Universidade Federal de Santa Catarina
Curso de Pós-Graduação em Ciência da Computação

Um Simulador para Validação de Sistemas Dependentes de Tempo Descritos em RT-LOTOS.

Dissertação submetida à Universidade Federal de Santa Catarina para a obtenção do grau de

MESTRE EM CIÊNCIA DA COMPUTAÇÃO

Roberto Milton Scheffel

Florianópolis, fevereiro de 1997

Um Simulador para Validação de Sistemas Dependentes de Tempo Descritos em RT-LOTOS.

Roberto Milton Scheffel

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Especialidade Sistemas de Computação, e aprovada em sua forma final pelo Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina

Prof. Murilo Silva de Camargo, Dr.
Orientador, INE, UFSC

Prof. Murilo Silva de Camargo, Dr.
Coordenador do Curso, INE, UFSC

Banca Examinadora:

Prof. Murilo Silva de Camargo, Dr.
Presidente, INE, UFSC

Prof. Daltro José Nunes, Dr.
Instituto de Informática, UFRGS

Prof. Jean-Marie Farines, Dr.
LCMI, UFSC

Prof. Rosvelter J. Coelho da Costa, Dr.
INE, UFSC

Agradecimentos

À todos aqueles que de uma forma ou de outra contribuíram para a elaboração deste trabalho.

Ao prof. Murilo Silva de Camargo, pela oportunidade e pela dedicação na elaboração deste trabalho.

Às secretárias do CPGCC, Verinha e Valdete.

À todos os colegas e professores do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Aos meus pais, Aldino e Adelyra, e aos meus irmãos, pelo apoio dado durante a realização deste trabalho. Em especial à minha irmã Roselí, por tudo que sempre fez por mim, e principalmente pelas palavras de incentivo.

À *diretoria* (Celso, Iwens e Bráulio), pela amizade demonstrada neste período, consolidada pela convivência, entre estudos, festas e dificuldades.

À minha namorada, Osnilde, por ter suportado a distância e pelo amor e compreensão que sempre demonstrou.

À Deus.

Sumário

Lista de Tabelas

Lista de Figuras

Resumo

LOTOS é uma técnica de descrição formal padronizada pela ISO desde 1988, a qual permite expressar a ordenação temporal de eventos, percebidas por um observador externo. Com a combinação de ações elementares com os operadores da linguagem, é possível descrever comportamentos complexos. Sua maior utilização ocorre na descrição de sistemas distribuídos. Existe atualmente um grande número de métodos e técnicas de verificação e validação suportando a técnica de descrição formal LOTOS.

Entretanto, uma grande limitação de LOTOS reside no fato da mesma não oferecer mecanismos de descrição de restrições temporais. Assim, LOTOS não pode ser facilmente utilizada na descrição de sistemas onde o tempo intervém. Este fato levou ao desenvolvimento de várias extensões da linguagem, permitindo representar explicitamente restrições de tempo na evolução do sistema.

Neste trabalho, apresenta-se uma ferramenta de validação de especificações de sistemas tempo real escritas na linguagem de descrição formal RT-LOTOS. Esta linguagem é uma extensão temporizada de LOTOS. A ferramenta permite a simulação do comportamento dos modelos descritos em RT-LOTOS, de modo a validar requisitos de qualidade dos mesmos. A simulação pode ser interativa ou automática. A simulação automática oferece a alternativa de geração de um traço de execução ou a alternativa de explorar todos os traços possíveis, gerando uma árvore de execução. Ao final do trabalho são apresentados estudos de caso, de modo a ilustrar a utilização da ferramenta.

Abstract

LOTOS is a formal description technique standardized by ISO since 1988, which allows to express the temporal ordering of events, as perceived by an external observer. The combination of the elementar actions and the language operators can be used to describe complex behaviours. This formal description technique is mostly used on the distributed systems description. Actually, there are a number of methods and techniques of verification and validation supporting the LOTOS formal description technique.

However, the greatest limitation of LOTOS is its lack of mechanisms for the description of temporal restrictions. Thus, LOTOS cannot be easily used for the description of systems where the time is crucial. This fact led to the development of various extensions to the language, which have mechanisms to explicitly describe temporal restrictions in the evolution of the system.

In this work we present a tool for the validation of real time systems specifications, written in the formal description language RT-LOTOS. This language is a temporal extension of LOTOS. The tool allows the simulation of the behaviour of models described with RT-LOTOS, with the objective of the quality requisites validation. The simulation can be interactive or automatic. The automatic simulation offers the alternative of generating an execution trace, or the alternative of exploring all possible traces, generating an execution tree. At the end of this work, some case studies are presented, to illustrate the use of the tool.

Capítulo 1

Introdução

Nos últimos anos, as Técnicas de Descrição Formal (FDT - *Formal Description Techniques*) têm sido cada vez mais utilizadas e aceitas. Isto se deve ao fato de que cada vez mais procura-se verificar e validar os sistemas antes da sua implementação. Para sistemas de grande complexidade, é ainda mais importante garantir o quanto antes sua correção. E pela facilidade de se modelar paralelismo e comportamentos complexos, as FDTs encontram um vasto campo de aplicação, destacando-se na especificação de sistemas distribuídos.

As FDTs padronizadas pelos organismos internacionais, tais como LOTOS e Estelle (padronizadas pela ISO) e SDL (padronizada pelo UTI) encontraram, nos últimos anos, grande aceitação nos meios acadêmicos e industriais. Várias ferramentas de validação e verificação foram desenvolvidas, permitindo uma análise completa das especificações. Até mesmo ferramentas de tradução automática de especificações para linguagens de programação existem, eliminando o processo de codificação manual de implementações das especificações. Esta tradução automatizada evita a introdução de erros na programação.

Uma das FDTs de maior aceitação tem sido LOTOS, devido ao seu grande poder de expressão, e pela sua boa fundamentação teórica, que permite a verificação formal de propriedades da especificação. LOTOS tem sua parte comportamental baseada em CCS e CSP, e a parte de dados baseada na tipo abstrato de dados ACT ONE. Porém, LOTOS apresenta uma limitação grave: não apresenta operadores ou mecanismos para a descrição de restrições de tempo nas especificações. Isto é, LOTOS fornece os mecanismos apropriados para se expressar a ordenação temporal em que as ações devem ocorrer, mas não há como representar o tempo quantitativamente. Em outras palavras, não há como expressar *quando* as ações devem acontecer ou as restrição de tempo que são impostas a uma ação.

Este trabalho apresenta uma extensão de LOTOS, denominada de RT-LOTOS (Real-Time LOTOS), proposta em [Camargo95]. Esta extensão permite expressar restrições temporais na ocorrência das ações das especificações. Estas restrições permitem expressar o intervalo de tempo no qual as ações pode ocorrer.

Porém, para uma completa verificação e validação de um sistema dependente de tempo, são necessárias ferramentas automatizadas. Estas ferramentas devem permitir a

verificação de propriedades sobre as especificações, bem como acompanhar o desenvolvimento do comportamento das especificações, de modo a garantir que o mesmo seja condizente com os requisitos de qualidade do sistema sendo especificado.

Neste trabalho apresenta-se uma ferramenta de simulação para especificações de sistemas dependentes de tempo, escritas em RT-LOTOS. Esta ferramenta apresenta uma série de funcionalidades para a simulação interativa e automática de especificações RT-LOTOS. Permite o acompanhamento do desenvolvimento da especificação sendo simulada. Também é possível gerar uma árvore de execução para uma especificação. Permite ainda a geração de diversos traços de execução da especificação, com diversas opções de disparo das ações nos intervalos de tempo a elas associadas. Ao final do trabalho são feitos alguns estudos de caso.

Esta dissertação é organizada da seguinte forma:

No capítulo 2 são apresentadas as características de sistemas dependentes de tempo, e as maneiras de representá-los. Em seguida é apresentada uma descrição da linguagem RT-LOTOS, utilizada para a especificação formal de sistemas dependentes de tempo, bem como métodos de validação e verificação das especificações.

No capítulo 3 é discutida a simulação de sistemas dependentes de tempo descritos em RT-LOTOS, com ênfase nos aspectos utilizados no desenvolvimento da ferramenta apresentada neste trabalho. Ainda neste capítulo é apresentada a arquitetura do simulador, com uma discussão dos algoritmos utilizados.

No capítulo 4 as funcionalidades da ferramenta são apresentadas, descrevendo-as e mostrando como utilizá-las na validação de requisitos de qualidade dos sistemas especificados em RT-LOTOS.

Na continuidade, no capítulo 5, são apresentados alguns estudos de caso, visando testar e ilustrar a utilização da ferramenta como método de validação.

Por fim, são apresentadas as conclusões sobre este trabalho, mostrando perspectivas futuras de desenvolvimento desta e de outras ferramentas.

Capítulo 2

Sistemas Dependentes de Tempo - Especificação, Verificação e Validação.

A complexidade dos sistemas computacionais modernos é muito grande. Esta complexidade de descrição de sistemas é ainda maior quando o problema envolve aspectos de tempo. Assim, a utilização de métodos formais na especificação e verificação de sistemas dependentes de tempo é uma alternativa das mais promissoras.

Neste capítulo serão abordadas as características dos sistemas dependentes de tempo, suas definições básicas e as principais classes de sistemas informáticos tempo real. Também serão apresentados os métodos formais de especificações destes sistemas. Um destes métodos, a álgebra de processos RT-LOTOS é mostrada em detalhes, com uma discussão de sua sintaxe e semântica. Em seguida serão descritas algumas abordagens de verificação e validação das especificações dependentes de tempo, com ênfase na simulação.

2.1 Sistemas Dependentes de Tempo

Sistemas dependentes de tempo, ou sistemas tempo real, são sistemas nos quais o tempo tem uma influência fundamental sobre seu comportamento [Camargo95]. Ou seja, além de terem que oferecer certos serviços, para que sejam considerados corretos estes sistemas devem oferecer os serviços dentro de intervalos de tempo determinados. Pode-se dizer assim que o tempo é um recurso a ser consumido pelo sistema.

Os sistemas dependentes de tempo incluem três grandes classes: sistemas tempo real, que reagem a estímulos do ambiente, protocolos de comunicação, que tratam da comunicação e sincronização de dispositivos, e aplicações multimídia, que tratam da recuperação, transmissão, sincronização e apresentação de fluxos de diferentes mídias.

Os *sistemas tempo real* apresentam suas restrições temporais na interação com seu ambiente. Devem reagir a estímulos do ambiente, incluindo a própria passagem do tempo. Apresentam características que permitem diferenciá-los e classificá-los, segundo [Audsley90] *et alli*, tais como:

- Correção Temporal (*Timeliness*): os sistemas reagem a estímulos do ambiente, dentro de intervalos de tempo especificados. Por exemplo, um sistema de piloto

automático de aeronave, ao detectar um obstáculo a frente, não deve somente alterar o curso para evitar a colisão, mas fazê-lo num tempo curto o suficiente.

- Relação de Ordem (*Orderliness*): a ordem de entrada dos dados deve ser respeitada nos dados de saída do sistema.
- Atualidade (*Freshness*): o sistema deve utilizar ou fornecer o dado mais atual possível, do ponto de vista temporal. Assim, por exemplo, não basta utilizar a última leitura de um sensor, mas sim garantir o tempo decorrente entre a leitura e a utilização do dado seja o menor possível. Isto pode implicar até mesmo no fato do sistema pedir uma nova leitura.
- Previsibilidade (*Temporal Predictability*): o comportamento funcional e temporal deve estar de acordo com sua especificação. Ou seja, deve ser possível saber de antemão qual será o comportamento do sistema, e o momento em que o mesmo vai ocorrer.
- Coordenação (*Coordination*): as ações do sistema no seu ambiente devem ser coordenadas, de modo que uma não prejudique a outra.

Além disso, os sistemas tempo real podem ainda ser classificados em dois grandes grupos:

- Sistemas Tempo Real Não-Críticos (*Soft Real-Time Systems*): são sistemas tempo real que não apresentam falhas críticas. Ou seja, caso uma ação não ocorra dentro do seu tempo pré-determinado, a falha resultante não é catastrófica. O custo da falha é da mesma ordem de grandeza que os benefícios da operação normal.
- Sistemas Tempo Real Críticos (*Hard Real-Time Systems*): São sistemas tempo real que apresentam conseqüências catastróficas em caso de falha. Os custos de uma falha podem exceder em várias ordens de magnitude os custos da operação normal. Por exemplo, o mal funcionamento de um sistema de controle de uma usina nuclear pode causar acidentes cujas conseqüências são muito maiores que o benefício da operação normal da usina. Os sistemas tempo real críticos apresentam duas situações assim caracterizadas:
  - Seguro em caso de falha (*Fail Safe*): um ou vários estados seguros para o comportamento do sistema podem ser alcançados em caso de falha do sistema.
  - Operacional em caso de falha (*Fail Operational*): quando não existe nenhum estado seguro identificável, mas o sistema computacional pode fornecer um serviço mínimo em caso de falha, evitando uma catástrofe.

Os *Protocolos de Comunicação* são normas e regras estabelecidas para a padronização da comunicação entre dispositivos. As principais características gerais dos protocolos de comunicação são as seguintes [BBBC93]:

- Concorrência: protocolos de redes de computadores são sistemas essencialmente paralelos, pois os nós de comunicação operam de modo autônomo, em paralelo.
- Interação: os protocolos de comunicação expressam uma interação entre os nós da rede. Esta interação pode se dar de diversas formas, tais como: memória compartilhada, comunicação síncrona, comunicação assíncrona, comunicação por difusão e outras.
- Imprevisibilidade (*unpredictability*): a interação entre os nós não é previsível, ou seja, mensagens podem ser perdidas, bem como não se pode garantir o tempo de transmissão de uma mensagem entre dois nós distantes.

Além destas características, outros mecanismos temporais devem ser considerados na descrição de protocolos de comunicação. Entre outros, pode-se citar erros por atraso (*timeout*), tempos máximos e mínimos de transmissão, duração de conexões, etc.

As *aplicações multimídia* são aplicações que geram, processam, armazenam, recuperam e apresentam informações de vários tipos, denominadas mídia. Estas informações podem ser, por exemplo, textos e arquivos (mídia estática) ou áudio e vídeo (mídia contínua). Uma discussão mais detalhada dos aspectos temporais de aplicações multimídia podem ser encontradas em [Camargo95].

Os aspectos temporais mais importantes nos sistemas multimídia, entre outros, são:

- Os atrasos de transmissão fim-a-fim de um fluxo de mídia. Variações nestes atrasos são denominados *jitter*, sendo que o mesmo pode ter um intervalo de tolerância.
- Sincronização intra-fluxo, na qual se expressa o intervalos de tempos mínimo e máximo entre a chegada de duas unidades de informação (pacotes) de um mesmo fluxo de mídia.
- Sincronização inter-fluxos, na qual se expressa uma relação entre os tempos de chegada de pacotes de dois ou mais fluxos de mídia relacionados. Por exemplo, a sincronização de um fluxo de vídeo com o fluxo de som associado.

2.2 Técnicas de Descrição Formal de Sistemas Dependentes de Tempo.

A necessidade de se expressar com exatidão as características dos sistemas dependentes de tempo levou ao desenvolvimento de diversas abordagens de tratamento. Entre as principais, pode-se citar:

- Redes de Petri Temporizadas
- Lógicas Temporais Tempo Real
- Álgebras de Processos Temporizadas

2.2.1 Redes de Petri Temporizadas

A teoria de Redes de Petri [Brams83] foi um dos primeiros formalismos introduzidos para tratar concorrência, não-determinismo e conexões causais entre eventos. É um formalismo gráfico e matemático que se adapta bem a um grande número de aplicações, onde as noções de evento e evoluções simultâneas são importantes.

As redes de Petri têm sua estrutura representada por lugares e transições, sendo que os lugares representam atividades do sistema, e as transições, ligadas a lugares de entrada e saída, representam a ocorrência de eventos. A evolução das redes de Petri é regida pelas fichas (*tokens* ou *jettons*) presentes nos lugares, que representam estados do sistema. A ocorrência de eventos no sistema se dá através do disparo das transições, com a retirada de fichas dos lugares de entrada e seu depósito nos lugares de saída.

As extensões temporais das redes de Petri apresentam variações quanto os elementos da rede aos quais o tempo é associado. Na literatura encontramos três formas básicas: Redes de Petri com tempo associado aos arcos (*Timed Arc Petri Nets*) [Walter83], Redes de Petri com tempo associado às transições (*Timed Transition Petri Nets*) [MeFa76] e, finalmente, Redes de Petri com tempo associado aos lugares (*Timed Place Petri Nets*) [CoRo83].

Porém, a abordagem por Redes de Petri apresenta restrições, tais como não apresentar um mecanismo de representação explícita de dados. Também são criticadas por sua fraca modularização, não oferecendo uma mecanismo nativo de composição. Requerem ainda um grande grau de abstração, para interpretar o que uma certa marcação representa no sistema por ela modelado.

2.2.2 Lógicas Temporais Tempo Real

Lógicas temporais são uma classe de lógicas modais, na qual os operadores modais são interpretados de uma maneira temporal [AlHe92]. São compostas de operadores lógicos proposicionais que, combinados com um conjunto de operadores temporais, permitem que as seguintes propriedades sejam satisfeitas:

- Invariância: propriedades que são sempre verdadeiras em um programa.
- Eventualidade: propriedades que devem tornar-se verdadeiras em algum tempo futuro no sistema.
- Precedências: asserções que estabelecem que um certo evento deve ocorrer antes de um outro.

Já as lógicas temporais tempo real estendem as lógicas temporais, incorporando possibilidades e tratamento de propriedades quantitativas, tais como periodicidade, esgotamento de tempo (*deadlines*) e atrasos (*delays*). Algumas das lógicas temporais mais conhecidas na literatura são: *Temporal Computation Tree Logic* (TCTL) [AlHe92]; *Real Time Temporal Logic* (RTTL) [Ostroff89]; e *Metric Temporal Logic* (MTL) [Koymans90].

2.2.3 Álgebras de Processos Temporizadas

As álgebras de processos têm sido desenvolvidas dentro do campo da teoria da concorrência. Seu início se deu com os trabalhos de Hoare, com seu *Communicating Sequential Processes* - CSP [Hoare85], e de Milner, com seu *Calculus of Communicating Systems* - CCS [Milner80].

As álgebras de processos conseguem representar, com poucos operadores, toda a complexidade de sistemas concorrentes ou de sistemas distribuídos. Os principais operadores existentes nas álgebras de processos são os de comunicação síncrona, escolha, não determinismo, concorrência, restrição e renomeação. A maioria das álgebras de processos implementa estes operadores, porém encontra-se diferenças na utilização e na semântica dos mesmos.

Dentre as álgebras de processos existentes, a que mais se sobressai nos últimos tempos é LOTOS, padronizada pela ISO (*International Organization for Standardization*) desde 1988 [ISO88]. A parte comportamental de LOTOS é baseada em CCS, com a comunicação síncrona com *rendezvous* múltiplo de CSP. Já a parte de descrição de dados é baseada na representação de dados algébricos ACT ONE.

Mas as álgebras de processos possuem limitações para a representação de tempo [BBBC93]. Não é possível representar o tempo preciso em que um evento deve ocorrer, nem é possível forçar a ocorrência de certos eventos. Para introduzir o tratamento de tempo nas álgebras de processos, duas são as alternativas: atribuir aos eventos um intervalo de tempo no qual os mesmos podem ocorrer; ou incluir um operador que represente explicitamente a passagem de uma ou mais unidades de tempo.

Várias são as extensões temporais das álgebras de processos existentes. Para fins de referência, pode-se citar algumas extensões de CCS, tais como [MoTo89] e [Regan90]. Já LOTOS foi alvo de um maior número de propostas para extensões temporais, entre as quais cita-se [Azcorra90] [Regan93] [LeLé94] [CoOl95] [Camargo95]. A evolução destas propostas fez com que um novo padrão de LOTOS fosse estudado, agora com suporte a aspectos temporais da especificação. Estes estudos deverão resultar num novo padrão denominado E-LOTOS (Enhancement to LOTOS) [ISO97], o qual já se encontra em estado de *Draft Proposal* pela ISO.

Na espera do novo padrão de LOTOS, neste trabalho será utilizada a extensão apresentada em [Camargo95], denominada RT-LOTOS, que é apresentada na seção seguinte.

2.3 A Técnica de Descrição Formal RT-LOTOS

RT-LOTOS é uma extensão da TDF padrão LOTOS, e apresenta mecanismos para a representação de restrições de tempo na ocorrência das ações da especificação[1]. Assim, além do conjunto de ações, é definido um domínio de tempo para a especificação. Tal domínio pode ser denso ou esparso, a única restrição é que seja enumerável. Esta restrição é necessária para que o modelo subjacente seja um sistema de transições rotulado.

Mais formalmente, o domínio de tempo $\mathcal{D}$ deve ser um monóide comutativo[2] $(\mathcal{D}, +, 0)$ que satisfaz as seguintes condições:

1. $d + d' = d \Leftrightarrow d' = 0$
2. a relação $\leq \subseteq \mathcal{D}\times\mathcal{D}$ definida por $d \leq d' \Leftrightarrow \exists d'' \in \mathcal{D}$ tal que $d + d'' = d'$ é uma ordem parcial.
3. a relação $< \subseteq \mathcal{D}\times\mathcal{D}$ definida por $d < d' \Leftrightarrow \exists d'' \in \mathcal{D}$ tal que $d + d'' = d'$, $d'' \neq 0$ é uma ordem parcial.

Além disso, as seguintes propriedades são válidas para o domínio de tempo $\mathcal{D}$:

- $\forall r \in \mathcal{D}, 0 \leq r$ (0 é o menor elemento de $\mathcal{D}$)
- $\forall d, d' \in \mathcal{D}$, se $d \leq d'$ então o elemento $d'' \in \mathcal{D}$ tal que $d + d'' = d'$ é único e é denotado por $d' - d$.

Denota-se por $\mathcal{D}^\infty$ o conjunto $\mathcal{D} \cup \{\infty\}$, onde $\infty \notin \mathcal{D}$ é um elemento distinguido, tal que $\forall d \in \mathcal{D}, d < \infty$ e $\infty + d = \infty$.

Uma discussão mais detalhada sobre o domínio de tempo em RT-LOTOS, tal como a definição de domínios densos e esparsos, pode ser encontrada em [Camargo95].

---

[1] Numa especificação, um sistema é visto como um processo, com a possibilidade deste ser constituído de vários sub-processos. Um processo é uma entidade capaz de realizar ações internas e não-observáveis, e de interagir com outros processos, através de ações externamente observáveis. Uma interação entre processos é denominada de *evento*, sendo que os eventos são atômicos, ou seja, ocorrem de forma instantânea, sem consumir tempo, num ponto de interação denominado *porta*. Neste trabalho comete-se um abuso de linguagem identificando com o termo *ação* tanto a porta quanto o evento associado a esta. Isto não traz maiores problemas, visto que os contextos são bem distintos, não havendo risco de confusão.

[2] Monóide: conjunto munido de uma operação binária associativa e de um elemento neutro. Além disso, se $(A, \bullet, 1)$ é um monóide e $a\bullet b = b\bullet a$ para qualquer $a, b \in A$, então diz-se que $(A, \bullet, 1)$ é um monóide comutativo. Por exemplo, o conjunto dos números naturais munido da adição é um monóide comutativo.

As ações de RT-LOTOS podem ser agrupadas em dois grandes conjuntos:

- as ações clássicas de LOTOS, que incluem as ações observáveis pertencentes ao conjunto $\mathcal{Act}$, a ação interna `i` e a ação de término com sucesso δ. $\mathcal{Act}^i$ denota o conjunto $\mathcal{Act} \cup \{i\}$, e $\mathcal{Act}^\delta$ denota o conjunto $\mathcal{Act} \cup \{\delta\}$.
- as ações específicas de RT-LOTOS, que são as *violações temporais* `a*`, pertencentes ao conjunto $\mathcal{Act}^*$. É suposto que existe uma bijeção entre $\mathcal{Act}$ e $\mathcal{Act}^*$.

Os intervalos de tempo associados às ações são da forma $[t_{min}, t_{max}]$. A semântica das ações temporizadas faz com que uma ação com um intervalo de tempo associado não seja oferecida antes do limite inferior deste intervalo. Uma vez alcançado o limite inferior, a ação é oferecida ao ambiente para a sincronização. Caso esta não ocorra até o limite superior do intervalo, ocorre uma ação semântica de *violação temporal*. Esta violação temporal pode ser tratada por um novo operador, denominado de *preempção temporal*.

Uma descrição detalhada da sintaxe e da semântica de RT-LOTOS, pode ser encontrada em [Camargo95]. O objetivo desta seção é dar somente as noções básicas da linguagem para um melhor entendimento do restante do trabalho.

Neste trabalho será utilizado somente RT-LOTOS básico, ou seja, RT-LOTOS sem a parte de manipulação de dados. Portanto, a discussão apresentada a seguir sobre a sintaxe e a semântica de RT-LOTOS omitirá os aspectos relativos aos dados.

2.3.1 Sintaxe da Linguagem

As expressões de comportamento das especificações RT-LOTOS são gerados por uma sintaxe que é uma extensão direta da linguagem LOTOS e é apresentada a seguir:

E ::=	stop	⇒ inação
|	exit	⇒ término com sucesso
|	$[t_{min}, t_{max}]$a; E	⇒ prefixação - ação observável
|	$[t_{min}, t_{max}]$i; E	⇒ prefixação - ação interna
|	E [] E	⇒ escolha
|	hide L in E	⇒ ocultação
|	E |[L]| E	⇒ composição paralela
|	E >> E	⇒ composição seqüencial
|	E [> E	⇒ preempção
|	E <L> $\{a_1:Q_1, \ldots, a_n:Q_n\}$	⇒ preempção temporal
|	P$[a_1, a_2, \ldots, a_n]$	⇒ instanciação de processo

A semântica dos operadores de RT-LOTOS pode ser descrita informalmente como segue:

`stop`: o operador de inação representa um processo que não realiza nenhuma ação, completamente inativo, apenas deixa o tempo progredir.

`exit`: o operador de término com sucesso representa um processo que no qual a ação semântica δ representa o fim bem sucedido do mesmo.

$[t_{min}, t_{max}]$ `a;E`: representa o processo que pode evoluir pela ação `a`, dentro do intervalo de tempo $[t_{min}, t_{max}]$. Antes de t_{min} a ação `a` não é oferecida ao ambiente para a sincronização. Depois de t_{min} e antes de t_{max} o processo pode passar a ter o comportamento de `E` pela ocorrência da ação `a`. No instante t_{max}, se a ação `a` não ocorrer, acontece o disparo de uma ação semântica `a*`, que representa a violação temporal do intervalo da ação `a`. Esta ação de violação temporal faz com que o processo evolua para `stop`. No caso da ação interna `i`, não é definida a violação temporal. Assim, no instante t_{max}, a ação `i` torna-se urgente e incontrolável, sendo a sua ocorrência obrigatória. Caso não seja especificado, o intervalo *default* para ações observáveis é $[0,\infty]$ e para a ação interna `i` é $[0,0]$.

