a apresentação do modelo abstrato dos sistemas de transição e de sua versão temporizada. Veremos como as sequências de execução de um sistema reativo que satisfazem a propriedade de *fairness* (que, conforme visto na seção 2.2, representam o comportamento do sistema) podem ser representadas por um sistema de transição.

Em seguida, um modelo concreto é considerado: um formalismo gráfico de diagramas de ação temporizados. Na verdade, examinamos separadamente o caso de sistemas multiprocessados e de sistemas multiprogramados. A tradução destas representações concretas para o modelo abstrato de sistemas de transição temporizados é discutida.

3.2 Sistemas de Transição

3.2.1 Definições

Um sistema de transição ([2, 14, 16]) consiste de um conjunto de estados, um conjunto de transições que transforma estados, e um conjunto de ações que rotulam as transições. Em sistemas de transição computacionais, nos quais estamos interessados, os estados correspondem aos estados da execução de um processo, as ações às instruções atômicas que compõem o processo, e as transições às execuções destas instruções.

Um estado do sistema de transição é descrito por um conjunto de fórmulas lógicas, podendo ser caracterizado pelo conjunto de fórmulas que representam enunciados verdadeiros nos estados correspondentes do processo representado. Nesta dissertação, para simplificar a exposição, fórmulas da Lógica Proposicional são usadas.[1] As fórmulas atômicas da Lógica Proposicional (i.e., as letras sentenciais) representam os enunciados sobre o processo nos quais estamos interessados, e são denominadas os *atributos* do processo.

Dado um conjunto de atributos $A = \{a, b, ...\}$ e um conjunto de ações $\Gamma = \{g, h, ...\}$, um sistema de transição S é definido como a quádrupla $(W, \rightarrow, I, w_0)$, onde

- W é o conjunto de *estados* (também chamados de "mundos possíveis");
- Para cada $g \in \Gamma$, $\xrightarrow{g} \subseteq W \times W$ é a *relação de transição* da ação g;
- $I : A \rightarrow 2^W$ é a função de *interpretação* dos atributos, onde cada $a \in A$ é mapeado para o conjunto de mundos onde a é verdadeiro;

[1] No caso de referências bibliográficas que utilizam Lógica de Primeira Ordem, como [5, 10, 11, 17], as definições foram adaptadas para a Lógica Proposicional.

- $w_0 \in W$ é o *estado inicial*.

Segundo estas definições, uma transição rotulada pela ação g é um par de estados (w, w'). Esta transição $(w, w') \in \xrightarrow{g}$ é geralmente escrita como $w \xrightarrow{g} w'$. w é chamado de *origem*, e w' de *destino* da transição.

Em um sistema de transição S, dizemos que uma ação g está *habilitada* (*enabled*) em um estado w se e somente se existe pelo menos uma transição rotulada com g cuja origem seja w, i.e., $w \xrightarrow{g} w'$ para algum $w' \in W$.

Definimos o conjunto $enable(g)$ como o conjunto de estados do sistema de transição onde a ação g está habilitada.

Uma *computação* de um sistema de transição S é uma sequência de estados $(\sigma_i)_{i \in Nat}$ onde

- $\sigma_0 = w_0$, i.e., o primeiro estado da sequência é o estado inicial de S;
- $\sigma_i \in W$ para todo $i \in Nat$; σ_i é chamado de i-ésimo *ponto* da computação.
- $\sigma_i \xrightarrow{g} \sigma_{i+1}$ para algum $g \in \Gamma$, para todo $i \in Nat$; uma computação define uma função $\sigma : Nat \longrightarrow \Gamma$, onde $\sigma(i) = g$ (g responsável pela transição de σ_i para σ_{i+1}).

3.2.2 Exemplos

3.2.2.1 Uma Variável Booleana

O exemplo a seguir, adaptado de [2], mostra como um sistema de transição pode representar o comportamento de uma variável booleana `b`:

Atributos: Apenas um atributo precisa ser considerado, relativo ao valor da variável; chamaremos este atributo de **valor**.

Ações: Estipulamos que as seguintes ações podem ser executadas sobre a variável `b`:

- Atribuição para verdadeiro (simbolizada por $b \leftarrow true$);
- Atribuição para falso (simbolizada por $b \leftarrow false$);
- Leitura de um valor verdadeiro (simbolizada por $true!$);
- Leitura de um valor falso (simbolizada por $false!$);
- A ação nula ($idle$), que não realiza nada (simbolizada por i).

Estados: O sistema de transição possui apenas dois estados, correspondentes aos valores possíveis da variável `b`. Chamaremos estes estados pelos mesmos nomes que os valores: **true** e **false**.

Transições: As seguintes transições estão presentes no sistema:

$$
\begin{aligned}
&\textbf{true} \xrightarrow{b \leftarrow true} \textbf{true} \\
&\textbf{true} \xrightarrow{b \leftarrow false} \textbf{false} \\
&\textbf{false} \xrightarrow{b \leftarrow true} \textbf{true} \\
&\textbf{false} \xrightarrow{b \leftarrow false} \textbf{false} \\
&\textbf{true} \xrightarrow{true!} \textbf{true} \\
&\textbf{false} \xrightarrow{false!} \textbf{false} \\
&\textbf{true} \xrightarrow{i} \textbf{true} \\
&\textbf{false} \xrightarrow{i} \textbf{false}
\end{aligned}
$$

Estado inicial: Estipulamos o estado inicial como sendo **false**.

Este sistema de transição, representado como um grafo direcionado, pode ser visto na figura 3.1. Os nós representam os estados, e as arestas, as transições.

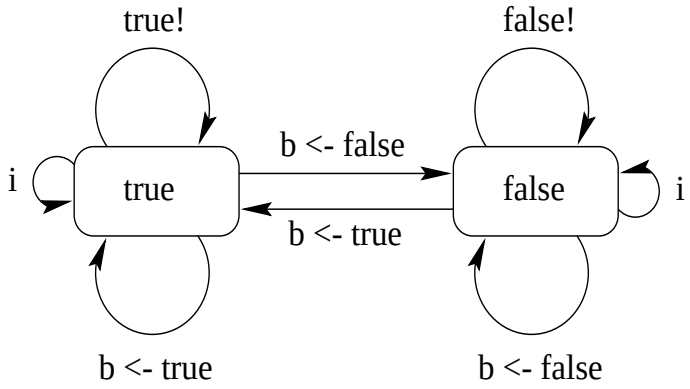

Figura 3.1: Sistema de transição correspondente a uma variável booleana

3.2.2.2 Um Programa Sequencial

Este outro exemplo, também adaptado de [2], ilustra como o comportamento de um programa sequencial pode ser modelado por um sistema de transição.[2] Seja o seguinte fragmento de programa

```
while true do
      1:  if not b then
          begin
      2:     b := true;
      3:     proc;
      4:     b := false;
          end
```

onde `proc` é a chamada de um procedimento que não altera o valor da variável booleana `b`. Os números de linha representam as instruções relevantes que podem ser

[2] [5] apresenta um mapeamento formal de programas em uma linguagem de programação simples para sistemas de transição.

apontadas pelo IP[3] durante a execução do programa.

Podemos decidir modelar o comportamento deste programa por um sistema de transição com apenas um atributo, que simboliza o valor do IP. Os estados do sistema, correspondentes aos valores possíveis do atributo único, são simbolizados por **1**, **2**, **3** e **4**, dos quais **1** é o estado inicial.[4]

As ações do alfabeto do sistema de transição são símbolos correspondentes às instruções numeradas:

- O teste da linha 1 dá origem a duas ações: uma, a leitura do valor **true** da variável `b`, é simbolizada por $b = true?$; a outra, a leitura do valor **false** da variável `b`, é simbolizada por $b = false?$;
- A atribuição da linha 2 dá origem à ação $b \leftarrow true$;
- A chamada do procedimento é representada pela ação $proc$;
- A atribuição da linha 4 dá origem à ação $b \leftarrow false$;

As transições do sistema são:

$$\begin{array}{lll} \mathbf{1} & \xrightarrow{b=true?} & \mathbf{1} \\ \mathbf{1} & \xrightarrow{b=false?} & \mathbf{2} \\ \mathbf{2} & \xrightarrow{b\leftarrow true} & \mathbf{3} \\ \mathbf{3} & \xrightarrow{proc} & \mathbf{4} \\ \mathbf{4} & \xrightarrow{b\leftarrow false} & \mathbf{1} \end{array}$$

[3] *Instruction Pointer*, ou Apontador de Instrução.

[4] Poderíamos ter incluído o valor da variável `b` como segundo atributo, aumentando o total de estados do sistema para 8. Isto seria perfeitamente correto, e só não foi feito aqui por razões didáticas, para que a composição dos sistemas dos dois exemplos – variável booleana e programa sequencial – fosse examinada mais adiante.

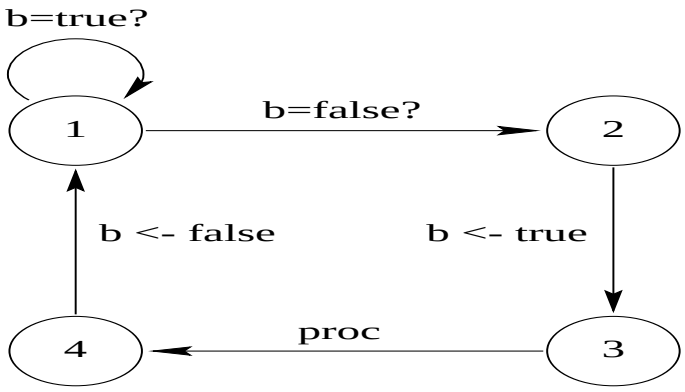

Figura 3.2: Sistema de transição correspondente a um programa sequencial

A representação gráfica deste sistema de transição pode ser vista na figura 3.2.

3.2.3 Compondo Sistemas de Transição

Um sistema de transição, a princípio, pode ser usado para modelar o comportamento de um único processo, como nos exemplos acima. No entanto, sistemas de transições distintos podem ser combinados para formar um novo sistema de transição, que representará, então, o comportamento do sistema composto pelos processos em questão. O fato de a composição de sistemas de transição gerar um novo sistema de transição está de acordo com o ponto de vista composicional defendido na seção 2.1.5: a distinção entre processos, ambiente e sistemas de processos não deve ser relevante para o estudo formal de sistemas reativos; todas estas entidades são analisadas de forma homogênea através do modelo abstrato de sistemas de transição.

Sejam n processos $P_1, P_2, \ldots, P_n$, modelados, respectivamente, pelos sistemas de transição $S_1, S_2, \ldots, S_n$. Existem duas formas básicas de encarar a execução concorrente dos n processos (denotada por $P_1 \mid\mid P_2 \mid\mid \ldots \mid\mid P_n$) e a composição dos sistemas

de transição correspondentes:

Se não existe qualquer interação entre os processos (i.e., suas execuções evoluem independentemente uma da outra, não havendo necessidade de compartilhamento de objetos, sincronização ou comunicação entre os processos), os sistemas de transição devem ser compostos através da operação de *produto simples*, onde não há restrição quanto à composição das transições.

Havendo interação entre os processos, o comportamento de $P_1 \mid\mid P_2 \mid\mid \ldots \mid\mid P_n$ é representado pelo *produto síncrono* dos sistemas de transição correspondentes. Nesta operação, a natureza das ações dos processos desempenha um papel na sincronização entre eles: uma transição de um processo só pode ocorrer em conjunto com uma outra transição específica de um outro processo, designada pela ação que a rotula.

Os dois tipos de produtos são definidos formalmente a seguir.

3.2.3.1 Produto Simples de Sistemas de Transição

O produto simples S de dois ou mais sistemas de transição, $S_1 \times S_2 \times \ldots \times S_n$, é definido do seguinte modo, segundo [2]:

- O conjunto de atributos A de S é a união disjunta dos conjuntos de atributos dos sistemas $S_1, S_2, \ldots, S_n$:

  $$A = A_1 \oplus A_2 \oplus \ldots \oplus A_n$$

  Ou seja, as proposições que representam enunciados sobre um dos sistemas não possuem relação com as proposições que representam enunciados sobre os outros sistemas.

- O conjunto de ações Γ de S é o produto cartesiano dos conjuntos de ações dos sistemas $S_1, S_2, \ldots, S_n$:

$$\Gamma = \Gamma_1 \times \Gamma_2 \times \ldots \times \Gamma_n$$

  Ou seja, uma ação do produto simples pode ser vista como sendo composta por ações dos sistemas componentes, uma ação de cada sistema.

- O conjunto de estados de S é o produto cartesiano dos conjuntos de estados dos sistemas $S_1, S_2, \ldots, S_n$:

$$W = W_1 \times W_2 \times \ldots \times W_n$$

  Ou seja, o estado global do produto simples é um registro do estado em que cada um dos sistemas componentes se encontra em um dado ponto de sua execução.

- As relações de transição de S são tais que, para cada $(g_1, g_2, \ldots, g_n) \in \Gamma$, $(w_1, w_2, \ldots, w_n) \stackrel{(g_1, g_2, \ldots, g_n)}{\longrightarrow} (w'_1, w'_2, \ldots, w'_n)$ se e somente se, para cada $i, 1 \leq i \leq n, w_i \stackrel{g_i}{\longrightarrow} w'_i$.

  Ou seja, uma transição do produto simples pode ser encarada como a ocorrência simultânea de uma transição de cada um dos sistemas componentes.

- A função de interpretação de S é $I : A \rightarrow 2^W$ tal que, se $a \in A$ é um atributo do sistema k, $I(a) = \{w \in W \mid (w \downarrow k) \in I_k(a)\}$.

  Ou seja, um atributo a de A, originalmente atributo do sistema S_k, é verdadeiro nos estados globais cuja k-ésima componente é um mundo onde a era verdadeiro segundo a função de valoração I_k, do sistema S_k.

- O estado inicial é $(w_{0_1}, w_{0_2}, \ldots, w_{0_n})$, o estado global composto pelos estados iniciais dos sistemas componentes.

O produto simples indica que os sistemas componentes realizam suas transições simultaneamente (i.e., cada sistema componente realiza uma transição em um dado

intervalo de tempo, como se um mesmo relógio comandasse todos os sistemas). Esta interpretação "síncrona" pode parecer contrariar a natureza irrestrita do produto simples, mas tal não é o caso: como o tempo não faz parte de nossa abstração (ver seção 2.2), e como não há qualquer interação entre os processos, as execuções separadas dos processos sempre podem ser encaradas como ocorrendo à mesma "velocidade".

3.2.3.2 Produto Simples e Intercalação de Ações

Na seção 2.2, fizemos uma suposição básica para justificar o uso de sequências de execução como modelos do comportamento de um sistema reativo: a suposição de que o efeito da execução simultânea de duas instruções atômicas g_1 e g_2 é o mesmo do que o efeito de qualquer uma das sequências g_1, g_2 ou g_2, g_1.

Isto possibilita uma definição alternativa de produto simples, que não envolve a criação de ações conjuntas do tipo (g_1, g_2). Esta definição alternativa difere da anterior nos seguintes itens:

- O conjunto de ações Γ de S é simplesmente a união dos conjuntos de ações dos sistemas $S_1, S_2, \ldots, S_n$:
$$\Gamma = \Gamma_1 \cup \Gamma_2 \cup \ldots \cup \Gamma_n$$
- As relações de transição de S são tais que, para cada $g_i \in \Gamma$ (onde i é o índice do sistema S_i a cujo conjunto de ações g_i pertencia originalmente), $(w_1, \ldots, w_i, \ldots, w_n) \xrightarrow{g_i} (w_1, \ldots, w_i', \ldots, w_n)$ se e somente se $w_i \xrightarrow{g_i} w_i'$ era, originalmente, uma transição do sistema S_i.

A intuição desta definição alternativa é a equivalência – garantida pela nossa suposição básica em 2.2 – entre as situações da figura 3.3, baseada na primeira definição de produto simples, e da figura 3.4, baseada na definição alternativa.

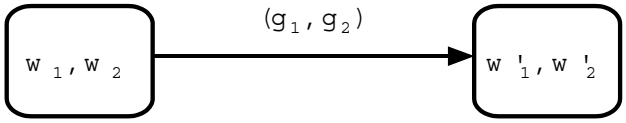

Figura 3.3: Ação conjunta no produto simples de sistemas de transição

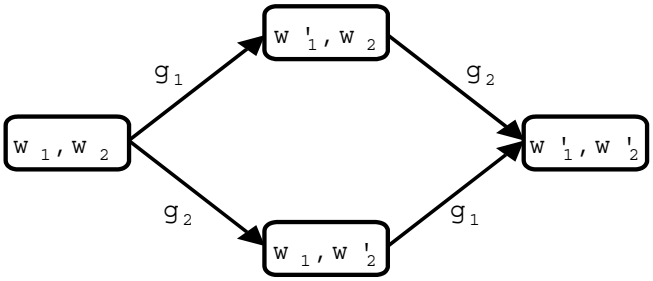

Figura 3.4: Ações individuais no produto simples de sistemas de transição

Na prática, porém, como estamos interessados em sistemas compostos por processos que interagem entre si de alguma forma, a noção de produto síncrono, definida a seguir, é mais útil do que a de produto simples.

3.2.3.3 Produto Síncrono de Sistemas de Transição

As relações de transição do produto simples de dois sistemas de transição S_1 e S_2 são tais que *qualquer* transição de S_1 pode ser tomada em conjunto com *qualquer* transição de S_2. Na prática, no entanto, as interações entre os dois processos representados podem proibir a ocorrência conjunta de determinadas transições. Isto ocorre no caso em que os processos precisam se comunicar de forma síncrona, trocando mensagens, por exemplo, ou no caso em que o funcionamento do sistema depende da sincronização entre os processos que compõem o sistema, como a situação a seguir:

Consideremos os sistemas de transição correspondentes à variável booleana (3.2.2.1) e ao programa sequencial (3.2.2.2). Na execução concorrente dos dois processos, não é razoável esperar que a transição **true** $\stackrel{b \leftarrow true}{\longrightarrow}$ **true** da variável booleana ocorra em conjunto com a transição **4** $\stackrel{b \leftarrow false}{\longrightarrow}$ **1** do programa sequencial, pois isto significaria que a variável `b` está assumindo o valor *true* ao mesmo tempo que o programa está atribuindo a ela o valor $false$! No produto síncrono, então, a transição conjunta

$$\textbf{(true, 4)} \stackrel{(b \leftarrow true, b \leftarrow false)}{\longrightarrow} \textbf{(true, 1)}$$

não pode ocorrer.

Na verdade, a relação de transição do produto síncrono de dois sistemas é um subconjunto da relação de transição do produto simples entre eles, pois as transições rotuladas por ações conjuntas proibidas, como a do parágrafo anterior, estão presentes no produto simples mas não no produto síncrono.

Na construção do produto síncrono de n sistemas de transição, as ações conjuntas

permitidas são especificadas por uma *restrição de sincronização* ([2]), composta por *vetores de sincronização*, que são n-uplas de ações. Somente as transições rotuladas por estas n-uplas podem estar presentes no produto síncrono.

Assim, o produto síncrono S de dois ou mais sistemas de transição, $(S_1, S_2, \ldots, S_n)$, é definido do seguinte modo:

- O conjunto de atributos A de S é definido de maneira idêntica ao conjunto de atributos do produto simples.
- O conjunto de ações Γ de S é o produto cartesiano dos conjuntos de ações dos sistemas $S_1, S_2, \ldots, S_n$, restrito aos vetores de sincronização, conforme explicado acima.
- O conjunto de estados de S é definido de maneira idêntica ao conjunto de estados do produto simples.
- As relações de transição de S são definidas de maneira semelhante às relações de transição do produto simples (segundo a definição envolvendo ações conjuntas na seção 3.2.3.1), apenas restritas ao conjunto de ações globais permitidas.
- A função de interpretação de S é definida de maneira idêntica à função de interpretação do produto simples.
- O estado inicial é definido de maneira idêntica ao estado inicial do produto simples.

3.2.3.4 Exemplo

Vamos examinar o produto síncrono do sistema correspondente ao programa sequencial, S_{prog}, com o sistema correspondente à variável booleana `b`, S_b. Repetimos abaixo as relações de transição destes dois sistemas:

S_{prog} :

1. **1** $\xrightarrow{b=true?}$ **1**
2. **1** $\xrightarrow{b=false?}$ **2**
3. **2** $\xrightarrow{b\leftarrow true}$ **3**
4. **3** $\xrightarrow{proc}$ **4**
5. **4** $\xrightarrow{b\leftarrow false}$ **1**

S_b :

1. **true** $\xrightarrow{b\leftarrow true}$ **true**
2. **true** $\xrightarrow{b\leftarrow false}$ **false**
3. **false** $\xrightarrow{b\leftarrow true}$ **true**
4. **false** $\xrightarrow{b\leftarrow false}$ **false**
5. **true** $\xrightarrow{true!}$ **true**
6. **false** $\xrightarrow{false!}$ **false**
7. **true** $\xrightarrow{i}$ **true**
8. **false** $\xrightarrow{i}$ **false**

O produto *livre* $S_{prog} \times S_b$ possui 40 transições. Muitas delas, porém, serão proibidas pela restrição de sincronização definida a seguir:

Quando o programa verifica que o valor de `b` é *true* (transição 1 de S_{prog}), a única ação compatível em S_b é a que simboliza a leitura do mesmo valor (transição 5). Considerações análogas podem ser feitas para o valor $false$ (transição 2 de S_{prog} e transição 6 de S_b).

Quando o programa atribui o valor $true$ à variável `b` (transição 3 de S_{prog}), a única ação compatível em S_b é aquela em que a variável assume o mesmo valor (transições 1 e 3). Considerações análogas podem ser feitas para o valor $false$ (transição 5 de S_{prog} e transições 2 e 4 de S_b).

Finalmente, quando o programa executa o procedimento *proc* (transição 4 de S_{prog}), o valor da variável `b` não é alterado; a única ação possível em S_b, então, é a ação nula (transições 7 e 8).

Estas considerações definem, assim, os seguintes vetores de sincronização para o produto (S_{prog}, S_b):

$$(b \leftarrow true, b \leftarrow true)$$
$$(b \leftarrow false, b \leftarrow false)$$
$$(true?, true!)$$
$$(false?, false!)$$
$$(proc, i)$$

onde i representa a ação nula (*idle*).

As transições do produto síncrono serão apenas aquelas rotuladas pelas ações conjuntas presentes em algum dos vetores de sincronização acima. Considerando o estado inicial **(1, false)** e apenas os estados globais alcançáveis a partir dele, as transições do produto síncrono serão as seguintes:

$$\begin{array}{lcl} \textbf{(1, false)} & \xrightarrow{(b=false?, false!)} & \textbf{(2, false)} \\ \textbf{(2, false)} & \xrightarrow{(b \leftarrow true, b \leftarrow true)} & \textbf{(3, true)} \\ \textbf{(3, true)} & \xrightarrow{(proc, i)} & \textbf{(4, true)} \\ \textbf{(4, true)} & \xrightarrow{(b \leftarrow false, b \leftarrow false)} & \textbf{(1, false)} \end{array}$$

É este sistema de transição que representa o comportamento do sistema $P_{prog} \mid\mid P_b$, onde P_{prog} é o processo correspondente ao programa sequencial de 3.2.2.2, e P_b o processo correspondente à variável booleana de 3.2.2.1.

3.2.3.5 Produto Síncrono e Intercalação de Ações

A exemplo do que foi feito com o produto simples na seção 3.2.3.2, podemos criar uma definição alternativa do produto síncrono onde ações conjuntas (n-uplas de ações) sejam substituídas por ações simples. No entanto, como o produto síncrono representa uma situação onde ocorre a execução simultânea de mais de uma ação, algumas ações conjuntas permanecerão, simbolizando exatamente as ações que *precisam* ser executadas simultaneamente.

Examinando o conjunto de vetores de sincronização (que contém exatamente as ações conjuntas que *podem* ocorrer no produto síncrono), decidimos substituir as ações conjuntas onde somente uma componente é diferente de *idle*. Assim, se tivermos um vetor de sincronização da forma

$$(idle, idle, ..., g_k, ..., idle, idle)$$

podemos substituir as ocorrências desta ação conjunta no produto síncrono simplesmente por

$$g_k$$

obtendo um sistema de transição equivalente, com menos ações conjuntas.

Usando este recurso, podemos definir um conjunto de vetores de sincronização onde as únicas ações conjuntas com mais de uma componente diferente de *idle* sejam aquelas cujas componentes representam ações que *precisam* ser executadas simultaneamente. Ações que podem ou não ser executadas simultaneamente passam a figurar somente em vetores de sincronização onde todas as outras componentes são iguais a *idle*.

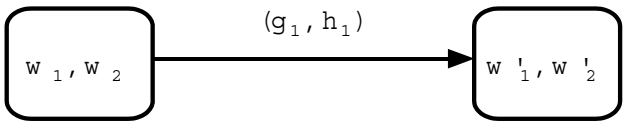

Figura 3.5: Ação conjunta no produto síncrono de sistemas de transição

Por exemplo, sejam os processos P_1 e P_2 representados pelos sistemas de transição S_1 e S_2. Sejam $\Gamma_1 = \{g_1, g_2, g_3\}$ o conjunto de ações de S_1 e $\Gamma_2 = \{h_1, h_2, h_3\}$ o conjunto de ações de S_2. Suponhamos que as únicas ações que precisam ser executadas simultaneamente são g_2 e h_2. Um conjunto de vetores de sincronização V_1 adequado é

$$(g_1, h_1)(g_1, h_3)(g_1, idle)(g_2, h_2)(g_3, h_1)(g_3, h_3)(g_3, idle)(idle, h_1)(idle, h_3)$$

Um conjunto alternativo V_2, mais simples, seria

$$(g_1, idle)(g_2, h_2)(g_3, idle)(idle, h_1)(idle, h_3)$$

No produto síncrono (S_1, S_2) usando V_1, a transição da figura 3.5 pode ocorrer:

Usando V_2, a transição da figura 3.5 não é possível, mas, segundo a abstração adotada por nós, o conjunto de transições da figura 3.6, com a ação $(g_1, idle)$ renomeada para g_1 e a ação $(idle, h_1)$ renomeada para h_1, gera computações que representam as mesmas sequências de execução de $P_1 \mid\mid P_2$.

Independentemente do conjunto de vetores de sincronização usado (V_1 ou V_2), as ações g_2 e h_2 só podem ocorrer em conjunto, na forma (g_2, h_2).

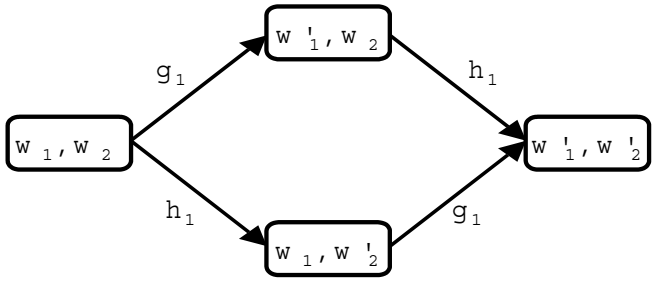

Figura 3.6: Ações individuais no produto síncrono de sistemas de transição

3.2.3.6 Uma Definição Alternativa de Produto Síncrono

Em [13], C.A.R. Hoare define o significado do operador de composição concorrente de processos || de forma diferente da exposta acima. Embora o modelo de comportamento de processos usado por ele não seja exatamente o de sistemas de transição, suas construções são facilmente traduzidas para as empregadas por nós. A seguir, tomaremos os significados das construções de Hoare em termos de sistemas de transição, comparando-as com a noção de produto síncrono definida acima.

Digamos que os comportamentos de dois processos P_1 e P_2 estejam representados, respectivamente, pelos sistemas de transição S_1 e S_2.[5] O produto síncrono destes sistemas de transição, segundo Hoare, é definido da seguinte forma:

- Se os conjuntos de ações Γ_1 de S_1 e Γ_2 de S_2 não possuírem nenhum símbolo de ação em comum, o produto síncrono de Hoare é igual ao produto simples, pois a ausência de ações em comum entre os dois sistemas significa que não há como haver interação entre eles.

- Se os conjuntos de ações Γ_1 de S_1 e Γ_2 de S_2 forem iguais, as únicas ações

[5]Como o operador || é associativo à esquerda – $P_1 \parallel P_2 \parallel P_2 \parallel \ldots \parallel P_n = (\ldots((P_1 \parallel P_2) \parallel P_3)\ldots \parallel P_n)$, vamos nos restringir ao caso da composição concorrente de dois processos.

conjuntas possíveis são aquelas em que a primeira componente é igual à segunda. Isto equivale a dizer que um processo só pode executar uma ação se o outro processo executar a mesma ação simultaneamente.

- Se os conjuntos de ações Γ_1 de S_1 e Γ_2 de S_2 forem diferentes, mas a interseção entre eles for não-vazia, duas possibilidades existem para cada ação g de Γ_1:
  - Se g também pertencer a Γ_2, g só pode figurar em ações conjuntas da forma (g, g). Isto significa que os processos só podem executar g de forma simultânea.
  - Se g não pertencer à interseção $\Gamma_1 \cap \Gamma_2$, g só pode figurar em ações conjuntas onde a outra componente seja uma ação do outro processo que também não pertença a $\Gamma_1 \cap \Gamma_2$. Assim, ações "locais" (conhecidas apenas por um processo) podem ser executadas em conjunto com qualquer outra ação "local" do outro processo.

O produto síncrono de Hoare pode ser expresso em termos da definição de produto síncrono usada por nós, bastando construir um conjunto apropriado de vetores de sincronização. Na verdade, no produto síncrono de Hoare existe um conjunto de vetores de sincronização definidos implicitamente, com base nos nomes das ações. De fato, dados dois sistemas de transição S_1 e S_2, com conjuntos de ações Γ_1 e Γ_2, suponhamos que os símbolos de ações envolvidos estejam particionados da seguinte forma:

- $\{g_1, g_2, \ldots, g_k\} \subset \Gamma_1 \cap \Gamma_2$
- $\{h_1, h_2, \ldots, h_l\} \subset \Gamma_1 \setminus \Gamma_2$
- $\{j_1, j_2, \ldots, j_m\} \subset \Gamma_2 \setminus \Gamma_1$

Os vetores de sincronização que fazem com que o produto síncrono (S_1, S_2) seja igual ao produto síncrono de Hoare são

- $\{(g_x, g_x) \mid 1 \leq x \leq k\}$
- $\{(h_x, j_y) \mid 1 \leq x \leq l, 1 \leq y \leq m\}$

O produto síncrono empregado por nós, porém, não pode ser definido em termos do produto síncrono de Hoare. Por exemplo, na composição apresentada em 3.2.3.4, as únicas ações comuns aos dois processos são $b \leftarrow true$ e $b \leftarrow false$. O produto síncrono de Hoare, então, permitiria combinações indesejáveis entre outras ações, como $(false?, true!)$, por exemplo.

É bem verdade que, neste caso, esta limitação do produto síncrono de Hoare poderia ser resolvida através da renomeação das ações envolvidas, mas existem situações em que tal renomeação não seria suficiente.

Por exemplo: suponhamos que a execução do procedimento `proc` pudesse alterar (ou não) o valor da variável `b`. Esta nova situação corresponde à adição dos seguintes vetores de sincronização aos já definidos em 3.2.3.4:

- $(proc, b \leftarrow true)$
- $(proc, b \leftarrow false)$

Não existe nenhuma renomeação adequada de ações capaz de gerar um produto síncrono de Hoare igual ao produto síncrono definido pelo novo conjunto de vetores de sincronização. A definição de produto síncrono empregada por nós, envolvendo vetores de sincronização definidos explicitamente, é mais geral e expressiva do que a de Hoare.[6]

[6]Isto não significa, de forma alguma, que o sistema em questão não possa ser expresso no formalismo de Hoare ([13]), no qual os tipos mais diversos de restrições de sincronização podem ser definidos usando construções como semáforos, etc.; estamos nos referindo apenas à definição do produto síncrono.

3.2.3.7 Sequências de Execução, Computações e *Fairness*

Uma sequência de execução de um sistema reativo corresponde a uma computação do sistema de transição que representa este sistema reativo.[7] Conhecido o estado inicial do sistema, cada ação na sequência de execução corresponde a uma transição da computação. Por exemplo, seja a sequência de execução

$$a_1, a_2, a_3, ..., a_n$$

onde cada ação $a_i, 1 \leq i \leq n$, provoca uma passagem do estado w_{i-1} para o estado w_i, e o estado inicial é w_0. Esta sequência corresponde à computação

$$w_0, w_1, w_2, w_3, ..., w_n$$

Como visto na seção 2.2, duas ações a_1 e a_2 executadas simultaneamente pelos componentes P_1 e P_2 de um sistema reativo S aparecem na sequência de execução intercaladas como a_1, a_2 ou como a_2, a_1. O sistema de transição que representa o comportamento de S, no entanto, é o produto síncrono dos sistemas de transição correspondentes a P_1 e P_2, e, neste produto síncrono, a execução simultânea de a_1 e a_2 é representada por uma transição rotulada pelo par (a_1, a_2).

Isto sugere que computações carregam mais informação (sobre a simultaneidade da execução das ações) do que sequências de execução. Isto ocorre, de fato; cabe esclarecer, porém, o papel de uma e outra construção no estudo de sistemas reativos: uma sequência de execução é o produto da *observação* do comportamento de um sistema; durante esta observação, optamos por não registrar a simultaneidade da ocorrência de duas ações a_1 e a_2, uma vez que o efeito é o mesmo que o das sequências a_1, a_2 e a_2, a_1.

[7] No caso não-determinístico, em que um único estado do sistema de transição pode originar mais de uma transição rotulada pela mesma ação, uma única sequência de execução pode corresponder a mais de uma computação.

A representação dos processos do sistema como sistemas de transição e a combinação destes sistemas de transição através do produto síncrono envolvem a definição de um conjunto de vetores de sincronização, onde é decidido, usando informação que pode não ser depreensível da observação de uma execução do sistema, quais ações devem/podem ocorrer simultaneamente.[8] É esta informação sobre a simultaneidade da execução das ações que está presente nas computações, mas não nas sequências de execução. Por isso, uma única computação pode corresponder a mais de uma sequência de execução.

Uma vez definido o sistema de transição correspondente a um sistema reativo, o comportamento do sistema reativo é definido, então, como um conjunto de computações inerentes ao sistema de transição. Porém, da mesma forma que restringimos as sequências de execução de interesse àquelas que satisfaziam alguma propriedade de *fairness*, aqui também restringimos o conjunto de computações de interesse às computações que satisfazem o equivalente formal da propriedade de *fairness* escolhida.

Neste ponto, podem ser introduzidas as versões mais precisas das propriedades de *fairness* geralmente usadas no estudo de sistemas reativos (ver [6]).

Sejam $P_1, P_2, ..., P_n$ os n processos componentes de um sistema reativo P. Seja S o sistema de transição correspondente a P. Seja $C = \sigma_0, \sigma_1, \sigma_2, ...$ uma computação infinita de S.[9]

Seja $executado(i)$ uma proposição verdadeira no ponto σ_i de C se e somente se a transição de σ_{i-1} para σ_i é rotulada por uma ação do processo P_i diferente da ação nula (ou rotulada por uma tupla de ações onde figura uma ação do processo P_i

[8]Esta informação "não depreensível" da observação de uma execução do sistema consiste de decisões do projetista a respeito de quais ações devem/podem ocorrer simultaneamente, decisões estas que podem não transparecer em uma dada execução.

[9]Uma execução *finita* é representada por uma computação infinita na qual, para algum $j \in Nat$, todos os pontos σ_k com $k \geq j$ consistem do mesmo estado w_f (o estado final da execução), e todas as transições $\sigma_k \rightarrow \sigma_{k+1}$ para $k \geq j$ são rotuladas por *idle*.

diferente da ação nula).

Seja *pronto*(i) uma proposição verdadeira nos estados de S (e apenas nestes estados de S) onde existe alguma ação do processo i habilitada (ver 3.2.1), ainda que a transição correspondente não esteja presente na computação C.

Definimos abaixo as seguintes propriedades possíveis de uma computação:

- Imparcialidade, também chamada de *fairness* incondicional;
- Justiça, também chamada de *weak fairness*;
- *Fairness*, também chamada de *strong fairness*.

Dizemos que a computação C satisfaz a propriedade de *fairness incondicional* (também chamada de *imparcialidade*) se e somente se, para todo processo $P_i, 1 \leq i \leq n$, para todo ponto σ_k de C, existe um ponto posterior $\sigma_l, l > k$ onde a proposição *executado*(i) é verdadeira. Ou seja, uma computação C é imparcial se e somente se todo processo é executado infinitas vezes em C.

Dizemos que a computação C satisfaz a propriedade de *weak fairness* (também chamada de *justiça*) se e somente se, para todo processo $P_i, 1 \leq i \leq n$, o fato de existir um ponto σ_k de C tal que *pronto*(i) é verdadeiro em todo ponto posterior σ_l, $l > k$, implica que, para todo ponto σ_p de C, existe um ponto posterior $\sigma_q, q > p$ onde a proposição *executado*(i) é verdadeira. Ou seja, uma computação C é justa se e somente se todo processo que se torna permanentemente pronto para executar é, de fato, executado infinitas vezes.

Dizemos que a computação C satisfaz a propriedade de *strong fairness* (também chamada simplesmente de *fairness*) se e somente se, para todo processo $P_i, 1 \leq i \leq n$, o fato de, para todo ponto σ_k de C, sempre existir um ponto posterior $\sigma_l, l > k$, onde *pronto*(i) é verdadeiro implica que, para todo ponto σ_p de C, existe um ponto posterior

$\sigma_q, q > p$ onde a proposição *executado*(i) é verdadeira. Ou seja, uma computação C é *fair* se e somente se todo processo que se torna pronto para executar infinitas vezes é, de fato, executado infinitas vezes.

Apesar de estas três variantes serem as mais frequentemente consideradas, existem outras definições de *fairness*, que não serão abordadas nesta dissertação (ver [8], por exemplo).

3.3 Sistemas de Transição Temporizados

Quando sistemas reativos de tempo real são o objeto de estudo, a informação contida em sistemas de transição não é suficiente. Um modelo formal para sistemas de tempo real deve incluir, obviamente, informações relativas à passagem do tempo ao longo da execução, bem como informações relativas às restrições de tempo que devem ser satisfeitas pelo sistema.

O modelo formal abstrato apresentado aqui é o de *sistemas de transição temporizados* (*timed transition systems* – ver [10, 11]).

3.3.1 Definições

Para modelar o tempo, usa-se um conjunto infinito totalmente ordenado (TIME, $\leq$) com mínimo 0. Pode ser usado o próprio conjunto dos naturais. Desde que seja suficiente uma precisão finita na mensuração do tempo (segundos, milissegundos, etc.) durante a execução do sistema, a escala utilizada sempre poderá ser posta em correspondência biunívoca com os naturais.[10]

[10]Existem formalismos onde o tempo é modelado de forma contínua, não discreta (ver [1]), mas estes formalismos não serão abordados nesta dissertação.

Um elemento ∞, comparável com todos os elementos t de TIME (tal que $t \leq \infty, \forall t \in \text{TIME}$), está disponível. Note que ∞ não pertence a TIME.

Dado este domínio para representar o tempo, a idéia é estender a definição de sistema de transição, atribuindo *limites de tempo* (*time bounds*) máximo e mínimo para cada uma das ações do conjunto Γ.

Um sistema de transição temporizado é definido como sendo uma sêxtupla

$$(W, \rightarrow, l, u, I, w_0)$$

onde

- W é o conjunto de estados, como em um sistema de transição;
- Para cada $g \in \Gamma$, $\xrightarrow{g} \subseteq W \times W$ é a *relação de transição* da ação g, como em um sistema de transição;
- l mapeia cada $g \in \Gamma$ em um elemento $l(g) \in \text{TIME}$;
- u mapeia cada $g \in \Gamma$ em um elemento $u(g) \in \text{TIME} \cup \{\infty\}$ tal que $u(g) \geq l(g)$;
- $I : A \rightarrow 2^W$ é a função de *interpretação* dos atributos, onde cada $a \in A$ é mapeado para o conjunto de mundos onde a é verdadeiro, como em um sistema de transição;
- w_0 é o estado inicial, como em um sistema de transição.