`E [] F`: representa a escolha entre duas expressões de comportamento. Esta escolha é decida pela ocorrência da primeira ação em um dos lados da escolha. Porém, a ocorrência de uma violação temporal não decide a escolha para evitar o chamado indeterminismo temporal.

`hide L in E`: representa a ocultação do conjunto de ações `L` na expressão de comportamento `E`. As ações pertencentes a `L` tornam-se invisíveis e urgentes. Porém, ao contrário da ação especial `i`, que deve ocorrer até o limite superior do seu intervalo, as ações que tornam-se invisíveis pelo operador de ocultação devem ocorrer no primeiro instante em que são oferecidas. Esta propriedade é chamada de *propriedade de progressão máxima.*

`E |[L]| F`: representa o processo que é a composição paralela dos processos `E` e `F`. Os processos podem evoluir independentemente para as ações que não estão no conjunto de sincronização `L`. Para as ações pertencentes a `L`, as mesmas são oferecidas ao ambiente no intervalo resultante da interseção dos intervalos locais em cada processo. Se esta interseção for vazia, a sincronização não será possível, ocorrendo a violação temporal em ambos os lados da composição paralela. Se a interseção não for vazia, a ação é oferecida ao ambiente. Porém, se não houver a sincronização, haverá também a violação temporal local nos processos em composição paralela. Esta violação temporal ocorre isoladamente em cada lado do operador.

`E >> F`: representa a composição seqüencial dos processo `E` e `F`. A composição se comporta como `E` até o seu término com sucesso, passando a se comportar como `F`.

`E [> F`: representa o processo que se comporta como `E`, porém pode ser interrompido por `F`, no momento da ocorrência de sua primeira ação, e passa a se

comportar como F. Porém, a ocorrência de uma violação temporal em F não interrompe E. Se E terminar com sucesso, todo o processo termina.

E <L] {a_1:Q_1, . . . , a_n:Q_n} : representa o processo que se comporta como E, porém fazendo o tratamento das violações temporais que por acaso ocorram em E. Caso ocorra a violação temporal de uma ação de L (a_1, . . . , a_n), a expressão de comportamento correspondente assume o lugar de E. Assim, se ocorrer a violação temporal a_i* em E, o processo passa a comportar-se como Q_i.

P[a_1, . . . , a_n] : representa a instanciação do processo P, com a troca das ações da definição do processo pelas ações que aparecem na instanciação.

2.3.2 Semântica Operacional de RT-LOTOS

A tabela 2.1 apresenta a semântica operacional de RT-LOTOS, no estilo SOS (*Structured Operational Semantics*) de Plotkin, e inclui regras de inferência para as ações clássicas, para as violações temporais e para a passagem do tempo.

2.3.3 Comentários sobre a semântica de RT-LOTOS

Como dito anteriormente, existe uma diferença entre a urgência da ação interna i e das ações ocultadas pelo operador hide. A ação interna i tem a garantia de ocorrer no seu intervalo, tornando-se urgente e incontrolável no limite superior do seu intervalo. Já as ações ocultadas, mesmo vistas pelo ambiente como ações internas normais, têm a *propriedade de progressão máxima*, ou seja, devem ocorrer no primeiro momento do seu intervalo.

Desta forma, a expressão de comportamento

```
[5,10]i;P
```

e a expressão de comportamento

```
hide [a] in [5,10]a;P
```

apresentam diferenças semânticas relevantes. Ambas são vistas, externamente, como uma ação interna seguida do comportamento de P. Porém, no aspecto temporal, têm comportamentos diferentes. No primeiro exemplo a ação i irá ocorrer no intervalo [5,10], tornando-se urgente e incontrolável no instante 10. Já no segundo exemplo a ação a, pela propriedade da progressão máxima, irá ocorrer no instante 5, sendo vista externamente como a ação i. A diferença semântica está na passagem do tempo, e não na visão externa das ações.

Inação	$\dfrac{-}{stop \xrightarrow{t} stop}$
Término com Sucesso	$\dfrac{-}{exit \xrightarrow{\delta} stop}$ $\qquad$ $\dfrac{-}{exit \xrightarrow{t} stop}$
Prefixação	$\dfrac{-}{[0,t]a;E \xrightarrow{a} E}$ $\qquad$ $\dfrac{-}{[0,0]a;E \xrightarrow{a^*} stop}(a \neq i)$ $\dfrac{-}{[0,t+s]a;E \xrightarrow{s} [0,t]a;E}$ $\qquad$ $\dfrac{-}{[t_1+s,t_2+s]a;E \xrightarrow{s} [t_1,t_2]a;E}$
Escolha	$\dfrac{E \xrightarrow{a} E'}{E[]F \xrightarrow{a} E' \quad F[]E \xrightarrow{a} E'}$ $\qquad$ $\dfrac{E \xrightarrow{a^*} E'}{E[]F \xrightarrow{a^*} E'[]F \quad F[]E \xrightarrow{a^*} F[]E'}$ $\dfrac{E \xrightarrow{t} E' \quad F \xrightarrow{t} F'}{E[]F \xrightarrow{t} E'[]F'}$
Ocultação	$\dfrac{E \xrightarrow{a} E' \quad (a \notin L)}{hide\ L\ in\ E \xrightarrow{a} hide\ L\ in\ E'}$ $\qquad$ $\dfrac{E \xrightarrow{a} E' \quad (a \in L)}{hide\ L\ in\ E \xrightarrow{i} hide\ L\ in\ E'}$ $\dfrac{E \xrightarrow{a^*} E' \quad (a \notin L)}{hide\ L\ in\ E \xrightarrow{a^*} hide\ L\ in\ E'}$ $\qquad$ $\dfrac{E \xrightarrow{a^*} E' \quad (a \in L)}{hide\ L\ in\ E \xrightarrow{i} hide\ L\ in\ E'}$ $\dfrac{E \xrightarrow{t} E' \quad (\forall a \in L \neg(E \xrightarrow{a}))}{hide\ L\ in\ E \xrightarrow{t} hide\ L\ in\ E'}$
Composição Paralela	$\dfrac{E \xrightarrow{a} E' \quad (a \notin L \cup \{\delta\})}{E\|[L]\|F \xrightarrow{a} E'\|[L]\|F \quad F\|[L]\|E \xrightarrow{a} F\|[L]\|E'}$ $\dfrac{E \xrightarrow{a^*} E'}{E\|[L]\|F \xrightarrow{a^*} E'\|[L]\|F \quad F\|[L]\|E \xrightarrow{a^*} F\|[L]\|E'}$ $\dfrac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F' \quad (a \in L \cup \{\delta\})}{E\|[L]\|F \xrightarrow{a} E'\|[L]\|F'}$ $\qquad$ $\dfrac{E \xrightarrow{t} E' \quad F \xrightarrow{t} F'}{E\|[L]\|F \xrightarrow{t} E'\|[L]\|F'}$
Composição Seqüencial	$\dfrac{E \xrightarrow{a} E' \quad (a \neq \delta)}{E >> F \xrightarrow{a} E' >> F}$ $\qquad$ $\dfrac{E \xrightarrow{\delta} E'}{E >> F \xrightarrow{i} F}$ $\dfrac{E \xrightarrow{a^*} E'}{E >> F \xrightarrow{a^*} E' >> F}$ $\qquad$ $\dfrac{E \xrightarrow{t} E' \quad \neg(E \xrightarrow{\delta})}{E >> F \xrightarrow{t} E' >> F}$
Preempção	$\dfrac{E \xrightarrow{a} E' \quad (a \neq \delta)}{E[> F \xrightarrow{a} E'[> F}$ $\qquad$ $\dfrac{F \xrightarrow{a} F'}{E[> F \xrightarrow{a} F'}$ $\dfrac{E \xrightarrow{\delta} E'}{E[> F \xrightarrow{\delta} E'}$ $\qquad$ $\dfrac{E \xrightarrow{a^*} E'}{E[> F \xrightarrow{a^*} E'[> F}$ $\dfrac{F \xrightarrow{a^*} F'}{E[> F \xrightarrow{a^*} E[> F'}$ $\qquad$ $\dfrac{E \xrightarrow{t} E' \quad F \xrightarrow{t} F'}{E[> F \xrightarrow{t} E'[> F'}$
Preempção Temporal	$\dfrac{E \xrightarrow{a} E' \quad (a \neq \delta)}{E < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\} \xrightarrow{a} E' < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\}}$ $\qquad$ $\dfrac{E \xrightarrow{\delta} E'}{E < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\} \xrightarrow{\delta} E'}$ $\dfrac{E \xrightarrow{a^*} E' \quad (a \notin L)}{E < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\} \xrightarrow{a^*} E' < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\}}$ $\qquad$ $\dfrac{E \xrightarrow{a_j^*} E' \quad (a_j \in L)}{E < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\} \xrightarrow{i} Q_j}$ $\dfrac{E \xrightarrow{t} E'}{E < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\} \xrightarrow{t} E' < L]\{a_1{:}Q_1,\ldots,a_n{:}Q_n\}}$
Instanciação	$\dfrac{E[a_1/g_1,\ldots,a_n/g_n] \xrightarrow{a} E', \quad P[g_1,\ldots,g_n] := E}{P[a_1,\ldots,a_n] \xrightarrow{a} E'}$ $\dfrac{E[a_1/g_1,\ldots,a_n/g_n] \xrightarrow{a^*} E', \quad P[g_1,\ldots,g_n] := E}{P[a_1,\ldots,a_n] \xrightarrow{a^*} E'}$ $\dfrac{E[a_1/g_1,\ldots,a_n/g_n] \xrightarrow{t} E', \quad P[g_1,\ldots,g_n] := E}{P[a_1,\ldots,a_n] \xrightarrow{t} E'}$

Tabela 2.1 - Semântica Formal de RT-LOTOS

Também o uso do operador de preempção temporal necessita de alguns cuidados, mais especificamente na instanciação recursiva no lado direito do operador. As regras semânticas fazem com que a recursão "acumule" o operador de preempção temporal na expressão de comportamento subseqüente. Um exemplo deste comportamento é o seguinte processo:

```
process P[a,b] :=
      [0,15]a; [20,30]b; P[a,b]<a,b>{a:Q[a], b:Q[b]}
endproc
```

Após a primeira recursão, o processo acima fica:

```
process P[a,b] :=
      [0,15]a; [20,30]b; P[a,b] <a,b>{a:Q[a], b:Q[b]} <a,b>{a:Q[a],
      b:Q[b]}
endproc
```

Ocorrendo agora a violação temporal da ação `a`, o primeiro operador de preempção temporal irá tratá-la, mas o segundo operador ficará ainda ativo. Desta forma, se houver a violação temporal de a no processo `Q`, a mesma será tratada pelo segundo operador de preempção temporal. Este comportamento se repetirá para quantas recursões houverem antes da primeira violação temporal.

Este comportamento pode ser evitado, no exemplo acima, utilizando-se a seguinte construção:

```
process P[a,b] :=
      P1[a,b]<a,b>{a:Q[a], b:Q[b]}
where process P1[a,b] :=
            [0,15]a; [20,30]b; P1[a,b]
      endproc
endproc
```

Neste novo processo não há mais o problema da acumulação do operador de preempção temporal, e o comportamento recursivo do lado esquerdo do operador de preempção temporal foi mantido.

2.4 Validação e Verificação de Sistemas Dependentes de Tempo

Quanto se fala de verificação e validação de especificações formais, algumas divergências são encontradas na literatura, principalmente quando os dois termos são confrontados.

Para [Ernb91] *et alli*, *validação* é um termo genérico que envolve todo o processo de detecção e correção de erros de uma dada especificação ou implementação. Neste processo há três atividades principais: *verificação*, *simulação* e *teste*. A *verificação* visa provar formalmente que uma especificação satisfaz certas propriedades desejáveis. A *simulação* é utilizada para gerar e observar um modelo executável. Já a atividade de *teste* serve para

avaliar resultados gerados por um protótipo observado. Os resultados dos testes são usados na correção das especificações.

Já para [Kirkwood94], *verificação* é uma prova formal das propriedades da especificação, obtida pela manipulação de axiomas. Por exemplo, utilizando a definição de relações de equivalência para provar a existência ou não de uma bissimulação entre duas especificações. A *validação* é uma demonstração convincente, geralmente obtida por experimentos, de que uma dada especificação está correta. Tais experimentos podem incluir simulação ou testes, por exemplo.

Na continuidade deste trabalho, o termo *verificação* será utilizado para designar uma prova formal das propriedades de uma especificação através de um método (manipulação de axiomas). Já o termo *validação* denotará uma análise (possivelmente não exaustiva) do comportamento de uma especificação por experimentos. Esta abordagem segue a linha de [Kirkwood94].

Para a verificação de sistemas dependentes de tempo, um método que pode ser utilizado é o da verificação de modelos (*Model Checking*) [AlCo90], a partir do qual procura-se verificar se o modelo do comportamento do sistema satisfaz um conjunto de propriedades estabelecidas. Para isto, o comportamento do sistema é representado a partir de um formalismo de descrição de seus estados, como grafos temporizados, sistema de transições rotulado, entre outros. As propriedades são descritas a partir de fórmulas de uma lógica temporal tempo real. Tal verificação pode se dar por uma abordagem numérica ou por uma abordagem simbólica. A abordagem numérica avalia a fórmula, utilizando um modelo do grafo temporizado, através da enumeração exaustiva de todos os estados do modelo. Já a abordagem simbólica avalia a fórmula diretamente sobre o grafo temporizado, sem a necessidade de construção de um modelo.

Para as especificações de sistemas utilizando RT-LOTOS, uma metodologia de verificação foi desenvolvida e apresentada em [Martins96]. Esta metodologia utiliza uma técnica baseada na verificação de modelos (*model checking*). Para verificação de propriedades de uma especificação em RT-LOTOS, são escritas fórmulas de uma Lógica Temporal Tempo Real chamada TCTL [AlHe92] para representar essas propriedades. Em seguida é feita uma tradução das especificações RT-LOTOS para um formalismo denominado *Grafo Temporizado* que é o modelo a ser verificado. Este formalismo é um grafo estendido, com relógios e condições lógicas associados aos estados, que permitem uma redução da complexidade do modelo. A tradução é feita pela ferramenta apresentada em [Martins96]. Em seguida, uma ferramenta auxiliar, denominada KRONOS [OlYo93], é utilizada sobre o grafo temporizado, para verificar a satisfação ou não das fórmulas TCTL que representam as propriedades especificadas para o sistema.

Esta metodologia restringe as expressões de comportamento RT-LOTOS sobre as quais se pode realizar verificações. São proibidas duas classes de especificações RT-LOTOS:

- especificações que representem um comportamento infinito;
- especificações que contenham expressões não guardadas.

Diz-se que uma especificação representa um comportamento infinito quando o comportamento da mesma não pode ser traduzido num grafo temporizado finito. Um exemplo de tal situação é o processo

```
P = Q ||| i; P
```

Este processo é uma expressão RT-LOTOS válida, porém não tem uma representação em grafo temporizado finito.

Já as expressões guardadas dizem respeito às expressões de comportamento nas quais a substituição recursiva das instanciações leva a uma expressão de comportamento onde todas as instanciações são precedidas de no mínimo uma ação. Um exemplo de expressão não guardada é o processo

```
P = P [] a; Q
```

Neste processo, a instanciação recursiva de P nunca levará a uma expressão de comportamento onde todas as instanciações são precedidas de uma ação.

Além das técnicas da verificação formal de propriedades de uma especificação, é útil a simulação das especificações, para um acompanhamento da evolução do comportamento da mesma. Este acompanhamento pode detectar evoluções não desejadas da especificação, nem sempre detectadas na verificação através de predicados lógicos.

A simulação pode ainda, dependendo da técnica utilizada, permitir a validação de especificação que não podem ser verificadas pelas técnicas normais. Um exemplo é o caso de comportamentos infinitos, nos quais o formalismo subjacente à álgebra de processos (grafos temporizados ou sistema de transições rotulado) não é finito.

Assim, pode-se dizer que a verificação de sistemas através da verificação de modelos, e a validação dos mesmos utilizando-se a simulação, são funções complementares. Na continuidade deste trabalho será abordada a simulação de especificações de sistemas dependentes de tempo, descritas em RT-LOTOS, através de uma metodologia de simulação que utiliza a transformação das especificações.

2.5 Conclusões

Neste capítulo foram discutidas as principais características dos sistemas dependentes de tempo. Foram também apresentadas diversas maneiras de se representá-los, tal como redes de Petri temporizadas, lógicas temporais tempo real e álgebras de processos temporizadas.

Uma álgebra de processos temporizada, denominada RT-LOTOS, foi escolhida para especificar sistemas dependentes de tempo, e sua sintaxe e semântica apresentadas, com maior ênfase sendo dada à semântica.

Foi mostrada a necessidade da utilização de meios formais de verificação das especificações, e que uma metodologia para tal já encontra-se disponível, definida em [Martins96]. Também foi mostrado que a simulação complementa a verificação através da verificação de modelos.

No capítulo seguinte é feita uma abordagem de simulação de sistemas dependentes de tempo especificadas em RT-LOTOS. As características de um simulador são discutidas, e a arquitetura interna de uma ferramenta de simulação é descrita.

Capítulo 3

Simulação de Especificações RT-LOTOS

A simulação de especificações formais envolve alguns aspectos de avaliação e transformação das mesmas, que podem determinar características importantes de uma ferramenta de simulação. Entre estes aspectos, o mais importante é a escolha entre a simulação diretamente sobre a especificação ou sobre o modelo semântico subjacente. Também é necessária a definição das funções que determinam as possibilidades de evolução da especificação, além das funções que representam a evolução propriamente dita. Outro aspecto importante é a definição do mecanismo de exploração das diferentes alternativas de evolução do comportamento da especificação.

Estes aspectos serão abordados neste capítulo, primeiramente com uma análise das alternativas disponíveis, seguida da descrição detalhada da alternativa escolhida para o desenvolvimento deste trabalho.

3.1 Modos de Simulação

Para a simulação de especificações formais de sistemas através de álgebras de processo, existem duas alternativas básicas:

- A tradução da especificação para o formalismo subjacente, tal como um grafo temporizado ou um sistema de transições rotulado. Após a tradução, é feita a simulação sobre o formalismo, sendo o processo de simulação basicamente o de disparo de ações e transições entre estados. Os estados já foram todos previamente calculados, com as transições possíveis já determinadas para cada estado. Basicamente, a simulação resume-se a percorrer os arcos do formalismo subjacente, como grafo temporizado ou sistema de transições rotulado.
- O cálculo das ações possíveis diretamente da especificação, com o disparo de uma ação implicando no cálculo do novo estado, através da transformação da especificação. Para cada novo estado alcançado, é necessário calcular-se novamente as ações possíveis. No disparo das ações e na passagem do tempo, ao invés de uma simples mudança de estado num modelo do formalismo subjacente, é necessário o cálculo da expressão de comportamento resultante.

As ferramentas de simulação para LOTOS, incorporadas nos maiores pacotes de verificação e simulação, utilizam a primeira abordagem. Algumas ferramentas deste tipo são: a ferramenta Caesar.open, do pacote Caesar/Aldebaran [Garavel94] e a ferramenta SMILE, do pacote MiniLite [Eertink92]. Estas ferramentas utilizam o sistema de transições rotuladas que são gerados por compiladores de especificações escritas em LOTOS. A simulação consiste basicamente em percorrer as transições do sistema de transições rotuladas.

Para sistemas dependentes de tempo, a mesma abordagem é possível. Apenas o formalismo subjacente muda, devido ao problema da explosão de estados quando da utilização de sistemas de transições rotulados para representar especificações de sistemas dependentes de tempo. Uma alternativa é utilizar um formalismo diferente, os grafos temporizados, que podem contornar este problema. Para o caso particular de RT-LOTOS, é a solução adotada, conforme mostrado em [Camargo95]. Um tradutor de RT-LOTOS para grafos temporizados já foi implementado em [Martins96]. Para a extensão temporal de LOTOS apresentada em [CoOl95], homônima à apresentada neste trabalho, foi implementada uma ferramenta de simulação, denominada RTL. Esta ferramenta realiza a simulação sobre o grafo temporizado utilizado como modelo semântico subjacente àquela linguagem.

Porém, como discutido em [CaFa95, Garavel89, Martins96], e já mostrado na seção 2.4, esta abordagem apresenta a restrição de poder apenas tratar especificações que levem a um sistema de transições finito. Utilizando esta abordagem, um simulador pode apenas lidar com especificações que tenham representação em um sistema de transições finito.

Em [Eijk89] é apresentada uma metodologia de construção de um simulador para especificações formais baseada na segunda alternativa, a da transformação das especificações. Esta metodologia foi utilizada no desenvolvimento da ferramenta de simulação Hippo [Tretmans89], do projeto ESPRIT/SEDOS. Esta abordagem contorna o problema da representação de especificações cujo comportamento não possa ser representado por um sistema de transições finito.

Esta segunda abordagem será utilizada no presente trabalho, pelo fato de possibilitar a validação de um espectro maior de especificações, complementando assim as funcionalidades da ferramenta já existente para RT-LOTOS, apresentada em [Martins96]. Porém, mesmo contornando a restrição dos comportamentos finitos, ainda é necessário que a especificação analisada pela simulação não apresente expressões não guardadas [Camargo95, Martins96], como mostrado na seção 2.4.

Na continuidade deste capítulo serão apresentadas as definições das funcionalidades do núcleo do simulador, bem como os algoritmos de simulação.

3.2 Funções do Núcleo de Simulação

Seguindo a metodologia proposta em [Eijk89], o primeiro passo para a construção de um simulador é a definição das funções do núcleo do mesmo. Estas funções dizem respeito à avaliação dos eventos que podem ocorrer num determinado estado da especificação, e como atualizar a especificação em decorrência de um evento.

Ao contrário de LOTOS, onde uma especificação pode apenas evoluir pela ocorrência de ações, nas especificações RT-LOTOS existem outras maneiras da especificação evoluir. Além das ações clássicas de LOTOS, existem as ações semânticas de violação temporal. Também a passagem do tempo deve ser administrada, havendo situações na especificação em que a mesma é permitida ou não, dependendo da existência ou não de ações ditas urgentes.

Ações urgentes são ações que devem ocorrer no momento atual da especificação. Assim, uma ação interna `i` com o intervalo [0, 0] associado é urgente. Ela deve ocorrer imediatamente, não permitindo a passagem do tempo antes de sua ocorrência. Ações observáveis com intervalo [0, 0] não são urgentes. Porém, neste instante, também há a possibilidade da ocorrência da violação temporal, e esta sim é urgente. Assim, neste instante não são permitidas ações de passagem do tempo até que ocorra ou a ação observável ou sua violação temporal.

Desta forma, foram identificados dois grupos de funções necessárias para o núcleo do simulador de especificações RT-LOTOS:

- funções que manipulam a evolução da especificação pela ocorrência de ações e violações temporais, e
- funções que manipulam a evolução da especificação pela passagem do tempo.

Para a manipulação da ocorrência de ações e violações temporais, foram identificadas as seguintes funções:

- `Actions(B)`: dada uma expressão de comportamento `B`, esta função retorna um conjunto de triplas (a, t_1, t_2), onde `a` é uma ação (observável, interna, término com sucesso ou violação temporal), e t_1 e t_2 são os limites inferior e superior, respectivamente, do intervalo no qual a ação pode ocorrer. O conjunto de triplas denota as ações oferecidas por `B`, prontas para ocorrer ou não.
- `Menu(M)`: esta função, aplicada sobre um conjunto de ações, retorna o subconjunto de todas os ações que estão prontas para ocorrer. Ou seja, pesquisa um conjunto de ações, procurando as ações que tenham seu limite inferior igual a zero.
- `Next(a,B)`: esta função deve calcular o novo estado da especificação, a partir da ocorrência do evento `a` no estado atual `B`. Este evento `a` pode ser uma das ações

da especificação (pertencentes ao conjunto `Act`), a ação interna (`i`), as ações de violação temporal (pertencente ao conjunto `Act*`) e a ação de término com sucesso (δ). Além da expressão de comportamento que represente o novo estado da especificação, a função `Next` deve retornar também a ação que deve ser vista pelo ambiente de `B`.

Para tratamento da parte temporal das especificações RT-LOTOS, três foram as funções identificadas como necessárias no núcleo do simulador. Estas funções são descritas a seguir.

- `MinTime(M)`: esta função pesquisa o conjunto de ações temporizadas de uma expressão de comportamento (calculada pela função `Actions`), e retorna o tempo mínimo que este conjunto permite passar. Este tempo pode ser o menor limite inferior, ou o menor limite superior, dependendo do que for menor. Em outras palavras, é o tempo mínimo para que uma nova ação seja habilitada, ou para que uma ação urgente ou violação temporal impeça a ocorrência de ações de passagem do tempo.
- `MaxTime(M)`: esta função também pesquisa o conjunto de ações temporizadas, retornando o menor limite superior deste conjunto. Pode-se dizer que é o tempo máximo que pode passar, antes que uma ação torne-se urgente. As violações temporais também são consideradas urgentes. Isto porque no limite superior do intervalo de uma ação, é necessário determinar se o que vai ocorrer é a ação ou sua violação temporal.
- `DoTime(t,B)`: esta função calcula o novo estado da especificação, resultante da passagem de `t` unidades de tempo no estado atual `B`. Este tempo não deverá ser maior que o tempo máximo permitido antes de uma ação urgente, obtido através da função `MaxTime(M)`.

Na seqüência, todas estas funções serão definidas formalmente, de modo a gerar um modelo de avaliação e evolução de especificações RT-LOTOS a ser utilizado no núcleo de um simulador. Para a definição formal destas funções, são necessárias algumas definições preliminares, feitas a seguir.

$\mathcal{D}$ é o domínio de tempo da especificação, $\mathcal{Act}$ é o conjunto das ações clássicas de RT-LOTOS, e $\mathcal{Act}^*$ é o conjunto das *violações temporais*, conforme definido na seção 2.3.

$\mathcal{M}$ é um conjunto das triplas na forma $(a,\ t_1,\ t_2)$, onde $a \in \mathcal{Act}^{i,\delta} \cup \mathcal{Act}^*$, $t_1 \in \mathcal{D}$ e $t_2 \in \mathcal{D}^\infty$. Cada elemento de $\mathcal{M}$ é uma tripla que denota uma ação a qualquer, com seu intervalo de tempo associado, onde t_1 e t_2 são os limites superior e inferior, respectivamente. É o conjunto de todas as ações da especificação em qualquer intervalo possível no domínio de tempo especificado. É importante ressaltar que o limite inferior não pode ser infinito.

Será denominado $\mathcal{B}$ o conjunto de todas as expressões de comportamento RT-LOTOS válidas. Ou seja, o conjunto de todas as especificações RT-LOTOS possíveis, que não possuam expressões não guardadas, conforme definido na seção 2.4.

Quando não indicado o contrário, $a \in \mathcal{Act}^{i,\delta}$ e $a^* \in \mathcal{Act}^*$. Nos intervalos de tempo $[t_1, t_2]$, $t_1 \in \mathcal{D}$ e $t_2 \in \mathcal{D}^\infty$.