As funções l e u representam, respectivamente, o limite mínimo (*lower bound*) e o limite máximo (*upper bound*) de cada ação de Γ. Intuitivamente, o limite mínimo de uma ação é o menor período de tempo durante o qual a ação deve estar habilitada antes de ser executada. O limite máximo de uma ação é o maior período de tempo durante o qual a ação pode permanecer habilitada sem ser executada. Formalmente,

os limites mínimo e máximo são definidos através do uso das noções de *sequência temporizada de estados* e de *computação temporizada*:

Uma *sequência temporizada de estados* (*timed state sequence*) de um sistema de transição temporizado é um par $\rho = (\sigma, \mathrm{T})$, onde σ é uma sequência infinita de estados ($\sigma_i \in W$) e T é uma sequência infinita de tempos correspondentes ($\mathrm{T}_i \in \mathrm{TIME}$), satisfazendo

monotonicidade: para todo $i \geq 0$, ou $\mathrm{T}_{i+1} = \mathrm{T}_i$, ou $\mathrm{T}_{i+1} > \mathrm{T}_i$ e $\sigma_{i+1} = \sigma_i$. Ou seja, mudanças de estado só podem ocorrer se o tempo permanecer inalterado: as transições são instantâneas.

progresso: para todo $t \in \mathrm{TIME}$, existe um $i \geq 0$ tal que $\mathrm{T}_i \geq t$. Ou seja, o tempo sempre avança.

Uma *computação temporizada* sobre um sistema de transição temporizado é uma sequência temporizada de estados (σ, T) tal que

- σ é uma computação do sistema de transição subjacente. Para todo $i \geq 0$, σ define uma transição $\sigma_i \xrightarrow{\sigma(i)} \sigma_{i+1}$;
- (limite mínimo): para todo $i \geq 0$ no domínio de σ, existe um $j \leq i$ tal que $\mathrm{T}_i - \mathrm{T}_j > l(\sigma(i))$ e $\sigma(i)$ está habilitada em todo estado σ_k para $j \leq k \leq i$.
- (limite máximo): para todo $g \in \Gamma$ e $i \geq 0$, existe $j \geq i$ com $\mathrm{T}_j - \mathrm{T}_i \leq u(g)$ tal que ou g não está habilitada em σ_j ou $g = \sigma(j)$.

3.3.2 Comentários

3.3.2.1 Transições instantâneas

Embora, intuitivamente, possamos esperar que a execução de uma ação possua alguma duração, as definições acima estipulam que as transições (i.e., as mudanças de estado) são instantâneas. Duas observações devem ser feitas a este respeito:

- Podemos supor que uma mudança de estado diz respeito às características *observáveis* do sistema, e o que observamos é que o sistema se encontra em um determinado estado durante um certo intervalo de tempo e, no instante seguinte, já se encontra em um estado diferente.
- Caso seja necessário considerar ações com duração, podemos
    1. "Simular" a duração através do limite inferior l da ação, caso no qual devemos exigir, ainda, que os efeitos observáveis da ação só se manifestem após a transição (que continua a ser considerada instantânea), ou
    2. Refinar o espaço de estados para conter estados intermediários da execução da ação.

3.3.2.2 Ações auto-desabilitadoras

A condição de limite mínimo não exclui a possibilidade de que uma ação g seja executada mais de uma vez em um intervalo de tempo de duração $l(g)$. Isto pode ocorrer se g permanecer habilitada após sua execução (i.e., o estado destino w' da transição $w \xrightarrow{g} w'$ também pertence a $enable(g)$). Se este fenômeno for indesejável, o espaço de estados pode ser refinado para evitar sua ocorrência. Ou, equivalentemente, pode-se exigir que todas as ações sejam auto-desabilitadoras (i.e., quando executadas, conduzem a estados onde não estão habilitadas).

3.3.2.3 Computações Temporizadas × Computações

Se tomarmos somente os componentes de estado de uma computação temporizada qualquer C sobre um sistema de transição temporizado S, o resultado será uma computação C^- do sistema de transição subjacente S^-. Isto implica que os métodos dedutivos para sistemas de transição (não-temporizados) são corretos para sistemas de transição temporizados. Toda propriedade (que não envolva o tempo) verdadeira em todas as computações de S^- será verdadeira em todas as computações temporizadas de S.

A recíproca, porém, não se verifica. Existem computações de S^- que não podem formar os componentes de estado de nenhuma computação temporizada de S, porque os limites de tempo das ações não são respeitados.

Um caso especial é o de uma ação que tenha o limite superior igual a ∞. Pela definição de limite superior, esta ação não pode ficar permanentemente habilitada sem ser executada. Isto equivale a uma espécie de propriedade de *justiça* para ações. Assim, qualquer computação formada pelos componentes de estado de uma computação temporizada é justa, ainda que o sistema de transição subjacente S^- contenha computações não-justas.

3.3.2.4 *Stuttering*

Não estão proibidos pares consecutivos repetidos em uma computação temporizada: $(\sigma_i, \mathrm{T}_i), (\sigma_{i+1}, \mathrm{T}_{i+1})$ com $\sigma_i = \sigma_{i+1}$ e $\mathrm{T}_i = \mathrm{T}_{i+1}$. Esta repetição é um exemplo de um passo-*stutter* ("gagueira", em inglês). Convenciona-se que a ação responsável pela transição de um passo-*stutter* é a ação nula (*idle*).[11]

[11] Da mesma forma que a ação nula é responsável pelos passos onde o tempo avança e o estado não muda – os tiques do "relógio".

O conjunto de computações temporizadas de um sistema de transição temporizado é fechado sobre a adição finita de passos-*stutter*.

3.3.2.5 Translação da Origem do Tempo

O conjunto de computações temporizadas de um sistema de transição temporizado é fechado sobre a adição de uma constante inteira a todos os componentes de tempo. Isto significa que um sistema de transição temporizado não pode fazer referência ao tempo absoluto. Assim, não há perda de generalidade em supor que o instante da primeira transição de uma computação temporizada seja 0.

3.3.2.6 O Produto Síncrono de Sistemas de Transição Temporizados

O produto síncrono de sistemas de transição temporizados é definido da mesma forma que o produto síncrono de sistemas de transição não-temporizados, salvo no que diz respeito aos limites mínimos e máximos das ações conjuntas permitidas pelos vetores de sincronização. Para que o produto síncrono de sistemas de transição temporizados seja também um sistema de transição temporizado, é razoável adotar a seguinte definição:

Sejam S_1 e S_2 dois sistemas de transição temporizados. Seja (g_1, g_2) uma ação conjunta permitida pelas restrições de sincronização. Os limites de tempo desta ação conjunta são definidos do seguinte modo:

- $l(g_1, g_2) = max(l(g_1), l(g_2))$. Ou seja, o limite inferior da ação conjunta deve ser o maior dos limites inferiores das ações que a compõem.
- $u(g_1, g_2) = min(l(g_1), l(g_2))$. Ou seja, o limite superior da ação conjunta deve ser o menor dos limites superiores das ações que a compõem.

3.3.2.7 Multiprocessamento × Multiprogramação

Quando os sistemas computacionais estudados não são de tempo real, a abstração adotada por nós (ver 2.2) identifica o paralelismo verdadeiro de várias CPUs com a execução intercalada de processos em um esquema multitarefa em uma única CPU. Quando há restrições de tempo, porém, não é possível ignorar a diferença entre multiprocessamento e multiprogramação, como mostra o seguinte exemplo simples, adaptado de [11]:

Os processos P_1 e P_2 consistem, cada um, de uma ação: respectivamente g_1 e g_2. O início da execução de P_1 é simultâneo ao início da execução de P_2. Os limites de tempo das ações são definidos como

$$l(g_1) = l(g_2) = u(g_1) = u(g_2) = 1$$

Ou seja, as duas ações devem permanecer habilitadas durante exatamente uma unidade de tempo antes de serem executadas, e sua execução deve ocorrer imediatamente após esta unidade de tempo. Em outras palavras: a execução conjunta de P_1 e P_2 deve terminar após uma unidade de tempo (visto que transições não possuem duração).

Em um ambiente com dois processadores, estas restrições são facilmente satisfatíveis: basta que as ações g_1 e g_2 sejam executadas em paralelo, simultaneamente. Em um ambiente multiprogramado, no entanto, onde um processador é compartilhado pelos dois processos, as restrições de tempo jamais serão satisfeitas, pois qualquer execução conjunta de P_1 e P_2 leva 2 unidades de tempo.[12]

Esta diferença entre multiprocessamento e multiprogramação no que diz respeito

[12]Desde que valham as seguintes considerações: (1) uma ação de um processo é considerada habilitada apenas quando o processo estiver ativo (rodando na CPU), e (2) a troca de processos (*swapping*) é instantânea. Estas condições serão formalizadas na próxima seção.

a sistemas de tempo real fez com que alguns autores (ver [15]) alegassem que o modelo de ações intercaladas não é adequado para a análise *quantitativa* de sistemas de tempo real. Esta alegação foi refutada em [11], onde são apresentados os modelos abordados na próxima seção desta dissertação.

3.4 Diagramas de Ação Temporizados

Diagramas de ação temporizados servem como modelos concretos de sistemas reativos. Nesta seção, definimos tais diagramas e discutimos mapeamentos deles para o modelo abstrato de sistemas de transição temporizados, separadamente para o caso de sistemas multiprocessados e para o caso de sistemas multiprogramados.

3.4.1 Definições

3.4.1.1 Multiprocessamento

Um *sistema multiprocessado* P composto por m processos tem a forma

$$\Theta[P_1 \mid\mid P_2 \mid\mid ... \mid\mid P_m]$$

onde Θ é um conjunto finito de sentenças lógicas que definem o estado inicial da memória do sistema. Cada um dos processos P_i pode ser representado por texto de programa ou por um diagrama de ação. No nosso caso, usaremos diagramas de ação temporizados, definidos a seguir.

Dados um conjunto de símbolos de atributo A, um conjunto de símbolos de ação Γ, um conjunto ordenado TIME e um elemento ∞ como na definição de sistemas de transição temporizados, um *diagrama de ação temporizado* é um grafo direcionado

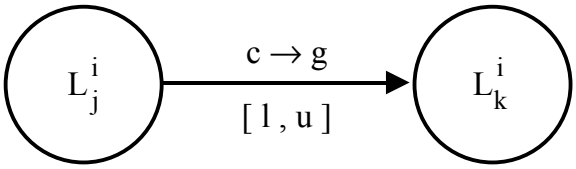

Figura 3.7: Aresta de um diagrama de ação temporizado

finito. Cada nó é chamado um local [13]. Se o diagrama do processo i tem n locais, os nós serão denotados $L_0^i, L_1^i, ..., L_{n-1}^i$. Cada aresta é rotulada por uma instrução com guarda $c \rightarrow g$ (onde c é uma expressão proposicional envolvendo atributos em A, e $g \in \Gamma$) e por um par $[l, u]$, com $l \in$ TIME e $u \in$ TIME$\cup\{\infty\}$ representando os limites de tempo da ação g. Uma aresta entre dois locais L_j^i e L_k^i, representando a instrução $c \rightarrow g$, com limites de tempo l e u, seria representada graficamente como na figura 3.7.

O significado pretendido desta aresta é o seguinte: quando o controle do processo i estiver no local L_j^i, a ação g pode ser executada somente se o controle permanecer em L_j^i, com a guarda c verdadeira, durante l unidades de tempo.

Além disso, o controle do processo não pode permanecer no local L_j^i durante mais de u unidades de tempo com a guarda c continuamente satisfeita. Para evitar que isto aconteça, ou o processo deve executar g e passar para o local L_k^i, ou algum outro processo deve falsificar a guarda c (supondo, obviamente, que a guarda envolva condições sobre objetos compartilhados).

Quando a ação g é executada (instantaneamente), o controle passa para o local

[13] Do inglês *location*.

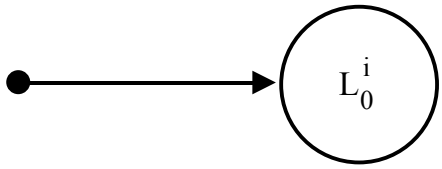

Figura 3.8: Local de entrada de um diagrama de ação temporizado

L_k^i.

Todo nó do grafo possui uma aresta dele para ele mesmo rotulada pela ação nula (*idle*), com limites de tempo 0 e ∞.

Além disso, o diagrama de cada processo P_i possui um *local de entrada* L_0^i, representado pelo nó da figura 3.8.

O local de entrada é onde o controle se encontra no início da execução do processo P_i. Não é obrigatório que todos os processos do sistema iniciem sua execução de forma síncrona.

3.4.1.2 Comunicação e Sincronização

Diagramas de ação temporizados podem incluir ações cuja execução deve ser sincronizada. Isto é necessário, por exemplo, quando dois processos trocam mensagens síncronas através de canais (ver [13]). Neste caso, duas instruções especiais de envio e recebimento de mensagens são acrescentadas à linguagem. A figura 3.9 mostra exemplos de ocorrências destas instruções.

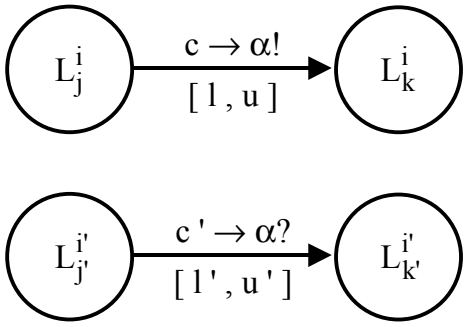

Figura 3.9: Instruções de envio e recebimento de mensagens

- No processo P_i, a aresta rotulada pela instrução $c \rightarrow \alpha!$, com limites de tempo $[l, u]$, significa que, se a guarda c for verdadeira, o processo envia uma mensagem pelo canal α. O controle só passa para o local de destino quando a mensagem for recebida por algum processo destinatário (escolhido de forma não-determinística). O recebimento da mensagem é representada pela aresta definida abaixo.

- No processo $P_{i'}$, a aresta rotulada pela instrução $c' \rightarrow \alpha?$, com limites de tempo $[l', u']$, significa que, se a guarda c' for verdadeira, o processo tenta receber uma mensagem através do canal α. O controle só passa para o local de destino quando a ação correspondente ao envio de uma mensagem por este canal (definida acima) for executada simultaneamente por algum outro processo.

Os limites de tempo são combinados do seguinte modo: suponhamos que, durante $max(l, l')$ unidades de tempo, o controle do processo P_i tenha permanecido no local L^i_j, o controle do processo $P_{i'}$ tenha permanecido no local $L^{i'}_{j'}$ e as guardas c e c' tenham sido satisfeitas continuamente. Então, P_i e $P_{i'}$ *podem* proceder, de forma síncrona, para os locais L^i_k e $L^{i'}_{k'}$, respectivamente.

Por outro lado, se P_i permanecer no local L^i_j e $P_{i'}$ permanecer no local $L^{i'}_{j'}$ e as guardas c e c' forem satisfeitas continuamente durante $min(u, u')$ unidades de tempo, então P_i e $P_{i'}$ *devem* proceder, de forma síncrona, para os locais L^i_k e $L^{i'}_{k'}$, respectivamente, ou algum outro processo deve falsificar c ou c'.

3.4.1.3 Multiprogramação

Um *sistema multiprogramado* P composto por m processos tem a forma

$$\Theta[P_1 \,|||\, P_2 \,|||\, ... \,|||\, P_m]$$

Como no caso de um sistema multiprocessado, Θ define o estado inicial, e cada P_i é um processo que pode ser representado por um diagrama de ação temporizado, definido da mesma forma. O significado informal da aresta da figura 3.7, no entanto, é modificado para refletir a existência de um único processador que executa alternadamente os m processos:

O limite mínimo garante que, quando o controle do processador único permanecer no local L^i_j durante pelo menos l unidades de tempo e a guarda c for verdadeira, o controle pode passar para o local L^i_k. O limite mínimo tem o mesmo significado que no caso multiprocessado.

O limite máximo estipula que, quando o controle tiver permanecido no local L^i_j durante u unidades de tempo com a guarda c verdadeira, o controle *deve* passar para L^i_k. Aqui há uma diferença importante em relação ao caso multiprocessado: não há a possibilidade de que outro processo falsifique a guarda c, já que há apenas um processador.

Transições nulas e locais de entrada são definidos como no caso de diagramas de sistemas multiprocessados.

3.4.1.4 Funcionalidade de uma Ação

Os símbolos de ação que rotulam as arestas de um diagrama de ação temporizado não trazem nenhuma informação sobre a funcionalidade da ação; i.e., como ela relaciona os valores-verdade dos atributos do estado de origem com os valores-verdade do estado destino.[14]

Para suprir esta informação, representamos a funcionalidade de cada ação $g \in \Gamma$ do seguinte modo:

$$(p_1, p_2, \ldots, p_n) \xrightarrow{g} (q_1, q_2, \ldots, q_n)$$

onde cada p_x e cada q_x, $1 \leq x \leq n$, é uma fórmula proposicional da forma a ou da forma $\neg a$, onde a é um atributo.

O significado pretendido é o seguinte: para cada $x, 1 \leq x \leq n$, se o estado de origem de uma transição rotulada por g satisfaz a fórmula p_x, então o estado destino da transição satisfaz a fórmula q_x.

Como exemplo, consideremos a ação *inverteSinal*, cujo efeito é o de multiplicar por -1 o valor de uma variável inteira v. Os atributos de interesse são as proposições *vzero* (interpretação: o valor de v é zero), *vpos*, (interpretação: o valor de v é positivo) e *vneg* (interpretação: o valor de v é negativo). A funcionalidade da ação *inverteSinal* pode ser expressa como

$$(vpos, vpos, vpos, vneg, vneg, vneg)$$
$$\xrightarrow{inverteSinal}$$
$$(vneg, \neg vpos, \neg vzero, vpos, \neg vneg, \neg vzero)$$

[14]Na literatura, é comum o uso de atributos tipados (variáveis) e ações com argumentos (geralmente, atribuições da forma $\bar{x} := \bar{e}$, onde $\bar{x}$ é um vetor de variáveis e $\bar{e}$ um vetor de expressões). Por simplicidade, utilizamos nesta dissertação atributos proposicionais e símbolos de ação atômicos, mas somos obrigados, então, a fornecer informações adicionais sobre os efeitos de uma ação, conforme explicado nesta seção.

3.4.1.5 Dependências entre Atributos

O conjunto A de atributos é escolhido de acordo com as propriedades do sistema nas quais estamos interessados. No entanto, A pode conter atributos cujos valores-verdade possuam alguma relação de dependência entre si, como as proposições $vzero$, $vpos$ e $vneg$ do exemplo acima. Neste caso, torna-se necessário especificar estas dependências através de um conjunto de fórmulas proposicionais:

$$vzero \leftrightarrow (\neg vpos \wedge \neg vneg)$$

$$\neg vzero \rightarrow (vneg \leftrightarrow \neg vpos)$$

A especificação destas dependências, aliás, permite que a funcionalidade da ação $inverteSinal$ seja expressa de forma mais compacta, pois os valores-verdade dos atributos não especificados explicitamente podem ser determinados através das fórmulas de dependência:

$$(vpos, vneg) \overset{inverteSinal}{\longrightarrow} (vneg, vpos)$$

3.4.2 Tradução para o Modelo Abstrato

O uso de diagramas de ação temporizados como modelos concretos de sistemas reativos exige um mapeamento para sistemas de transição temporizados, pois são estes objetos que constituem a semântica da linguagem lógica apresentada no próximo capítulo.

São apresentados, a seguir, mapeamentos de diagramas de ação temporizados para sistemas de transição temporizados. Estes mapeamentos são diferentes para os casos de multiprocessamento e de multiprogramação.

3.4.2.1 Multiprocessamento

Um diagrama de ação temporizado correspondente a um sistema reativo multiprocessado da forma

$$\Theta[P_1 \mid\mid P_2 \mid\mid ... \mid\mid P_m]$$

é mapeado para o seguinte sistema de transição temporizado S:

1. O conjunto de símbolos de atributo A inclui os símbolos de atributo sobre os quais o diagrama de ação temporizado foi construído, mais as proposições especiais da forma atL_j^i para cada processo i, $1 \leq i \leq m$, e cada local L_j^i do processo i. O significado pretendido de atL_j^i é o de que o controle do i-ésimo processo se encontra no local L_j.

2. O conjunto de símbolos de ação Γ é o mesmo sobre o qual o diagrama de ação temporizado foi construído.

3. O conjunto de estados W é o conjunto de todas as valorações possíveis dos símbolos de atributo de A (incluindo as proposições especiais da forma atL_j^i).

4. Cada aresta do diagrama com o nó L_j^i como origem e o nó L_k^i como destino, rotulada pela ação com guarda $c \rightarrow g$ e limites de tempo $[l, u]$, é mapeada em um conjunto de transições rotuladas pela ação g. Mais precisamente, dados w e $w' \in W$, $w \xrightarrow{g} w'$ se e somente se

   (a) atL_j^i é verdadeiro em w;

   (b) atL_k^i é verdadeiro em w';

   (c) c é verdadeiro em w;

   (d) Para todo p_x e todo q_x presentes na especificação da funcionalidade de g (ver 3.4.1.4), se p_x é verdadeiro em w, então q_x é verdadeiro em w';

(e) Para todo atributo $a \in A$ cujo valor-verdade não puder ser determinado pelas relações de dependência (ver 3.4.1.5) a partir da verdade de todos os q_x tais que p_x é verdadeiro em w, a é verdadeiro em w se e somente se a for verdadeiro em w'. Mais formalmente, para todo $a \in A \setminus \{a_x \mid w \text{ satisfaz } p_x \wedge (q_x \rightarrow a_x \vee q_x \rightarrow \neg a_x)\}$, w satisfaz a se e somente se w' satisfaz a.

Além das transições geradas pelas arestas do diagrama, existem transições rotuladas pela ação nula ($idle$) $w \xrightarrow{idle} w$ para todo $w \in W$.

Transições de entrada e_i são criadas para cada processo P_i. Dois estados w, w' estão ligados por uma transição de entrada e_i ($w \xrightarrow{e_i} w'$) se e somente se

- atL_j^i é falso em w para todo j, e atL_0^i é verdadeiro em w'.
- Todo atributo verdadeiro em w é verdadeiro em w', e todo atributo falso em w é falso em w', com exceção de atL_0^i.

5. As funções l e u mapeiam cada transição nos valores l e u que rotulavam a aresta do diagrama que deu origem à transição. Em especial, $l(idle) = 0$ e $u(idle) = \infty$, e $l(e_i) = 0$ e $u(e_i) = \infty$.

6. A função de interpretação dos atributos I mapeia cada atributo $a \in A$ no conjunto de estados que satisfazem a.

7. O estado inicial w_0 é aquele no qual todas as sentenças de Θ são verdadeiras, e todas as proposições especiais da forma atL_j^i são falsas (i.e., nenhum dos processos iniciou sua execução ainda, e o local do controle é indefinido). Para que o estado inicial seja único, Θ deve determinar o valor verdade inicial de todos os atributos de A.

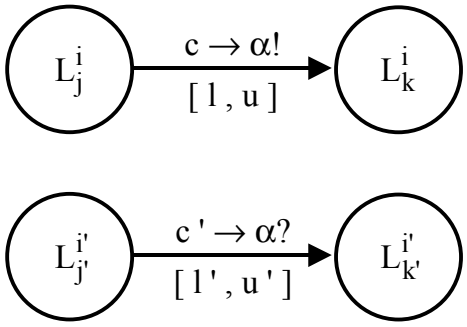

Figura 3.10: Instruções de envio e recebimento de mensagens

3.4.2.2 Comunicação e Sincronização

Se os diagramas de ação temporizados de dois processos P_i e $P_{i'}$ contiverem arestas rotuladas por ações de envio (em P_i) e recebimento (em $P_{i'}$) de mensagens síncronas do tipo $\alpha!$ e $\alpha?$ (ver 3.4.1.2), estas são mapeadas em transições rotuladas pelo par $(\alpha!, \alpha?)$ – i.e., uma ação conjunta representando a ocorrência simultânea do envio e do recebimento da mensagem. Mais precisamente:

Suponha que uma aresta do diagrama de P_i tenha o nó L^i_j como origem e o nó L^i_k como destino, sendo rotulada pela ação com guarda $c \rightarrow \alpha!$ e pelos limites de tempo $[l, u]$. Suponha também que uma aresta do diagrama de $P_{i'}$ tenha o nó $L^{i'}_{j'}$ como origem e o nó $L^{i'}_{k'}$ como destino, sendo rotulada pela ação com guarda $c' \rightarrow \alpha?$ e pelos limites de tempo $[l', u']$ (ver figura 3.10).

Dados dois estados w e w', $w \xrightarrow{(\alpha!,\alpha?)} w'$ no sistema de transição temporizado se e somente se

1. atL^i_j e $atL^{i'}_{j'}$ são verdadeiras em w;
2. atL^i_k e $atL^{i'}_{k'}$ são verdadeiras em w';

3. c e c' são verdadeiras em w;

4. Para todo p_x e todo q_x presentes na especificação da funcionalidade de $\alpha!$ (ver 3.4.1.4), se p_x é verdadeiro em w, então q_x é verdadeiro em w';

5. Para todo p_x e todo q_x presentes na especificação da funcionalidade de $\alpha?$, se p_x é verdadeiro em w, então q_x é verdadeiro em w';

6. Atributos cujos valores-verdade não são determinados pelas funcionalidades de $\alpha!$ e $\alpha?$ e pelas dependências entre os atributos têm o mesmo valor-verdade em w e em w' (ver item 4e em 3.4.2.1).

3.4.2.3 Exemplo

Um exemplo simples do uso de diagramas de ação temporizados é apresentado a seguir. Um processo P verifica, repetidas vezes, se um determinado evento ocorreu, enquanto um dado limite de tempo (*timeout*) não for ultrapassado.

Apenas o diagrama do processo P é apresentado. A ocorrência do evento aguardado é representada pela proposição c (c se torna verdadeira pela ação de algum outro processo rodando concorrentemente com P). A instrução $c \rightarrow nop$, onde nop é uma ação especial, sem funcionalidade, que passa para o local seguinte mas não altera o restante da memória, é abreviada para $test(c)$.

O processo P é especificado pelo diagrama da figura 3.11:

No início da execução, P encontra-se no local de entrada L_0. Durante 10 unidades de tempo, P pode passar para o local L_1, verificando se a proposição c é verdadeira (i.e. se o evento aguardado ocorreu). Se nenhum outro processo rodando em paralelo com P tornar c verdadeira durante estas 10 unidades de tempo, o controle passa imediatamente para o local L_2.

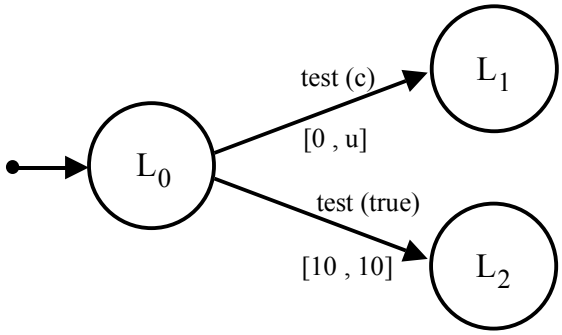

Figura 3.11: Diagrama de ação temporizado do exemplo

O valor de u determina a frequência com que o valor da proposição c é testado por P. Se u for maior do que 10, pode acontecer que P nem teste o valor de c antes de passar para L_2. Se u estiver entre 0 e 10, P precisa verificar o valor de c pelo menos uma vez a cada u unidades de tempo.

3.4.2.4 Multiprogramação

No mapeamento de um sistema multiprogramado da forma

$$\Theta[P_1 \,|||\, P_2 \,|||\, ... \,|||\, P_m]$$

para um sistema de transição temporizado, dois fatores devem ser considerados:

- Em qualquer instante, o controle do sistema se encontra em um local de um único processo (o processo ativo).
- A estratégia de escalonamento dos processos deve ser levada em consideração no estudo do comportamento do sistema.

Como no caso de sistemas multiprocessados, o conjunto de atributos A do sistema de transição temporizado é aumentado com proposições especiais do tipo atL^i_j;

no entanto, um conjunto adicional de atributos da forma $ativo_i$ (significando que o processo i é o processo ativo) torna-se necessário. Obviamente, em qualquer estado w do sistema de transição temporizado, no máximo um dos atributos $ativo_i$ pode ser verdadeiro.[15] Os outros processos j, com $j \neq i$, são ditos *suspensos*.

O estado inicial w_0 do sistema de transição temporizado é aquele em que Θ é satisfeito, todas as proposições da forma atL_j^i são falsas, e todas as proposições da forma $ativo_i$ são falsas (i.e., no início da execução do sistema, nenhum processo está ativo).

As arestas do diagrama geram transições exatamente do mesmo modo que no caso multiprocessado. Da mesma forma, transições rotuladas pela ação nula e transições de entrada também são criadas.

Além destas, *transições de escalonamento* precisam ser acrescentadas. Em uma transição de escalonamento $w \xrightarrow{esc} w'$, os valores verdade de no máximo dois atributos $ativo_i$ podem variar; mais precisamente, para algum i o valor de $ativo_i$ passa de verdadeiro para falso (significando que o processo i foi suspenso); opcionalmente, para algum j o valor verdade de $ativo_j$ passa de falso para verdadeiro (significando que o processo j foi reativado).

O conjunto exato de transições de escalonamento depende da estratégia usada pelo escalonador, uma informação que não está contida no diagrama de ação temporizado, devendo ser especificada à parte. Um exemplo seria uma estratégia "gulosa", onde um processo ativo assim permanece até que sua execução termine (i.e., todas suas ações estejam desabilitadas); neste momento, algum outro processo com uma ação habilitada, caso exista, é ativado. Para esta estratégia, as transições de escalonamento $w \xrightarrow{esc_g} w'$ geradas são tais que

1. Em w, $ativo_i$ é verdadeiro para algum processo i;

[15] Esta restrição pode ser acrescentada às dependências entre os atributos.

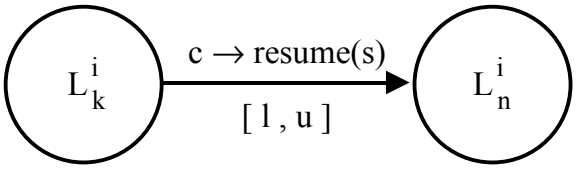

Figura 3.12: Aresta com instrução *resume*

2. Em w', todos os atributos têm o mesmo valor que em w, com exceção de $ativo_i$ (i representando o processo que está sendo suspenso) e possivelmente $ativo_j$ (j representando o processo que está sendo ativado);

3. Nenhuma outra ação (salvo a ação nula) está habilitada em w;

4. Existe alguma ação (além da ação nula) habilitada em w'.

Para permitir maior flexibilidade nas estratégias de escalonamento, instruções especiais de escalonamento podem ser acrescentadas à linguagem dos diagramas de ação temporizados. A instrução $resume(s)$, onde $s \subseteq \{1, 2, ..., m\}$, suspende o processo atual e ativa, de modo não-determinístico, um processo P_j com $j \in s$. Quando s é um conjunto unitário $\{j\}$, abreviamos $resume(\{j\})$ como $resume(j)$. A abreviatura *suspend* representa $resume\{j \mid 1 \leq j \leq m, j \neq i\}$.

No mapeamento para sistemas de transição temporizados, cada aresta da forma da figura 3.12 é associada a transições $w \stackrel{resume(s)}{\longrightarrow} w'$, onde:

1. $ativo_i$ é verdadeiro em w;

2. $ativo_j$ é verdadeiro em w' para algum $j \in s$.

3. atL_k^i é verdadeiro em w;

4. atL_n^i é verdadeiro em w';

5. c é verdadeira em w;

6. Todos os outros atributos têm o mesmo valor em w e em w'.

Se a troca de processos pode ser considerada instantânea, os limites de tempo das transições de escalonamento podem ser definidos como $l = u = 0$. Caso contrário, os valores devem ser ajustados para refletir o tempo necessário para o escalonador suspender um processo e ativar outro.

3.4.2.5 Exemplo

O exemplo a seguir mostra como representar uma situação de espera não-ocupada: um processo P_i deve liberar o processador para outro processo P_j durante 10 unidades de tempo, e então ser reativado. Para isto, um processo P_T (um *timer*) é rodado em paralelo *em um processador diferente* (i.e., o sistema é denotado por $\Theta[(P_i \;|||\; P_j) \;||\; P_T]$. O processo P_T é representado pelo diagrama da 3.13. A ação $test(t) \rightarrow setat$ torna o atributo t falso somente se t for verdadeiro.

Suponhamos que o controle do processo P_i esteja no local L_j^i e o controle do processo P_t esteja no local t_0. O processo P_i ativa o *timer* e suspende a si mesmo, cedendo o processador para o processo P_j, através da sequência da figura 3.14. A ação *setat* torna o atributo t verdadeiro.

10 unidades de tempo depois, o *timer* reativa o processo P_i.

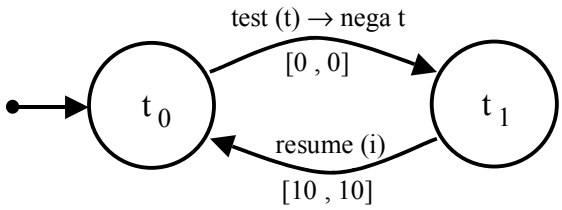

Figura 3.13: Diagrama de ação temporizado do processo P_T do exemplo

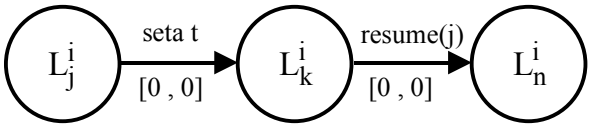

Figura 3.14: Diagrama de ação temporizado da ativação do *timer* do exemplo

3.4.2.6 Múltiplos Processadores e Outras Extensões

A discussão acima sobre multiprogramação supôs a existência de um único processador que deve ser compartilhado pelos processos de um sistema reativo. O formalismo pode ser estendido, no entanto, (como mostra [11]) para uma situação em que existe um *pool* de processadores aos quais os processos podem ser alocados estática ou dinamicamente.

A mesma referência discute, ainda, o uso de prioridades e interrupções para a especificação de estratégias de escalonamento mais poderosas e semelhantes às utilizadas em sistemas operacionais modernos ([19]).

Embora discussões deste tipo sejam importantes para ilustrar a adequação de diagramas de ação temporizados e sistemas de transição temporizados como modelos de sistemas reativos, elas se encontram além do escopo deste capítulo, cujo propósito foi apenas o de apresentar estes formalismos. No próximo capítulo, uma linguagem lógica para raciocinar sobre sistemas de transição temporizados será proposta.

Capítulo 4

RETOOL

4.1 O Operador δ

Em [18], Segerberg introduziu a noção de um operador δ, o qual, quando aplicado a uma dada proposição q, denota o conjunto de ações que tornam q verdadeiro. Neste capítulo, definimos uma lógica de ações dotada de um operador semelhante, que tem, como modelos, sistemas de ação temporizados, apresentados no capítulo anterior.

O operador δ de Segerberg serviu como base para uma extensão da lógica dinâmica ([9]) contendo elementos de lógicas de ações, onde δq denota ações que levam a estados onde q é verdadeiro. Em [4, 7], o operador foi estendido de duas maneiras diferentes: a primeira extensão foi introduzida para raciocinar sobre pré-condições e pós-condições, com a forma diádica $p\delta q$. A segunda extensão, relativa a aplicações de tempo real, acrescentou limites de tempo máximos e mínimos às ações: nela, o termo de ação $p_l\delta^u q$ denota as ações que tornam q verdadeiro em menos de u unidades de tempo, desde que tenham permanecido habilitadas em estados que satisfazem p durante no mínimo l unidades de tempo.

Em [4, 7], esta última extensão foi chamada de RETOOL (*Real-Time Object-Oriented Logic*, uma referência ao objetivo maior de produzir uma formalização de sistemas de tempo real orientados a objetos). Não foi atingido um consenso, no entanto, quanto à denotação exata de um termo de ação $p\delta q$. Nos dois trabalhos citados, duas semânticas diferentes foram propostas para RETOOL; esta dissertação sugere uma terceira, oferecendo uma axiomatização adequada, acompanhada de provas de correção e completude fraca:

- [4] definiu $p\delta q$ como denotando o conjunto de todas as ações que tornam q verdadeiro a partir de estados onde p vale, com a exigência adicional de que estas ações estejam habilitadas em *todos* os estados onde p vale; chamamos esta definição de *Semântica da Condição de Habilitação* (SCH) do operador δ.

- Em [7], $p\delta q$ foi definido como denotando todas as ações g tais que, se g parte de um estado onde p vale, então g conduz a um estado onde q vale; chamamos esta definição de *Semântica da Implicação Material* (SIM) do operador δ.

- Nesta dissertação, apresentamos uma definição de $p\delta q$ onde este termo significa o conjunto de ações que partem apenas de estados onde p vale e tornam q verdadeiro; esta é a *Semântica da Condição Necessária* (SCN) do operador δ.

As abordagens semânticas mais promissoras para RETOOL como uma linguagem lógica para descrever sistemas de transição computacionais parecem ser justamente a SCN e a SCH. A próxima seção define a linguagem de RETOOL. Em seções posteriores, a SCN é apresentada, acompanhada de uma axiomatização adequada, com provas de correção e completude fraca.

4.2 A Linguagem de RETOOL

As entidades sintáticas primitivas de RETOOL são os *símbolos de atributo* (que, nesta versão proposicional da lógica, são simples letras proposicionais) e *símbolos de ação*. Denotamos o conjunto de símbolos de atributos como A, e o conjunto de símbolos de ação como Γ.

A lógica pressupõe um conjunto ordenado (TIME, $\leq$), com mínimo 0. Uma constante ∞, que não é elemento de TIME, está disponível, tal que $t \leq \infty$ para todo $t \in$ TIME. As outras categorias sintáticas são:

Definição 4.2.1 (Proposições de estado (PE)**)**

$$p ::= a \mid \neg p \mid p \rightarrow p'$$

onde $a \in A$.

Definição 4.2.2 (Termos de ação (TA)**)**

$$t ::= g \mid p_l \delta^u q$$

onde $g \in \Gamma, p, q \in PE, l \in$ *TIME,* $u \in$ *TIME* $\cup \{\infty\}, l \leq u$.

Definição 4.2.3 (Fórmulas)

$$\phi ::= a \mid t_1 \supset t_2 \mid \neg\phi \mid \phi \rightarrow \phi' \mid [t]\phi \mid [\,]p$$

onde $a \in A$, $t, t_1, t_2 \in TA$, $p \in PE$.

4.3 Semântica da Condição Necessária (SCN)

A semântica de RETOOL é definida sobre sistemas de transição temporizados (ver 3.3.1) com conjunto de símbolos de atributo A e conjunto de símbolos de ação Γ. Dado um sistema de transição temporizado $(W, \rightarrow, l, u, I, w_0)$, valem as seguintes definições:

Definição 4.3.1 *Uma proposição de estado p denota o seguinte conjunto de estados:*

- $[\![a]\!] = I(a)$ *para* $a \in A$*;*
- $[\![\neg p]\!] = W \setminus [\![p]\!]$*;*
- $[\![p \rightarrow p']\!] = (W \setminus [\![p]\!]) \cup [\![p']\!]$.

Em especial, $[\![\top]\!] = W$ *e* $[\![\bot]\!] = \emptyset$.

Definição 4.3.2 *Um termo de ação t denota o seguinte conjunto de transições:*

- $[\![g]\!] = \{(w, w') \mid w \xrightarrow{g} w'\}$*;*
- $[\![p_l \delta^u q]\!] =$

$$\{(w, w') \mid \exists g \in \Gamma \quad [\\ (w \xrightarrow{g} w') \wedge \\ enable(g) \subseteq [\![p]\!] \wedge \\ \forall v, v'((v \xrightarrow{g} v') \Rightarrow (v' \in [\![q]\!])) \wedge \\ (l \leq l(g) \leq u(g) \leq u) \\ ] \\ \}$$

onde $enable(g)$ *significa o conjunto de estados do sistema de transição nos quais a ação g está habilitada.*

Definição 4.3.3 *A satisfação de uma fórmula em um estado w de um sistema de transição temporizado S é definida como:*

$S, w \models p$ *sss* $w \in [\![p]\!]$;

$S, w \models (t_1 \supset t_2)$ *sss* $[\![t_1]\!] \subseteq [\![t_2]\!]$;

$S, w \models \neg\phi$ *sss* $S, w \not\models \phi$;

$S, w \models \phi \rightarrow \phi'$ *sss* $S, w \models \phi$ *implica* $S, w \models \phi'$;

$S, w \models [t]\phi$ *sss* $S, w' \models \phi$ *para todo* w' *tal que* $(w, w') \in [\![t]\!]$;

$S, w \models [\,]p$ *sss* $S, w_0 \models p$.