3.2.1 A Função Actions

A função `Actions` calcula o conjunto de todas as ações de uma expressão de comportamento. Estas ações podem estar prontas para serem realizadas ou não. Mais formalmente, a função `Actions` é definida da seguinte forma:

$\texttt{Actions} : \mathcal{B} \rightarrow 2^{\mathcal{M}}$

onde $2^{\mathcal{M}}$ representa o conjunto das partes de $\mathcal{M}$.

A partir de uma expressão de comportamento $B \in \mathcal{B}$, a função `Actions` calcula um conjunto $M \subset \mathcal{M}$, de ações com intervalos de tempo associados presentes em B. A seguir, a função `Actions` é definida recursivamente sobre os operadores de RT-LOTOS.

$\text{Actions(stop)} = \varnothing$

O processo `stop` não oferece nenhuma ação.

$\text{Actions(exit)} = \{(\delta, 0, \infty)\}$

O processo `exit` pode realizar a ação de término com sucesso δ, sendo seu limite inferior 0 e o superior infinito.

$$\text{Actions}([t_1,t_2]a;\ B) = \begin{cases} \{(a,t_1,t_2),(a^*,t_2,t_2)\} & \text{se } t_2 \neq \infty \wedge a \neq i \\ \{(a,t_1,t_2)\} & \text{se } t_2 = \infty \wedge a \neq i \\ \{(i,t_1,t_2)\} & \text{se } a = i \end{cases}$$

$$\text{Actions}(a;B) = \begin{cases} \{(a,0,\infty)\} & \text{se } a \neq i \\ \{(i,0,0)\} & \text{se } a = i \end{cases}$$

Na prefixação, as ações observáveis podem ocorrer no seu intervalo. As violações temporais ocorrerem no limite superior do intervalo da ação observável, somente se este limite não for infinito. Para a ação interna, não é definida a violação temporal. Intervalos *default* são assumidos quando não definidos explicitamente.

$\text{Actions}(B_1 [] B_2) = \text{Actions}(B_1) \cup \text{Actions}(B_2)$

Na escolha, o conjunto de ações é a simples união dos conjuntos de ações de cada lado do operador.

$$\begin{aligned}\text{Actions}(B_1|[L]|B_2) = \; & \{(a,t_1,t_2)|(a,t_1,t_2) \in \text{Actions}(B_1) \wedge a \notin L \cup \{\delta\}\} \cup \\ & \{(a,t_1,t_2)|(a,t_1,t_2) \in \text{Actions}(B_2) \wedge a \notin L \cup \{\delta\}\} \cup \\ & \{(a,t_1,t_2)|(a,t_3,t_4) \in \text{Actions}(B_1) \wedge (a,t_5,t_6) \in \text{Actions}(B_2) \wedge \\ & a \in L \cup \{\delta\} \wedge t_1 = \max(t_3,t_5) \wedge t_2 = \min(t_4,t_6) \wedge t_1 \leq t_2\}\end{aligned}$$

Na composição paralela, o conjunto de ações é a união das ações das expressões de comportamento de cada lado do operador, que não pertençam ao conjunto de sincronização. Para as ações do conjunto de sincronização, é necessário que sejam oferecidas dos dois lados do operador e que haja uma interseção do intervalo de tempo associado de cada lado do operador. É o que denota a definição do terceiro conjunto na expressão acima. É importante notar que cada ação a sincronizar oferecida de um lado do operador, poderá sincronizar-se com qualquer ocorrência da mesma ação do outro lado do operador. Assim, todas as possibilidades de sincronização são determinadas.

$$\begin{aligned}\text{Actions}(B_1 >> B_2) = \; & \{(a,t_1,t_2)|(a,t_1,t_2) \in \text{Actions}(B_1) \wedge a \neq \delta\} \cup \\ & \{(i,t_1,t_1)|(\delta,t_1,t_2) \in \text{Actions}(B_1)\}\end{aligned}$$

Na composição seqüencial, a ação de término com sucesso δ é ocultada. Por este motivo, é necessário substituir suas ocorrências pela ação interna `i`, no conjunto de ações possíveis para esta expressão de comportamento. Como passa a ser urgente, o limite superior do intervalo é atualizado, sendo igualado ao limite inferior (geralmente 0).

$$\text{Actions}(B_1 [> B_2) = \text{Actions}(B_1) \cup \text{Actions}(B_2)$$

Na preempção o conjunto de ações é a união dos conjuntos de ações que podem ocorrer dos dois lados do operador.

$$\begin{aligned}\text{Actions}(\text{hide L in B}) = \; & \{(a,t_1,t_2)|(a,t_1,t_2) \in \text{Actions}(B) \wedge a \notin L\} \cup \\ & \{(i,t_1,t_1)|(a,t_1,t_2) \in \text{Actions}(B) \wedge a \in L\}\end{aligned}$$

No operador de ocultação, as ações a serem ocultadas são substituídas pela ação interna `i`. Pela propriedade de progressão máxima, devem ocorrer no primeiro instante possível. Assim, o limite superior é igualado ao limite inferior.

$$\begin{aligned}\text{Actions}(B < L]\{a_1:Q_1,\ldots,a_n:Q_n\}) = \; & \{(a,t_1,t_2)|(a,t_1,t_2) \in \text{Actions}(B) \wedge a \in \text{Act}^{i,\delta}\} \cup \\ & \{(a^*,t_2,t_2)|(a^*,t_2,t_2) \in \text{Actions}(B) \wedge a \notin L\} \cup \\ & \{(i,t_2,t_2)|(a^*,t_2,t_2) \in \text{Actions}(B) \wedge a \in L\}\end{aligned}$$

O operador de preempção temporal oculta a ocorrência das violações temporais das ação de `L`. Assim, todas as violação temporais a serem tratadas são substituídas pela ação interna `i`.

$$\begin{aligned}\text{Actions}(P[a_1,a_2,\ldots,a_n]) = \; & \text{Actions}(B[a_1 / g_1, a_2 / g_2,\ldots,a_n / g_n]) \\ & \text{se } P[g_1,g_2,\ldots,g_n] = B\end{aligned}$$

As ações possíveis numa instanciação são as ações possíveis na expressão de comportamento que define o processo instanciado. É necessário apenas fazer a substituição das ações da definição do processo pelas ações passadas na instanciação.

3.2.2 A Função Menu

A função `Menu(M)` calcula um subconjunto do conjunto `M` das ações possíveis de `B`, calculado pela função `Actions`. Esta função "filtra" o conjunto `M`, de modo a determinar as ações prontas para sincronização. Em outras palavras, as ações com limite inferior igual a 0. Mais formalmente, a função menu é definida como:

`Menu` : $2^{\mathcal{M}} \rightarrow 2^{\mathcal{M}}$

O cálculo do conjunto retornado pela função `Menu` pode ser definido da seguinte forma:

$$\text{Menu}(M) = \{(a, t_1, t_2) | (a, t_1, t_2) \in M \wedge t_1 = 0\}$$

Ou seja, para um conjunto `M` $\subset \mathcal{M}$ de ações com intervalos associados, a função `Menu` retorna um outro conjunto `M'` $\subseteq$ `M`, onde as ações têm o limite inferior do intervalo igual a 0. Ou seja, as ações prontas a ocorrer.

3.2.3 A Função Next

A função `Next(a, B)` calcula a expressão de comportamento resultante da ocorrência da ação `a` na expressão de comportamento `B`. Este cálculo na verdade são as próprias regras da semântica formal de RT-LOTOS, sem a parte de tempo. A passagem do tempo é tratada na função `DoTime(t, B)`. Além da expressão de comportamento resultante da ocorrência da ação, `Next(a, B)` retorna a ação que é vista pelo observador externo do comportamento. A forma geral da função é:

$$\text{Next}(a, B) = (B', g)$$

onde `a`, `g` $\in (\mathcal{Act}^{i,\delta} \cup \mathcal{Act}^*)$, `B`, `B'` $\in \mathcal{B}$, e `a` é a ação que ocorreu, `B` é a expressão de comportamento onde `a` ocorreu, `B'` é a expressão de comportamento resultante, e `g` é como a ação `a` é vista pelo ambiente de `B`.

Mais formalmente, a função `Next` é definida por:

`Next` : $(\mathcal{Act}^{i,\delta} \cup \mathcal{Act}^*, \mathcal{B}) \rightarrow (\mathcal{B}, \mathcal{Act}^{i,\delta} \cup \mathcal{Act}^*)$

A definição recursiva da função `Next(a,B)` sobre os operadores de RT-LOTOS é descrita a seguir, sendo separada a definição para a evolução pelas ações de violação temporal.

$$\text{Next}(\delta, \text{exit}) = (\text{stop}, \delta)$$

O processo `exit`, através da ação semântica δ, de término com sucesso, evolui para `stop`.

$$\text{Next}(a,[0,t]a;B) = (B,a) \qquad a \in \text{Act}^i$$
$$\text{Next}(a^*,[0,0]a;B) = (\text{stop},a^*) \qquad a \in \text{Act}$$

A prefixação pode evoluir para o comportamento subseqüente quando da ocorrência da ação, desde que o limite inferior do intervalo de tempo seja 0. Já a ocorrência de uma violação temporal pode ocorrer somente quando o limite superior do intervalo de tempo for 0, e leva o comportamento para `stop`. Note-se que a violação temporal só é definida na prefixação para ações observáveis. Para outras ocorrências, a função não é definida, portanto não pode ser aplicada. Por exemplo, a ocorrência da violação temporal numa prefixação com limite superior diferente de 0.

$$\text{Next}(a,B_1[]B_2) = \begin{cases} (B_1',a) & \text{se Next}(a,B_1) = (B_1',a) \\ (B_2',a) & \text{se Next}(a,B_2) = (B_2',a) \end{cases}$$
$$\text{Next}(a^*,B_1[]B_2) = \begin{cases} (B_1'[]B_2,a^*) & \text{se Next}(a^*,B_1) = (B_1',a^*) \\ (B_1[]B_2',a^*) & \text{se Next}(a^*,B_2) = (B_2',a^*) \end{cases}$$

A escolha irá evoluir pelo lado no qual se deu a ação, descartando o outro lado, decidindo assim o não determinismo. Já a ocorrência de uma violação temporal não decide a escolha.

$$\text{Next}(a,B_1|[L]|B_2) = \begin{cases} (B_1'|[L]|B_2,a) & \text{se Next}(a,B_1) = (B_1',a) \wedge a \notin L \\ (B_1|[L]|B_2',a) & \text{se Next}(a,B_2) = (B_2',a) \wedge a \notin L \\ (B_1'|[L]|B_2',a) & \text{se Next}(a,B_1) = (B_1',a) \wedge \\ & \text{Next}(a,B_2) = (B_2',a) \wedge a \in L \end{cases}$$
$$\text{Next}(a^*,B_1|[L]|B_2) = \begin{cases} (B_1'|[L]|B_2,a^*) & \text{se Next}(a,B_1) = (B_1',a^*) \\ (B_1|[L]|B_2',a^*) & \text{se Next}(a,B_2) = (B_2',a^*) \end{cases}$$

As expressões de comportamento numa composição paralela irão evoluir individualmente para as ações que não estejam no conjunto de sincronização, bem como para as violações temporais. Para as ações que pertencem ao conjunto de sincronização, as expressões de comportamento dos dois lados do operador devem poder evoluir pela mesma ação.

$$\text{Next}(a,B_1>>B_2) = \begin{cases} (B_1'>>B_2,a) & \text{se Next}(a,B_1) = (B_1',a) \wedge a \neq \delta \\ (B_2,i) & \text{se Next}(a,B_1) = (B_1',a) \wedge a = \delta \end{cases}$$

$$\text{Next}(a^*,B_1>>B_2) = (B_1'>>B_2,a^*) \quad \text{se Next}(a^*,B_1) = (B_1',a^*)$$

A composição seqüencial evolui pela ações da expressão de comportamento a esquerda do operador, até a ocorrência da ação de término com sucesso δ, quando então passa a comportar-se como a expressão de comportamento a direita do operador. Note-se que ocorrência da ação de término com sucesso δ é ocultada pelo operador de habilitação, sendo sua ocorrência vista pelo ambiente como a ação interna `i`.

$$\text{Next}(a, B_1[>B_2) = \begin{cases} (B_1'[>B_2, a) & \text{se Next}(a, B_1) = (B_1', a) \wedge a \neq \delta \\ (B_1', a) & \text{se Next}(a, B_1) = (B_1', a) \wedge a = \delta \\ (B_2', a) & \text{se Next}(a, B_2) = (B_2', a) \end{cases}$$

$$\text{Next}(a^*, B_1[>B_2) = \begin{cases} (B_1'[>B_2, a^*) & \text{se Next}(a^*, B_1) = (B_1', a^*) \\ (B_1[>B_2', a^*) & \text{se Next}(a^*, B_2) = (B_2', a^*) \end{cases}$$

A expressão de comportamento do lado esquerdo do operador de preempção pode evoluir até a ocorrência da primeira ação da expressão de comportamento do lado direito do operador. Caso ocorra o término com sucesso da expressão de comportamento do lado esquerdo do operador, todo o processo termina. Note-se que a ocorrência de uma violação temporal não causa a preempção.

$$\text{Next}(a, \text{hide L in B}) = \begin{cases} (B', a) & \text{se Next}(a, B) = (B', a) \wedge a \notin L \\ (B', i) & \text{se Next}(a, B) = (B', a) \wedge a \in L \end{cases}$$

$$\text{Next}(a^*, \text{hide L in B}) = \begin{cases} (B', a^*) & \text{seNext}(a^*, B) = (B', a^*) \wedge a \notin L \\ (B', i) & \text{se Next}(a^*, B) = (B', a^*) \wedge a \in L \end{cases}$$

A ocultação pode evoluir por qualquer ação que faça a expressão de comportamento afetada pelo operador evoluir. A diferença está em como o ambiente percebe a evolução. Caso seja uma ação ocultada, o ambiente vê a expressão de comportamento evoluindo por uma ação interna `i`. Caso contrário o ambiente vê a evolução normal da expressão de comportamento. A violação temporal de uma ação do conjunto de portas ocultadas também é ocultada. Porém só é possível para ações ocultadas cujo intervalo associado seja pontual. Ou seja, seu intervalo de tempo deve ser do tipo `[t, t]`. Caso contrário, a propriedade de progressão máxima impede a ocorrência da violação temporal em ações ocultadas.

$$\text{Next}(a, B < L]\{\Phi\}) = \begin{cases} (B' < L]\{\Phi\}, a) & \text{seNext}(a, B) = (B', a) \wedge a \neq \delta \\ (B', a) & \text{se Next}(a, B) = (B', a) \wedge a = \delta \end{cases}$$

onde $\Phi = a_1 : Q_1, \ldots, a_n : Q_n$

$$\text{Next}(a^*, B < L]\{\Phi\}) = \begin{cases} (B' < L]\{\Phi\}, a^*) & \text{seNext}(a^*, B) = (B', a^*) \wedge a \notin L \\ (Q_j, i) & \text{se Next}(a_j^*, B) = (B', a_j^*) \wedge a = a_j \wedge a \in L \end{cases}$$

onde $\Phi = a_1 : Q_1, \ldots, a_n : Q_n$

A evolução de uma expressão de comportamento que apresente o operador de preempção temporal é normal, a não ser quando da evolução se dá através da ação de

término com sucesso δ, que implica no término de todo o processo que inclui o operador de preempção temporal. Quando ocorre uma violação temporal que deve ser tratada pelo operador, o mesmo faz a transição para o comportamento correspondente "silenciosamente". Ou seja, para o observador externo, a transição se dá através de uma ação interna `i`.

$$Next(a, P[a_1,...,a_n]) = (B'[g_1 / a_1,...,g_n / a_n], a) \quad se\ P[a_1,...,a_n] = B \wedge$$
$$Next(a, B[g_1 / a_1,...,g_n / a_n]) = (B'[g_1 / a_1,...,g_n / a_n], a)$$

$$Next(a^*, P[a_1,...,a_n]) = (B'[g_1 / a_1,...,g_n / a_n], a^*) \quad se\ P[a_1,...,a_n] = B \wedge$$
$$Next(a^*, B[g_1 / a_1,...,g_n / a_n]) = (B'[g_1 / a_1,...,g_n / a_n], a^*)$$

A evolução de uma instanciação se dá pela evolução da expressão de comportamento que a define, substituindo apenas os nomes das ações pelos nomes das ações da instanciação.

As funções que determinam a possibilidade de passagem do tempo numa expressão de comportamento RT-LOTOS são descritas a seguir. Também é descrita a função que transforma uma especificação RT-LOTOS quando da passagem do tempo.

3.2.4 A Função MinTime

A função `MinTime(M)`, `M` $\subset \mathcal{M}$, varre o conjunto de ações possíveis numa especificação RT-LOTOS, e determina o tempo mínimo que é necessário passar até que alguma nova ação tenha seu limite mínimo alcançado. Ou então, qual o tempo mínimo necessário para que uma ação urgente bloqueie a passagem do tempo. Por exemplo, na expressão de comportamento `([0,10]a; exit [] [5,10]b; exit)` o tempo mínimo que deve passar é de 5 unidades de tempo, momento no qual a ação `b` passa a ser oferecida ao ambiente.

Já no caso da expressão de comportamento `([0,5]i; exit ||| [10,15]a; exit)` o tempo necessário para que uma nova ação seja habilitada é de 10 unidades de tempo. Porém, o tempo que pode passar é de 5 unidades de tempo, quando então a semântica da ação interna bloqueia a passagem do tempo, e a ação `i` torna-se urgente e incontrolável.

Mais formalmente, a função `MinTime(M)` pode ser definida como:

`MinTime` : $2^{\mathcal{M}} \rightarrow \mathcal{D}^{\infty}$

O cálculo do tempo mínimo de um conjunto de ações temporizadas pode ser definido como:

$$\text{MinTime(M)} = \begin{cases} \infty & \text{se } M = \varnothing \\ \min\{t \mid t = t_1,\ \forall(a,t_1,t_2) \in M \wedge t_1 > 0, \text{ou} \\ \qquad t = t_2,\ \forall(a,t_1,t_2) \in M \wedge t_1 = 0\} & \text{se } M \neq \varnothing \end{cases}$$

O tempo mínimo é o menor valor entre os limites inferiores dos intervalos de tempo das ações, que sejam diferentes de 0, e os limites superiores das ações que tenham seu limite inferior igual a 0.

Se o valor for 0, é sinal de que há uma ação urgente impedindo que ações de passagem do tempo ocorram. Se não houver nenhuma ação possível na expressão de comportamento (o processo `stop`, por exemplo) `MinTime` retornará infinito.

3.2.5 A Função MaxTime

A função `MaxTime(M)`, $M \subset \mathcal{M}$, procura, no conjunto de ações temporizadas `M`, o tempo máximo que pode passar antes que uma ação urgente ou uma violação temporal bloqueie a passagem do tempo. Em outras palavras, procura o menor limite superior das ações temporizadas. A definição formal da função `MaxTime` é a seguinte:

`MaxTime` : $2^{\mathcal{M}} \rightarrow \mathcal{D}^{\infty}$

O cálculo do tempo mínimo de um conjunto de ações temporizadas pode ser definido como:

$$\text{MaxTime(M)} = \begin{cases} \infty & \text{se } M = \varnothing \\ \min\{t \mid t = t_2, \forall(a,t_1,t_2) \in M\} & \text{se } M \neq \varnothing \end{cases}$$

Como na função `MinTime`, se `MaxTime` retornar 0 é um indicativo de que há uma ação urgente ou uma violação temporal bloqueando a passagem do tempo. `MaxTime` também retorna infinito quando não houver ações temporizadas na expressão de comportamento. Porém, `MaxTime` também retornará infinito quando as ações da expressão de comportamento forem todas ações observáveis sem intervalo de tempo explícito associado. Neste caso é o intervalo [0, ∞].

3.2.6 A Função DoTime

Além da ocorrência das ações e das violações temporais, a passagem do tempo também implica na transformação das expressões de comportamento RT-LOTOS. Desta forma, é necessária a definição da função de transformação `DoTime(t, B)`. Esta função irá calcular a nova expressão de comportamento resultante da passagem de `t` unidades de tempo na expressão de comportamento `B`.

Basicamente, `DoTime(t, B)`, $t \in \mathcal{D}^{\infty}$, $B \in \mathcal{B}$, implementa as regras da semântica de RT-LOTOS que tratam a passagem do tempo. A definição formal da função `DoTime` é a seguinte:

$$DoTime : (\mathcal{D}^{\infty}, \mathcal{B}) \rightarrow \mathcal{B}$$

A definição recursiva da função `DoTime(t, B)`, sobre os operadores de RT-LOTOS, é a seguinte:

$$DoTime(t, stop) = stop$$

A passagem do tempo não altera o comportamento do processo `stop`.

$$DoTime(t, exit) = exit$$

Como o processo `stop`, o processo `exit` não sofre alteração com a passagem do tempo.

$$DoTime(t, [t_1, t_2]a; B) = \begin{cases} [t_1 - t, t_2 - t]a; B & \text{se } t_1 > t \wedge t_2 \geq t \\ [0, t_2 - t]a; B & \text{se } t_1 \leq t \wedge t_2 \geq t \end{cases}$$

A passagem do tempo, na prefixação, pode ser maior que o limite inferior do intervalo da ação. O limite superior não pode ser excedido. Caso isto aconteça, a função não está definida. Para fins de implementação, antes da aplicação da função, o tempo a transcorrer é comparado com o tempo máximo, obtido pela função `MaxTime`. Se for maior que o tempo máximo, a passagem do tempo é ignorado. Se for menor, garante-se que nunca haverá um limite superior menor que o tempo a passar.

$$DoTime(t, B_1 [] B_2) = DoTime(t, B_1) \wedge DoTime(t, B_2)$$

O tempo irá passar na mesma proporção dos dois lados da escolha.

$$DoTime(t, B_1 |[L]| B_2) = DoTime(t, B_1) \wedge DoTime(t, B_2)$$

Também na composição paralela o tempo passa na mesma proporção dos dois lados do operador.

$$DoTime(t, B_1 >> B_2) = DoTime(t, B_1)$$

Na composição seqüencial, a passagem do tempo afeta apenas o processo do lado direito do operador.

$$DoTime(t, B_1 [> B_2) = DoTime(t, B_1) \wedge DoTime(t, B_2)$$

Na preempção o tempo evolui na mesma proporção em ambos os lados do operador.

$$DoTime(t, \text{hide } L \text{ in } B) = DoTime(t, B)$$

O tempo irá passar normalmente para a expressão de comportamento sobre o qual é feito a ocultação de ações.

$$DoTime(t, B < L]\{a_1:Q_1,\ldots,a_n:Q_n\}) = DoTime(t, B)$$

O operador de preempção temporal também não altera a passagem do tempo no processo que tem suas violações temporais tratadas pelo operador.

$$DoTime(t, P[a_1,\ldots,a_n]) = DoTime(t, B[g_1 / a_1,\ldots,g_n / a_n]) \text{ se } P[a_1,\ldots,a_n] = B$$

Na instanciação de processo, a passagem do tempo implica na atualização da expressão de comportamento da definição do processo, com as devidas substituições das ações.

Uma vez definidas as funções que são necessárias para o núcleo do simulador, serão feitas a seguir considerações sobre a arquitetura do simulador. Primeiramente será feita uma abordagem da implementação do núcleo do simulador, com a compilação da especificação a ser validada, e sua tradução para uma representação interna. A seguir, a implementação das funções do núcleo do simulador, sobre esta representação interna, é discutida. Por último, os algoritmos de simulação interativa e automática são apresentados.

3.3 Implementação do Simulador

Nesta seção serão descritos alguns aspectos arquiteturais da ferramenta de simulação de especificações RT-LOTOS. Primeiramente a estrutura básica dos módulos de simulação é mostrada, de maneira bastante genérica. O processo geral de compilação e representação interna das especificações é descrita. Em seguida, os módulos de simulação interativa e automática são descritos.

3.3.1 Arquitetura Básica do Simulador

A figura 3.1 apresenta a estrutura básica da ferramenta de simulação. O simulador é dividido em 3 módulos principais, descritos a seguir:

- Núcleo: este módulo é responsável pela compilação das especificações, com sua tradução para a representação interna. É também no núcleo que estão implementadas as funções descritas na seção 3.2, que são responsáveis por toda evolução da especificação, entre outras.
- Módulo de simulação interativa: é responsável pela interação com o usuário. Ou seja, é responsável pela interpretação de comandos na simulação interativa. Interage com o núcleo para a evolução da especificação.
- Módulo de simulação automática: responsável pela obtenção das opções de simulação automática, bem como pela implementação dos algoritmos de simulação. Também interage com o núcleo para a evolução da especificação.

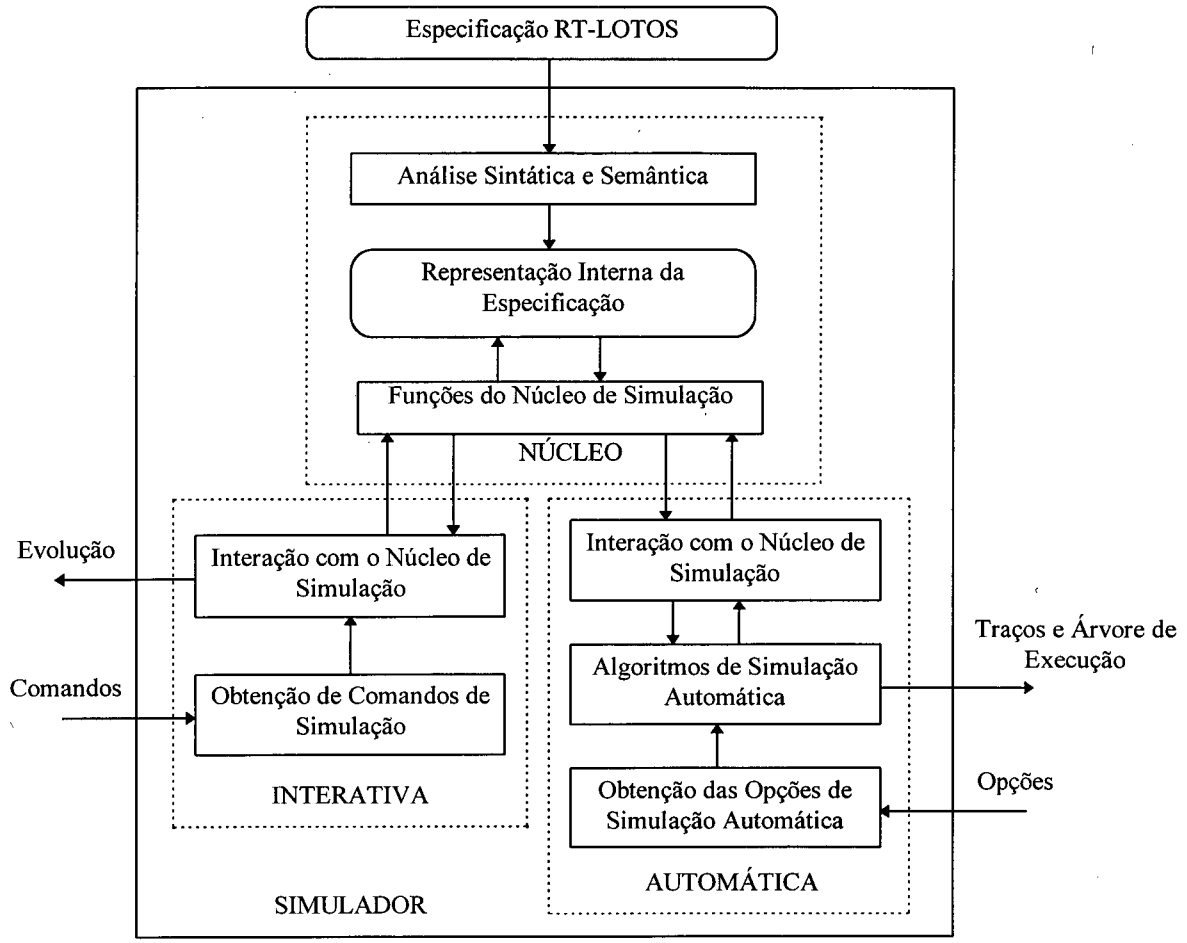


Figura 3.1 - Ferramenta de Simulação de Especificações RT-LOTOS.