4.3.1 Comentários

4.3.1.1 Termos de Ação como Instruções

A denotação de um termo de ação da forma $p_l\delta^u q$ é um conjunto de transições tal que, se uma transição rotulada por um símbolo de ação g estiver incluído na denotação, então *todas* as transições rotuladas por g também estão incluídas na denotação. Isto equivale a dizer que, se os símbolos de ação representam instruções atômicas, então é a *instrução* representada por g (em vez de uma ou outra *execução* da instrução) que está associada às condições p e q e aos limites de tempo l e u.

4.3.1.2 Subsunção

"$\supset$" é o operador de subsunção. Dizer que um termo de ação t_1 está subsumido em um termo t_2 significa que toda ação denotada por t_1 também é denotada por t_2 (mas não necessariamente vice-versa). A subsunção pode ser vista como um refinamento de ações: as ações denotadas por t_1 refinam as denotadas por t_2. Note que a verdade de uma fórmula de subsunção independe do estado w em questão.

4.3.1.3 Termos de Ação Especiais

Alguns termos de ação com denotações interessantes podem ser construídos com as constantes $\bot$ e $\top$. Para proposições de estado p e q diferentes de $\top$ e $\bot$ (omitimos l e u destes termos, pois a ênfase, aqui, é nas proposições, não nos limites de tempo):

$[\![p\delta\top]\!]$ = todas as transições rotuladas por ações com condição necessária p. Não é especificada qualquer pós-condição.

$[\![\top\delta q]\!]$ = todas as transições rotuladas por ações com pós-condição q. Não é especificada qualquer condição necessária.

$[\![p\delta\bot]\!] = [\![\bot\delta\bot]\!] = [\![\bot\delta\top]\!] = [\![\top\delta\bot]\!] = [\![\bot\delta q]\!] = \emptyset$;

4.3.1.4 Ações produtivas

Ao dizer que uma dada ação g torna q verdadeiro, pode parecer razoável exigir que q seja falso no estado de origem; i.e., a ação altera, de fato, o valor-verdade de q, produzindo uma situação que não existia antes. Uma ação deste tipo é chamada de *ação produtiva*. A semântica do operador δ dada acima não impõe tal condição. No entanto, um operador produtivo δ' pode ser definido em termos do operador não-produtivo δ do seguinte modo:

$$p_l\delta'^u q = (p \wedge \neg q)_l\delta^u q$$

4.3.1.5 Habilitação de uma Ação

Na seção 3.2.1, definimos, para cada ação $g \in \Gamma$, o conjunto $enable(g)$ como

$$enable(g) = \{w \in W \mid \exists w' \text{ tal que } (w, w') \in \xrightarrow{g}\}$$

Este é o conjunto de todos os estados onde g está habilitada. Dizer que um estado w pertence a $enable(g)$ equivale a dizer que a fórmula $\neg(g \supset \top\delta\bot)$ é verdadeira em w, ou ainda, que a fórmula $[g]\bot$ é falsa em w. De agora em diante, usaremos a abreviatura $enabled(g)$ para $\neg[g]\bot$. Um estado w pertencerá a $enable(g)$ se e somente se a fórmula $enabled(g)$ for verdadeira em w.

Para um termo de ação $t = p_l\delta^u q$, $enable(t)$ é o conjunto de todos os estados onde pelo menos uma ação $g \in [\![p_l\delta^u q]\!]$ está habilitada. A abreviatura $enabled(p_l\delta^u q)$ significa, então, $\neg[p_l\delta^u q]\bot$.

Assim, temos, para $t \in TA$,

$$enabled(t) = \neg[t]\bot$$

4.4 Uma Axiomatização para a SCN

Esta seção apresenta uma axiomatização de RETOOL em sua versão SCN. As seguintes definições caracterizam o cálculo dedutivo:

Definição 4.4.1 (Sequência de Derivação, Teorema) *Seja ϕ uma fórmula RETOOL. Escrevemos $\vdash \phi$ se e somente se houver uma sequência finita de fórmulas $\phi_1, \phi_2, ..., \phi_n$, com $\phi_n = \phi$, tal que cada $\phi_i (1 \leq i \leq n)$ ou é uma instância dos esquemas de axioma de RETOOL ou é a conclusão de uma regra de inferência onde cada premissa aparece como ϕ_j na sequência para algum $j < i$. Denominamos tal sequência uma* sequência de derivação de ϕ. *Dizemos que ϕ é um* teorema *de RETOOL.*

Definição 4.4.2 (Relação de Derivabilidade) *Para todo conjunto Λ de fórmulas RETOOL e toda fórmula RETOOL ϕ, escrevemos $\Lambda \vdash \phi$ se e somente se houver uma sequência finita de fórmulas $\phi_1, \phi_2, ..., \phi_n$, com $\phi_n = \phi$, tal que cada $\phi_i (1 \leq$*

$i \leq n$) ou é um elemento de Λ, ou é uma instância de um esquema de axioma de RETOOL, ou é a conclusão de uma regra de inferência onde cada premissa aparece como ϕ_j na sequência para algum $j < i$. Denominamos tal sequência uma sequência de derivação de ϕ a partir de Λ. *Dizemos que* Λ deriva ϕ. *Denominamos* $\vdash$ *a* relação de derivabilidade. *Para* Λ *unitário* $= \{\psi\}$, *escrevemos* $\psi \vdash \phi$.

4.4.1 Axiomas e Regras

Os seguintes esquemas de axioma e regras de inferência compreendem o cálculo dedutivo de RETOOL.[1] A prova de correção, na próxima seção, inclui comentários sobre os esquemas e regras mais importantes.

[1] Assumimos a existência de um cálculo dedutivo adequado para provar propriedades que digam respeito apenas ao operador $\leq$ e aos elementos de TIME $\cup\{\infty\}$.

(CP) Todos os axiomas e regras do Cálculo Proposicional

(K) $[t](\phi \rightarrow \psi) \rightarrow ([t]\phi \rightarrow [t]\psi)$

$[\,](p \rightarrow q) \rightarrow ([\,]p \rightarrow [\,]q)$

(I) $[\,]p \rightarrow [t][\,]p$

$[t][\,]p \rightarrow (\mathit{enabled}(t) \rightarrow [\,]p)$

$[\,]\neg p \leftrightarrow \neg[\,]p$

$\neg[\,]\bot$

(N) $\dfrac{\Lambda \vdash \phi}{\Lambda \vdash [t]\phi} \quad \dfrac{\Lambda \vdash p}{\Lambda \vdash [\,]p}$

(δ) $\dfrac{\Lambda \vdash \mathit{enabled}(t) \rightarrow p \quad \Lambda \vdash [t]q \quad \Lambda \vdash l \leq l(t) \leq u(t) \leq u}{\Lambda \vdash t \supset p_l \delta^u q}$

(S1) $t \supset t$

(S2) $(t_1 \supset t_2) \rightarrow ((t_2 \supset t_3) \rightarrow (t_1 \supset t_3))$

(S3) $(t_1 \supset t_2) \rightarrow ([t_2]\phi \rightarrow [t_1]\phi)$

(CN) $(t \supset p_l \delta^u q) \rightarrow (\mathit{enabled}(t) \rightarrow p)$

(Pos) $(t \supset p_l \delta^u q) \rightarrow ([t]q)$

(Lim) $(t \supset p_l \delta^u q) \rightarrow (\mathit{enabled}(t) \rightarrow l \leq l(t) \leq u(t) \leq u)$

(Global-$\supset$) $(t_1 \supset t_2) \leftrightarrow [t](t_1 \supset t_2)$

(Lim-δ) $l(p_l \delta^u q) = l \quad u(p_l \delta^u q) = u$

4.5 Correção

Nesta seção, a correção da axiomatização acima é provada em relação à Semântica da Condição Necessária. Para tanto, definimos as noções de satisfatibilidade, verdade, validade e consequência para determinar quando um sistema de transição temporizado é um modelo de um conjunto de fórmulas RETOOL:

Definição 4.5.1 (Satisfatibilidade) *Uma fórmula RETOOL ϕ é satisfatível se e somente se existe algum estado w de algum sistema de transição temporizado S tal que ϕ vale em w (escrito $S, w \models \phi$).*

Definição 4.5.2 (Verdade em um Sistema de Transição Temporizado) *Uma fórmula RETOOL ϕ é verdadeira em um sistema de transição temporizado S (escrito $S \models \phi$) se e somente se ela é satisfeita em todos os estados de S.*

Definição 4.5.3 (Validade) *Uma fórmula RETOOL ϕ é válida (escrito $\models \phi$) se e somente se ela é verdadeira em todos os sistemas de transição temporizados.*

Definição 4.5.4 (Consequência) *Uma fórmula RETOOL ϕ é consequência de um conjunto Λ de fórmulas RETOOL (escrito $\Lambda \models \phi$) se e somente se, para todo sistema de transição temporizado S, se todas as fórmulas de Λ forem verdadeiras em S, então ϕ também é verdadeira em S.*

Esta noção de consequência é chamada de "consequência global" em [9]. É uma definição mais fraca do que a definição alternativa de consequência (que estipula que $\Lambda \models \phi$ se e somente se, para todo estado w de todo sistema de transição temporizado S, se todas as fórmulas de Λ são verdadeiras em w, então ϕ também é verdadeira em w). Achamos razoável adotar esta definição de consequência porque, na descrição de um sistema reativo S, esperamos que as fórmulas de uma especificação Λ sejam verdadeiras em todos os estados do sistema de transição temporizado que representa S. Queremos, com a lógica, deduzir propriedades decorrentes de Λ (i.e. verdadeiras em todos os estados do sistema em questão), ainda que tais propriedades não sejam válidas (i.e., verdadeiras em todos os estados de todos os sistemas).

A correção da axiomatização consiste na prova da seguinte proposição:

Teorema 4.5.1 (Correção) *Para toda fórmula ϕ da linguagem de RETOOL*

$$\vdash \phi \quad \Rightarrow \quad \models \phi$$

A prova é feita por indução sobre o comprimento da sequência de derivação de ϕ. Como casos-base, temos os teoremas derivados através de sequências de derivação de comprimento 1 (i.e., os axiomas). O passo da indução consiste na prova de que, para toda regra de inferência, se as premissas são fórmulas verdadeiras, então a conclusão também é uma fórmula verdadeira.

4.5.1 Prova e Comentários

Na prova de correção a seguir, omitimos o esquema (CP) – todos os axiomas e regras do Cálculo Proposicional. Os mais importantes dos outros axiomas e regras

são acompanhados de comentários sobre as características de sistemas de transição temporizados que eles buscam capturar.

(K) $[t](\phi \to \psi) \to ([t]\phi \to [t]\psi)$
$[\,](p \to q) \to ([\,]p \to [\,]q)$

Prova:

Seja S um sistema de transição temporizado. Suponha que $S, w \models [t](\phi \to \psi)$ em um estado w. Então, para todo $g \in [\![t]\!]$, $w \xrightarrow{g} w'$ implica $S, w' \models \phi \to \psi$. Suponha, para fins de contradição, que $S, w \not\models [t]\phi \to [t]\psi$, i.e., que $S, w \models [t]\phi$ mas $S, w \not\models [t]\psi$. Então existe um estado w'' tal que $w \xrightarrow{g} w''$, $g \in [\![t]\!]$ e $S, w'' \not\models \psi$. Além disso, para todo estado w', $w \xrightarrow{g} w'$ implica $S, w' \models \phi$. Então $S, w'' \models \phi$ e $S, w'' \not\models \psi$, e, portanto, $S, w'' \not\models \phi \to \psi$, contradizendo a suposição inicial de que $\phi \to \psi$ vale em todos os t-sucessores de w.

Um raciocínio semelhante se aplica ao caso da modalidade $[\,]$, onde $S, w \models [\,]p$ significa que p vale no estado inicial w_0.

□

(I) $[\,]p \to [t][\,]p$
$[t][\,]p \to (\mathit{enabled}(t) \to [\,]p)$
$[\,]\neg p \leftrightarrow \neg[\,]p$
$\neg[\,]\bot$

Prova:

- Para $[\,]p \to [t][\,]p$: em um sistema de transição temporizado S, para todo

$w \in W$, temos que

$$\begin{aligned}
& S, w \models [\,]p && \Leftrightarrow \\
& S, w_0 \models p && \Rightarrow \\
& \forall g \in [\![t]\!] : (w \xrightarrow{g} w' \Rightarrow S, w_0 \models p) && \Leftrightarrow \\
& \forall g \in [\![t]\!] : (w \xrightarrow{g} w' \Rightarrow S, w' \models [\,]p) && \Leftrightarrow \\
& S, w \models [t][\,]p
\end{aligned}$$

- Para $[t][\,]p \rightarrow (\mathit{enabled}(t) \rightarrow [\,]p)$: em um sistema de transição temporizado S, para todo $w \in W$, temos que

$$\begin{aligned}
& S, w \models [t][\,]p && \Leftrightarrow \\
& \forall g \in [\![t]\!] : (w \xrightarrow{g} w' \Rightarrow S, w' \models [\,]p) && \Rightarrow \\
& (S, w \models \mathit{enabled}(t)) \Rightarrow (\exists w' : S, w' \models [\,]p) && \Leftrightarrow \\
& S, w_0 \models p && \Leftrightarrow \\
& S, w \models [\,]p
\end{aligned}$$

- Para $[\,]\neg p \leftrightarrow \neg[\,]p$: em um sistema de transição temporizado S, para todo $w \in W$, temos que

$$\begin{aligned}
& S, w \models \neg[\,]p && \Leftrightarrow \\
& S, w_0 \not\models p && \Leftrightarrow \\
& S, w_0 \models \neg p && \Leftrightarrow \\
& S, w \models [\,]\neg p
\end{aligned}$$

- Para $\neg[\,]\bot$: em um sistema de transição temporizado S, nunca é o caso que $S, w_0 \models \bot$.

□

(S1) $t \supset t$

Prova:

Para todo sistema de transição temporizado S e estado w,

$$(S, w \models t \supset t) \Leftrightarrow ([\![t]\!] \subseteq [\![t]\!]).$$

□

(S2) $(t_1 \supset t_2) \rightarrow ((t_2 \supset t_3) \rightarrow (t_1 \supset t_3))$

Prova:

Suponha que $(t_1 \supset t_2)$ e $(t_2 \supset t_3)$ valham em um estado w de um sistema de transição temporizado S. Pela transitividade de $\subseteq$, temos que $[\![t_1]\!] \subseteq [\![t_3]\!]$. Portanto, $S, w \models t_1 \supset t_3$.

□

(S3) $(t_1 \supset t_2) \rightarrow ([t_2]\phi \rightarrow [t_1]\phi)$

Prova:

Em um sistema de transição temporizado S, suponha $S, w \models t_1 \supset t_2$ em um estado w. Então $[\![t_1]\!] \subseteq [\![t_2]\!]$. Suponha também $S, w \models [t_2]\phi$. Então, em todo estado w' tal que $(w, w') \in [\![t_2]\!]$, ϕ vale. Suponha, para fins de contradição, que $S, w \not\models [t_1]\phi$. Então, haveria um estado w'' tal que $(w, w'') \in [\![t_1]\!]$ e $S, w'' \not\models \phi$. Mas, pela suposição inicial, temos que $(w, w'') \in [\![t_1]\!]$ implica $(w, w'') \in [\![t_2]\!]$, e portanto ϕ precisa valer em w''. Assim, $S, w \models [t_1]\phi$.

□

(CN) $(t \supset p_l \delta^u q) \rightarrow (enabled(t) \rightarrow p)$

Comentário:

Este axioma trata de uma condição necessária para que um termo de ação t seja subsumido por um termo de ação da forma $p_l \delta^u q$: sempre que t estiver habilitado em um estado w, p deverá ser verdadeiro em w.

Prova:

Suponha que $S, w \models t \supset p_l \delta^u q$ em um estado w de um sistema de transição temporizado S. Então, $enable(t) \subseteq [\![p]\!]$, e, para toda ação $g \in [\![t]\!]$, $S, w \models enabled(g)$ implica $S, w \models p$.

□

(Pos) $(t \supset p_l \delta^u q) \rightarrow ([t]q)$

Comentário:

Este axioma simplesmente afirma que qualquer ação que tenha q como pós-condição de fato torna q verdadeiro ao ser executada.

Prova:

Suponha que $S, w \models t \supset p_l \delta^u q$ em um estado w de um sistema de transição temporizado S. No caso de nenhuma ação $g \in [\![t]\!]$ estar habilitada em w, $S, w \models [t]q$ vacuamente. Se há uma ação $g \in [\![t]\!]$ habilitada em w, pela definição do operador δ, g sempre conduz a estados onde q vale.

□

(Lim) $(t \supset p_l\delta^u q) \rightarrow (enabled(t) \rightarrow l \leq l(t) \leq u(t) \leq u)$

Comentário:

Este axioma simplesmente afirma que todas as ações subsumidas por $p_l\delta^u q$ respeitam os limites de tempo l e u.

Prova:

Suponha que $S, w \models t \supset p_l\delta^u q$ em um estado w de um sistema de transição temporizado S. Se há uma ação $g \in [\![t]\!]$ habilitada em w, pela definição do operador δ, $l \leq l(g) \leq u(g) \leq u$.

□

(Global-$\supset$) $(t_1 \supset t_2) \leftrightarrow [t](t_1 \supset t_2)$

Comentário:

Este axioma diz respeito ao caráter global do operador de subsunção: se uma subsunção é verdadeira em um estado de um sistema de transição temporizado, ela é verdadeira em todos os estados do sistema.

Prova:

Pela própria definição da relação de satisfação de fórmulas RETOOL (ver 4.3.3).

□

(Lim-δ) $l(p_l\delta^u q) = l \quad u(p_l\delta^u q) = u$

Comentário:

Estes axiomas definem os limites de tempo de um termo de ação construído com o operador δ.

Prova:

Pela própria definição do operador δ.

□

(N) $\frac{\Lambda\vdash\phi}{\Lambda\vdash[t]\phi} \quad \frac{\Lambda\vdash p}{\Lambda\vdash[\,]p}$

Prova:

Suponha que p vale em todos os estados de um sistema de transição temporizado no qual Λ é verdadeiro. Em especial, p vale em w_0; portanto, $S, w \models [\,]p$ para todo estado w de S. De forma análoga, para todo w, ϕ vale em todos os estados w' tais que $w \xrightarrow{g} w'$ para algum $g \in [\![t]\!]$; assim, $S, w \models [t]\phi$.

□

(δ) $\frac{\Lambda\vdash enabled(t)\rightarrow p \quad \Lambda\vdash[t]q \quad \Lambda\vdash l\leq l(t)\leq u(t)\leq u}{\Lambda\vdash t\supset p_l\delta^u q}$

Comentário:

Esta regra captura uma condição suficiente para que um termo de ação t seja

subsumido por um termo de ação da forma $p_l \delta^u q$. A regra permite que o operador δ seja introduzido para construir termos de ação a partir de proposições de estado.

Prova:

Suponha que as premissas sejam verdadeiras. Então, para toda ação $g \in [\![t]\!]$, $enable(g) \subseteq [\![p]\!]$. Além disso, em todo sistema de transição temporizado S no qual Λ seja verdadeiro, para todo par de estados (w, w') tal que $w \xrightarrow{g} w'$, temos que $w' \models q$. Temos também que $l \leq l(g) \leq u(g) \leq u$. Estes três fatos implicam em que $[\![t]\!] \subseteq [\![p_l \delta^u q]\!]$. Logo, $t \supset p_l \delta^u q$ também será uma fórmula verdadeira em S.

□

Corolário 4.5.1 *O Teorema da Correção equivale ao seguinte enunciado:*

Para toda fórmula ϕ,

$$(\exists S \exists w : S, w \models \phi) \quad \Rightarrow \quad (\not\forall \neg\phi)$$

Prova:

$$\begin{array}{lll} \vdash \phi \quad \Rightarrow \quad \models \phi & & \Leftrightarrow \\ \not\models \phi \quad \Rightarrow \quad \not\forall \phi & & \Leftrightarrow \\ (\exists S \exists w : S, w \models \neg\phi) \quad \Rightarrow \quad \not\forall \phi & & \Leftrightarrow \\ (\exists S \exists w : S, w \models \psi) \quad \Rightarrow \quad \not\forall \neg\psi & & \end{array}$$

□

Isto significa que, se uma fórmula ψ é satisfatível, então $\neg\psi$ não é um teorema.

4.6 Regras Derivadas e Teoremas Interessantes

Apresentamos, a seguir, algumas regras derivadas e teoremas de RETOOL. Estes resultados serão úteis na prova de completude da próxima seção.

1. (CN-Enfraq):

$$\frac{\Lambda \vdash enabled(p_l\delta^u q) \rightarrow r}{\Lambda \vdash p_l\delta^u q \supset r_l\delta^u q}$$

   Uma ação com condição necessária p também tem uma proposição mais fraca r como condição necessária. (Note que $\vdash enabled(p_l\delta^u q) \rightarrow p$.)

   Prova:

   1. $\Lambda \vdash enabled(p_l\delta^u q) \rightarrow r$
   2. $\Lambda \vdash p_l\delta^u q \supset p_l\delta^u q$ (S1)
   3. $\Lambda \vdash (p_l\delta^u q \supset p_l\delta^u q) \rightarrow [p_l\delta^u q]q$ (Pos)
   4. $\Lambda \vdash [p_l\delta^u q]q$ (Modus Ponens 2, 3)
   5. $\Lambda \vdash l \leq l \leq u \leq u$ (TIME)
   6. $\Lambda \vdash p_l\delta^u q \supset r_l\delta^u q$ (δ 1, 4, 5)

   □

2. (Pos-Enfraq):

$$\frac{\Lambda \vdash [p_l\delta^u q]r}{\Lambda \vdash p_l\delta^u q \supset p_l\delta^u r}$$

   Uma ação com pós-condição necessária q também tem uma proposição mais fraca r como pós-condição (bastando que $q \rightarrow r$ valha nos estados de destino da referida ação).

   Prova:

1. $\Lambda \vdash [p_l \delta^u q] r$
2. $\Lambda \vdash p_l \delta^u q \supset p_l \delta^u q$ (S1)
3. $\Lambda \vdash (p_l \delta^u q \supset p_l \delta^u q) \rightarrow (enabled(p_l \delta^u q) \rightarrow p)$ (CN)
4. $\Lambda \vdash enabled(p_l \delta^u q) \rightarrow p$ (Modus Ponens 2, 3)
5. $\Lambda \vdash l \leq l \leq u \leq u$ (TIME)
6. $\Lambda \vdash p_l \delta^u q \supset p_l \delta^u r$ (δ 1, 4, 5)

□

3. (Subs-δ):

$$\frac{\Lambda \vdash t_1 \supset p_l \delta^u q \quad \Lambda \vdash t_2 \supset r_m \delta^n s \quad \Lambda \vdash t_1 \supset t_2}{\Lambda \vdash (enabled(t_1) \rightarrow (p \wedge r)) \wedge [t_1] s}$$

Se uma ação é subsumida por outra, quando a primeira está habilitada, as condições necessárias e pós-condições de ambas se combinam conforme estipulado pela conclusão da regra.

Prova:

1. $\Lambda \vdash t_1 \supset p_l \delta^u q$
2. $\Lambda \vdash t_2 \supset r_m \delta^n s$
3. $\Lambda \vdash t_1 \supset t_2$
4. $\Lambda \vdash t_1 \supset p_l \delta^u q \rightarrow (enabled(t_1) \rightarrow p)$ (CN)
5. $\Lambda \vdash enabled(t_1) \rightarrow p$ (Modus Ponens 1, 4)
6. $\Lambda \vdash t_1 \supset t_2 \rightarrow ((t_2 \supset r_m \delta^n s) \rightarrow (t_1 \supset r_m \delta^n s))$ (S2)
7. $\Lambda \vdash t_1 \supset r_m \delta^n s$ (2X Modus Ponens 2, 3, 6)
8. $\Lambda \vdash (t_1 \supset r_m \delta^n s) \rightarrow (enabled(t_1) \rightarrow r)$ (CN)
9. $\Lambda \vdash enabled(t_1) \rightarrow r$ (Modus Ponens 7, 8)
10. $\Lambda \vdash enabled(t_1) \rightarrow (p \wedge r)$ (CP 5, 9)
11. $\Lambda \vdash (t_1 \supset r_m \delta^n s) \rightarrow [t_1] s$ (Pos)
12. $\Lambda \vdash [t_1] s$ (Modus Ponens 7, 11)
13. $\Lambda \vdash (enabled(t_1) \rightarrow (p \wedge r)) \wedge [t_1] s$ (CP 10, 12)

□

4. (Lim-Enfraq):

$$\frac{\Lambda \vdash (l' \leq l \leq u \leq u')}{\Lambda \vdash p_l \delta^u q \supset p_{l'} \delta^{u'} q}$$

Um ação com limites de tempo l e u também satisfaz limites de tempo mais liberais l' e u'.

Prova:

1.	$\Lambda \vdash (l' \leq l \leq u \leq u')$	
2.	$\Lambda \vdash l(p_l \delta^u q) = l$	(Lim-δ)
3.	$\Lambda \vdash u(p_l \delta^u q) = u$	(Lim-δ)
4.	$\Lambda \vdash l' \leq l(p_l \delta^u q) \leq u(p_l \delta^u q) \leq u'$	(TIME)
5.	$\Lambda \vdash \mathit{enabled}(p_l \delta^u q) \rightarrow p$	(como na prova anterior)
6.	$\Lambda \vdash [p_l \delta^u q] q$	(como na prova anterior)
7.	$\Lambda \vdash p_l \delta^u q \supset p_{l'} \delta^{u'} q$	(δ 4, 5, 6)

□

5. (Subs-Enabled):

$$\vdash (t_1 \supset t_2) \rightarrow (\mathit{enabled}(t_1) \rightarrow \mathit{enabled}(t_2))$$

Prova:

1.	$\vdash (t_1 \supset t_2) \rightarrow ([t_2]\bot \rightarrow [t_1]\bot)$	(S3)
2.	$\vdash ([t_2]\bot \rightarrow [t_1]\bot) \leftrightarrow (\neg[t_1]\bot \rightarrow \neg[t_2]\bot)$	(CP)
3.	$\vdash (t_1 \supset t_2) \rightarrow (\neg[t_1]\bot \rightarrow \neg[t_2]\bot)$	(CP)
4.	$\vdash (t_1 \supset t_2) \rightarrow (\mathit{enabled}(t_1) \rightarrow \mathit{enabled}(t_2))$	(Def)

□

6. (Pos-Falso):

$$\vdash \neg enabled(p_l\delta^u\bot)$$

Prova:

1.	$\vdash p_l\delta^u\bot \supset p_l\delta^u\bot$	(S1)
2.	$\vdash (p_l\delta^u\bot \supset p_l\delta^u\bot) \rightarrow [p_l\delta^u\bot]\bot$	(Pos)
3.	$\vdash [p_l\delta^u\bot]\bot$	(Modus Ponens 1, 2)
4.	$\vdash \neg enabled(p_l\delta^u\bot)$	(Def)

□

7. (Nec-Falso):

$$\vdash \neg enabled(\bot_l\delta^u q)$$

Prova:

1.	$\vdash \bot_l\delta^u q \supset \bot_l\delta^u q$	(S1)
2.	$\vdash (\bot_l\delta^u q \supset \bot_l\delta^u q) \rightarrow (enabled(\bot_l\delta^u q) \rightarrow \bot)$	(CN)
3.	$\vdash enabled(\bot_l\delta^u q) \rightarrow \bot$	(Modus Ponens 1, 2)
4.	$\vdash \neg enabled(\bot_l\delta^u q)$	(CP)

□

4.7 Completude Fraca

4.7.1 Definições e Considerações Iniciais

Para provar a completude fraca de RETOOL, são necessárias as seguintes definições e resultados:

Definição 4.7.1 (Consistência) *:*

1. ϕ *é* consistente *se e somente se* $\nvdash \neg\phi$ *(o que equivale a* $\phi \nvdash \bot$*).*

2. *Um conjunto finito* $\{\phi_1, \phi_2, ..., \phi_k\}$ *é consistente se e somente se a conjunção* $\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_k$ *for consistente.*

3. *Um conjunto infinito* Φ *de fórmulas é consistente se e somente se todo subconjunto finito de* Φ *for consistente.*

4. *Uma fórmula ou conjunto que não seja consistente é chamado de* inconsistente.

Definição 4.7.2 (Conjuntos maximais consistentes) *Um conjunto* Φ *de fórmulas é dito* maximal consistente *se e somente se:*

1. Φ *for consistente, e*

2. *Para toda fórmula* ϕ *de RETOOL, ou* $\phi \in \Phi$*, ou* $\neg\phi \in \Phi$*.*

4.7.1.1 Propriedades Envolvendo Conjuntos Maximais Consistentes

Lema 4.7.1 *Todo conjunto consistente* Φ *de fórmulas pode ser estendido para um conjunto maximal consistente.*

Prova:

Seja Φ um conjunto consistente. Seja a seguinte enumeração de todas as fórmulas RETOOL:

$$\phi_0, \phi_1, \phi_2, ..., \phi_n, ...$$

Definimos a seguinte sequência de conjuntos

$$\Phi_0 \subseteq \Phi_1 \subseteq \Phi_2 \subseteq ... \subseteq \Phi_n \subseteq ...$$

com $\Phi_0 = \Phi$ e os demais conjuntos definidos como

$$\Phi_{n+1} = \begin{cases} \Phi_n & \text{se } \Phi_n \cup \{\phi_n\} \text{ é inconsistente;} \\ \Phi_n \cup \{\phi_n\} & \text{caso contrário} \end{cases}$$

Definimos, então,

$$\Phi_\omega = \bigcup_{n \geq 0} \Phi_n$$

Φ_ω é consistente, pois, se não fosse, haveria um subconjunto finito inconsistente $\Psi \subseteq \Phi_\omega$. Mas, neste caso, Ψ deveria estar contido em Φ_n para algum valor de n. No entanto, para todo n, Φ_n é consistente, por construção. Logo, chegaríamos a uma contradição.

Φ_ω é maximal consistente, pois, se não fosse, haveria uma fórmula ϕ tal que $\phi \notin \Phi_\omega$ e $\Phi_\omega \cup \{\phi\}$ é consistente. No entanto, esta fórmula ϕ pertence à enumeração de fórmulas construída no início desta prova. Digamos que $\phi = \phi_n$ na enumeração. Se $\phi_n \notin \Phi_\omega$, é porque ϕ_n não foi incluída em Φ_n, o que acarreta $\Phi_\omega \cup \{\phi_n\}$ inconsistente. Isto nos levaria a uma contradição.

□

Lema 4.7.2 *Todo conjunto maximal consistente* Φ *é fechado para* $\vdash$*; i.e., para toda fórmula* ϕ*,*

$$\Phi \vdash \phi \Rightarrow \phi \in \Phi$$

Prova:

Suponha $\Phi \vdash \phi$. Se $\phi \notin \Phi$, então $\neg\phi \in \Phi$, pela definição de conjunto maximal consistente. Mas, então, pela definição de derivabilidade, $\Phi \vdash \neg\phi$, e daí $\Phi \vdash \bot$, contradizendo a suposição inicial de que Φ é consistente.

□

Corolário 4.7.1 *Seja Φ um conjunto maximal consistente. Então, para todas as fórmulas ϕ e ψ:*

1. *$\vdash \phi$ implica $\phi \in \Phi$ (i.e., Φ contém todos os axiomas e teoremas de RETOOL).*

2. *$\phi \in \Phi$ se e somente se $\neg\phi \notin \Phi$.*

3. *Se $\phi \in \Phi$ implica $\psi \in \Phi$, então $(\phi \rightarrow \psi) \in \Phi$, e vice-versa.*

Lema 4.7.3 *Seja Λ um conjunto qualquer de fórmulas. Seja ϕ uma fórmula. Se, para todo conjunto maximal consistente Φ, $\Lambda \subseteq \Phi$ implica $\phi \in \Phi$, então $\Lambda \vdash \phi$.*

Prova:

Suponha que não seja o caso que $\Lambda \vdash \phi$. Então $\Lambda \cup \{\neg\phi\}$ é consistente. Como todo conjunto consistente pode ser estendido para um conjunto maximal consistente (lema 4.7.1), existe um conjunto maximal consistente contendo Λ e $\neg\phi$, o que contradiz a suposição inicial.

□

4.7.2 Visão Geral

O resultado discutido nesta seção é a recíproca do Teorema 4.5.1 (Correção). Desejamos provar que, para toda fórmula RETOOL ϕ, se ϕ é válida, então ϕ é um teorema:

Teorema 4.7.1 (Completude Fraca) [2]

$$\models \phi \quad \Rightarrow \quad \vdash \phi$$

O enunciado deste Teorema é equivalente a

$$\not\vdash \phi \quad \Rightarrow \quad \not\models \phi$$

i.e., para algum estado w de algum sistema de transição temporizado S,

$$\not\vdash \phi \quad \Rightarrow \quad S, w \models \neg\phi$$

substituindo ϕ por $\neg\psi$, isto equivale a

$$\not\vdash \neg\psi \quad \Rightarrow \quad S, w \models \psi$$

para algum estado w de algum sistema de transição temporizado S.

Assim, provar a completude fraca de RETOOL equivale a provar que toda fórmula consistente é satisfatível. Como toda fórmula consistente está contida em um conjunto consistente e todo conjunto consistente está contido em um conjunto maximal consistente (lema 4.7.1), basta provar que todo conjunto maximal consistente é satisfatível. Isto será feito através da construção de um modelo RETOOL S^c contendo, para cada conjunto maximal consistente Φ, um estado w_Φ tal que $S^c, w_\Phi \models \Phi$. Este modelo, chamado *modelo canônico*, é definido na próxima subseção.

[2] O termo "fraca" diferencia este tipo de completude da noção de "completude forte", que significa que, para todo conjunto Θ e toda fórmula ϕ, é o caso que $\Theta \models \phi \Rightarrow \Theta \vdash \phi$. A completude forte de RETOOL não é examinada nesta dissertação.

4.7.3 O Modelo Canônico

4.7.3.1 Definição

O modelo canônico, como qualquer modelo RETOOL, deve ser definido com base em um conjunto de atributos A^c e um conjunto de símbolos de ação Γ^c. Embora possamos definir A^c como contendo atributos arbitrários $a_1, a_2, ...$, não podemos fazer o mesmo com Γ^c, pois RETOOL é uma lógica para raciocinar, em última análise, sobre a funcionalidade de ações (i.e., suas condições necessárias, pós-condições e limites de tempo). Símbolos de ação arbitrários $g_1, g_2, ...$ simplesmente não carregam este tipo de informação funcional.

Para resolver este impasse, construímos um conjunto de símbolos de ação Γ^c e, ao mesmo tempo, definimos a funcionalidade de todos os símbolos $g \in \Gamma^c$ através de um conjunto Σ de fórmulas RETOOL.

Sendo A um conjunto de símbolos de atributos arbitrários, definimos as seguintes classes de equivalência a partir do conjunto de proposições de estados (ver definição 4.2.1) $PE = \{p, q, ...\}$:

Definição 4.7.3 *Para cada* $p \in PE$, $\tilde{p} = \{r \mid \vdash p \leftrightarrow r\}$

Feito isto, definimos o conjunto de símbolos de ação Γ^c da seguinte forma:

Definição 4.7.4 $\Gamma^c = \{\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”} \mid p, q \in PE \setminus \tilde{\bot},\ l \in TIME\ ,\ u \in TIME \cup \{\infty\}\}$

A intenção é criar um símbolo de ação para cada quádrupla (p, q, l, u) composta pelas proposições de estados satisfatíveis p e q (a menos de equivalência tautológica) e pelos limites de tempo l e u. Isto é, desejamos que haja um símbolo de ação para cada combinação possível (e satisfatível) de condição necessária, pós-condição

e limites de tempo. Para que os símbolos tenham o significado desejado, definimos a funcionalidade de cada "$\tilde{p}_l\delta^u\tilde{q}$" $\in \Gamma^c$ através do seguinte conjunto Σ de fórmulas RETOOL:

$$\Sigma =$$

$$\{(\text{“}\tilde{p}_l\delta^u\tilde{q}\text{”} \supset p_l\delta^u q) \mid p, q \in PE \setminus \tilde{\bot},\ \ l \in \text{TIME}\ ,\ \ u \in \text{TIME}\ \cup \{\infty\}\}$$

$$\cup$$

$$\{(p_l\delta^u q \supset \text{“}\tilde{p}_l\delta^u\tilde{q}\text{”}) \mid p, q \in PE \setminus \tilde{\bot},\ \ l \in \text{TIME}\ ,\ \ u \in \text{TIME}\ \cup \{\infty\}\}$$

Assim, em todo modelo S que satisfaça Σ, cada símbolo de ação "$\tilde{p}_l\delta^u\tilde{q}$" servirá como "testemunha" do termo de ação $p_l\delta^u q$, com a denotação do símbolo correspondendo exatamente à denotação do termo (o que será provado mais adiante).

Além da funcionalidade dos símbolos de ação, precisamos fornecer outro tipo de informação extra-lógica para a construção correta do modelo canônico: a especificação do estado inicial. Para tanto, definimos um conjunto consistente Θ contendo todas as proposições de estado $x_1, x_2, ...$ que desejamos que sejam verdadeiras no estado inicial, necessitadas com a modalidade $[\,]$:

$$\Theta = \{[\,]x_1, [\,]x_2, ...\}$$

Exigimos, ainda, que o conjunto $\{x \mid [\,]x \in \Theta\}$ seja consistente.

Uma consequência da escolha de Γ^c e dos conjuntos Σ e Θ é que não estaremos mais provando o teorema da completude para todas as fórmulas RETOOL, mas apenas para as fórmulas que são válidas na classe de modelos RETOOL restrita àqueles que satisfazem $\Sigma \cup \Theta$.

Além disso, a possibilidade da existência de conjuntos Θ diferentes que satisfaçam nossas exigências nos leva a uma *classe* de modelos canônicos. Nesta classe, o conjunto

de estados é igual em todos os modelos, mas o estado inicial é diferente para cada modelo, dependendo da escolha de Θ.

O teorema da completude fraca a ser provado, então, se torna:

Teorema 4.7.2 (Completude Fraca com Γ^c)

$$\Sigma \cup \Theta \models \phi \quad \Rightarrow \quad \Sigma \cup \Theta \vdash \phi$$

Isto equivale a mostrar que

$$\Sigma \cup \Theta \nvdash \neg\phi \quad \Rightarrow \quad S, w \models \phi$$

para algum estado w de algum sistema de transição temporizado S no qual $\Sigma \cup \Theta$ seja verdadeiro.

Segue-se a definição completa do modelo canônico S^c:

Definição 4.7.5 (Modelo Canônico) *Para um conjunto de símbolos de atributo $A^c = \{a_1, a_2, ...\}$, o conjunto de símbolos de ação Γ^c e os conjuntos Σ e Θ definidos acima, o modelo canônico de RETOOL é definido como $S^c = (W^c, \rightarrow^c, l^c, u^c, I^c, w_0^c)$, onde*

- *Conjunto de estados:*

$$W^c = \{w \mid w \text{ maximal consistente e } (\Sigma \cup \Theta) \subseteq w\}$$

- *Transições: para todo par de estados w e w'*

$$w \xrightarrow{\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}} w' \quad \Leftrightarrow \quad w \setminus [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}] \subseteq w'$$

onde $w \setminus [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}] \quad = \quad \{\phi \mid [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\phi \in w\}$

- *Funções de limites de tempo:*

$$l^c(\text{“}\tilde{p}_l\delta^u\tilde{q}\text{”}) = l$$

$$u^c(\text{“}\tilde{p}_l\delta^u\tilde{q}\text{”}) = u$$

- *Função de interpretação:*

$$I^c(a) = \{w \in W^c \mid S^c, a \in w\}$$

- *Estado inicial:*

  w_0^c *é algum estado designado de* W^c *tal que* $w_0 \models \Theta \setminus [\,]$

4.7.3.2 Comentários

Estados do modelo canônico: Um estado do modelo canônico é definido como sendo o próprio conjunto maximal consistente que ele satisfaz. Por economia de símbolos, usamos, então, w em vez de Φ ou w_Φ.