Esta modularização do simulador permite que alterações nas funcionalidades de um módulo não impliquem na alteração de outros. Por exemplo, caso sejam inseridos novos algoritmos de simulação automática (como novas distribuições de probabilidade), somente o módulo de simulação automática precisa ser alterado.

A seguir, as funções de cada módulo são descritas, nos seus aspectos de implementação. A funcionalidades, a nível de interação com usuário, serão abordadas no capítulo 4.

3.3.2 O Núcleo de Simulação

O primeiro passo do núcleo de simulação é a compilação da especificação. A análise sintática e semântica consiste basicamente na verificação da especificação com relação à gramática da linguagem, apresentada no Anexo A.

Para a análise sintática e semântica da especificação, é utilizada como ferramenta auxiliar o sistema SINTAX [BoDe88, BoDe89], descrito com mais detalhes no anexo B. Esta ferramenta gera um conjunto de tabelas de dados relacionados à estrutura da

linguagem, escritas em linguagem C, que serão utilizadas durante toda a compilação. A ferramenta também gera um programa em C, que é o esqueleto do programa de análise semântica.

Nos níveis léxico e sintático, a ferramenta SINTAX também facilita a detecção e possível correção de erros. O programa detecta e identifica tais erros e, se for possível, faz sua correção e prossegue com a compilação, alertando o ocorrido posteriormente.

A construção da árvore abstrata da especificação é feita de acordo com a teoria de Sintaxe Abstrata [Meyer90]. Durante esta construção, as análises semânticas da especificação são realizadas, e simultaneamente é feita a tradução da especificação para a representação interna do simulador.

A representação interna da especificação é baseada diretamente na árvore abstrata. Consiste numa árvore, onde cada nó pode ser:

- um operador da linguagem, e os seus filhos os operandos;
- um processo básico de RT-LOTOS (`exit` e `stop`)
- uma instanciação de processo.

A construção desta árvore obedece a precedência de operadores de LOTOS, apresentada em [BoBr87]. O operador de preempção temporal, introduzido em RT-LOTOS, tem a menor precedência, ou seja, as precedências são:

Prefixação > Escolha > Composição Paralela > Preempção > Composição Seqüencial > Ocultação > Preempção Temporal

A representação interna dos operadores é descrita a seguir. Sua representação gráfica, para uma melhor compreensão, pode ser vista na tabela 3.1.

- Exit: o processo `exit` é representado num nó da árvore, sem filhos. Será sempre uma folha da árvore.
- Stop: o processo `stop`, como o processo `exit`, é representado num nó da árvore, sem filhos. Será também sempre uma folha da árvore.
- Prefixação (`[t1,t2]a; B`): é representada por um nó que contém a ação e seu intervalo de tempo. Aponta para um nó que é o comportamento que deve seguir a ocorrência da ação.
- Escolha (`B1[]B2`): a escolha é representada por um nó que a denota, apontando para dois filhos, cada qual é uma árvore representando a expressão de comportamento de um lado da escolha.
- Composição Paralela (`B1|[L]|B2`): para representar a composição paralela, é utilizado um nó que armazena as ações a sincronizar. Este nó aponta para dois

filhos, cada qual é uma árvore representando a expressão de comportamento de um lado da composição paralela.

- Composição Seqüencial (B1>>B2): também é representada na forma de um nó que a denota, apontando para as expressões de comportamento dos dois lados do operador. Preempção (B1 [>B2): Novamente representada por um nó com dois filhos, representando os dois lados do operador.

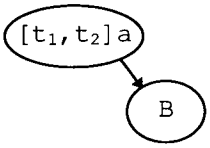
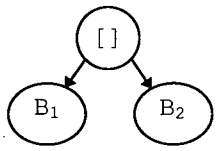
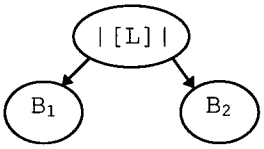
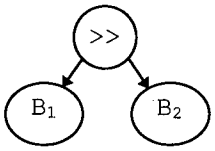
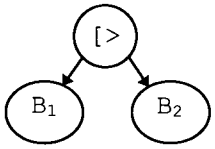
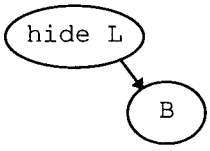
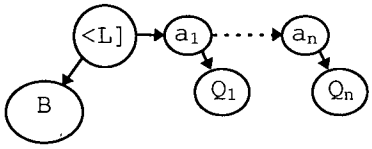
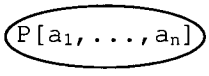
Inação stop	
Término com Sucesso exit	
Prefixação $[t_1,t_2]a;$ B	
Escolha $B_1[]B_2$	
Composição Paralela $B_1\|[L]\|B_2$	
Composição Seqüencial $B_1>>B_2$	
Preempção $B_1[>B_2$	
Ocultação hide L in B	
Preempção Temporal $B<L]\{a_1:Q_1,\ldots,a_n:Q_n\}$	
Instanciação $P[a_1,\ldots,a_n]$	

Tabela 3.1 - Representação interna dos operadores RT-LOTOS.

- Ocultação (`hide L in B`): Este operador é representado por um nó que contém as ações a serem ocultadas. Este nó aponta para a expressão de comportamento onde as ações ocorrem.
- Preempção Temporal (`B <L] {`a_1`:`Q_1`, . . . ,`a_n`:`Q_n`}`): A preempção temporal tem um representação mais complexa. O nó da árvore que representa este operador armazena a lista de ações cujas violações temporais serão tratadas. Este nó aponta para a expressão de comportamento onde estas ações irão ocorrer. Além disso, armazena uma lista, na qual cada elemento é a ação cuja violação temporal será tratada. Junto com esta ação encontra-se a árvore da expressão de comportamento que irá tratar a violação temporal.
- Instanciação (`P[a1,...,an]`): A instanciação é representada por um nó somente, que contém o nome do processo instanciado, e a lista de ações que tomarão o lugar das ações presentes na declaração do processo. Isto permite o *relabeling* das ações do processo.

Para ilustrar a construção da árvore que representa uma expressão de comportamento RT-LOTOS, será considerado o seguinte exemplo:

```
Exemplo := a; exit ||| c; exit [] b; Q[a]
```

A construção da representação interna desta expressão de comportamento é baseada na sua árvore abstrata, gerada pela ferramenta SINTAX. Cada nó da árvore abstrata é visitado duas vezes, uma na sua inserção na árvore (visita hereditária), e outra no término da construção da sua sub-árvore (visita de síntese). O processo é explicado com o acompanhamento da construção da representação interna do processo exemplo acima, que tem a árvore abstrata da figura 3.2

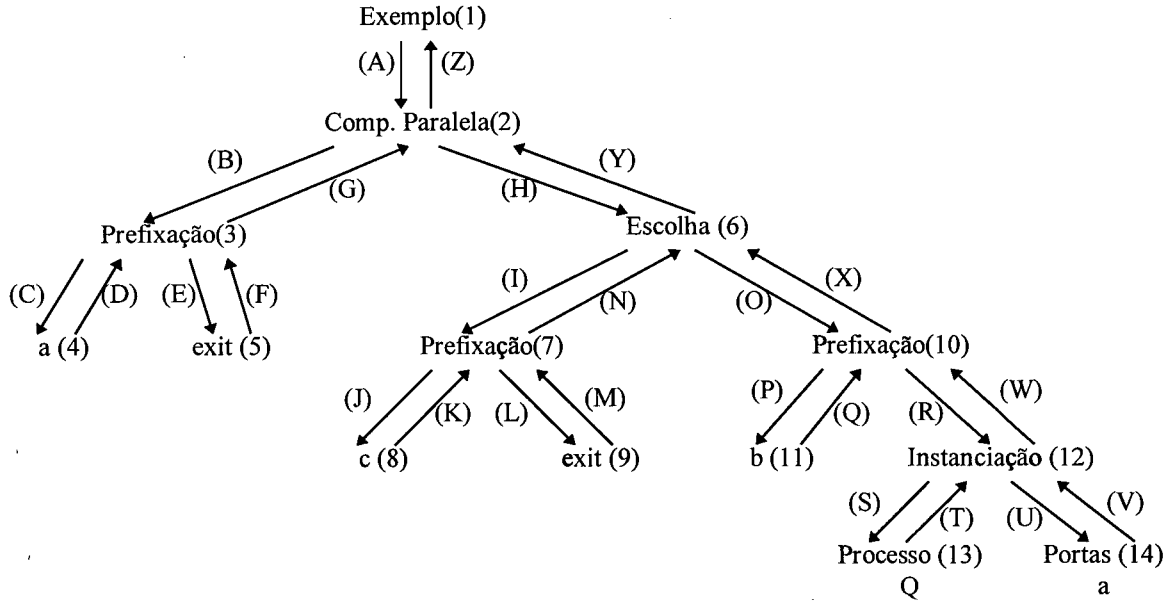


Figura 3.2 - Árvore Abstrata do Processo Exemplo

Na compilação, o programa gerado pela ferramenta SINTAX percorre os arcos da árvore abstrata na ordem alfabética apresentada na figura 3.2. Ou seja, de cima para baixo e da esquerda para a direita. O processo da construção da árvore de representação interna utiliza um pilha de nós genéricos, que são utilizados pelos nós seguintes da árvore. Esta pilha, no início da compilação, sempre conterá um nó inicial, que será a raiz da árvore.

Na primeira visita a cada nó da árvore abstrata, se o nó for um operador de RT-LOTOS, o nó genérico do topo da pilha é retirado, e atualizado para representar o operador sendo analisado. A seguir, novos nós são empilhados, dependendo da quantidade de operandos do operador. No exemplo acima, na visita ao nó 2, o nó genérico do topo da pilha é retirado, e atualizado para representar uma composição paralela. Dois novos nós são empilhados, para representar os dois lados da composição paralela. Na visita ao nó 3, o primeiro nó é desempilhado, atualizado para representar uma prefixação, e um novo nó é empilhado, para representar o comportamento seguinte à prefixação.

Quando as folhas da árvore abstrata são visitadas, apenas o último nó desempilhado é atualizado. As exceções são os nós `exit` e `stop`, que necessitam um desempilhamento antes da atualização. Por exemplo, quando o nó 5 for visitado, através de (E), teremos 2 nós genéricos empilhados, e a representação interna, ainda parcial, é a representada na figura 3.3:

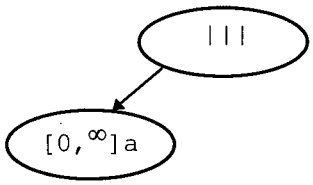


Figura 3.3 - Representação interna parcial do processo Exemplo.

Neste momento, um nó genérico é desempilhado, atualizado para representar o processo `exit` e conectado ao nó de prefixação. Assim, a representação parcial fica como representado na figura 3.4.

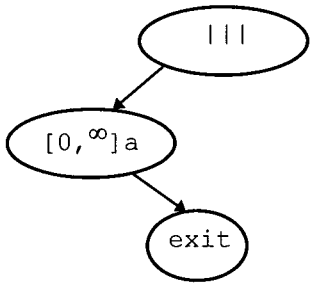


Figura 3.4 - Representação interna parcial do processo Exemplo.

A visita de retorno (síntese) dos nós na árvore abstrata é utilizada para ajustar o nó corrente da representação interna, o qual terá o próximo nó como filho. Assim, na volta ao nó 2 pelo arco (G), o nó corrente passa a ser o nó que representa a composição paralela. Na visita ao nó 6, por (H), o último nó é desempilhado, conectando como filho do nó corrente

e atualizado para representar uma escolha. Dois novos nós genéricos (os lados da escolha) são empilhados. Teremos assim a representação interna mostrada na figura 3.5.

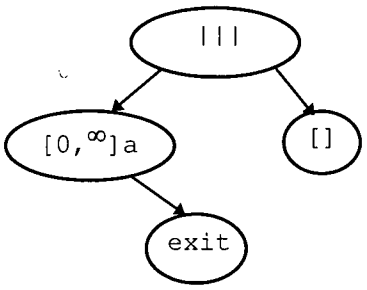


Figura 3.5 - Representação interna parcial do processo Exemplo.

O processo é repetido de forma análoga para todos os nós da árvore abstrata. Note-se que o processo é genérico. Quando um operador é analisado, não há preocupação com o tipo do operando, já que basta empilhar nós genéricos na mesma quantidade dos operandos.

Na visita a um nó representando uma instanciação, um nó genérico é desempilhado. A visita aos filhos da instanciação apenas atualiza os dados do nó desempilhado. No exemplo, a visita ao nó 12 desempilha o último nó genérico, ajustando-o para representar uma instanciação. A visita aos nós 13 e 14 apenas ajustam o nó da instanciação, com o nó 13 informando que o processo instanciado é o processo `Q`, e o nó 14 informando que a porta a ser passada na instanciação é a porta `a`.

Ao final da análise da árvore abstrata do processo Exemplo, a representação interna da expressão de comportamento (agora completa) será uma árvore como a representada na figura 3.6.

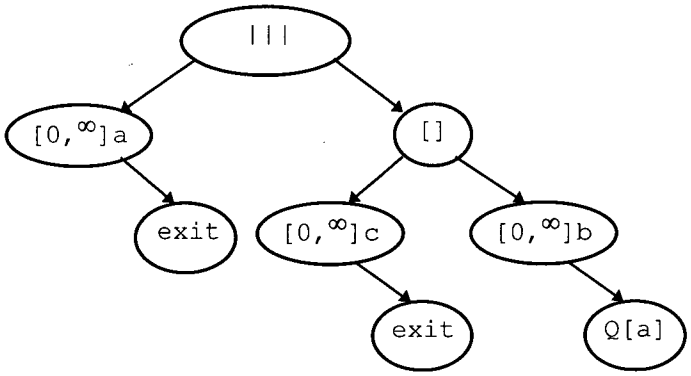


Figura 3.6 - Representação interna completa do processo Exemplo.

A representação interna é utilizada pela implementação das funções do núcleo do simulador, descritas formalmente na seção 3.2. Antes da aplicação das funções, as instanciações que não forem guardadas (com no mínimo uma ação antecedendo-as) são substituídas recursivamente pela expressões de comportamento que as definem. Caso seja detectada uma recursão nesta substituição, a simulação é interrompida, pelo fato de haver expressões não guardadas na especificação, o que é proibido.

A função `Actions` percorre a representação interna, determinando as ações que podem ocorrer. Possíveis substituição são feitas nesta lista, de acordo com a aplicação de cada operador. Por exemplo, a substituição de uma ação observável pela ação interna, num operador de ocultação, ou a criação de uma ação sincronizada a partir de duas ações sincronizáveis num operador de composição paralela.

A função `Menu` simplesmente percorre a lista criada pela expressão `Actions`, retirando apenas as ações com o limite inferior do intervalo igual a `0`.

A função `Next` é a mais complicada. Deve atualizar a árvore, refletindo a ocorrência de ações clássicas e de violações temporais. O mecanismo básico é o seguinte: para cada ação possível, é associado o nó da árvore onde a mesma ocorre (para as ações da composição paralela, são armazenados todos os nós onde a ação ocorrerá simultaneamente). Cada nó, depois de atualização, determina como a ação que ocorreu é vista pelo operador acima, e notifica-o. Desta forma, as múltiplas opções presentes na definição formal da função `Next` são decididas pela sinalização vinda da sub-árvore.

Por exemplo, a ocorrência da ação `c` na árvore da figura 3.6 substitui a prefixação pelo comportamento subseqüente (`exit`), e notifica o nó pai (escolha) que uma ação ocorreu na sub-árvore da esquerda. Este nó, de acordo com a definição da função `Next`, deve substituir toda a escolha pelo comportamento resultante do lado onde ocorreu a ação. Assim, a escolha será substituída pelo processo `exit`, e o nó da composição paralela será notificado que ocorreu a ação `c` do lado direito do operador. Como `c` não faz parte do conjunto de sincronização, e a composição paralela é o nó raiz, a atualização da árvore termina, e a representação interna do processo passa a ser a representada na figura 3.7.

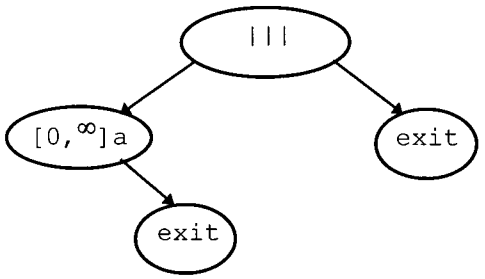


Figura 3.7 - Representação interna do Processo Exemplo depois da ação c.

As funções `MinTime` e `MaxTime` apenas percorrem a lista gerada pela função `Actions`. A função `DoTime` não altera a estrutura da árvore. Apenas a percorre, atualizando os intervalos das ações nas prefixações, de acordo com a definição formal da função.

Além das funções definidas na seção 3.2, o núcleo do simulador ainda implementa outras funções que manipulam diretamente a árvore. Estas funções são relacionadas à ação *undo* e à manipulação de estados, como sua marcação e a mudança para um estado já marcado.

A função `Undo` utiliza os nós retirados da árvore pela função `Next`, e os recoloca na árvore na ordem inversa em que foram retirados, restaurando assim o estado anterior à ocorrência da ação. A passagem do tempo é desfeita somente pelo retorno dos intervalos aos valores antigos.

A marcação de estados é feita através de cópias da árvore de representação interna da especificação. O retorno a um estado marcado é feito pela substituição da árvore da especificação pela cópia previamente guardada.

3.3.3 O Módulo de Simulação Interativa

O módulo de simulação interativa basicamente obtém comandos do usuário, gerando transformações sobre a expressão de comportamento sendo simulada. Este módulo obtém a lista de ações e transições de tempo possíveis do núcleo de simulação, e permite que o usuário escolha qual disparar.

Também permite que o usuário desfaça as últimas ações, bem como veja o traço das ações que o usuário disparou.

O usuário também pode ver o estado atual e a especificação original. A marcação de estados e o retorno a um estado previamente marcado são possíveis.

Todas as funções são implementadas no núcleo, a não ser a armazenagem do traço de ações, que é responsabilidade do módulo. O módulo de simulação interativa é apenas uma interface de leitura de comandos de simulação.

3.3.4 O Módulo de Simulação Automática

Este módulo, ao contrário do módulo de simulação interativa, apresenta algoritmos mais elaborados, de modo a guiar a simulação automática. Pode ser visto como sendo composto de dois sub-módulos, um para a geração de traços de simulação, e outro para a geração da árvore de execução.

A geração do traço de simulação segue os seguintes passos:

1. Obtenção das opções de traço, no que diz respeito ao momento de disparo, à ocorrência de violações temporais, à condição de parada da simulação e ao traço a ser gerado.
2. Para todas as ações prontas a disparar, é determinado o momento em que as mesmas deverão ocorrer, com base nas opções acima.
3. Se houver ação que deve disparar, a mesma é disparada. Se houver mais de uma, é feita a escolha aleatória de uma delas. O traço de simulação apropriado é gerado.
4. O tempo que pode transcorrer é determinado. Se houver uma ação que deve disparar antes do tempo mínimo obtido por `MinTime`, o tempo passa até o

momento do disparo. Caso contrário, o tempo mínimo passa, e as novas ações têm seu momento de disparo determinado pelas opções de simulação.

5. Os passos 3 e 4 são repetidos até que a condição de parada seja alcançada.

A geração aleatória do momento de disparo das ações segue uma distribuição de probabilidade uniforme. Futuramente, pretende-se implementar outras distribuições de probabilidade para o disparo das ações. Também não há mecanismos de se determinar prioridade de uma ação sobre as demais. Todas as ações que podem ocorrer num determinado instante tem a mesma probabilidade de serem disparadas.

A geração da árvore de execução obedece um algoritmo que procura disparar todas as combinações possíveis de ações. Em outras palavras, tenta explorar todas alternativas de execução possíveis. Porém, como pode haver traços de execução infinitos, é fornecido um mecanismo para limitar a profundidade desta exploração. Além disso, para cada novo estado, é feita uma verificação se existe um estado equivalente já explorado. Se já existir um estado igual, a exploração não é necessária, pois a mesmo já foi realizada. A geração da árvore de execução é detalhada mais adiante nesta mesma seção.

Um estado é igual a outro se ambos representarem a mesma expressão de comportamento, com operadores iguais, as mesmas ações e os mesmos intervalos de tempo associados. São feitas algumas simplificações nas expressões de comportamento dos estados, de modo a determinar equivalências. Estas simplificações são baseadas na relação de bissimulação temporal forte, definida em [Camargo95]. Esta relação de bissimulação é definida como:

Definição: Seja $\mathcal{L} = \mathcal{A}ct^{i,\delta} \cup \mathcal{A}ct^{*} \cup \mathcal{D}^{\infty}$ um conjunto de ações. Uma relação $\mathcal{R} \subseteq \mathcal{B} \times \mathcal{B}$ (ou seja, uma relação binária de comportamentos) é uma bissimulação temporal forte se e somente se: $\forall (P, Q) \in \mathcal{R}$, $\forall\ a \in \mathcal{L}$, temos:

1. sempre que $P \xrightarrow{a} P'$, então, para algum Q', $Q \xrightarrow{a} Q'$ e $(P', Q') \in \mathcal{R}$

2. sempre que $Q \xrightarrow{a} Q'$, então, para algum P', $P \xrightarrow{a} P'$ e $(P', Q') \in \mathcal{R}$.

As simplificações nos estados, feitas durante a geração da árvore de execução, são dadas pelas bissimulações temporais fortes abaixo. As provas são diretas, e omitidas neste trabalho.

- hide [L_1] in (hide [L_2] in B) $\sim_t$ hide [$L_1 \cup L_2$] in B
- hide [L] in stop $\sim_t$ stop
- hide [L] in exit $\sim_t$ exit
- exit ||| B $\sim_t$ B
- stop [] B $\sim_t$ B
- stop >> B $\sim_t$ stop
- stop <L>{a_1:Q_1,...,a_n:Q_n} $\sim_t$ stop

Desta forma, dois estados são equivalentes se, depois de aplicadas as simplificações acima, resultarem em comportamentos iguais, ou seja, com a mesma árvore de representação interna, descrita na seção 3.3.2.

Uma árvore de execução que, para uma determinada profundidade, tiver todos os seus ramos terminando num estado de *deadlock*, ou apontando para um outro estado equivalente, é uma *árvore completa*. Em outras palavras, é uma árvore que representa, ainda que de modo limitado no aspecto temporal, todas a possibilidades de evolução da especificação. Representa todas as possibilidades de ordem de ocorrência das ações, mas não em todos os momentos possíveis.

É importante ressaltar que não existe uma representação finita (seção 2.4) para especificações com comportamento infinito. Para tais especificações a árvore de execução, por maior que seja a profundidade exploradas, nunca chegará a um estado equivalente a outro previamente explorado. Desta forma, será impossível gerar uma árvore completa.

Na geração do traço de execução, cada estado é avaliado para determinar:

a) o conjunto de ações que podem ser realizadas naquele estado, e

b) o tempo máximo que pode passar, sem que uma nova ação seja sensibilizada ou que uma ação urgente tenha que ocorrer.

A exploração de cada estado da árvore, a partir do inicial, é feita em dois passos:

1. Para cada ação possível no estado, a mesma é disparada, e o estado resultante é explorado.

2. Se houver a possibilidade de passagem do tempo, é calculado o tempo máximo que pode passar no estado atual. Este cálculo obedece o seguinte critério: "*É o menor tempo para que se habilite uma nova ação, ou para que uma ação urgente ou uma violação temporal não permita a ocorrência de ações de passagem do tempo.*" Porém, o tempo máximo pode ser infinito, por exemplo, quando existem apenas ações observáveis não temporizadas. Neste caso, a exploração do estado pára depois do primeiro passo. Caso contrário, é feita a passagem do tempo máximo, e o estado resultante é explorado.

Desta forma, todas as possibilidades de evolução da especificação são cobertas. É importante ressaltar que uma ação temporizada é disparada no mínimo duas vezes: no seu limite inferior e no seu limite superior. No limite superior, também a ocorrência da violação temporal é explorada. Para ações não temporizadas, o disparo é feito no primeiro instante em que a ação é oferecida. As ações internas são disparadas no limite inferior e no limite superior. Além disso, todas as ações podem ser disparadas mais vezes, quando houver outras ações sendo habilitadas durante seu intervalo de tempo.

Existem alguns aspectos na geração da árvore de execução, relativos ao operador de composição paralela, que devem ser discutidos.

O *interleaving* das ações não pertencentes ao conjunto de sincronização faz com que as ações sejam disparadas no meio de seu intervalo de tempo, além de serem disparadas no limite inferior e no limite superior. Um exemplo simples é a expressão de comportamento

```
[5,10]a; exit ||| [6, 9]b; exit
```

Neste exemplo, a ação a será disparada no instante 5 (seu limite inferior), no instante 6 (quando a habilitação de b determinou o tempo máximo), no instante 9 (quando a violação temporal de b determinou o tempo máximo) e no instante 10 (seu limite superior). Já a ação b será disparada somente nos seu limite inferior e superior.

A sincronização das ações temporizadas é garantida somente quando o intervalo de tempo global das mesmas apresentar uma interseção. Este intervalo de tempo global é determinado pela soma dos limites inferiores e superiores das ações que precedem a ação a sincronizar.

Seja a seguinte composição paralela genérica:

$[t_a^1, t_b^1]a_1; [t_a^2, t_b^2]a_2; \ldots; [t_a^n, t_b^n]a_n; [t_1, t_2]c; B1$

$|[c]|$

$[t_c^1, t_d^1]b_1; [t_c^2, t_d^2]b_2; \ldots; [t_c^m, t_d^m]b_m; [t_3, t_4]c; B2$

A sincronização somente será possível se houver uma interseção dos intervalos de tempo definidos por:

$[T_a + t_1, T_b + t_2]$, onde $T_a = \sum_{i=1}^{n} t_a^i$ e $T_b = \sum_{i=1}^{n} t_b^i$

$[T_c + t_3, T_d + t_4]$, onde $T_c = \sum_{i=1}^{m} t_c^i$ e $T_d = \sum_{i=1}^{m} t_d^i$

Em outras palavras, deve haver uma interseção entre as evoluções mais rápidas e mais lentas dos dois lados da composição paralela. Para exemplificar, serão analisados dois exemplos simples.

Seja o primeiro exemplo o processo definido por:

```
[5,10]a; [0,5]c; exit |[c]| [10,20]b; [6,10]c; exit
```

Esta expressão de comportamento não poderá evoluir pela sincronização de da ação c, pois no lado esquerdo da composição paralela, a ação c poderá realizar-se no intervalo de tempo global [5, 15] (calculado pela fórmula acima, [5+0, 10+5]). Já no lado direito do operador, a ação c será oferecida no intervalo de tempo global [16, 30]. Como os intervalos são disjuntos, não há possibilidade de sincronização.

Considere-se agora um processo definido como:

```
[5,10]a; [5,5]b; [10,15]c; exit
|[c]|
[10,15]b; [6,10]c; exit
```

Neste caso a sincronização é possível, pois existe uma interseção do intervalo global da ação c em cada lado da composição paralela. Estes intervalos são [20, 30] no lado esquerdo do operador, e [16, 25] no lado direito. Desta forma, o intervalo de sincronização será [20, 25]. A ferramenta irá disparar a ação sincronizada nestes dois instantes, no mínimo. Outros disparos podem ser possíveis, devido ao *interleaving*, conforme discutido acima. Pode-se assim garantir que, quando a sincronização for possível, a mesma aparecerá no mínimo uma vez na árvore de execução, quando a interseção do intervalo for um instante apenas, do tipo [t, t]. Ou duas vezes, quando o intervalo for do tipo $[t_1, t_2]$, $t_1 \neq t_2$.

A inclusão de ações não temporizadas (com intervalo [0, ∞]) não impede a sincronização, pois a mesma será disparada no primeiro instante em que é oferecida. Assim, as somatórias acima não se alteram. Porém, a ferramenta pode deixar de explorar possíveis caminhos de execução, que tornam-se possíveis somente quando uma ação não temporizada é atrasada num tempo determinado. Tome-se por exemplo a expressão de comportamento:

```
a; [10, 20]c; exit |[c]| [35, 40]c; exit
```

Neste processo, a análise acima indica que o intervalo de sincronização é disjunto, uma vez que o intervalo da ação a não muda as somas dos tempos. Realmente, se a ação a ocorrer no instante 0, a sincronização não será possível. Porém, se a ação a for retardada entre 25 e 30 unidades de tempo, a sincronização torna-se possível.

A geração da árvore de execução não determina esta possibilidade, pois a análise da evolução das ações é feita somente sobre as ações habilitadas. Não é feita nenhuma análise dos comportamentos futuros, tentando ajustar intervalos de tempo. Isto porque podem haver infinitas possibilidades de evoluções futuras que exijam ajustes diferentes de tempo em ações não temporizadas. Além disso, uma expressão de comportamento como a do exemplo anterior não representa nenhum processo útil na especificação de sistemas dependentes de tempo, uma vez que a ocorrência da sincronização é totalmente indeterminística.

Pode-se concluir assim que somente as expressões de comportamento que exijam um ajuste de tempo em ações não temporizadas, de modo a garantir uma sincronização futura, não serão exploradas exaustivamente pelo algoritmo de geração da árvore de execução.

3.4 Conclusões

Neste capítulo foram apresentadas as alternativas de implementação de um simulador de especificações RT-LOTOS. Foi escolhida a alternativa da simulação através da

manipulação direta das especificações, por poder tratar especificações com comportamentos infinitos.

As funções principais de um núcleo para simulação de especificações RT-LOTOS foram identificadas e definidas, de modo a poder avaliar e atualizar as especificações, refletindo a evolução do comportamento através da ocorrência de ações e pela passagem do tempo.

Em seguida, a arquitetura interna de uma ferramenta de simulação foi apresentada, suportando simulação interativa e automática. A simulação automática suporta a geração de traços de simulação e a geração de uma árvore de execução. Os algoritmos da simulação automática foram discutidos.

No próximo capítulo, as funcionalidades da ferramenta serão apresentadas do ponto de vista do projetista. As funções do simulador serão apresentadas, acompanhadas de considerações do seu uso para a validação de especificações RT-LOTOS.

Capítulo 4

A Ferramenta RTLS

A ferramenta de simulação, apresentada neste trabalho, é denominada RTLS (RT-LOTOS Simulator), e complementa a ferramenta para análise de sistemas dependentes de tempo descritos em RT-LOTOS apresentada em [Martins96]. O simulador apresenta uma série de funcionalidades, descritas neste capítulo, que permitem a validação de propriedades da especificação. Ambas as ferramentas têm seu desenvolvimento inserido no projeto ProTeM III intitulado DAMD - Design de Aplicações Multimídia Distribuídas. Um dos objetivos deste projeto é o desenvolvimento de um conjunto de ferramentas, utilizadas na verificação e validação de especificações RT-LOTOS. Em especial, especificações de sistemas multimídia, as quais, por sua natureza, apresentam restrições de tempo que devem ser tratadas explicitamente.

A ferramenta RTLS apresenta basicamente duas funcionalidades, a serem utilizadas na validação de especificações RT-LOTOS, descritas a seguir:

- Simulação Interativa: em cada estado da especificação, é possível a escolha de uma das ações a disparar, entre as ações clássicas, as violações temporais e a passagem do tempo. A cada disparo é alcançado um novo estado. A ferramenta permite a marcação de estados da especificação. É possível o retorno a estes estados posteriormente, e a exploração de outro traço de simulação. Permite-se ainda a visualização do estado corrente da especificação, dado na forma de uma expressão de comportamento RT-LOTOS. A simulação interativa é crucial para a depuração das especificações, já que permite acompanhar a evolução da especificação passo a passo.
- Simulação Automática: permite que se gere um traço de simulação da especificação, com diferentes parâmetros. Estes parâmetros definem o momento de disparo das ações, a ocorrência ou não das violações temporais, o tamanho do traço de simulação a ser gerado, e também o tipo de traço a ser mostrado. É possível assim analisar a evolução da especificação sob diversas condições. A simulação automática também permite a geração de uma árvore de execução da especificação, de modo que todas as possíveis execuções até uma determinada

profundidade de execução sejam mostradas. É obtida assim uma representação de todos os comportamentos possíveis da especificação.

A seguir, todas as funcionalidades da ferramenta serão descritas, de modo mais detalhado. A utilização de cada funcionalidade é analisada do ponto de vista da validação de sistemas dependentes de tempo descritos em RT-LOTOS.

4.1 Simulação Interativa

A simulação interativa permite a validação de especificações através do acompanhamento, passo a passo, da evolução do comportamento das mesmas. Os disparos de todas as transições são controlados, permitindo assim um controle total da evolução da especificação. Este controle e acompanhamento permite que os pontos exatos da especificação onde ocorram possíveis erros sejam encontrados.

Na medida em que a especificação formal é uma maneira de *descrever* um sistema, uma ferramenta de simulação também pode ser utilizada como um *protótipo*. Acompanhando a evolução da especificação, através da simulação, programadores podem aprender exatamente como o sistema deve se comportar. Assim, um simulador pode ser visto também como uma ferramenta para o aprendizado dos requisitos do sistema, para auxiliar as pessoas que irão fazer a implementação real.

Na simulação interativa, a ferramenta RTLS permite que se escolha as ações da especificação que devem acontecer, entre ações clássicas e violações temporais. Se não houver nenhuma ação urgente, é possível realizar transições de passagem do tempo. Os disparos de ações e a passagem do tempo podem ser desfeitos. A cada disparo de ação ou tempo, o estado da especificação é atualizado. O estado atual pode ser visualizado, na forma de uma especificação RT-LOTOS. Também a especificação original pode ser visualizada. O traço de disparo de ações, que levou a especificação até o estado atual, pode ser inspecionado. Em qualquer ponto da especificação, o estado atual pode ser marcado para posterior retorno, de modo a permitir a exploração de outros caminhos de simulação.

No início da simulação, é permitida a escolha entre a simulação da especificação completa ou apenas de um processo da mesma. A simulação da especificação completa implica na simulação da expressão de comportamento da cláusula behaviour da especificação. A simulação de um processo faz com que a declaração do processo seja considerada a especificação. Os nomes das portas da declaração do processo passam a valer.

A ferramenta permite assim, os seguintes comandos na simulação interativa:

- Actions/Time: permite ver a lista de ações possíveis no momento na especificação. Também permite a realização de ações de passagem do tempo, quando estas puderem ocorrer.

- Undo: permite desfazer as últimas ações. *Undos* consecutivos levam de volta ao estado de onde se iniciou a simulação.
- State: permite ver o estado atual da especificação. Este estado é mostrado na forma de uma expressão de comportamento RT-LOTOS.
- Specification: permite ver a especificação original que está sendo simulada.
- Trace: permite ver a lista das ações pelas quais a especificação evoluiu, desde o estado de onde começou a simulação até o estado atual.
- Mark State: marca o estado atual para posterior retorno. O estado marcado recebe um número, que é o número seqüencial de estados previamente marcados. O primeiro recebe o número 1, o segundo o número 2, e assim por diante. Este número deve ser informado num posterior retorno ao estado.
- Go To State: vai para um estado previamente marcado. Para isto, deve ser informado o número que o estado recebeu quando da sua marcação.

Todas as possibilidades da simulação interativa são agora descritas em detalhes. Para isto será feito o acompanhamento de uma pequena especificação modelo. Tal especificação tenta mostrar todos os operadores de RT-LOTOS. A mesma pode ser vista na figura 4.1

```
specification Teste[a, b, c, d, e]
behaviour
     P[a,b,c] |[a]| Q[a,d,e] <a,b] {a: c; exit, b: P[a,b,c]}
where
     process P[x,y,z] :=
           hide [z] in ([0,10]y; [5,10]x; z; exit)
     endproc
     process Q[l,m,n] :=
           (    [5,20]m; l; exit
            [] [5,10]n; l; exit) >> [10,20] n; exit
     endproc
endspec
```

Figura 4.1 - Especificação RT-LOTOS exemplo

Na simulação desta especificação, as ações serão vistas com os nomes `a`, `b`, `c`, `d` e `e`. Já na simulação do processo `P`, por exemplo, a evolução da especificação seria vista como ocorrendo através das ações `x`, `y` e `z`. E somente o comportamento interno de `P` seria simulado.

4.1.1 Ações e passagem do tempo

Para a escolha da alternativa de evolução da especificação, é apresentada uma lista das ações que podem ser disparadas no estado atual da especificação. Estas ações incluem as ações clássicas, bem como as ações de violação temporal. As ações clássicas são representadas por seu nome. As ações de violação temporal são representadas pelo nome da ação seguida

de um asterisco. Já as ações internas (a ação especial `i` ou as ações ocultas) são representadas pela ação `i`, seguidas de seu nome original entre parênteses.

Para a especificação exemplo da figura 4.1, no estado inicial, a lista de ações a disparar é a seguinte:

```
[0, 10]b
```

Além do disparo das ações, também é possível permitir a passagem do tempo na especificação. Para isto, o simulador informa, quando for o caso, dois limites de tempo. O primeiro é limite mínimo, antes do qual não haverá mudanças na lista de ações a disparar. O segundo é o limite máximo, no qual haverá uma ação urgente que deverá obrigatoriamente ser disparada. Para uma verdadeira evolução da especificação, o tempo a passar deve estar compreendido entre estes dois limites. Se o tempo for menor que o limite inferior, não haverá mudança no conjunto de ações que a especificação pode realizar. Apenas os intervalos de tempo mudarão, para as ações que já eram oferecidas. Um tempo maior que o limite superior não é permitido.

Continuando com simulação da especificação do exemplo da figura 4.1, caso se deseje uma passagem do tempo antes de qualquer ação, o simulador informa que o tempo a passar está entre 5 e 10 unidades de tempo. Passando 5 unidades de tempo, o conjunto das ações possíveis na especificação passa a ser:

```
[0, 5]b
[0, 15]d
[0, 5]e
```

A ferramenta RTLS permite também a inspeção do traço de execução. Um comando de *trace* faz com que a ferramenta mostre as ações que ocorreram, precedidas do momento da ocorrência. A passagem do tempo é denotada pela quantidade de tempo transcorrida. Supondo que se continue a simulação da especificação exemplo, disparando a ação `d`, seguida da ação `b` e da passagem de 5 unidades de tempo. Neste caso, o traço ficaria:

```
<0 - 5>
<5 - e>
<5 - b>
<5 - 5>
```

O simulador RTLS permite a visualização do estado atual da especificação. Ainda no exemplo sendo seguido, o comando para visualizar o estado atual da especificação, após o traço acima, resultaria na seguinte expressão de comportamento:

```
(hide [c] in [0, 5]a; c; exit)
|[a]|
(a; exit >> [10,20]e; exit) <a,b]{a:c; exit, b:P[a,b,c]}
```

É importante notar que todas as instanciações de processo necessárias para a avaliação da expressão de comportamento foram feitas. As instanciações são feitas na medida em que se tornam necessárias, para uma melhor visualização dos estados. Quando

for necessário ver a declaração dos processos, basta utilizar a opção da simulação interativa que mostra a especificação original. A mesma apenas imprime a especificação dada como entrada para a ferramenta, com todas as declarações de processo.

Para ilustrar a ocorrência das violações temporais, basta permitir a evolução da especificação por mais 5 unidades de tempo, a partir do estado anterior. Depois disto, a lista de ações possíveis será:

```
[0, 0]a
[0, 0]i(a*)
```

A violação temporal está representada por uma ação interna, pois a mesma é tratada pelo operador de violação temporal. Caso contrário, a mesma apareceria como uma violação normal, sem a indicação de ação interna. A ferramenta não faz o disparo automático das violações temporais. Seria possível permitir que se determinasse a passagem de uma quantidade arbitrária de tempo. O simulador se encarregaria de disparar todas as ações urgentes e violações temporais, que ocorreriam dentro deste intervalo. Porém, como é desejada a maior interatividade possível, decidiu-se que todas as ações devem ser determinadas pelo usuário da ferramenta.

Após o disparo a violação temporal, o estado da especificação será:

```
c; exit
```

Este é exatamente o expresso no operador de preempção temporal da especificação exemplo.

Os disparos de ações e a passagem do tempo podem ser desfeitas pela operação de *undo*, oferecida pela ferramenta. Isto implicará na volta da especificação ao estado anterior à ocorrência da ação ou da passagem do tempo. *Undos* consecutivos são permitidos. Considerando o exemplo sendo simulado, dois comandos de *undo* farão com que a especificação volte ao penúltimo estado mostrado. A violação temporal e a passagem do tempo foram desfeitas.

4.1.2 Marcação e visualização de estados

A ferramenta RTLS permite que, durante a simulação, estados da especificação sejam marcados, para posterior retorno. Esta funcionalidade permite que caminhos alternativos sejam seguidos, sem a necessidade de se reiniciar toda a simulação.

Quando da marcação de um estado, é informado ao usuário o número interno que este estado recebeu (um número seqüencial aos estados já marcados anteriormente). Quando quiser retornar a um determinado estado, o usuário deverá informar o número do mesmo.

Na especificação exemplo sendo simulada, foi verificado o que acontece quando da ocorrência da violação temporal da ação a. Posteriormente, foram realizados dois comandos de *undo* para voltar ao estado onde a começou a ser oferecida. Ao invés destes *undos*, o

estado onde a começou a ser oferecida poderia ter sido marcado. Com isto, o estado receberia o número 1, uma vez que foi a primeira marcação de estado. Após toda a exploração do comportamento que envolve a violação temporal de a, ao invés dos *undos* consecutivos, poderia ter sido utilizada a volta ao estado marcado, bastando informar o número do mesmo. Para especificações grandes esta funcionalidade é importante para exploração de comportamentos alternativos.

No momento da mudança de estado, as listas de traço e de *undo* são eliminadas. Pode-se dizer que a simulação reinicia, como se o estado para o qual o usuário retornou fosse uma nova especificação. Desta forma, o traço de simulação sempre diz respeito ao estado do qual se reiniciou a simulação. Enquanto não for feita nenhuma mudança de estado, o traço diz respeito ao estado inicial da especificação original. Esta restrição é feita para obter-se economia de memória para especificações grandes, pois seria necessário armazenar as listas de traço e *undo* para cada estado marcado.

Voltando novamente ao exemplo da figura 4.1 sendo simulado, caso houvesse sido utilizada esta funcionalidade da ferramenta, quando da mudança de estado, a lista de *undos* e o traço seriam esvaziados. Para o simulador, é como se a simulação recomeçasse. Esta restrição é necessária por motivos de economia de recursos computacionais (memória) na simulação. Como a mudança de estados pode ser feita de forma aleatória, para frente e para trás, manter estas listas implicaria em duplicá-las. Este consumo de recursos de máquina foi evitado, para utilizá-los em aspectos mais importantes da simulação.

4.2 Simulação Automática

A simulação automática permite que se faça a exploração de um ou mais caminhos da especificação sem a intervenção do usuário. Permite que se automatizem alguns testes da especificação, de modo que o usuário não tenha que fazer a simulação manualmente.

Dois tipos de simulação automática são possíveis: a *geração de um traço de simulação*, ou a *geração de uma árvore de execução*. A geração de um traço é mais rápida, já que uma seqüência aleatória de disparos é gerada, sendo que sua análise fica a cargo do usuário. Já a árvore de execução faz uma análise de todos os caminhos possíveis de execução para uma especificação, detectando recursividade. Esta funcionalidade faz uma exploração, a partir do estado inicial da especificação, de todos os traços de simulação possíveis. Cada ramo da árvore é explorado até se encontrar uma recursão, um *deadlock*, ou a profundidade especificada pelo usuário.

Neste capítulo apenas serão descritas as diversas opções da simulação automática, sem a apresentação de exemplos. A utilização das funcionalidades da simulação automática podem ser vistas no capítulo 5, onde serão feitos alguns estudos de caso.

As duas formas de simulação automática são descritas com mais detalhes a seguir.

4.2.1 Geração de Traços

A geração de traços de simulação é feita baseada na geração de valores aleatórios, que irão determinar o momento de ocorrência de cada ação. Também é utilizado um número aleatório para escolher entre duas ou mais ações que possam ocorrer no mesmo instante. Para a geração dos números aleatórios, é utilizado a função da biblioteca padrão da linguagem C chamada *random(n)*, onde *n* é o valor máximo no número a ser gerado. Os números aleatórios são gerados sobre uma semente (um número natural) que os determina. Para a mesma semente, a mesma seqüência de números é gerada. Na simulação, esta semente é fornecida pelo usuário. Desta forma, seqüências de simulação podem ser repetidas, bastando para isso apenas que o usuário informe a mesma semente.

Para a geração de traços de ações da especificação, o usuário pode determinar parâmetros no que diz respeito ao momento de ocorrência das ações temporizadas, à ocorrência ou não das violações temporais, à condição de parada da simulação, bem como ao traço a ser fornecido pelo simulador. Estas opções são descritas sucintamente a seguir.

Com relação ao momento de ocorrência de uma ação, o usuário tem a opção de fazer com que as ações ocorram nos seguintes instantes do seu intervalo de tempo associado:

- Limite inferior do intervalo: as ações são disparadas no primeiro instante em que são sensibilizadas. Esta opção faz com que seja gerado um traço que representa a evolução do sistema no menor tempo possível.
- Limite superior do intervalo: as ações são retardadas ao máximo dentro do seu intervalo de tempo associado. Assim, o traço gerado representa a evolução do sistema no maior tempo possível. Para ações não temporizadas, o usuário deve informar um tempo máximo para sua ocorrência. Isto para que não haja estouro das variáveis internas de controle de tempo. Esta é uma limitação computacional, e não da semântica da linguagem.
- Instante aleatório dentro do intervalo: é gerado um instante aleatório para o disparo da ação. Esta geração aleatória obedece uma distribuição de probabilidade uniforme. Esta opção gera um traço de execução que representa uma evolução da especificação que dura um tempo qualquer entre a progressão máxima e o retardo máximo das ações.

Também é oferecido o controle sobre a ocorrência das violações temporais. As opções para o controle das violações temporais são as seguintes:

- Violações temporais nunca ocorrem: na evolução da especificação as violações temporais nunca ocorrem. Mesmo no limite superior, sempre é disparada a ocorrência da ação, e não da violação temporal. Desta forma, pode-se analisar a evolução da especificação se o ambiente estiver sempre pronto para a sincronização. Ou seja, a evolução normal da especificação.

- Violações temporais sempre ocorrem: ao contrário da opção anterior, sempre que houver a possibilidade de ocorrência de uma violação temporal, a mesma é disparada. Desta forma, o traço gerado reflete o caso em que o ambiente não está pronto para sincronizar-se. Permite a análise do comportamento da especificação quando há uma falha geral do ambiente.
- Violações temporais ocorrem aleatoriamente: com esta opção as violações temporais podem ou não ocorrer, aleatoriamente. Desta forma, é gerado um traço que permite a análise do comportamento da especificação num ambiente em que pode estar pronto ou não para a sincronização.

A geração de um traço de simulação irá gerar uma seqüência de ações, até encontrar uma condição de parada. Esta condição pode ser um estado de *deadlock*, no qual nenhuma ação está pronta para ocorrer, nem existem ações que estão esperando para que o limite mínimo do seu intervalo seja alcançado. Neste caso o sistema informa se a especificação terminou com sucesso (por um processo `exit`) ou não (*deadlock* por violação temporal, por exemplo).

Também é possível especificar o comprimento do traço que se deseja gerar. Desta forma, o traço será gerado até a especificação chegar a um estado de *deadlock*, ou até que uma das seguintes condições for alcançada, o que acontecer primeiro:

- Tempo transcorrido: a simulação segue até que o tempo transcorrido seja igual ao tempo especificado pelo usuário.
- Tamanho do traço: o usuário determina o número de ações que quer sejam disparadas. Após este número de disparos, a geração do traço de simulação pára.
- Indefinidamente: a geração do traço pára somente com a especificação chegando a um estado de *deadlock*. Se a especificação não chegar a um *deadlock*, a simulação continua indefinidamente, até que a capacidade de memória da máquina seja excedida, por exemplo.

Para o acompanhamento do traço de simulação, o usuário pode ajustar o tipo de saída que deseja que o simulador forneça. O traço de execução pode ser ajustado para mostrar os disparos das ações e do tempo de acordo com as seguintes opções:

- Todas ações e tempo: todos os disparos de ações e de tempo são mostradas. Permite que a evolução detalhada da especificação seja acompanhada, durante a geração do traço.
- Ações observáveis e tempo: somente as ações observáveis são mostradas, e as transições de tempo. As ações internas não são mostradas, escondendo a evolução interna da especificação.

- Somente ações observáveis: para restringir um pouco o traço gerado na opção anterior, pode-se retirar as transições de tempo. Isto é feito por esta opção, que mostra apenas a ocorrência das ações observáveis da especificação.
- Ações selecionadas: o usuário pode especificar uma lista de ações, cuja ocorrência quer inspecionar. O traço de simulação irá apresentar apenas estas ações, na ordem em que ocorrerem. Nomes de ações que não existirem na especificação são ignoradas, após um aviso ao usuário. Esta opção é útil para analisar a periodicidade de uma ação, ou o intervalo de tempo entre duas ações, por exemplo.

A utilização do traço de simulação para a análise de especificações RT-LOTOS depende da combinação das diversas opções disponíveis. É importante ressaltar que as diferentes opções não interferem entre elas.

Por exemplo, se for escolhida a opção de disparar as ações no limite inferior, e a opção de disparar sempre as violações temporais, somente as ações com intervalo pontual (do tipo [t, t]) associado irão ter sua violação temporal disparada. Para gerar um traço que simule a falha geral do ambiente, é necessário utilizar o disparo no limite superior dos intervalos, juntamente com a opção de disparar sempre as violações temporais.

4.2.2 Árvore de Execução

A árvore de execução permite representar, de uma forma simples, todos os comportamentos possíveis de uma especificação. Ao contrário da simulação interativa e da geração de traços, a árvore de execução permite uma análise exaustiva do comportamento.

Por exemplo, uma especificação que deveria ter um comportamento infinito, ou seja, livre de *deadlocks*, pode ter um traço de simulação em que um *deadlock* ocorre. Na simulação interativa, mesmo que o usuário se esforce ao máximo para cobrir todas as possibilidades de execução, pode acabar deixando de lado justamente o caminho que leve a um *deadlock*. Na geração da árvore de execução este problema não ocorre, pois todas as possibilidades de execução de ações observáveis, internas e violações temporais são examinadas, com as simplificações dos aspectos temporais descritas na seção 3.3.4.

A árvore de execução pode representar todos os comportamentos possíveis da especificação, na medida em que o comportamento da especificação possa ser representado de forma finita. A cada novo estado gerado pela ocorrência de uma ação ou pela passagem do tempo, é verificado se não existe um estado equivalente já analisado. Se houver, uma referência é feita.

Assim, se todos os ramos da árvore de execução terminarem num *deadlock* ou apontando para um estado em outro lugar da árvores, todos os comportamentos possíveis da especificação estarão representados na árvore, podendo o usuário analisar a sua

especificação em todos os aspectos comportamentais e temporais, validando os requisitos da mesma.

É evidente que comportamentos infinitos, que não apresentem recursão de comportamento (as transições acabam levando de volta para um estado já explorado), não podem ser analisados, nem mesmo utilizando a árvore de execução. Isto se deve ao fato de que, para comportamentos infinitos não recursivos, a árvore de execução também é infinita. Note-se que comportamentos infinitos recursivos têm uma representação finita do seu comportamento. Já para comportamentos infinitos não-recursivos não é possível gerar uma representação finita, o que inviabiliza sua análise.

Cada linha na árvore de execução pode estar em uma das quatro formas abaixo:

`Nº do estado - ação disparada (tempo)`: o número do estado indica em qual estado a ação ocorreu. Todas as linhas que começarem com o mesmo número identificam os disparos de ações e tempo que são possíveis num mesmo estado. A ação disparada é a transição que leva ao próximo estado. O tempo, entre parênteses, é o momento, a partir do início da especificação, em que a ação ocorreu. Por exemplo, uma entrada na árvore da forma

```
35 - lower (50)
```

indica que no estado 35 a ação `lower` foi disparada 50 unidades de tempo após o início do traço que leva até o estado 35.