Transições: As transições são definidas sintaticamente no modelo canônico de acordo com o que esperamos da modalidade $[g], g \in \Gamma^c$: se $w \vdash [g]\phi$, então $w' \vdash \phi$ para todo w' tal que $w \xrightarrow{g} w'$.

Derivabilidade nos estados do modelo canônico: Em algumas ocasiões, no restante desta prova de completude, diremos que uma dada fórmula ϕ pertence a um dado estado w do modelo canônico “por um dado axioma” ou “por uma dada regra de inferência”, referindo-nos ao fato de que um conjunto maximal consistente w é fechado para a relação de derivabilidade (lema 4.7.2 e corolário 4.7.1); i.e., ϕ é uma instância de um esquema de axioma ou da conclusão de uma regra cujas premissas também pertencem a w.

4.7.3.3 Propriedades do Modelo Canônico

Para provar que o modelo canônico satisfaz, de fato, todas as extensões maximais consistentes de $\Sigma \cup \Theta$, precisamos mostrar que, em S^c, a relação de derivabilidade $\vdash$ coincide com a relação de satisfação $\models$.[3] Antes disso, porém, resultados mais básicos precisam ser provados:

Lema 4.7.4 *O conjunto de fórmulas $\Sigma \cup \Theta$ é consistente; i.e., o conjunto de estados do modelo canônico é não-vazio.*

Prova:

Segundo a definição de consistência (def. 4.7.1), um conjunto infinito será consistente se e somente se qualquer um de seus subconjuntos finitos for consistente. Para mostrar que $\Sigma \cup \Theta$ é consistente, mostraremos que um subconjunto finito arbitrário de $\Sigma \cup \Theta$ é consistente. Chamemos este subconjunto finito de Σ_f.

Um dos corolários do Teorema 4.5.1 (Correção) é o que afirma que um conjunto de fórmulas é consistente se ele for satisfatível (corolário 4.5.1). Assim, basta encontrar um modelo RETOOL contendo um estado que satisfaça Σ_f para provar que Σ_f (e, por consequência, $\Sigma \cup \Theta$) é consistente. Vamos construir este modelo a partir de uma análise do conteúdo de Σ_f:

[3] Na definição do modelo canônico, isto foi garantido apenas no que diz respeito aos símbolos de atributo e, por extensão, às proposições de estado (ver a definição 4.3.3 – relação de satisfação – e o item relativo à função de interpretação I^c na definição 4.7.5).

Σ_f é um conjunto finito de fórmulas da forma

$$
\begin{array}{ll}
\{ & g_1 \supset p_{1\,l_1}\delta^{u_1} q_1, \\
& g_2 \supset p_{2\,l_2}\delta^{u_2} q_2, \\
& \vdots \\
& g_k \supset p_{k\,l_k}\delta^{u_k} q_k, \\
& r_{1\,m_1}\delta^{v_1} s_1 \supset h_1, \\
& r_{2\,m_2}\delta^{v_2} s_2 \supset h_2, \\
& \vdots \\
& r_{n\,m_n}\delta^{v_n} s_n \supset h_n \\
& [\,]x_1, [\,]x_2, ...[\,]x_t \quad \}
\end{array}
$$

As seguintes restrições valem para as fórmulas de Σ_f:

- Todos os símbolos de ação $g_1, g_2, ..., g_k$ são diferentes entre si.
- Todos os símbolos de ação $h_1, h_2, ..., h_n$ são diferentes entre si.
- Se existem i e j tal que $g_i = h_j$, então $p_i = r_j$, $q_i = s_j$, $l_i = m_j$ e $u_i = v_j$.
- Todas as proposições de estado $p_1, p_2, ..., p_k$ são diferentes entre si.
- Todas as proposições de estado $q_1, q_2, ..., q_k$ são diferentes entre si.
- Todas as proposições de estado $r_1, r_2, ..., r_n$ são diferentes entre si.
- Todas as proposições de estado $s_1, s_2, ..., s_n$ são diferentes entre si.

Construímos o seguinte modelo para Σ_f, baseado no conjunto de símbolos de ação $\{g_1, g_2, ...g_k, h_1, h_2, ..., h_n\}$:

- O conjunto de estados consiste em todas as valorações possíveis do conjunto de proposições de estado envolvidas:[4]

$$\{p_1, p_2, ..., p_k, q_1, q_2, ..., q_k, r_1, r_2, ..., r_n, s_1, s_2, ..., s_n, x_1, x_2, ..., x_t\}$$

- Para cada fórmula de Σ_f da forma

$$g_i \supset p_{i\,l_i} \delta^{u_i} q_i$$

criamos transições de cada estado que satisfaz p_i com todos os estados que satisfazem q_i.

- Para cada fórmula de Σ_f da forma

$$r_{j\,m_j} \delta^{v_j} s_j \supset h_j$$

criamos transições de cada estado que satisfaz r_j com todos os estados que satisfazem s_j. Não há conflito com as transições criadas no item anterior, justamente porque se $g_i = h_j$, então $p_i = r_j$, $q_i = s_j$, $l_i = m_j$ e $u_i = v_j$; i.e., as mesmas transições são criadas nos dois itens.

- Definimos as funções de limites de tempo l e u de modo que

$$l(g_i) = l_i \quad u(g_i) = u_i \quad 1 \leq i \leq k$$
$$l(h_j) = l_j \quad u(h_j) = u_j \quad 1 \leq j \leq n$$

- A função de valoração I leva cada atributo a ao conjunto de estados nos quais a é verdadeiro.

- O estado inicial w_0 é um estado designado de W tal que as fórmulas $x_1, x_2, ...x_t$ são verdadeiras em w_0. Como exigimos que $\Theta \setminus [\,]$ fosse consistente, seu subconjunto finito $\{x_1, x_2, ...x_t\}$ também o será, e existirá uma valoração que satisfaz este subconjunto finito.

[4]Se houver dependências entre as proposições de estado envolvidas, o número de valorações possíveis será menor do que se as proposições de estado forem independentes entre si.

O sistema de transição temporizado construído desta forma satisfaz todas as fórmulas de Σ_f. Como Σ_f é um subconjunto finito arbitrário de $\Sigma \cup \Theta$, temos, então, que $\Sigma \cup \Theta$ é satisfatível e, consequentemente, consistente.

□

Definimos o conjunto de símbolos de ação Γ^c com a intenção de que um símbolo de ação "$\tilde{p}_l \delta^u \tilde{q}$" tenha o mesmo significado que o termo de ação $p_l \delta^u q$. Que esta coincidência de denotações ocorre é provado através dos próximos lemas.

Lema 4.7.5 (Coincidência na relação de derivação) *Para todo $w \in W^c$, para todo $p, q \in PE \setminus \tilde{\perp}$, para todo $l \in TIME$, $u \in TIME \cup \{\infty\}$, para todo termo de ação t, e para toda fórmula ϕ:*

$$\begin{aligned} w \vdash t \supset p_l \delta^u q \quad &\Leftrightarrow \quad w \vdash t \supset \text{“}\tilde{p}_l \delta^u \tilde{q}\text{”} \\ w \vdash p_l \delta^u q \supset t \quad &\Leftrightarrow \quad w \vdash \text{“}\tilde{p}_l \delta^u \tilde{q}\text{”} \supset t \\ w \vdash [p_l \delta^u q]\phi \quad &\Leftrightarrow \quad w \vdash [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\phi \end{aligned}$$

Prova:

Lembramos que todas as fórmulas de Σ pertencem a todo $w \in W^c$. Em especial,

$$p_l \delta^u q \supset \text{“}\tilde{p}_l \delta^u \tilde{q}\text{”} \tag{4.1}$$

e

$$\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”} \supset p_l \delta^u q \tag{4.2}$$

pertencem a w.

Para a primeira equivalência:

$$\begin{aligned} &w \vdash t \supset p_l \delta^u q \quad \Leftrightarrow \quad \text{(por (4.1) e pelo axioma (S2))} \\ &w \vdash t \supset \text{“}\tilde{p}_l \delta^u \tilde{q}\text{”} \end{aligned}$$

Para a segunda equivalência:

$$w \vdash t \supset \text{“}\tilde{p}_l \delta^u \tilde{q}\text{”} \quad \Leftrightarrow \quad \text{(por (4.2) e pelo axioma (S2))}$$
$$w \vdash t \supset p_l \delta^u q$$

Para a terceira equivalência:

Suponha $w \vdash [p_l \delta^u q]\phi$. Por (4.1) e pelos axiomas (S2) e (S3),

$$w \vdash [p_l \delta^u q]\phi \rightarrow [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\phi$$

Suponha $w \vdash [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\phi$. Por (4.2) e pelos axiomas (S2) e (S3),

$$w \vdash [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\phi \rightarrow [p_l \delta^u q]\phi$$

□

Lema 4.7.6 (Coincidência para a fórmula enabled()) *Para todo $w \in W^c$, para todo $p, q \in PE \setminus \tilde{\perp}$, para todo $l \in TIME$, $u \in TIME \cup \{\infty\}$:*

$$w \in enable(p_l \delta^u q) \quad \Leftrightarrow \quad w \vdash enabled(p_l \delta^u q)$$

Pelo lema anterior, a equivalência vale também para a fórmula enabled (“$\tilde{p}_l \delta^u \tilde{q}$”).

Prova:

($\Rightarrow$) :

Suponha que $w \in enable(p_l \delta^u q)$. Então existe w' tal que $w \setminus [p_l \delta^u q] \subseteq w'$. Se $w \nvdash enabled(p_l \delta^u q)$, então, como w é maximal consistente, seria o caso que $w \vdash \neg enabled(p_l \delta^u q)$. Mas $\neg enabled(p_l \delta^u q)$ é uma abreviatura para $[p_l \delta^u q]\perp$, o que significaria que $\perp \in w'$. Logo, $w \vdash enabled(p_l \delta^u q)$.

($\Leftarrow$) :

Suponha que $w \vdash \mathit{enabled}(p_l\delta^u q)$; i.e., $\mathit{enabled}(p_l\delta^u q) \in w$, o que é uma abreviatura para $\neg[p_l\delta^u q]\bot \in w$. Devemos mostrar que existe w' tal que $w \setminus [p_l\delta^u q] \subseteq w'$. Isto equivale a mostrar que $w \setminus [p_l\delta^u q]$ é consistente, o que é feito provando-se a seguinte afirmação:

Afirmação: *Se w é maximal consistente e $w \setminus [p_l\delta^u q]$ é inconsistente, então* $w \vdash [p_l\delta^u q]\bot$.

Prova da afirmação:

Seja w um conjunto para o qual valham as premissas. Então, algum subconjunto finito de $w \setminus [p_l\delta^u q]$ é inconsistente. Sendo este subconjunto $\{\phi_1, \phi_2, ..., \phi_k\}$, temos que $\vdash \neg(\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_k)$. Temos também que $\vdash \neg(\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_k \wedge \neg\phi)$ para qualquer fórmula ϕ; i.e., $\{\phi_1, \phi_2, ..., \phi_k, \neg\phi\}$ é inconsistente para qualquer fórmula ϕ.

Pelo cálculo proposicional, temos que

$$\vdash \phi_1 \rightarrow (\phi_2 \rightarrow (...(\phi_k \rightarrow \phi)...))$$

Pela regra (N),

$$\vdash [p_l\delta^u q](\phi_1 \rightarrow (\phi_2 \rightarrow (...(\phi_k \rightarrow \phi)...)))$$

Chamando $(\phi_2 \rightarrow (...(\phi_k \rightarrow \phi)...))$ de ξ, temos

$$\vdash [p_l\delta^u q](\phi_1 \rightarrow \xi)$$

e, por (K)

$$\vdash [p_l\delta^u q]\phi_1 \rightarrow [p_l\delta^u q]\xi$$

Mas, como $\phi_1 \in w \setminus [p_l\delta^u q]$, temos que $[p_l\delta^u q]\phi_1 \in w$. Logo, por *modus ponens*, $[p_l\delta^u q]\xi \in w$. Repetindo-se este argumento k vezes, chega-se à conclusão de que

$[p_l\delta^u q]\phi \in w$. Como ϕ era uma fórmula arbitrária, mostramos que $[p_l\delta^u q]\bot \in w$, contradizendo a suposição inicial do lema. Logo, $w \setminus [p_l\delta^u q]$ não pode ser inconsistente, e existe w' tal que $w \setminus [p_l\delta^u q] \subseteq w'$; portanto, $w \in enable(p_l\delta^u q)$.

□

O lema a seguir fornece algumas implicações do fato de dois estados w e w' do modelo canônico estarem ligados por uma transição:

Lema 4.7.7 *Para todo par de estados $w, w' \in W^c$, se $w \overset{\text{“}\tilde{p}_l\delta^u\tilde{q}\text{”}}{\longrightarrow} w'$, então*

1. $w \vdash enabled(p_l\delta^u q)$
2. $w \vdash p$
3. $w \vdash [p_l\delta^u q]q$
4. $w' \vdash q$

Prova:

Sejam w e w' tais que $w \overset{\text{“}\tilde{p}_l\delta^u\tilde{q}\text{”}}{\longrightarrow} w'$. Então

1. $w \setminus [\text{“}\tilde{p}_l\delta^u\tilde{q}\text{”}] \subseteq w'$. Mas, pelo lema 4.7.5, isto equivale a $w \setminus [p_l\delta^u q] \subseteq w'$. Assim, $[p_l\delta^u q]\bot$ não pertence a w, e $w \vdash enabled(p_l\delta^u q)$.
2. Pelo axioma (S1), $w \vdash p_l\delta^u q \supset p_l\delta^u q$; pelo axioma (CN) e *modus ponens*, $w \vdash enabled(p_l\delta^u q) \rightarrow p$; por *modus ponens* de novo, usando o item anterior, $w \vdash p$.
3. Pelo axioma (S1) e pelo axioma (Pos), $w \vdash [p_l\delta^u q]q$.

4. Pelo lema 4.7.5, $w \vdash [p_l \delta^u q]q \quad \Leftrightarrow \quad w \vdash [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]q$; logo, $w' \vdash q$.

□

Por fim, o seguinte lema afirma que as denotações de “$\tilde{p}_l \delta^u \tilde{q}$” e de $p_l \delta^u q$ coincidem no modelo canônico.

Lema 4.7.8 *No modelo canônico,* $[\![\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\!] = [\![p_l \delta^u q]\!]$.

Prova:

($\subseteq$)

Seja $(w, w') \in [\![\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\!]$. Pelo lema 4.7.7, temos que $w \vdash p$ e $w' \vdash q$. Como p e q são proposições de estado, derivabilidade coincide com satisfação; logo, $w \models p$ e $w' \models q$. Pela definição de Γ^c, temos que $l \leq l(\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}) \leq u(\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}) \leq u$.

Assim, estão satisfeitas as condições para que $(w, w') \in [\![p_l \delta^u q]\!]$.

($\supseteq$)

Seja $(w, w') \in [\![p_l \delta^u q]\!]$. Então existe um $g \in \Gamma^c$ tal que $enable(g) \subseteq [\![p]\!], \forall v, v' : (v \xrightarrow{g} v' \Rightarrow v' \models q)$, e $l \leq l(g) \leq u(g) \leq u$. Chamemos este g de “$\tilde{r}_x \delta^y \tilde{s}$”.

Se $\tilde{r} = \tilde{p}, \tilde{s} = \tilde{q}, x = l$ e $y = u$, a prova está terminada. Suponha que não.

Temos $enable(\text{“}\tilde{r}_x \delta^y \tilde{s}\text{”}) \subseteq [\![p]\!]$ e $enable(\text{“}\tilde{r}_x \delta^y \tilde{s}\text{”}) \subseteq [\![r]\!]$; daí, $enable(\text{“}\tilde{r}_x \delta^y \tilde{s}\text{”}) \subseteq [\![p \wedge r]\!]$.

Analogamente, quanto às pós-condições, temos que $\forall v, v' : (v \xrightarrow{\text{“}\tilde{r}_x \delta^y \tilde{s}\text{”}} v' \Rightarrow v' \models q \wedge s)$.

Quanto aos limites de tempo, temos que $l \leq x \leq y \leq u$.

Assim, $[\![\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}]\!] \subseteq [\![\text{“}(\widetilde{p \wedge r})_l\delta^u(\widetilde{q \wedge s})\text{”}]\!]$.

Sejam v e v' dois estados quaisquer do modelo canônico. Então, temos

$$(v, v') \in [\![\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}]\!] \quad \Rightarrow \quad v \vdash p \wedge r$$

Pelo lema 4.7.7, isto equivale a

$$v \vdash \mathit{enabled}(\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}) \quad \Rightarrow \quad v \vdash p \wedge r$$

Como v é maximal consistente,

$$v \vdash \mathit{enabled}(\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}) \rightarrow (p \wedge r)$$

Como v é qualquer conjunto maximal consistente contendo $\Sigma \cup \Theta$, pelo lema 4.7.3,

$$\Sigma \cup \Theta \vdash \mathit{enabled}(\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}) \rightarrow (p \wedge r)$$

Como $\vdash (p \wedge r) \rightarrow p$, temos

$$\Sigma \cup \Theta \vdash \mathit{enabled}(\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}) \rightarrow p \tag{4.3}$$

Para as pós-condições, sabemos que

$$(v, v') \in [\![\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}]\!] \quad \Rightarrow \quad v \vdash [\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}](q \wedge s)$$

Mas, também, ainda que $(v, v') \notin [\![\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}]\!]$, também $v \vdash [\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}](q \wedge s)$; daí, conclui-se que

$$\Sigma \cup \Theta \vdash [\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}](q \wedge s)$$

Como $\vdash (q \wedge s) \rightarrow q$, temos

$$\Sigma \cup \Theta \vdash [\text{“}\tilde{r}_x\delta^y\tilde{s}\text{”}]q \tag{4.4}$$

Para os limites de tempo, temos

$$\Sigma \cup \Theta \vdash l \leq x \leq y \leq u \tag{4.5}$$

Por (4.3), (4.4), (4.5) e pela regra (δ), concluímos que

$$\Sigma \cup \Theta \vdash \text{“}\tilde{r}_x \delta^y \tilde{s}\text{”} \supset \text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}$$

Pelo axioma (S3), para qualquer fórmula ϕ,

$$\Sigma \cup \Theta \vdash [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\phi \rightarrow [\text{“}\tilde{r}_x \delta^y \tilde{s}\text{”}]\phi$$

Assim, para w e w' da suposição inicial, como $w \setminus [\text{“}\tilde{r}_x \delta^y \tilde{s}\text{”}] \subseteq w'$, também $w \setminus [\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}] \subseteq w'$, e $(w, w') \in [\![\text{“}\tilde{p}_l \delta^u \tilde{q}\text{”}]\!]$.

□

4.7.4 O Lema da Coincidência

Para mostrar que o modelo canônico satisfaz todos os conjuntos maximais consistentes contendo $\Sigma \cup \Theta$, precisamos provar a coincidência entre derivabilidade e satisfação no modelo. Isto é feito no

Lema 4.7.9 (Coincidência) *: Para todo estado $w \in W^c$, para toda fórmula ϕ*

$$\phi \in w \quad \Leftrightarrow \quad S^c, w \models \phi$$

Prova:

A prova é feita por indução na formação da fórmula ϕ:

(Caso 1) $\phi = p \in PE$:

Pela definição de I^c.

(Caso 2) $\phi = t_1 \supset t_2$

Como os símbolos de ação primitivos podem ser substituídos por termos de ação construídos com δ, gerando fórmulas equivalentes no modelo canônico (lema 4.7.5), e como as denotações dos símbolos de ação coincidem com as denotações dos termos de ação (lema 4.7.8), podemos nos restringir ao caso em que

$$t_1 = p_l \delta^u q$$

$$t_2 = r_m \delta^n s$$

$(\Rightarrow)$

Suponha que $p_l \delta^u q \supset r_m \delta^n s \in w$.

Queremos mostrar que $w \models p_l \delta^u q \supset r_m \delta^n s$, o que equivale a $[\![p_l \delta^u q]\!] \subseteq [\![r_m \delta^n s]\!]$.

Seja $(w, w') \in [\![p_l \delta^u q]\!]$. Então

$p_l \delta^u q \supset r_m \delta^n s \in w$	$\Rightarrow$	(pelo axioma (S3))
$\forall \phi \quad [r_m \delta^n s]\phi \rightarrow [p_l \delta^u q]\phi \quad \in w$	$\Rightarrow$	(por w ser maximal consistente)
$\forall \phi \quad [r_m \delta^n s]\phi \in w \quad \Rightarrow \quad [p_l \delta^u q]\phi \in w$	$\Rightarrow$	(porque $w \setminus [p_l \delta^u q] \subseteq w'$)
$w \setminus [r_m \delta^n s] \subseteq w'$	$\Rightarrow$	
$(w, w') \in [\![r_m \delta^n s]\!]$		

$(\Leftarrow)$

Suponha que $w \models p_l \delta^u q \supset r_m \delta^n s$. Isto significa que $[\![p_l \delta^u q]\!] \subseteq [\![r_m \delta^n s]\!]$. Então, para todo estado w do modelo canônico,

$[\![p_l\delta^u q]\!] \subseteq [\![r_m\delta^n s]\!] \quad \Rightarrow$

$w \models enabled(p_l\delta^u q) \quad \Rightarrow \quad w \models r \quad \Leftrightarrow$ (pelo lema 4.7.6 e pela definição de I^c)

$w \vdash enabled(p_l\delta^u q) \quad \Rightarrow \quad w \vdash r \quad \Leftrightarrow$ (por w ser maximal consistente)

$w \vdash enabled(p_l\delta^u q) \rightarrow r \quad \Leftrightarrow$ (w arbitrário contendo $\Sigma \cup \Theta$)

$\Sigma \cup \Theta \vdash enabled(p_l\delta^u q) \rightarrow r$

No que diz respeito à pós-condição, temos que, para todo estado w do modelo canônico,

$[\![p_l\delta^u q]\!] \subseteq [\![r_m\delta^n s]\!] \quad \Rightarrow$

$w \models [p_l\delta^u q]s \quad \Rightarrow$ (pelo lema 4.7.7)

$w \vdash [p_l\delta^u q]s \quad \Leftrightarrow$ (w arbitrário contendo $\Sigma \cup \Theta$)

$\Sigma \cup \Theta \vdash [p_l\delta^u q]s$

Quanto aos limites de tempo, temos que $m \leq l \leq u \leq n$.

Logo, pelas regras derivadas (CN-Enfraq) e (Pos-Enfraq) e pelo teorema (Lim-Enfraq) (ver seção 4.6),

$$\Sigma \cup \Theta \vdash p_l\delta^u q \supset r_m\delta^n s$$

Concluímos que $w \vdash p_l\delta^u q \supset r_m\delta^n s$.

(Caso 3) $\phi = \neg\psi$

(Hipótese indutiva: $\psi \in w \quad \Leftrightarrow \quad S^c, w \models \psi$)

$\neg\psi \in w \quad \Leftrightarrow$ (por w ser maximal consistente)

$\psi \notin w \quad \Leftrightarrow$ (pela h.i.)

$w \not\models \psi \quad \Leftrightarrow$ (pela definição de $\models$)

$w \models \neg\psi$

(Caso 4) $\phi = \psi_1 \rightarrow \psi_2$

(Hipótese indutiva: $\psi_i \in w \quad \Leftrightarrow \quad S^c, w \models \psi_i, i \in \{1, 2\}$)

$$
\begin{array}{lll}
\psi_1 \rightarrow \psi_2 \in w & \Leftrightarrow & \text{(por } w \text{ ser maximal consistente)} \\
\psi_1 \in w \Rightarrow \psi_2 \in w & \Leftrightarrow & \text{(pela h.i.)} \\
w \models \psi_1 \Rightarrow w \models \psi_2 & \Leftrightarrow & \text{(pela definição de } \models) \\
w \models \psi_1 \rightarrow \psi_2 & &
\end{array}
$$

(Caso 5) $\phi = [t]\psi$

(Hipótese indutiva: $\psi \in w \quad \Leftrightarrow \quad S^c, w \models \psi$)

($\Rightarrow$)

Suponha que $[t]\psi \in w$. Então, $\psi \in w \setminus [t]$.

Para todo estado $w', \quad (w, w') \in [\![t]\!] \;\Rightarrow\; w \setminus [t] \subseteq w'$.

I.e., para todo estado $w', \quad (w, w') \in [\![t]\!] \;\Rightarrow\; \psi \in w'$.

Pela hipótese indutiva, $w' \models \psi$.

Como para todo estado $w', \quad (w, w') \in [\![t]\!] \;\Rightarrow\; w' \models \psi$, temos que $w \models [t]\psi$.

($\Leftarrow$)

Suponha que $w \models [t]\psi$.

Afirmação $w \setminus [t] \cup \{\neg\psi\}$ *é inconsistente.*

Prova da afirmação:

Suponha que $w \setminus [t] \cup \{\neg\psi\}$ seja consistente. Então, pelo lema 4.7.1, existe uma extensão maximal consistente w', e como $w \setminus [t] \cup \{\neg\psi\} \subseteq w'$, temos que $(w, w') \in [\![t]\!]$. Além disso, $\neg\psi \in w'$, e pela hipótese indutiva e pelo caso (3) desta prova, $w' \models \neg\psi$, e, portanto, $w \models \neg[t]\psi$, contradizendo a suposição inicial.

Desta afirmação, concluímos que algum subconjunto finito de $w \setminus [t] \cup \{\neg\psi\}$ é inconsistente. Seja este subconjunto $\{\phi_1, \phi_2, ..., \phi_k, \neg\psi\}$ (incluímos $\neg\psi$, pois, mesmo que o conjunto sem $\neg\psi$ já seja inconsistente, certamente continuará sendo inconsistente com $\neg\psi$).

A inconsistência deste subconjunto implica que

$$\vdash \neg(\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_k \wedge \psi)$$

Pelo cálculo proposicional, temos que

$$\vdash \phi_1 \rightarrow (\phi_2 \rightarrow (...(\phi_k \rightarrow \psi)...))$$

Pela regra (N),

$$\vdash [t](\phi_1 \rightarrow (\phi_2 \rightarrow (...(\phi_k \rightarrow \psi)...)))$$

Chamando $(\phi_2 \rightarrow (...(\phi_k \rightarrow \psi)...))$ de ξ, temos

$$\vdash [t](\phi_1 \rightarrow \xi)$$

e, por (K)

$$\vdash [t]\phi_1 \rightarrow [p_l \delta^u q]\xi$$

Mas, como $\phi_1 \in w \setminus [t]$, temos que $[t]\phi_1 \in w$. Logo, por *modus ponens*, $[t]\xi \in w$. Repetindo-se este argumento k vezes, chega-se à conclusão de que $[t]\phi \in w$.

(Caso 6) $\phi = [\,]p$

(Hipótese indutiva: $p \in w \quad \Leftrightarrow \quad S^c, w \models p$)

Como todos os estados do modelo canônico contêm o conjunto $\Theta = \{[\,]x_1, [\,]x_2, ...\}$ e como o estado inicial w_0^c do modelo canônico satisfaz, por construção, $\Theta \setminus [\,] = \{x_1, x_2, ...\}$, temos que, para qualquer estado $w \in W^c$,

$$[\,]x \in w \quad \Leftrightarrow \quad S^c, w_0 \models x \quad \Leftrightarrow \quad S^c, w \models [\,]x$$

□

4.8 Mapeamento DATs → RETOOL

Esta seção mostra como traduzir um diagrama de ação temporizado (DATs – ver seção 3.4) para um conjunto de fórmulas RETOOL que descreve o comportamento do sistema retratado no diagrama.

Dado um diagrama de ação temporizado D, com conjunto de atributos A_d, conjunto de símbolos de ação Γ_d e condições iniciais Θ, produzimos um conjunto de fórmulas RETOOL baseadas nos seguintes atributos e símbolos de ação:

- $A = A_d \cup \{atL_0, atL_1, ..., atL_{n-1}\}$, onde $L_i, 0 \leq i < n$ são os locais do diagrama. Estes atributos da forma atL_i são interpretados como significando que o controle da execução se encontra no local L_i (ver mapeamento DATs → Sistemas de Transição Temporizados nas seções 3.4.2.1 e 3.4.2.4).
- $\Gamma = \Gamma_d \cup \{entry, idle\}$. O novo símbolo de ação *entry* rotula a transição de entrada do sistema (ver seção 3.4.2.1). O símbolo de ação *idle* corresponde à ação nula.

As fórmulas RETOOL geradas para o diagrama D são:

Local do controle :

$$atL_i \rightarrow (\bigwedge_{i \neq j} \neg atL_j)$$

Em cada estado, o controle da execução se encontra em no máximo um local.

Estado inicial :

$$[\,]((\bigwedge_{0 \leq i < n} \neg atL_i) \wedge \Theta)$$

O estado inicial do sistema é aquele que satisfaz as condições iniciais Θ e no qual o controle da execução não se encontra ainda em nenhum local L_i.

Transição de entrada :

$$entry \quad \supset \quad ((\bigwedge_{0 \leq i < n} \neg atL_i) \wedge \Theta) \quad {}_0\delta^{\infty} \quad (atL_0)$$

A transição de entrada tem como condições necessárias as fórmulas que caracterizam o estado inicial. Além disso, a transição de entrada deve conduzir ao local L_0 em algum instante entre 0 e ∞.

Transições *idle* :

$$enabled(idle)$$

$$\phi \leftrightarrow [idle]\phi$$

A ação *idle* está sempre habilitada, e preserva a verdade de todas as fórmulas do estado de onde se origina (i.e., o estado de destino é o mesmo que o de origem).

Transições comuns : Para cada ação g que rotula arestas do diagrama como as mostradas na figura 4.1 a seguinte subsunção é gerada:

$$\begin{aligned} g \supset \quad & ((atL_{i_1} \wedge c_1) \vee (atL_{i_2} \wedge c_2) \vee ... \vee (atL_{i_k} \wedge c_k)) \\ & {}_l\delta^u \\ & (atL_{j_1} \vee atL_{j_2} \vee ... \vee atL_{j_k}) \end{aligned}$$

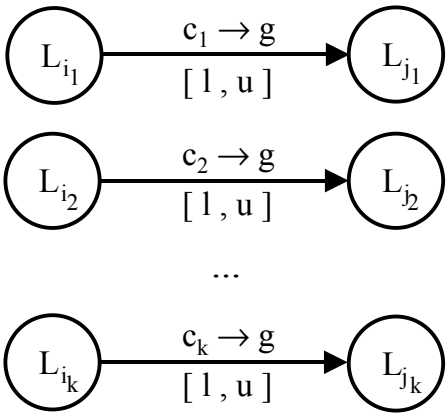

Figura 4.1: Arestas de um diagrama de ação temporizado

Além disso, as seguintes fórmulas são incluídas:

$$\begin{aligned} atL_{i_1} \wedge c_1 &\rightarrow [g] atL_{j_1} \\ atL_{i_2} \wedge c_2 &\rightarrow [g] atL_{j_2} \\ &\ldots \\ atL_{i_k} \wedge c_k &\rightarrow [g] atL_{j_k} \end{aligned}$$

Funcionalidade das ações : a notação utilizada na seção 3.4.1.4 para definir a funcionalidade de uma ação é traduzida para fórmulas RETOOL da forma

$$\begin{aligned} p_1 &\rightarrow [g] q_1 \\ p_2 &\rightarrow [g] q_2 \\ &\ldots \\ p_n &\rightarrow [g] q_n \end{aligned}$$

Dependências entre os atributos : por fim, as fórmulas proposicionais que descrevem as dependências entre os atributos são incluídas (ver seção 3.4.1.5).

4.9 Sobre as Semânticas Alternativas

Como comentado no início deste capítulo, duas outras alternativas foram definidas para a lógica apresentada aqui. Cada uma delas se baseia em um significado diferente para um termo de ação da forma $p_l\delta^u q$.

4.9.1 SIM - Semântica da Implicação Material

Nesta semântica, apresentada em [7], a denotação de um termo construído com o operador δ é a seguinte:

$$\begin{aligned}[\![p_l\delta^u q]\!] = \quad \{(w,w') \mid \exists g \in \Gamma \quad & (\ (w \xrightarrow{g} w') \wedge \\ & (w \models p \ \ \Rightarrow \ \ w' \models q) \wedge \\ & (l \leq l(g) \leq u(g) \leq u) \\ &) \\ \}\end{aligned}$$

A SIM carrega o mesmo tipo de informação que as triplas de Hoare ([12]) da forma $\{p\}g\{q\}$, que significam que, se uma proposição p é satisfeita no estado de origem de um programa g e a execução do programa g terminar, então a proposição q será satisfeita no estado destino.

Como nas triplas de Hoare, não há qualquer garantia sobre execuções de g que se iniciem em estados nos quais p não vale. Isto permite a inclusão, em $[\![p_l\delta^u q]\!]$, de pares de estados (w,w') onde p não vale em w nem q vale em w'.

Para a SIM, p é considerada uma pré-condição relativa das ações contidas em $[\![p_l\delta^u q]\!]$.

Além disso, $[\![p_l\delta^u q]\!]$ pode incluir um subconjunto próprio das transições rotuladas por um certo símbolo de ação g (apenas aquelas onde w e w' satisfazem a implicação

da definição de δ). Isto significa que $[\![p_l \delta^u q]\!]$ representa um conjunto de *execuções* de certas ações, em vez de *todas* as execuções de certas ações, como no caso da nossa semântica.

4.9.2 SCH - Semântica da Condição de Habilitação

A SCH, apresentada em [4], difere da nossa semântica apenas no significado da proposição p. Na SCH, a denotação de um termo construído com o operador δ é

$$
\begin{aligned}
[\![p_l \delta^u q]\!] = \quad \{(w, w') \mid \exists g \in \Gamma \quad [\\
& (w \xrightarrow{g} w') \wedge \\
& \mathit{enable}(g) = [\![p]\!] \wedge \\
& \forall v, v' ((v \xrightarrow{g} v') \Rightarrow (v' \in [\![q]\!])) \wedge \\
& (l \leq l(g) \leq u(g) \leq u) \\
] \\
\}
\end{aligned}
$$

Para a SCH, p é considerada a condição de habilitação das ações contidas em $[\![p_l \delta^u q]\!]$: estas ações estão habilitadas em um estado w se e somente se p for satisfeita em w. É uma definição mais forte, para a qual, no entanto, ainda não foi encontrada uma axiomatização adequada.

Capítulo 5

Conclusões

5.1 Sumário

Nesta dissertação, apresentamos uma breve discussão sobre sistemas computacionais e os modelos formais usados para representar o comportamento destes sistemas. A lógica de ações RETOOL foi definida para raciocinar sobre o modelo formal de sistemas de transição temporizados, com ênfase nas condições necessárias, pós-condições e limites de tempo das ações envolvidas. A correção e a completude fraca de uma axiomatização recursiva de RETOOL foram provadas.

5.2 RETOOL e MTL

Como a semântica do operador δ se baseia em parte na noção de computação (no que diz respeito aos limites de tempo), é natural examinar uma combinação de RETOOL com uma lógica temporal de tempo linear, cujos modelos são justamente computações. Nesta conclusão, descrevemos brevemente a natureza desta combinação, que já foi

estudada para a Semântica da Implicação Material (SIM) de RETOOL em [7] e que pode ser facilmente adaptada para a Semântica da Condição Necessária (SCN) apresentada nesta dissertação.

A lógica temporal escolhida foi MTL (ver [5]), que é dotada de operadores temporais com limites de tempo $\mathbf{X}_{Rc}$ (próximo estado) e $\mathbf{U}_{Rc}$ (até), onde $c \in \text{TIME} \cup \{\infty\}$ e R é uma relação sobre $\text{TIME} \cup \{\infty\}$. Estendemos esta linguagem, acrescentando os termos de ação de RETOOL como proposições primitivas; estipulamos que um termo de ação t é verdadeiro no passo i de uma computação se e somente se a denotação de t contém a ação responsável pela transição tomada no passo i.

Seguem-se alguns exemplos de regras de inferência (tiradas de [7]) que relacionam RETOOL e MTL:

$$\frac{\text{enabled}(t) \rightarrow r \qquad t \supset p_0 \delta^\infty q}{t \rightarrow r \wedge \mathbf{X}_{=0} q}$$

Esta regra afirma que uma ação denotada por t estabelece q no próximo estado sempre que p for verdadeiro no estado atual e que esta ação for responsável pela transição atual. De acordo com a definição de computação, a transição é instantânea.

$$\frac{p \rightarrow \neg\text{enabled}(t) \qquad t \supset \top_x \delta^\infty \top}{p \rightarrow \mathbf{G}_{\leq x} \neg t}$$

Esta regra captura uma propriedade de *safety* para o limite inferior de uma ação: se a verdade de p significa que uma ação t está desabilitada, e t possui limite inferior x, concluímos que, p valendo no estado atual, t não será tomada pelo menos durante as próximas x unidades de tempo (o operador temporal $\mathbf{G}_{\leq x}$ significa "durante as próximas x unidades de tempo").

Outras regras e um exemplo de descrição de um sistema computacional podem ser encontradas em [7], para a semântica SIM. Eles podem ser facilmente adaptados para a semântica SCN adotada nesta dissertação.

5.3 Trabalhos Futuros

Trabalhos futuros envolvendo RETOOL e MTL incluem:

- A definição de um axiomatização adequada para a semântica SCH (aliás, a primeira semântica proposta para RETOOL em [4]);
- Um estudo comparativo da SCN e da SCH em um contexto de especificação e verificação de sistemas reativos e de sistemas orientados a objetos;
- Um aprofundamento do estudo da combinação de RETOOL e MTL descrita na seção anterior.

Bibliografia

[1] Alur, R., Henzinger, T. "Logics and Models of Real Time: A Survey", *Real Time: Theory in Practice*, LNCS 600, 74–106, Springer-Verlag, 1992.

[2] Arnold, A., *Finite Transition Systems*, Prentice Hall, 1992.

[3] Ben-Ari, M., *Principles of Concurrent and Distributed Programming*, Prentice Hall, 1990.

[4] Carvalho, S., Fiadeiro, J. e Haeusler, E.H., "A Formal Approach to Real-Time Object-Oriented Software", in *Proc. 22nd IFAC/IFIP Workshop on Real-Time Programming WRTP'97*, Elsevier 1997.

[5] Chang, E., *Compositional Verification of Reactive and Real-Time Systems*, PhD Thesis, Stanford University, 1995.

[6] Emerson, E.A., "Temporal and Modal Logic", in *Handbook of Theoretical Computer Science*, pp. 995–1072, Elsevier, 1990.

[7] Fiadeiro, J. and Haeusler, E.H., "Bringing It About On Time (Extended Abstract)", in *Proc. I IMLLAI, Fortaleza, CE, Brazil*, 1998.

[8] Francez, N., *Fairness*, Springer, 1986.

[9] Harel, D., "Dynamic Logic", in *Handbook of Philosophical Logic Vol II* (Dov Gabbay, F. Guenthner (eds.)), D. Reidel, 1984.

[10] Henzinger, T.A., *The Temporal Specification and Verification of Real-Time Systems*, Tese de PhD, Dept. of Computer Science, Stanford University 1991.

[11] Henzinger, T.A., Manna, Z., Pnueli, A., "Temporal Proof Methodologies for Timed Transition Systems". *Information and Computation* 112, 273–337, 1994.

[12] Hoare, C.A.R., "An Axiomatic Basis for Computer Programming", in *Comm. ACM* 12, 1967.

[13] Hoare, C.A.R., *Communicating Sequential Processes*, Prentice Hall, 1985.

[14] Keller, R.M., "Formal Verification of Paralell Programs". *Communications of the ACM*, 19(7):371–384, 1976.

[15] Koymans, R., Shyamasundar, R.K., de Roever, W.P., Gerth, R., Arun-Kumar, S., "Compositional Semantics for Real-Time Distributed Computing". *Information and Computation*, 79(3):210–256, 1988.

[16] Pnueli, A., "The Temporal Logic of Programs". *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, 46–57, IEEE Computer Society Press, 1977.

[17] Pnuelli, A., "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: a Survey of Current Trends", in *Current Trends in Concurrency* (G. Goos e J. Hartmanis (eds.)), LNCS 224, Springer-Verlag, 1986.

[18] Segerberg, K., "Bringing It About", in *Journal of Philosophical Logic* 18, 1989.

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