`Nº do estado - recursion detected - estado (tempo)`: indica que existe um estado igual no traço de execução que leva até o estado sendo analisado. O número deste estado é informado depois da indicação de recursão. O tempo, entre parênteses, é o tempo que transcorreu até a chegada ao estado atual. Para continuar a análise, deve-se passar para o estado equivalente, substituindo os tempos das ações pelo tempo indicado, e atualizando os tempos dos estados subseqüentes. Por exemplo, a seguinte linha na árvore de execução:

```
37 - Recursion detected - 25 (150)
```

indica que o estado 37 é igual ao estado 25. A partir do estado 25 é possível disparar uma seqüência de ações e passagem do tempo que leva ao estado 37. Para continuar a análise do estado 37, deve-se retornar ao estado 25, substituindo os tempos no mesmo por 150. Os tempos dos estados alcançados partir do estado 25 devem ser atualizados. Esta atualização é feita somando-se com 150 a diferença do momento de ocorrência das ações com o tempo das ações no estado 25. Por exemplo, se o tempo no estado 25 for 30, deve ser substituído por 150. Se o tempo num estado seguinte ao estado 25 for 80, deve-se somar 50 (80 - 30) a 150, ou seja, o novo tempo deste estado será de 200. Como é uma recursão, pode-se afirmar que o estado 37 será alcançado novamente nos instantes 270 (150+ (150-30)), 390, 510, e assim por diante.

Nº do estado - Analized elsewhere - estado (tempo): é uma indicação idêntica à indicação de recursão, porém o estado equivalente não leva ao estado atual, por nenhuma seqüência de disparos. São estados equivalentes da especificação alcançados por caminhos diferentes. A análise é igual à análise explicada acima, com a substituição dos tempos.

Nº do estado - Deadlock (tempo): indica que o estado atual é um estado de deadlock. Ou seja, nenhuma ação é mais possível. Se a transição que leva até este estado for rotulada com EXIT, é um estado no qual a especificação terminou com sucesso. Senão, é um estado no qual a especificação não pode realizar nenhuma ação, podendo ser representada pelo processo stop. É o caso de uma violação temporal sem tratamento, por exemplo.

A análise da árvore de execução será exemplificada no capítulo 5, com os estudos de casos.

4.3 Conclusões

Neste capítulo, foram apresentadas as funcionalidades da ferramenta RTLS, do ponto de vista da validação de especificações RT-LOTOS. A simulação interativa foi mostrada com o acompanhamento de uma pequena especificação exemplo. Pode-se concluir que a simulação interativa é útil para a depuração de especificações, pois é possível um acompanhamento detalhado da evolução da especificação. Também pode ser utilizada para transmitir os requisitos do sistema para os responsáveis pela implementação da especificação.

Já as funcionalidades de simulação automática foram apenas descritas, ficando sua utilização para o capítulo seguinte, onde serão feitos estudos de caso. Estes estudos de caso irão ilustrar a utilização da simulação automática para a validação de especificações. A validação de restrições sobre as ações pode ser feita utilizando-se a combinação das opções de simulação automática, gerando traços que confirmam ou não a satisfação das restrições. A árvore de execução pode ser utilizada para a validação de requisitos mais genéricos, tais como a não ocorrência de deadlocks.

A utilização das funcionalidades de simulação automática para a validação de propriedades das especificações ficará mais clara com os exemplos discutidos no capítulo 5.

Capítulo 5

Exemplos de Aplicação da Ferramenta de Simulação

Neste capítulo, algumas aplicações são apresentadas, de modo a ilustrar a utilização das funcionalidades da ferramenta de simulação apresentadas no capítulo anterior. A simulação automática será utilizada para a validação das propriedades das aplicações.

As aplicações são as mesmas apresentadas em [Martins96]. Como as especificações são ambas em RT-LOTOS, há a possibilidade de se confrontar os resultados obtidos pela verificação, através de Grafos Temporizados e Lógica TCTL, com os resultados obtidos pela simulação automática e pela árvore de execução. Também é comparada um especificação com comportamento infinito, que apresenta a mesma funcionalidade, porém não pode ser analisada com a ferramenta apresentada em [Martins96]. Pretende-se assim mostrar a complementaridade das ferramentas.

Todos os exemplos foram executados em um computador do tipo PC 486 DX4, 100Mhz, com 24 megabytes de memória, rodando o sistema operacional Linux com *kernel* versão 2.0.0.

5.1 O Protocolo Tick-Tock

O exemplo a seguir foi descrito inicialmente em [LLD94] e verificado também em [DOY94]. Trata-se de um protocolo de comunicação simples, composto de três entidades principais: *sender, receiver* e *service*. A validação se dará somente sobre a especificação se *service*, um vez que aí estão presentes as restrições temporais básicas do protocolo.

5.1.1 Descrição Informal do Protocolo

O envio de informações através da entidade *service* se dá através da interação das entidades *sender* e *receiver*, nos pontos de comunicação SS-SAP e SR-SAP. A figura 5.1 ilustra o protocolo, de forma simplificada.

A entidade *service* aceita dados da entidade *sender*, e os transmite para a entidade *receiver*. As trocas de dados nos SAPs são feitas de forma atômica e instantânea. Assume-se que uma célula de dados está associada a cada interação nos SAPs.

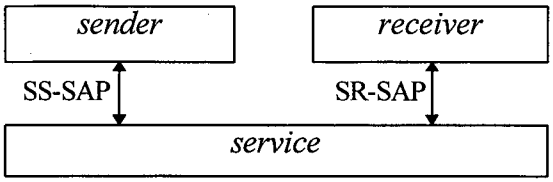


Figura 5.1 - O protocolo Tick-Tock

No que tange aos aspectos temporais, a entidade *service* deverá satisfazer as seguintes condições:

- Isocronismo: As células de dados são aceitas pela entidade *service* em intervalos regulares de tempo. Ou seja, depois do envio de uma célula de informação, a próxima somente poderá ser enviada π unidades de tempo depois. Somente uma célula pode ser enviada de cada vez.
- Atrasos de transmissão: Após o envio pela entidade *sender*, a informação será oferecida para a entidade *receiver* dentro de um intervalo de tempo $[\tau_{min}, \tau_{max}]$. Ou seja, o atraso de transmissão está compreendido neste intervalo.
- Espaço entre entregas: Existirá um atraso de, pelo menos, α unidades de tempo entre duas ofertas sucessivas de células para SR-SAP.
- Aceitação imediata: Se uma célula de dados for oferecida para a entidade *receiver*, mas não for aceita imediatamente, esta célula será perdida.
- Transmissão sem falhas: A entidade *service* é um meio de transmissão confiável. Ou seja, uma vez que a célula de dados é recebida pela entidade, a mesma será oferecida para a entidade *receiver*, sendo perdida somente no caso descrito na restrição anterior.

5.1.2 Especificação Formal do Protocolo Tick-Tock com Comportamento Finito

A especificação formal completa da entidade *service*, escrita em RT-LOTOS, é apresentada a seguir. Esta especificação é a mesma apresentada em [Martins96], apresentando comportamento finito, ou seja, pode ser representada por um grafo temporizado finito.

```
specification Service[SS_SAP, DELIVER, SR_SAP]
behaviour
            Isochronous[SS_SAP]
            |[SS_SAP]|
            (hide [DELIVER] in ((Trans_Dels[SS_SAP, DELIVER]
            |[DELIVER]|
            Spacing_Deliveries[DELIVER])
            |[DELIVER]|
            Imm_Accept[DELIVER, SR_SAP]))
where
```

```
        process Isochronous[SS_SAP]:=
                ([0]SS_SAP; [π]i; exit
                 <SS_SAP]{SS_SAP: [π]i; exit}) >> isochronous[SS_SAP]
        endproc
        process Trans_Dels[SS_SAP, DELIVER]:=
                SS_SAP; [τmin, τmax]i; DELIVER; Trans_Dels[SS_SAP, DELIVER]
        endproc


        process Spacing_Deliveries[DELIVER]:=
                DELIVER; [α]i; Spacing_Deliveries[DELIVER]
        endproc


        process Imm_Accept[DELIVER, SR_SAP]:=
                (DELIVER;[0]SR_SAP; exit
                 <SR_SAP]{SR_SAP: exit}) >> Imm_Accept[DELIVER, SR_SAP]
        endproc
endspec
```

As propriedades descritas acima são representadas do seguinte modo na especificação:

Isocronismo

Esta restrição, como o próprio nome indica, está representada no processo *Isochronous*. A sincronização com a entidade *sender* (SS_SAP) é oferecida somente num instante. Após a sincronização é necessário um intervalo de π unidades de tempo. Caso a sincronização não ocorra, uma próxima oportunidade será dada à entidade *sender* somente π unidades de tempo depois. Isto está representado no operador de preempção temporal.

Atraso de Transmissão

O processo *Trans_Dels* trata desta restrição da especificação. Como descrito acima, depois de receber um dado para transmissão, há um atraso para que o dado seja entregue. Este atraso está no intervalo $[\tau_{min}, \tau_{max}]$. Depois deste atraso, o processo *Trans_Dels* sinaliza que pode ser feita a entrega, através da ação DELIVER.

Espaços entre Entregas

O comportamento do processo *Spacing_Deliveries* garante o atendimento da restrição do espaçamento das entregas. Após uma entrega, outra somente poderá ser feita depois de α unidades de tempo.

Aceitação Imediata

O processo *Imm_Accept* irá garantir a restrição da aceitação imediata do dado recebido. Uma vez que o processo sincroniza na ação DELIVER, o que indica um novo dado está pronto para a entidade *receiver*, a mesma deverá sincronizar no mesmo instante. Após esta sincronização, o processo novamente fica esperando um novo sinal em DELIVER. Caso *receiver* não esteja pronto para receber o dado, o intervalo pontual de SR_SAP garante o

dado não fique disponível por mais tempo. Neste caso o operador de preempção temporal garante que *Imm_Accept* volte a esperar um sinal em DELIVER.

5.1.3 Simulação da Especificação

Nesta seção serão utilizadas as funcionalidades da ferramenta RTLS para a validação das propriedades da especificação. A especificação acima é simulada com os seguintes valores associados às constantes: $\pi = 100$, $\tau_{min} = 50$, $\tau_{max} = 80$ e $\alpha = 90$. Estes valores são os mesmo atribuídos em [DOY94].

As propriedades serão demonstradas na ordem em que foram apresentadas. Para isto serão utilizados traços de simulação. Para a geração dos traços de simulação será utilizada sempre a mesma semente. A validação das propriedades através da árvore de execução também são comentadas.

5.1.3.1 Traços de Simulação

Isocronismo

Como a transmissão pode ocorrer somente em intervalos de 100 unidades de tempo, serão gerados três traços de simulação, observando apenas a ação SS_SAP, que é a ação que recebe os dados da entidade *sender*.

O primeiro traço será gerado com as ações temporizadas ocorrendo no seu limite superior de tempo, sem violações temporais, analisando 15 disparos e gerando o traço da ação SS_SAP.

```
<0 - SS_SAP>          | <500 - SS_SAP>          | <1000 - SS_SAP>
<100 - SS_SAP>        | <600 - SS_SAP>          | <1100 - SS_SAP>
<200 - SS_SAP>        | <700 - SS_SAP>          | <1200 - SS_SAP>
<300 - SS_SAP>        | <800 - SS_SAP>          | <1300 - SS_SAP>
<400 - SS_SAP>        | <900 - SS_SAP>          | <1400 - SS_SAP>
```

Já o segundo traço será igual ao primeiro, com a diferença de que as violações temporais sempre ocorrerão. Isto modela o caso em que *sender* nunca está pronto para transmitir.

```
<0 - i(SS_SAP*)>      | <500 - i(SS_SAP*)>      | <1000 - i(SS_SAP*)>
<100 - i(SS_SAP*)>    | <600 - i(SS_SAP*)>      | <1100 - i(SS_SAP*)>
<200 - i(SS_SAP*)>    | <700 - i(SS_SAP*)>      | <1200 - i(SS_SAP*)>
<300 - i(SS_SAP*)>    | <800 - i(SS_SAP*)>      | <1300 - i(SS_SAP*)>
<400 - i(SS_SAP*)>    | <900 - i(SS_SAP*)>      | <1400 - i(SS_SAP*)>
```

No terceiro caso será simulado o caso em que a entidade *sender* pode estar pronta para transmitir ou não. Neste caso o traço de simulação será uma intercalação da ação SS_SAP e da sua violação temporal.

```
<0 - i(SS_SAP*)>      | <500 - i(SS_SAP*)>      | <1000 - i(SS_SAP*)>
<100 - SS_SAP>        | <600 - SS_SAP>          | <1100 - SS_SAP>
<200 - i(SS_SAP*)>    | <700 - i(SS_SAP*)>      | <1200 - i(SS_SAP*)>
<300 - i(SS_SAP*)>    | <800 - SS_SAP>          | <1300 - i(SS_SAP*)>
<400 - SS_SAP>        | <900 - i(SS_SAP*)>      | <1400 - SS_SAP>
```

Analisando os três traços gerados acima, pode-se deduzir que a ação SS_SAP será oferecida à entidade *sender* em intervalos regulares de 100 unidades de tempo. A transmissão deverá ser feita na hora, sendo que *sender* terá nova chance de transmitir somente depois de transcorridas mais 100 unidades de tempo. O mesmo acontece caso não ocorra uma transmissão (violação temporal de SS_SAP). Valida-se assim o requisito de isocronismo.

Atraso na transmissão

Para a validação desta propriedades, dois traços serão gerados. No primeiro caso será gerado o traço da ação DELIVER, permitindo a ocorrência aleatória de violações temporais, disparando as ação no limite inferior de seu intervalo de tempo.

```
<150 - i(DELIVER)>      | <1450 - i(DELIVER)>     | <2250 - i(DELIVER)>
<450 - i(DELIVER)>      | <1650 - i(DELIVER)>     | <2350 - i(DELIVER)>
<650 - i(DELIVER)>      | <1750 - i(DELIVER)>     | <2450 - i(DELIVER)>
<850 - i(DELIVER)>      | <1850 - i(DELIVER)>     | <2550 - i(DELIVER)>
<1150 - i(DELIVER)>     | <2050 - i(DELIVER)>     | <2750 - i(DELIVER)>
```

No segundo traço as ações serão disparadas no seu limite superior de tempo. Novamente serão permitidas as ocorrências de violações temporais.

```
<180 - i(DELIVER)>      | <1480 - i(DELIVER)>     | <2280 - i(DELIVER)>
<480 - i(DELIVER)>      | <1680 - i(DELIVER)>     | <2380 - i(DELIVER)>
<680 - i(DELIVER)>      | <1780 - i(DELIVER)>     | <2480 - i(DELIVER)>
<880 - i(DELIVER)>      | <1880 - i(DELIVER)>     | <2580 - i(DELIVER)>
<1180 - i(DELIVER)>     | <2080 - i(DELIVER)>     | <2780 - i(DELIVER)>
```

Pela análise dos traços gerados acima, pode-se ver que:

- o atraso mínimo na transmissão é de 50 unidades de tempo. Disparando as ações da especificação no limite inferior de seu intervalo de tempo, a ação DELIVER, que representa o término da transmissão, nunca ocorre antes de 50 unidades de tempo após o recebimento do dado (SS_SAP).
- seguindo o mesmo raciocínio anterior, pode-se ver que o atraso máximo da ação DELIVER em relação ao recebimento do dado é de 80 unidades de tempo. Esta propriedade pode ser vista no segundo traço de simulação.

Assim, pode-se considerar validado o requisito do atraso de transmissão compreendido entre dois limites, que no exemplo aqui analisado são de 50 e 80 unidades de tempo, respectivamente. Note que não há violações temporais da ação DELIVER, uma vez que a mesma é urgente, por ser ocultada pelo operador *hide*.]

Espaço entre entregas

Será gerado um traço para validar esta propriedade, que exige que as entregas (ação SR_SAP) tenham um intervalo de no mínimo α unidades de tempo (α = 90). Assim, o traço a ser inspecionado é o da ação SR_SAP, sem violações temporais (sempre haverá transmissão e recepção). O disparo será feito em momentos aleatórios, para que haja diferenças de tempo. O intervalo de transmissão deve variar, para que a entrega não seja afetada pela periodicidade da ação SS_SAP. Se os disparos forem feitos no limite inferior ou superior, a

transmissão será feita sempre no mesmo tempo, e as entregas serão feitas na periodicidade do envio.

O traço gerado é o seguinte:

```
<52 - SR_SAP>
<159 - SR_SAP>
<266 - SR_SAP>
<377 - SR_SAP>
<479 - SR_SAP>
<569 - SR_SAP>
<670 - SR_SAP>
<767 - SR_SAP>
<872 - SR_SAP>
<978 - SR_SAP>
<1068 - SR_SAP>
<1174 - SR_SAP>
<1274 - SR_SAP>
<1369 - SR_SAP>
<1469 - SR_SAP>
<1559 - SR_SAP>
<1674 - SR_SAP>
<1771 - SR_SAP>
<1868 - SR_SAP>
<1962 - SR_SAP>
<2076 - SR_SAP>
<2172 - SR_SAP>
<2262 - SR_SAP>
<2377 - SR_SAP>
<2467 - SR_SAP>
<2568 - SR_SAP>
<2658 - SR_SAP>
```

A análise do traço gerado mostra que todas as entregas são feitas respeitando o intervalo mínimo entre elas, de 90 unidades de tempo. Os quatro últimos disparos mostram o tempo mínimo de espaço entre entregas. Todos os outros disparos da ação SR_SAP têm intervalos maiores ou iguais entre eles.

Aceitação imediata

Nesta propriedade, o que se pretende verificar é: assim que o meio termina a transmissão, podendo oferecer o dado para a entidade *receiver* (ação DELIVER), não há passagem do tempo antes que a mensagem seja entregue (ação SR_SAP) ou perdida (violação temporal de SR_SAP). Para isto será gerado um traço, no qual ações irão ocorrer no limite superior de seu intervalo de tempo. O traço a ser observado é o das ações DELIVER e SR_SAP.

```
<175 - i(DELIVER)>
<175 - i(SR_SAP*)>
<466 - i(DELIVER)>
<466 - SR_SAP>
<675 - i(DELIVER)>
<675 - SR_SAP>
<765 - i(DELIVER)>
<765 - i(SR_SAP*)>
<972 - i(DELIVER)>
<972 - i(SR_SAP*)>
<1264 - i(DELIVER)>
<1264 - i(SR_SAP*)>
<1455 - i(DELIVER)>
<1455 - SR_SAP>
<1753 - i(DELIVER)>
<1753 - i(SR_SAP*)>
<1973 - i(DELIVER)>
<1973 - i(SR_SAP*)>
<2068 - i(DELIVER)>
<2068 - i(SR_SAP*)>
<2462 - i(DELIVER)>
<2462 - SR_SAP>
<2570 - i(DELIVER)>
<2570 - SR_SAP>
```

Pelo traço pode-se ver que entre o sinal de que uma transmissão terminou (ação DELIVER) e o oferecimento do dado à entidade *receiver* não há passagem do tempo. Isto valida o requisito da aceitação imediata, pois não há atraso entre o término da transmissão e a disponibilidade do dado para *receiver*. Por outro lado implica que se a entidade *receiver* não estiver pronta para receber o dado assim que o mesmo chegar, este será perdido.

Transmissão sem Falhas

Para a validação desta propriedade será gerado um traço de simulação, disparando as ações em momentos aleatórios de seus intervalos, permitindo violações temporais. O traço a ser avaliado é o das ações SS_SAP e DELIVER.

```
<0 - i(SS_SAP*)>
<100 - SS_SAP>
<175 - i(DELIVER)>
<200 - i(SS_SAP*)>
<300 - i(SS_SAP*)>
<400 - SS_SAP>
<466 - i(DELIVER)>
<500 - i(SS_SAP*)>
<600 - SS_SAP>
<675 - i(DELIVER)>
<700 - SS_SAP>
<765 - i(DELIVER)>
<800 - i(SS_SAP*)>
<900 - SS_SAP>
<972 - i(DELIVER)>
<1000 - i(SS_SAP*)>
<1100 - i(SS_SAP*)>
<1200 - SS_SAP>
<1264 - i(DELIVER)>
<1300 - i(SS_SAP*)>
<1400 - SS_SAP>
<1455 - i(DELIVER)>
<1500 - i(SS_SAP*)>
<1600 - i(SS_SAP*)>
```

Analisando o traço acima, pode-se perceber que a cada vez que há um dado é recebido da entidade *sender* (ação SS_SAP), existe uma ação DELIVER num tempo compreendido no intervalo de atraso de transmissão. Se não for recebido o dado de *sender*, depois do intervalo de isocronismo, novamente há a chance de transmissão, sem que ocorra a ação DELIVER. Valida-se assim requisito de transmissão sem falhas. Toda vez que um dado é colocado na entidade *service*, a mesma acusa a sua transmissão, para oferecer o dado para a entidade *receiver* (quando então o dado pode ser perdido por uma não recepção por parte da mesma).

5.1.3.2 Árvore de Execução

A árvore de execução da especificação do protocolo Tick-Tock pode ser vista abaixo. As propriedades discutidas acima podem ser todas validadas pela análise da mesma, mesmo sendo esta análise um pouco mais complicada do que a análise dos traços de execução.

```
1 - SS_SAP - (0)
 2 - time(50) - (0)
  3 - i (i) - (50)
   4 - i (DELIVER) - (50)
    5 - SR_SAP - (50)
     6 - i (EXIT) - (50)
      7 - time(50) - (50)
       8 - i (i) - (100)
        9 - i (EXIT) - (100)
         10 - SS_SAP - (100)
          11 - time(40) - (100)
           12 - i (i) - (140)
            13 - time(10) - (140)
             14 - Recursion detected - 3 (150)
         10 - i (SS_SAP*) - (100)
          15 - time(40) - (100)
           16 - i (i) - (140)
            17 - time(60) - (140)
             18 - i (i) - (200)
              19 - i (EXIT) - (200)
               20 - Recursion detected - 1 (200)
    5 - i (SR_SAP*) - (50)
     21 - i (EXIT) - (50)
      22 - Analized elsewhere - 7 (50)
  3 - time(30) - (50)
   23 - i (i) - (80)
    24 - i (DELIVER) - (80)
     25 - SR_SAP - (80)
      26 - i (EXIT) - (80)
       27 - time(20) - (80)
        28 - i (i) - (100)
         29 - i (EXIT) - (100)
          30 - SS_SAP - (100)
           31 - time(50) - (100)
            32 - i (i) - (150)
             33 - time(20) - (150)
              34 - i (i) - (170)
               35 - i (DELIVER) - (170)
                36 - SR_SAP - (170)
                 37 - i (EXIT) - (170)
                  38 - time(30) - (170)
                   39 - i (i) - (200)
                    40 - i (EXIT) - (200)
                     41 - SS_SAP - (200)
                      42 - time(50) - (200)
                       43 - i (i) - (250)
                        44 - time(10) - (250)
```

```
                                45 - i (i) - (260)
                                 46 - i (DELIVER) - (260)
                                  47 - SR_SAP - (260)
                                   48 - i (EXIT) - (260)
                                    49 - time(40) - (260)
                                     50 - i (i) - (300)
                                      51 - i (EXIT) - (300)
                                       52 - SS_SAP - (300)
                                        53 - time(50) - (300)
                                         54 - i (i) - (350)
                                          55 - i (i) - (350)
                                           56 - Analized elsewhere - 4 (350)
                                         54 - i (i) - (350)
                                          57 - Recursion detected - 3 (350)
                                        52 - i (SS_SAP*) - (300)
                                         58 - time(50) - (300)
                                          59 - i (i) - (350)
                                           60 - time(50) - (350)
                                            61 - Analized elsewhere - 18 (400)
                                  47 - i (SR_SAP*) - (260)
                                   62 - i (EXIT) - (260)
                                    63 - Analized elsewhere - 49 (260)
                             43 - time(10) - (250)
                              64 - i (i) - (260)
                               65 - Analized elsewhere - 45 (260)
                              64 - i (i) - (260)
                               66 - i (i) - (260)
                                67 - Analized elsewhere - 46 (260)
                               66 - time(20) - (260)
                                68 - Recursion detected - 23 (280)
                           41 - i (SS_SAP*) - (200)
                            69 - time(60) - (200)
                             70 - i (i) - (260)
                              71 - time(40) - (260)
                               72 - Analized elsewhere - 18 (300)
                       36 - i (SR_SAP*) - (170)
                        73 - i (EXIT) - (170)
                         74 - Analized elsewhere - 38 (170)
                  32 - time(20) - (150)
                   75 - i (i) - (170)
                    76 - Analized elsewhere - 34 (170)
                   75 - i (i) - (170)
                    77 - i (i) - (170)
                     78 - Analized elsewhere - 35 (170)
                    77 - time(10) - (170)
                     79 - Recursion detected - 23 (180)
               30 - i (SS_SAP*) - (100)
                80 - time(70) - (100)
                 81 - i (i) - (170)
                  82 - time(30) - (170)
                   83 - Analized elsewhere - 18 (200)
          25 - i (SR_SAP*) - (80)
           84 - i (EXIT) - (80)
            85 - Analized elsewhere - 27 (80)
1 - i (SS_SAP*) - (0)
 86 - time(100) - (0)
  87 - Analized elsewhere - 18 (100)
```

Esta árvore de execução foi gerada em menos de 1 segundo, na máquina descrita no início deste capítulo.

A seguir são apresentadas as análises da árvore de execução que validam as propriedades da especificação.

Isocronismo

Todos os pontos da árvore onde aparece a ação SS_SAP, ou sua violação temporal, são alcançados em momentos que são múltiplos do intervalo de isocronismo (100 unidades de tempo). Esta propriedade é mantida quando há estados que apontam para outro lugar da árvore, pois sabe-se que os intervalos de tempo em estados equivalentes são iguais, apenas ocorre a mudança dos valores absolutos.

Atraso na Transmissão

Esta propriedade é validada pelo fato de que todas as ocorrências da ação DELIVER ocorrem num intervalo de $[\tau_{min}, \tau_{max}]$ unidades de tempo após a ocorrência da ação SS_SAP. Não há ramo da árvore que possa levar de SS_SAP (não da sua violação temporal) para DELIVER. Assim, a propriedade de transmissão sem falhas é validada.

Espaço entre entregas

Esta propriedade é validada pela análise dos tempos entre as ocorrências da ação SR_SAP. Pela inspeção da árvore, podemos ver que o tempo mínimo entre as ocorrências é o especificado (90 unidades de tempo). A progressão máxima pode ser vista claramente no ramo que leva do estado 3 ao estado 56. Nele, todas as ocorrências de SR_SAP estão espaçadas exatamente por 90 unidades de tempo. Outros ramos, que implicam a passagem entre estados equivalentes, têm tempo maior entre as ocorrências de SR_SAP. Um exemplo é o estado 14, que é um recursão para o estado 3. Para chegar ao estado 14, SR_SAP ocorreu no instante 50. A próxima ocorrência será possível somente no instante 150 (valor obtido pela substituição dos tempos indicada no estado 14). Este intervalo de 100 unidades entre as entregas é maior que o mínimo exigido. Análises semelhantes são obtidas para os outros ramos da árvore onde SR_SAP ocorre. Pode-se considerar assim a propriedade validada.

Aceitação Imediata

Todas as ocorrências da ação DELIVER, que representa o final da transmissão, são sucedidas da ação SR_SAP (recepção do dado) ou sua violação temporal (perda do dado pelo fato de *receiver* não estar pronto). Não há passagem do tempo possível após a ocorrência de DELIVER (estados 5, 25, 36 e 47). Desta forma, valida-se a propriedade da aceitação imediata, já que ao final da transmissão o dado deve ser aceito imediatamente, ou será perdido.

Transmissão sem Falhas

Todas as ocorrências da ação SS_SAP são seguidos de ramos que levam até a ação DELIVER. As ocorrências da violação temporal de SS_SAP não levam a ocorrência de DELIVER antes de outra ação SS_SAP. Desta forma, pode-se deduzir que uma vez colocado o dado na entidade *service*, a transmissão é feita. Não há transmissão sem que tenha sido colocado um dado na entidade *service* (o que seria um absurdo).

5.1.4 Especificação Formal do Protocolo Tick-Tock com Comportamento Infinito

A especificação formal completa da entidade *service*, escrita em RT-LOTOS, é apresentada a seguir. Esta especificação, ao contrário daquela apresentada na seção 5.1.2, não apresenta comportamento finito. Em outras palavras, não pode ser representada por um grafo temporizado finito.

```
specification Service[SS_SAP, DELIVER, SR_SAP]
behaviour
       Isochronous[SS_SAP]
       |[SS_SAP]|
       (hide [DELIVER] in ((Trans_Dels[SS_SAP, DELIVER]
       |[DELIVER]|
       Spacing_Deliveries[DELIVER])
       |[DELIVER]|
       Imm_Accept[DELIVER, SR_SAP]))
where
       process Isochronous[SS_SAP]:=
              [π]i; Isochronous[SS_SAP] ||| [0]SS_SAP; exit
       endproc
       process Trans_Dels[SS_SAP, DELIVER]:=
              SS_SAP; [τmin, τmax]i; DELIVER; Trans_Dels[SS_SAP, DELIVER]
       endproc
       process Spacing_Deliveries[DELIVER]:=
              DELIVER; [α]i; Spacing_Deliveries[DELIVER]
       endproc
       process Imm_Accept[DELIVER, SR_SAP]:=
              (DELIVER;[0]SR_SAP; exit
               <SR_SAP]{SR_SAP: exit}) >> Imm_Accept[DELIVER, SR_SAP]
       endproc
endspec
```

As propriedades descritas acima são representadas nesta especificação da mesma forma que são representadas na especificação da seção 5.1.2, com exceção do isocronismo.

Ao contrário da especificação apresentada na seção 5.1.2, nesta especificação o processo *Isochronous* utiliza o lançamento periódico de processo para garantir a periodicidade da ação *SS_SAP*. O mecanismo de lançamento periódico de processo é uma construção simples em RT-LOTOS. Para disparar uma instância de um processo `P` a cada `t` unidades de tempo, basta utilizar um processo do tipo

```
Q := P ||| [t]i; Q
```

Porém, por gerar um comportamento infinito (devido a recursão de `Q` na composição paralela), este tipo de construção não pode ser analisada utilizando a tradução para grafo temporizado. A ferramenta RTLS, por realizar a simulação diretamente sobre a sintaxe e a semântica de RT-LOTOS, pode validar especificações que apresentam tal construção (entre outras que geram comportamentos infinitos).

5.1.5 Simulação da Especificação

Nesta seção serão utilizadas as funcionalidades da ferramenta RTLS para a validação das propriedades da especificação. A especificação acima é simulada com os mesmos valores associados às constantes apresentados na seção 5.1.3, ou seja: $\pi = 100$, $\tau_{min} = 50$, $\tau_{max} = 80$ e $\alpha = 90$.

As propriedades serão demonstradas na ordem em que foram apresentadas. Para isto serão utilizados traços de simulação. Para a geração dos traços de simulação será utilizada a mesma semente utilizada na seção 5.1.3. Porém, as seqüências podem ser diferentes, pois as ações possíveis em cada estado da especificação são diferentes nesta especificação, pela mudança do processo *Isochronous*.

5.1.5.1 Traços de Simulação

Isocronismo

Novamente o primeiro traço será gerado com as ações temporizadas ocorrendo no seu limite superior de tempo, sem violações temporais, analisando 15 disparos e gerando o traço da ação SS_SAP.

```
<0 - SS_SAP>      | <500 - SS_SAP>    | <1000 - SS_SAP>
<100 - SS_SAP>    | <600 - SS_SAP>    | <1100 - SS_SAP>
<200 - SS_SAP>    | <700 - SS_SAP>    | <1200 - SS_SAP>
<300 - SS_SAP>    | <800 - SS_SAP>    | <1300 - SS_SAP>
<400 - SS_SAP>    | <900 - SS_SAP>    | <1400 - SS_SAP>
```

O segundo traço será igual ao primeiro, com a diferença de que as violações temporais sempre ocorrerão. Isto modela o caso em que *sender* nunca está pronto para transmitir.

```
<0 - SS_SAP*>     | <500 - SS_SAP*>   | <1000 - SS_SAP*>
<100 - SS_SAP*>   | <600 - SS_SAP*>   | <1100 - SS_SAP*>
<200 - SS_SAP*>   | <700 - SS_SAP*>   | <1200 - SS_SAP*>
<300 - SS_SAP*>   | <800 - SS_SAP*>   | <1300 - SS_SAP*>
<400 - SS_SAP*>   | <900 - SS_SAP*>   | <1400 - SS_SAP*>
```

No terceiro caso será simulado o caso em que a entidade *sender* pode estar pronta para transmitir ou não. Neste caso o traço de simulação será uma intercalação da ação SS_SAP e da sua violação temporal.

```
<0 - SS_SAP*>     | <500 - SS_SAP*>   | <1000 - SS_SAP*>
<100 - SS_SAP*>   | <600 - SS_SAP>    | <1100 - SS_SAP*>
<200 - SS_SAP*>   | <700 - SS_SAP>    | <1200 - SS_SAP*>
<300 - SS_SAP>    | <800 - SS_SAP>    | <1300 - SS_SAP>
<400 - SS_SAP*>   | <900 - SS_SAP>    | <1400 - SS_SAP>
```

Analisando os três traços gerados acima, pode-se deduzir que a ação SS_SAP será oferecida à entidade *sender* em intervalos regulares de 100 unidades de tempo. O comportamento é o mesmo da especificação da seção 5.1.2. Porém o lançamento periódico do oferecimento da ação *SS_SAP* é mais intuitivo do que todo o tratamento de violação temporal feito na primeira especificação.

Atraso na transmissão

Para a validação desta propriedades, novamente serão gerados dois traços. No primeiro caso será gerado o traço da ação DELIVER, permitindo a ocorrência aleatória de violações temporais, disparando as ação no limite inferior de seu intervalo de tempo.

```
<350 - i(DELIVER)>     <1350 - i(DELIVER)>     <2050 - i(DELIVER)>
<650 - i(DELIVER)>     <1450 - i(DELIVER)>     <2250 - i(DELIVER)>
<750 - i(DELIVER)>     <1650 - i(DELIVER)>     <2350 - i(DELIVER)>
<850 - i(DELIVER)>     <1750 - i(DELIVER)>     <2850 - i(DELIVER)>
<950 - i(DELIVER)>     <1950 - i(DELIVER)>     <3150 - i(DELIVER)>
```

No segundo traço as ações serão disparadas no seu limite superior de tempo. Novamente serão permitidas as ocorrências de violações temporais.

```
<380 - i(DELIVER)>     <1380 - i(DELIVER)>     <2080 - i(DELIVER)>
<680 - i(DELIVER)>     <1480 - i(DELIVER)>     <2280 - i(DELIVER)>
<780 - i(DELIVER)>     <1680 - i(DELIVER)>     <2380 - i(DELIVER)>
<880 - i(DELIVER)>     <1780 - i(DELIVER)>     <2880 - i(DELIVER)>
<980 - i(DELIVER)>     <1980 - i(DELIVER)>     <3180 - i(DELIVER)>
```

Pela análise dos traços gerados acima, pode-se ver que o intervalo de transmissão, como na especificação da seção 5.1.2, está entre 50 e 80 unidades de tempo depois da aceitação do dado, pela ação SS_SAP.

Espaço entre entregas

Também será gerado um traço para validar esta propriedade, que exige que as entregas (ação SR_SAP) tenham um intervalo de no mínimo α unidades de tempo (α = 90). Assim, o traço a ser inspecionado é o da ação SR_SAP, sem violações temporais (sempre haverá transmissão e recepção). Novamente, o disparo será feito em momentos aleatórios, para que haja diferenças de tempo. O traço gerado é o seguinte:

```
<52 - SR_SAP>          <777 - SR_SAP>          <1471 - SR_SAP>
<158 - SR_SAP>         <867 - SR_SAP>          <1561 - SR_SAP>
<270 - SR_SAP>         <972 - SR_SAP>          <1653 - SR_SAP>
<360 - SR_SAP>         <1073 - SR_SAP>         <1753 - SR_SAP>
<479 - SR_SAP>         <1178 - SR_SAP>         <1874 - SR_SAP>
<569 - SR_SAP>         <1268 - SR_SAP>         <1968 - SR_SAP>
<676 - SR_SAP>         <1363 - SR_SAP>         <2058 - SR_SAP>
```

A análise do traço gerado mostra que todas as entregas são feitas respeitando o intervalo mínimo entre elas, de 90 unidades de tempo.

Aceitação imediata

Como para a especificação da seção 5.1.2, será gerado um traço, no qual as ações irão ocorrer no limite superior de seu intervalo de tempo. Violações temporais serão permitidas. O traço a ser observado é o das ações DELIVER e SR_SAP.

```
<379 - i(DELIVER)>     <1050 - i(SR_SAP*)>     <1663 - i(DELIVER)>
<379 - i(SR_SAP*)>     <1169 - i(DELIVER)>     <1663 - SR_SAP>
<469 - i(DELIVER)>     <1169 - i(SR_SAP*)>     <1768 - i(DELIVER)>
<469 - SR_SAP>         <1270 - i(DELIVER)>     <1768 - i(SR_SAP*)>
<877 - i(DELIVER)>     <1270 - i(SR_SAP*)>     <1858 - i(DELIVER)>
<877 - i(SR_SAP*)>     <1564 - i(DELIVER)>     <1858 - i(SR_SAP*)>
<1050 - i(DELIVER)>    <1564 - i(SR_SAP*)>     <2153 - i(DELIVER)>
```

Pelo traço pode-se ver que entre o sinal de que uma transmissão terminou (ação DELIVER) e o oferecimento do dado à entidade *receiver* (ação SR_SAP)não há passagem do tempo. Isto valida o requisito da aceitação imediata, como para a especificação da seção 5.1.2.

Transmissão sem Falhas

Para a validação desta propriedade, novamente será gerado um traço de simulação, disparando as ações em momentos aleatórios de seus intervalos, permitindo violações temporais. O traço a ser avaliado é o das ações SS_SAP e DELIVER.

```
<0 - SS_SAP*>
<100 - SS_SAP*>
<200 - SS_SAP*>
<300 - SS_SAP>
<379 - i(DELIVER)>
<400 - SS_SAP>
<469 - i(DELIVER)>
<500 - SS_SAP*>
<600 - SS_SAP*>
<700 - SS_SAP*>
```

```
<800 - SS_SAP>
<877 - i(DELIVER)>
<900 - SS_SAP*>
<1000 - SS_SAP>
<1050 - i(DELIVER)>
<1100 - SS_SAP>
<1169 - i(DELIVER)>
<1200 - SS_SAP>
<1270 - i(DELIVER)>
<1300 - SS_SAP*>
```

```
<1400 - SS_SAP*>
<1500 - SS_SAP>
<1564 - i(DELIVER)>
<1600 - SS_SAP>
<1663 - i(DELIVER)>
<1700 - SS_SAP>
<1768 - i(DELIVER)>
<1800 - SS_SAP>
<1858 - i(DELIVER)>
<1900 - SS_SAP*>
```

Analisando o traço acima, pode-se perceber que a cada vez que há um dado é recebido da entidade *sender* (ação SS_SAP), existe uma ação DELIVER num tempo compreendido no intervalo de atraso de transmissão. Como para a especificação da seção 5.1.2, pode-se considerar a propriedade da transmissão sem falhas validada.

5.1.5.2 Árvore de Execução

Como dito no capítulo 3, para especificação com comportamento infinito, não é possível gerar uma árvore de execução finita. Abaixo é apresentada a árvore de execução para a especificação do protocolo Tick-Tock com comportamento infinito. A profundidade de exploração de 30 disparos.

```
1 - SS_SAP - (0)
 2 - time(50) - (0)
  3 - i (i) - (50)
   4 - i (DELIVER) - (50)
    5 - SR_SAP - (50)
     6 - i (EXIT) - (50)
      7 - time(50) - (50)
       8 - i (i) - (100)
        9 - SS_SAP - (100)
         10 - time(40) - (100)
          11 - i (i) - (140)
           12 - time(10) - (140)
            13 - Recursion detected - 3 (150)
        9 - SS_SAP* - (100)
         14 - time(40) - (100)
          15 - i (i) - (140)
           16 - time(60) - (140)
            17 - i (i) - (200)
             18 - SS_SAP - (200)
              19 - time(50) - (200)
               20 - i (i) - (250)
                21 - i (DELIVER) - (250)
                 22 - SR_SAP - (250)
                  23 - i (EXIT) - (250)
                   24 - time(50) - (250)
                    25 - i (i) - (300)
```

```
                    26 - SS_SAP - (300)
                     27 - time(40) - (300)
                      28 - i (i) - (340)
                       29 - time(10) - (340)
                        30 - Recursion detected - 20 (350)
                    26 - SS_SAP* - (300)
                     31 - time(40) - (300)
                      32 - i (i) - (340)
                       33 - time(60) - (340)
                        34 - i (i) - (400)
                         35 - SS_SAP - (400)
                          36 - time(50) - (400)
                           37 - i (i) - (450)
                            38 - i (DELIVER) - (450)
                           37 - time(30) - (450)
                            39 - i (i) - (480)
                         35 - SS_SAP* - (400)
                          40 - time(100) - (400)
                           41 - i (i) - (500)
                            42 - SS_SAP - (500)
                            42 - SS_SAP* - (500)
                22 - i (SR_SAP*) - (250)
                 43 - i (EXIT) - (250)
                  44 - Analized elsewhere - 24 (250)
               20 - time(30) - (250)
                45 - i (i) - (280)
                 46 - i (DELIVER) - (280)
                  47 - SR_SAP - (280)
                   48 - i (EXIT) - (280)
                    49 - time(20) - (280)
                     50 - i (i) - (300)
                      51 - SS_SAP - (300)
                       52 - time(50) - (300)
                        53 - i (i) - (350)
                         54 - time(20) - (350)
                          55 - i (i) - (370)
                           56 - i (DELIVER) - (370)
                            57 - SR_SAP - (370)
                             58 - i (EXIT) - (370)
                            57 - i (SR_SAP*) - (370)
                             59 - i (EXIT) - (370)
                        53 - time(20) - (350)
                         60 - i (i) - (370)
                          61 - Analized elsewhere - 55 (370)
                         60 - i (i) - (370)
                          62 - i (i) - (370)
                           63 - Analized elsewhere - 56 (370)
                          62 - time(10) - (370)
                           64 - Recursion detected - 45 (380)
                      51 - SS_SAP* - (300)
                       65 - time(70) - (300)
                        66 - i (i) - (370)
                         67 - time(30) - (370)
                          68 - Analized elsewhere - 34 (400)
                  47 - i (SR_SAP*) - (280)
                   69 - i (EXIT) - (280)
                    70 - Analized elsewhere - 49 (280)
            18 - SS_SAP* - (200)
             71 - time(100) - (200)
              72 - Analized elsewhere - 34 (300)
  5 - i (SR_SAP*) - (50)
   73 - i (EXIT) - (50)
    74 - Analized elsewhere - 7 (50)
3 - time(30) - (50)
 75 - i (i) - (80)
  76 - i (DELIVER) - (80)
   77 - SR_SAP - (80)
    78 - i (EXIT) - (80)
     79 - time(20) - (80)
      80 - i (i) - (100)
       81 - SS_SAP - (100)
        82 - time(50) - (100)
```

```
            83 - i (i) - (150)
             84 - time(20) - (150)
              85 - i (i) - (170)
               86 - i (DELIVER) - (170)
                87 - SR_SAP - (170)
                 88 - i (EXIT) - (170)
                  89 - time(30) - (170)
                   90 - i (i) - (200)
                    91 - SS_SAP - (200)
                     92 - time(50) - (200)
                      93 - i (i) - (250)
                       94 - time(10) - (250)
                        95 - i (i) - (260)
                         96 - i (DELIVER) - (260)
                          97 - SR_SAP - (260)
                           98 - i (EXIT) - (260)
                            99 - time(40) - (260)
                             100 - i (i) - (300)
                              101 - SS_SAP - (300)
                              101 - SS_SAP* - (300)
                          97 - i (SR_SAP*) - (260)
                           102 - i (EXIT) - (260)
                            103 - Analized elsewhere - 99 (260)
                       93 - time(10) - (250)
                        104 - i (i) - (260)
                         105 - Analized elsewhere - 95 (260)
                        104 - i (i) - (260)
                         106 - i (i) - (260)
                          107 - Analized elsewhere - 96 (260)
                         106 - time(20) - (260)
                          108 - Recursion detected - 75 (280)
                     91 - SS_SAP* - (200)
                      109 - time(60) - (200)
                       110 - i (i) - (260)
                        111 - time(40) - (260)
                         112 - Analized elsewhere - 17 (300)
                87 - i (SR_SAP*) - (170)
                 113 - i (EXIT) - (170)
                  114 - Analized elsewhere - 89 (170)
            83 - time(20) - (150)
             115 - i (i) - (170)
              116 - Analized elsewhere - 85 (170)
             115 - i (i) - (170)
              117 - i (i) - (170)
               118 - Analized elsewhere - 86 (170)
              117 - time(10) - (170)
               119 - Recursion detected - 75 (180)
          81 - SS_SAP* - (100)
           120 - time(70) - (100)
            121 - i (i) - (170)
             122 - time(30) - (170)
              123 - Analized elsewhere - 17 (200)
       77 - i (SR_SAP*) - (80)
        124 - i (EXIT) - (80)
         125 - Analized elsewhere - 79 (80)
1 - SS_SAP* - (0)
 126 - time(100) - (0)
  127 - Analized elsewhere - 17 (100)
```

Esta árvore foi gerada em 1 segundo, na máquina descrita. Outra árvore foi gerada, explorando uma profundidade de 700 disparos. Esta árvore apresentou 3.436 estados diferentes, com um total de 4.333 estados analisados em 5 minutos e 38 segundos. Nenhum *deadlock* foi encontrado. Pode-se concluir que a especificação é livre de *deadlocks*. Foi feita a tentativa de gerar uma árvore com profundidade de 800 disparos, porém não a memória da máquina não foi suficiente.

A seguir são apresentadas as análises da árvore de execução que validam as propriedades da especificação.

Isocronismo

Todos os pontos da árvore onde aparece a ação SS_SAP, ou sua violação temporal, são alcançados em momentos que são múltiplos do intervalo de isocronismo (100 unidades de tempo). Esta característica é a mesma encontrada na árvore de execução para a especificação da seção 5.1.2.

Atraso na Transmissão

Esta propriedade é validada pelo fato de que todas as ocorrências da ação DELIVER ocorrem num intervalo de $[\tau_{min}, \tau_{max}]$ unidades de tempo após a ocorrência da ação SS_SAP. Não há ramo da árvore que possa levar de SS_SAP (não da sua violação temporal) para DELIVER. Assim, a propriedade de transmissão sem falhas é novamente validada.

Espaço entre entregas

Esta propriedade é validada pela análise dos tempos entre as ocorrências da ação SR_SAP. Pela inspeção da árvore, podemos ver que o tempo mínimo entre as ocorrências é o especificado (90 unidades de tempo). Valida-se assim também esta propriedade.

Aceitação Imediata

Todas as ocorrências da ação DELIVER, que representa o final da transmissão, são sucedidas da ação SR_SAP (recepção do dado) ou sua violação temporal (perda do dado pelo fato de *receiver* não estar pronto). Não há passagem do tempo possível após a ocorrência de DELIVER, validando-se, desta forma, a propriedade da aceitação imediata, já que ao final da transmissão o dado deve ser aceito imediatamente, ou será perdido.

Transmissão sem Falhas

Todas as ocorrências da ação SS_SAP são seguidos de ramos que levam até a ação DELIVER. As ocorrências da violação temporal de SS_SAP não levam a ocorrência de DELIVER antes de outra ação SS_SAP. Como para a especificação da seção 5.1.2, esta propriedade também é validada para a especificação com comportamento infinito.

5.2 Sistema Telefônico

O exemplo a seguir foi descrito e verificado em [Yovine93]. É uma simplificação de um sistema telefônico real, apresentando algumas propriedades tempo real interessantes.

O sistema é composto de vários terminais e de uma Central Telefônica. A Central Telefônica é dividida em três unidades: Unidade de Controle (UC), Unidade de Gestão do Número (UGN) e Unidade de Gestão do Som (UGS). A figura 5.2 representa esta divisão da Central Telefônica.

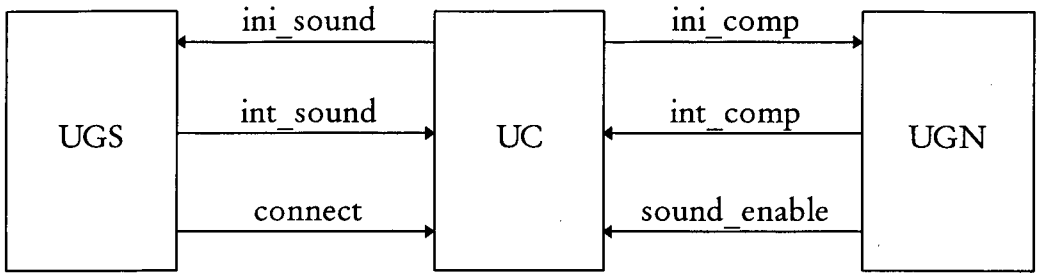


Figura 5.2 - Central Telefônica

A comunicação entre a UGS e a UC é realizada através das ações:

ini_sound → indica o início do toque do número chamado.

int_sound → indica a interrupção do toque e a anulação da conexão.

connect → indica o estabelecimento da conexão entre os terminais.

A comunicação entre a UGN e a UC se faz através das ações:

ini_comp → indica o início da composição do número a ser chamado.

sound_enable → indica que a composição do número foi terminada e que a central está pronta para provocar o toque do telefone chamado.

int_comp → indica a interrupção da composição do número.

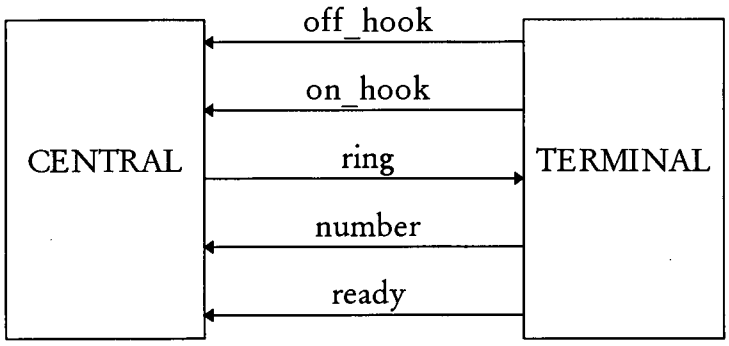


Figura 5.3 - Comunicação Central Telefônica - Terminal

Por outro lado, a comunicação entre um terminal e a Central Telefônica ocorre da forma apresentada na figura 5.3.

Esta comunicação entre a Central Telefônica e um Terminal se faz através das ações:

off_hook → indica a retirada do telefone do gancho, para solicitar uma conexão.

on_hook → indica o retorno do telefone ao gancho.

ring → sinalização para o toque (sonoro) de chamada do terminal.

number → indica um dígito do número do terminal chamado.

ready → indica a retirada do telefone chamado do gancho, para o início de uma conexão.

5.2.1 Descrição Informal dos Componentes da Central Telefônica

A Unidade de Controle tem como função atender da melhor forma possível a demanda de comunicação proveniente dos terminais, sendo que uma demanda ocorre quando um telefone é retirado do gancho. Por outro lado, supõe-se que uma central não trata de mais de uma chamada, e que uma conexão não pode ser interrompida por uma outra chamada. No momento em que um telefone é retirado do gancho, a central está pronta para gerenciar a composição de um número de chamada. Esta função é realizada pela Unidade de Gestão do Número. Após o final da composição do número do terminal a ser chamado, a Central Telefônica inicia o tom de chamada através da Unidade de Gestão do Som, que sinaliza para a Unidade Central se a conexão foi estabelecida ou não. A duração da comunicação é limitada, ou seja, se o chamador não finalizar a comunicação até um determinado instante *tm*, expresso em segundo após o início da chamada, a comunicação é interrompida.

A função da Unidade de Gestão do Número consiste em gerar a composição do número chamado. Supõe-se que os números são compostos de quatro algarismos, e que a sua composição deve obedecer as seguintes restrições:

- O terminal chamador deve discar o primeiro número em até *tp* segundos após o tom de discar.
- O tempo entre os algarismos não deve ultrapassar o limite de *ts* segundos.
- A composição de um número não deve ultrapassar o limite de *tc* segundos.

A função da Unidade de Gestão do Som consiste em provocar um toque, de duração de *tn* segundos, no terminal chamado. Se este não responder em até *tr* segundos, a chamada será cancelada. Caso contrário, o estabelecimento da conexão é sinalizado para a Unidade Central.

5.2.2 Especificação Formal da Central Telefônica

A especificação formal em RT-LOTOS da Central Telefônica é mostrada a seguir. As portas das instanciações de processos são omitidas, para facilitar a leitura. Não há nenhum *relabeling* nas instanciações, portanto esta omissão não prejudica a compreensão da especificação.

```
SPECIFICATION Central[   ring, off_hook, on_hook, int_comm, ready, number1, number2,
                         number3, number4, interrupt ]

BEHAVIOUR
      hide [ free, ini_comp, int_comp, sound_enable, ini_sound, int_sound,
             connect, sync_end ] in
```

```
        (( UC[...] |[ini_comp, int_comp, sound_enable, sync_end]|
           UGN[...])
        |[ini_sound, int_sound, connect]|
        UGS[...])
WHERE
    PROCESS UC[ free, off_hook, ini_comp, int_comp, sound_enable, ini_sound,
                int_sound, connect, on_hook, int_comm, sync_end ] :=
        free; off_hook; ini_comp;
                (int_comp; sync_end; UC[...]
                []
                sound_enable; sync_end; ini_sound;
                        ( int_sound; UC[...]
                        []
                        connect;
                                ( on_hook; UC[...]
                                []
                                (([tm]i; [0]int_comm; exit
                                 <int_comm]{int_comm: exit}) >>UC[...])
                                )
                        )
                )
    ENDPROC
    PROCESS UGN[  ini_comp, sound_enable, on_hook, int_comp, number1, number2,
                  number3, number4, interrupt, sync_end ] :=
        ini_comp;
                (((
                        [0, tp]number1; [0, ts]number2; [0, ts]number3;
                        [0, ts]number4; [0]interrupt; stop
                        [>
                        (       [tc]i; int_comp; exit
                                []
                                on_hook; int_comp; exit
                        )
                   ) <number1, number2, number3, number4, interrupt]
                   {number1: int_comp; exit,
                    number2: int_comp; exit,
                    number3: int_comp; exit,
                    number4: int_comp; exit,
                    interrupt: sound_enable; exit}
                 ) >> sync_end; UGN[...]
                )
    ENDPROC
    PROCESS UGS[ini_sound, ring, int_sound, ready, on_hook, connect]:=
        ini_sound;
        (       Make_new_sound[ring]
                [>
                (       [tr]i; int_sound; UGS[...]
                        []
                        ready; connect; UGS[...]
                        []
                        on_hook; int_sound; UGS[...]
                )
        )
    WHERE
        PROCESS Make_new_sound[ring]:=
                ([0]ring; [tn]i; exit < ring ] { ring : [tn]i; exit })
                >> Make_new_sound[ring]
        ENDPROC
    ENDPROC
ENDSPEC
```

Na especificação da Central, as ações que não são de comunicação com o terminal são ocultadas, por serem ações utilizadas somente para sincronização interna dos processo, e portanto não devem ser percebidas pelo ambiente externo. O comportamento geral do sistema é a composição paralela das unidades da central, descritas no início da seção 5.2.

A comunicação com o terminal se dá de acordo com o descrito na figura 5.3. Porém algumas observações são necessárias. Os números na especificação são vistos como 4

números distintos. Isto serve apenas para diferenciar a ordem da discagem. Na verdade poderia se modelar a discagem como a repetição da mesma ação `number`. As ações `interrupt` e `int_comm` servem para modelar uma queda da ligação causada pela central, implicando no término da conexão.

A ação `free` apenas sinaliza que a central está livre, podendo aceitar outra conexão. É utilizada para a validação das propriedades. A ação `sync_end` é uma sincronização interna para a retomada dos comportamentos pelas unidades da central.

A Unidade de Controle é descrita no processo UC, que controla a retirada do telefone do gancho, até que a conexão se encerre, num tempo máximo de *tm* segundos. Este tempo máximo é garantido por um mecanismo de *timeout*, construído em RT-LOTOS por um processo do tipo (`P [] [tm]i; Q.`) , onde o processo `P` tem *tm* segundos para iniciar, ou a escolha é decidida para o processo `Q`, por uma ação interna urgente no instante *tm*. A interrupção da conexão pela UC é representada pela ação `int_comm`.

A Unidade de Gestão do Número é descrita no processo UGN, que controla a composição do número a ser chamado. A discagem do primeiro dígito deve ser feita antes de *tp* segundos, ou a composição é interrompida. Depois do início da composição do número chamado, os números seguintes devem ser discados em intervalos de no máximo *ts* segundos. Também há a restrição do tempo total da composição do número, que não deve exceder *tc* segundos. Este tempo total é gerenciado por um mecanismo de *watchdog*, especificado em RT-LOTOS por uma construção do tipo (`P [> [tc]i; Q`), onde o processo `P` tem *tc* segundo para terminar com sucesso, ou é interrompido pela ação interna `i`, sendo que o processo `Q` toma seu lugar. O operador de preempção temporal interrompe a composição quando o tempo máximo entre a discagem de cada dígito é excedido.

A Unidade de Gestão do Som faz o controle do som da campainha no telefone chamado. O processo `Make_new_sound` representa o toque da campainha, efetuado em intervalos de `tn` segundos. O toque da campainha pode ser interrompido por três eventos: quando o fone chamado for retirado do gancho (`ready`), quando o terminal que originou a chamada desistir (`on_hook`), ou quando o tempo máximo de chamada (*tr*) for excedido.

5.2.3 Validação da Especificação Através da Simulação

As propriedades que serão validadas na especificação da central telefônica são as seguintes:

- *Tempo de conexão*: após estabelecida uma conexão, a mesma deve ser finalizada em até *tm* segundo. Esta finalização deve ser feita pelo usuário ou por uma interrupção pela Central Telefônica.
- *Tempo de utilização*: após uma demanda de comunicação, a Central Telefônica deverá estar livre no mais tardar depois de *tc* + *tr* + *tm* segundos.

- *Tempo de composição do número*: o primeiro dígito deverá ser obtido em até *tp* segundos, e o número completo deverá ser obtido em até *tc* segundos. Caso contrário, a Central Telefônica encerra a composição do número e torna-se disponível novamente.
- *Tempo de chamada*: caso o número chamado não responder em até *tr* segundos, a Central Telefônica encerra esta chamada e torna-se disponível novamente.

Os valores utilizados para a especificação a ser simuladas, para as constantes acima, são: $tm = 180$, $tc = 40$, $tp = 20$, $ts = 10$, $tr = 60$, $tn = 3$.

5.2.3.1 Traços de Simulação

Tempo de conexão

A geração do traço para validar o tempo máximo da conexão utiliza os disparos no limite inferior de tempo para completar a discagem em todas as possibilidades. A ocorrência de violações temporais é aleatória, podendo assim uma chamada ser interrompida ou completar-se normalmente. O que se deseja verificar é que, depois de completada a ligação (`connect`), ou a ligação é encerrada pelo chamador (`on_hook`) ou a central interrompe a chamada quando o tempo máximo é alcançado (`int_comm`). Assim, o traço destas três ações é inspecionado. O tempo máximo para as ações não temporizadas é de 400 segundos. Assim, cria-se um não determinismo entre o término da chamada pelo usuário ou pela interrupção da central. Se este tempo fosse menor que 180, as chamadas seria sempre terminadas pelo usuário.

```
<1015 - on_hook>
<1733 - on_hook>
<2643 - i(connect)>
<2724 - on_hook>
<4129 - on_hook>
<5811 - i(connect)>
<5958 - on_hook>
<11201 - on_hook>
```

```
<14944 - on_hook>
<17099 - on_hook>
<20115 - i(connect)>
<20258 - on_hook>
<21561 - i(connect)>
<21741 - i(int_comm*)>
<21981 - on_hook>
<22682 - i(connect)>
```

```
<22862 - int_comm>
<24104 - on_hook>
<24534 - on_hook>
<25304 - i(connect)>
<25484 - i(int_comm*)>
<26526 - on_hook>
<26591 - i(connect)>
<26721 - on_hook>
```

A inspeção do traço acima mostra que depois de estabelecida a conexão (`connect`), ou o usuário termina a chamada (`on_hook`), ou depois de 180 unidades de tempo (*tm*) a central interrompe a chamada (`int_comm`). Este interrupção é vista pelo usuário como uma queda de linha. Valida-se assim a propriedade do tempo máximo de conexão.

Tempo de utilização

Esta propriedade exige que, após uma demanda de comunicação, a central esteja livre em $tc + tr + tm$ segundos, ou seja, depois de 280 segundos. A demanda de comunicação começa com a retirada do fone do gancho (`off_hook`). A central estará livre novamente quando ocorrer a ação `free`. Assim, o traço a ser inspecionado é o destas duas ações. O disparo deverá ser feito com as ações ocorrendo em instantes aleatórios de tempo. Isto porque uma conexão se dá com uma combinação de tempos na discagem. Se as ações forem disparadas no limite superior, não há possibilidade de completar a ligação. Se forem

disparadas no limite inferior, o tempo máximo não pode ser determinado. As violações temporais também devem ser aleatórias, já que a conexão exige que todas as ações ocorram normalmente, com exceção da ação `interrupt`, que não deve ocorrer. O traço gerado é o seguinte:

```
<0 - i(free)>
<72 - off_hook>
<112 - i(free)>
<318 - off_hook>
<401 - i(free)>
<625 - off_hook>
<665 - i(free)>
<864 - off_hook>
<945 - i(free)>
<1339 - off_hook>
<1420 - i(free)>
<1621 - off_hook>
<1701 - i(free)>
<1970 - off_hook>
<2045 - i(free)>
<2329 - off_hook>
<2369 - i(free)>
<2514 - off_hook>
<2604 - i(free)>
<2773 - off_hook>
<2855 - i(free)>
<3161 - off_hook>
<3238 - i(free)>
<3284 - off_hook>
```

A inspeção do traço mostra que depois da ação `off_hook`, a ação `free` ocorre sempre num tempo menor do que o limite estabelecido (280). O traço mostrado é pequeno, por restrições de espaço. A geração de traços muito grandes dificulta a análise, pois cada diferença tem que ser calculada manualmente.

Uma funcionalidade em estudos para a implementação na ferramenta RTLS é a geração de gráficos de intervalos de tempo. Esta funcionalidade será útil para análises como a acima, já que pode analisar um traço de simulação muito grande, e ao final pode reportar automaticamente o intervalo máximo e mínimo entre a ocorrência de duas ações (ou a periodicidade de uma ação).

Tempo de composição do número

O primeiro número deve ser obtido em até *tp* segundos. Para validar esta propriedade, deve-se verificar se após a retirada do fone do gancho (`off_hook`), o primeiro número (`number1`) é obtido em até 20 segundos, que é valor de *tp* adotado na especificação. O traço destas duas ações é analisado, e o disparo é feito no limite superior das ações. O traço obtido é o seguinte:

```
<72 - off_hook>
<92 - i(number1*)>
<471 - off_hook>
<491 - i(number1*)>
<697 - off_hook>
<717 - number1>
<1004 - off_hook>
<1024 - number1>
<1338 - off_hook>
<1358 - i(number1*)>
<1583 - off_hook>
<1603 - number1>
<1938 - off_hook>
<1958 - i(number1*)>
<2330 - off_hook>
<2350 - number1>
<2595 - off_hook>
<2615 - number1>
<2686 - off_hook>
<2706 - i(number1*)>
<2930 - off_hook>
<2950 - i(number1*)>
<2954 - off_hook>
<2974 - i(number1*)>
```

O traço gerado mostra que após a retirada do fone do gancho, ou a discagem do primeiro dígito começa em 20 segundos, ou neste instante ocorre a violação temporal da ação `number1`, que implica na interrupção da composição do número. Valida-se desta forma a propriedade do tempo máximo para o início da discagem.

A propriedade pede ainda que a composição do número total não exceda o tempo de *tc* segundos. Será gerado um traço sob as mesmas condições do anterior, porém observando agora todos os números (`number1`, `number2`, `number3` e `number4`). Além disso, a interrupção da composição do número pela central (`int_comp`) ou pelo usuário (`on_hook`) também são observados. O traço gerado é:

```
<72 - off_hook>
<92 - on_hook>
<92 - i(int_comp)>
<439 - off_hook>
<459 - on_hook>
<459 - i(int_comp)>
<777 - off_hook>
<797 - number1>
<807 - on_hook>
<807 - i(int_comp)>
<972 - off_hook>
```

```
<992 - number1>
<1002 - number2>
<1012 - i(int_comp)>
<1082 - off_hook>
<1102 - on_hook>
<1102 - i(int_comp)>
<1135 - off_hook>
<1155 - number1>
<1165 - number2>
<1175 - number3>
<1175 - i(int_comp)>
```

```
<1506 - off_hook>
<1526 - on_hook>
<1526 - i(int_comp)>
<1850 - off_hook>
<1870 - number1>
<1880 - on_hook>
<1880 - i(int_comp)>
<2211 - off_hook>
<2231 - number1>
<2241 - number2>
<2251 - i(int_comp)>
```

A inspeção do traço acima mostra que durante a discagem dos dígitos, ou o usuário interrompe a composição (on_hook), ou a central interrompe a composição no limite superior do tempo máximo de composição (40 segundo). Note que o quarto dígito (number4) nunca é discado. Isto porque a soma dos tempos máximos dos três números anteriores já se iguala ao tempo máximo. Como os disparos estão sendo feito nos limites superiores, o quarto dígito não aparece na simulação. Desta forma a restrição do tempo máximo de composição do número é validada.

Tempo de chamada

Quando um número é discado, a central começa a enviar, repetitivamente, sinais de toque de campainha para o telefone chamado. A propriedade sendo analisada exige que, depois do primeiro toque (ring), se o telefone chamado não atender em *tr* segundos (connect), a ligação é encerrada pela central (int_sound). Para a validação desta propriedade, será gerado um traço observando as ações acima. As ações ocorrem em momentos aleatórios do intervalo, e as violações temporais podem ou não ocorrer. O traço gerado é o seguinte:

```
<633 - ring>
...
<645 - i(ring*)>
<646 - i(connect)>
<970 - ring>
...
<1030 - ring>
```

```
<1030 - i(int_sound)>
<1363 - i(ring*)>
...
<1420 - i(ring*)>
<1423 - i(int_sound)>
<1723 - i(ring*)>
...
```

```
<1750 - ring>
<1752 - i(connect)>
<2086 - i(ring*)>
...
<2146 - i(ring*)>
<2146 - i(int_sound)>
<2861 - i(ring*)>
```

As ocorrências repetitivas de ring e ring* foram retiradas do traço para facilidade de visualização. Pode-se ver que depois do primeiro toque da campainha (ring ou ring*), ou o telefone chamado atende (connect), ou a central interrompe a chamada (int_sound) depois de 60 segundos. Valida-se assim a propriedade do tempo máximo de chamada.

Note-se que a ação ring pode ou não ocorrer. Isto porque a modelagem da central preocupa-se apenas em enviar periodicamente o sinal de toque de campainha para o terminal chamado. O fato da campainha efetivamente tocar (ring) ou não (ring*) não implica em comportamentos diferentes da especificação.

5.2.3.2 Árvore de Execução

A árvore de execução completa da especificação da Central Telefônica contém 195 estados diferentes, com 575 estados analisados no total. A geração desta árvore levou 10

segundos para ser concluída na máquina descrita no início deste capítulo. Não será apresentada aqui para economia de espaço.

A inspeção da árvore mostra que a especificação é livre de deadlocks, o que demonstra que todas as possíveis condições de erro são recuperadas pela especificação. Por ser uma árvore completa, pode-se afirmar que a especificação apresenta um comportamento infinito recursivo.

A análise das propriedades acima exige uma inspeção rigorosa da árvore, sendo omitida neste trabalho. Apenas a propriedade do tempo máximo de conexão é fácil. A única ocorrência da ação `connect` na árvore é a seguinte:

```
23 - i (connect) - (0)
 24 - on_hook - (0)
  25 - Recursion detected - 1 (0)
 24 - time(180) - (0)
  26 - on_hook - (180)
   27 - Recursion detected - 1 (180)
  26 - i (i) - (180)
   28 - int_comm - (180)
    29 - i (EXIT) - (180)
     30 - Recursion detected - 1 (180)
   28 - i (int_comm*) - (180)
    31 - i (EXIT) - (180)
     32 - Recursion detected - 1 (180)
```

A inspeção deste trecho da árvore mostra que depois da chamada ser completada, ou ocorre o desligamento expontâneo (`on_hook`) em até 180 segundos, ou a ligação é encerrada pela central (`int_comm`). Assim, a primeira propriedade é validada pela árvore de execução.

Como já dito, todas as outras propriedades podem ser validadas de forma análoga, pela análise da árvore, sendo omitidas neste trabalho.

5.3 Cruzamento Rodovia-Ferrovia

O problema descrito a seguir é o clássico Cruzamento Rodovia-Ferrovia (*RailRoad Crossing*) [AlHe92]. Em [HeLy94] foi proposta uma forma generalizada deste problema, onde um número genérico de trens viaja pelo cruzamento. Porém, por não suportar a parte de dados, a ferramenta RTLS não permite a representação genérica, tendo que o número de trens ser arbitrado. Com esta restrição, é possível representar o problema em RT-LOTOS básico, validando a especificação com a ferramenta proposta.

5.3.1 Descrição Informal do Problema

O problema consiste de um sistema controlador que deve operar um portão, controlando o acesso a este cruzamento. Assim, um cruzamento I se concentra em uma região R, e um conjunto de trens viaja através de R em vários trilhos. Um sistema de sensores indica quando um trem entra e sai da região R, que corresponde ao local onde o sensor está localizado até a saída do cruzamento I.

O sistema de controle deve satisfazer as seguintes propriedades:

- Segurança: O portão deve ser fechado quando um trem se aproximar do cruzamento.
- Utilização: O portão deve permanecer aberto quando nenhum trem estiver no cruzamento. Quando estiver fechado e com um trem saindo do cruzamento, deve ser aberto quando não houver um novo trem a caminho do cruzamento.

São definidas algumas constantes para os tempos no sistema que controla o cruzamento, a saber:

ε_1 → limite inferior no tempo para um trem entrar em R e chegar em I (trem mais veloz)

ε_2 → limite superior no tempo para um trem entrar em R e chegar em I (trem mais lento).

γ_{down} → tempo máximo que o portão leva para fechar completamente.

γ_{up} → tempo máximo que o portão leva para abrir completamente

β → constante que representa o atraso do sinal enviado pelo sensor.

Para estes valores, algumas restrições são necessárias:

$\varepsilon_1 \leq \varepsilon_2$ → logicamente, o tempo do trem mais veloz é menor que o do trem mais lento.

$\varepsilon_1 > \gamma_{down} + \gamma_{up}$ → com esta restrição, é garantido que o trem mais veloz alcança I em um tempo superior ao tempo necessário para abrir e fechar o portão, mais o tempo mínimo útil do portão.

5.3.2 Especificação Formal do Cruzamento Rodovia-Ferrovia

O problema do cruzamento Rodovia-Ferrovia foi especificado em RT-LOTOS conforme a especificação a seguir. Novamente, as portas nas instanciações dos processo foram omitidas para uma melhor visualização, sem prejuízo para o entendimento da especificação.

```
specification RailRoad[in_r, in, out, up, down]
behaviour
      hide [ stop_the_train, approach, leaving, lower, raise, siren,
             warning_control_center] in
            ((( Train[...] ||| Train[...] )
            |[approach, leaving, stop_the_train]|
            Controller[...] )
            |[lower, raise, siren]|
            Gate[...] )
where
      process Train[   in_r, approach, in, out, leaving, warning_control_center,
                       stop_the_train ] :=
            TrainMotion[...] [> Emergency[stop_the_train]
      where
```

```
        process TrainMotion[   in_r, approach, in, out, leaving,
                               warning_control_center ] :=
                (in_r; approach; [ε1, ε2]in; out; leaving; exit
                <in] {in: warning_control_center; stop})
                >> TrainMotion[...]
        endproc
        process Emergency[stop_the_train]:=
                stop_the_train; stop
        endproc
endproc
process Controller[   approach, leaving, lower, raise, siren, stop_the_train
                      ] :=
        Control[...] [> Exception[...]
where
        process Control[ approach, leaving, lower, raise ] :=
                approach; [β]i; lower; Control_Entries[...]
                >> [β]i; raise; Control[...]
        where
                process Control_Entries[approach, leaving]:=
                        leaving; exit
                        []
                        approach; leaving; leaving; exit
                endproc
        endproc
        process Exception[siren, stop_the_train]:=
                siren; stop_the_train; stop_the_train; stop
        endproc
endproc
process Gate[  lower, raise, up, down, warning_control_center, siren] :=
        ((      lower; [0, γdown]down; exit
                []
                raise; ([0, γup]up; exit [] lower; [0, γdown]down; exit)
         )
        <up, down]
        {       up: warning_control_center; stop,
                down: siren; stop
        }) >> Gate[...]
endproc
endspec
```

A especificação do cruzamento foi restrito ao controle de dois trens, devido à impossibilidade de representar o problema de forma genérica. Uma especificação genérica envolveria a manipulação de dados, não suportada pela ferramenta.

A especificação consiste na composição paralela de dois processos que modelam a passagem dos trens pelo cruzamento, que evoluem independentemente. Também o comportamento do portão é modelado, representando o fechamento e a abertura do cruzamento, dentro dos limites de tempo estabelecidos. Todos estes processos por sua vez sincronizam-se com o controle do cruzamento, que faz o controle da entrada e saída dos trens, e a abertura e o fechamento do portão, tratando as possíveis situações de erro.

Somente as ações `in_r`, `in`, `out`, `up` e `down` são ações externas ao sistema, portanto devem ser visíveis. As demais ações são sincronizações internas dos componentes do sistema, devendo portanto serem ocultadas do ambiente.

O processo `Train` modela o comportamento de um trem. Após a entrada do trem na região `R`, representado pela ação `in_r`, é enviado para o controlador um sinal de aproximação (`approach`). Em seguida, o trem deverá entrar no cruzamento (`in`), dentro do

limite de tempo determinado ([ε_1, ε_2]); caso isto não ocorra, um aviso é enviado para a central de controle, pois ocorreu algum problema com o trem. Quando o trem sair do cruzamento (out), ele deverá sinalizar a sua saída ao controlador (leaving). O processo Emergency é utilizado para que, a qualquer instante, o trem possa ser parado pelo controlador (stop_the_train).

O portão é modelado pelo processo Gate. O controlador enviará um sinal para que o portão seja abaixado (lower), o que ocorrerá efetivamente dentro do limite de tempo possível ([0, γ_{down}]). Porém, caso ocorra algum problema com o portão, representado pela violação temporal da ação down, será acionado um alarme, fazendo os trens pararem. Da mesma forma, o controlador enviará um sinal para que o portão seja levantado (raise), que também ocorrerá efetivamente dentro do limite de tempo possível ([0, γ_{up}]). Se ocorrer alguma falha nesta situação, a central de controle é avisada. Se antes de completar a abertura do portão um novo trem entrar no cruzamento, a abertura é cancelada e o portão torna a fechar-se.

A central de controle é representada no processo Controller. Após receber o sinal de aproximação de um trem (approach), o controlador sinaliza ao portão para que este seja fechado, caso ele já não esteja. O atraso (β) é utilizado somente para representar uma possível correção do sistema, tal como um atraso do sinal enviado. Caso um trem saia do cruzamento (leaving), o controlador verifica se não existe outro trem a caminho do cruzamento ou até dentro do cruzamento, sinalizando para a abertura do portão se não ocorrer nenhuma destas possibilidades. O processo Control_Entries trata exatamente destas situações. Se o portão sinalizar alguma situação de alarma (siren), o controlador promove a parada dos trens.

5.3.3 Validação da Especificação Através da Simulação

Na especificação analisada, os seguintes valores foram atribuídos às constantes do problema: $\varepsilon_1 = 35$, $\varepsilon_1 = 40$, $\gamma_{down} = 10$ $\gamma_{down} = 12$ e $\beta = 1$.

As seguintes propriedades da especificação serão validadas através de simulação:

- *Segurança*: Após a entrada de um trem no cruzamento, o portão deverá estar fechado no seu limite de tempo γ_{down}, ou então sinalizar a falha ao controlador, para que este promova a parada dos trens. O portão também pode já estar fechado devido ao fato do outro trem já estar no cruzamento.
- *Utilização*: Após a saída de um trem do cruzamento, o portão deverá ser aberto até o seu limite de tempo γ_{up}, ou sinalizar uma falha para o controlador, ou então permanecer fechado se existir outro trem no cruzamento.

5.3.3.1 Traços de Simulação

Segurança

Esta propriedade exige que o portão seja fechado no seu limite de tempo γ_{down}. Desta forma, depois de receber um sinal de que um novo trem entrou na região R (`approach`), o controlador deverá enviar um sinal para o portão (`lower`), para que este feche. Este envio tem o atraso de transmissão β. Desta forma, temos a restrição de que, após a ocorrência da ação `approach`, a ação `down` deverá ocorrer em $\gamma_{down} + \beta = 11$ segundos. Porém, para entradas enquanto o portão está fechado, não é necessária a restrição. Assim, é necessário um `down` depois de um `approach` somente depois da ação `raise`.

Para validar esta propriedade, será gerado um traço das ações `approach`, `raise` e `down`, com o disparo das ações no limite superior, sem ocorrência de violações temporais.

```
<59 - i(approach)>
<70 - down>
<72 - i(approach)>
<108 - i(raise)>
<120 - i(approach)>
<121 - i(approach)>
<131 - down>
<254 - i(raise)>
<266 - i(approach)>
```

```
<275 - i(approach)>
<277 - down>
<349 - i(raise)>
<349 - i(approach)>
<360 - i(approach)>
<360 - down>
<491 - i(raise)>
<497 - i(approach)>
<508 - down>
```

```
<513 - i(approach)>
<625 - i(raise)>
<649 - i(approach)>
<660 - i(approach)>
<660 - down>
<766 - i(raise)>
<778 - i(approach)>
<786 - i(approach)>
<789 - down>
```

Inspecionando o traço acima, pode-se ver que para todas as ocorrências de `approach` após uma ação `raise`, a ação `down` ocorrem no limite de tempo especificado. Valida-se assim parte da propriedade de segurança.

A propriedade de segurança diz ainda que se um trem entrar na região R e o portão não fechar no tempo especificado, os trens devem ser parados. Para validar esta restrição, será gerado um traço de ações ocorrendo no limite superior do intervalo de tempo, e forçando as violações temporais. Todas as ações são observadas. O traço gerado é mostrado a seguir:

```
<0 - 57>
<57 - in_r>
<57 - i(approach)>
<57 - 1>
<58 - i(i)>
<58 - i(lower)>
```

```
<58 - 10>
<68 - i(down*)>
<68 - i(siren)>
<68 - i(stop_the_train)>
<68 - i(stop_the_train)>
```

No traço acima, vê-se que somente um trem está no cruzamento no momento da falha do portão (`down*`). Porém, como especificado na propriedade de segurança, todos os trens são parados. Este traço leva a especificação a um estado de *deadlock*. Ou seja, uma falha pára todo o sistema.

Utilização

A propriedade de utilização pede que, uma vez que todos os trens estejam fora do cruzamento, o portão esteja aberto no máximo em γ_{up} segundos. Com o atraso de transmissão, pode-se dizer que depois do último trem deixar o cruzamento, o portão deverá

estar aberto em $\gamma_{up} + \beta = 13$ segundos, caso não entre outro trem no cruzamento neste intervalo.

Para validar esta propriedade, será gerado um traço com as ações ocorrendo no limite superior do intervalo, sem a ocorrência de violações temporais. O traço gerado é apresentado a seguir:

```
<59 - i(approach)>
<72 - i(approach)>
<104 - i(leaving)>
<107 - i(leaving)>
<120 - i(approach)>
<120 - up>
<121 - i(approach)>
<214 - i(leaving)>
<253 - i(leaving)>
<266 - i(approach)>
<266 - up>
<275 - i(approach)>
<311 - i(leaving)>
<348 - i(leaving)>
<349 - i(approach)>
<360 - i(approach)>
<446 - i(leaving)>
<490 - i(leaving)>
<497 - i(approach)>
<513 - i(approach)>
<571 - i(leaving)>
<624 - i(leaving)>
<637 - up>
<649 - i(approach)>
<660 - i(approach)>
<756 - i(leaving)>
<765 - i(leaving)>
<778 - i(approach)>
<778 - up>
<786 - i(approach)>
<823 - i(leaving)>
<870 - i(leaving)>
<883 - up>
<884 - i(approach)>
<895 - i(approach)>
<1001 - i(leaving)>
<1003 - i(leaving)>
<1016 - i(approach)>
<1016 - up>
<1017 - i(approach)>
<1148 - i(leaving)>
<1158 - i(leaving)>
<1171 - up>
<1220 - i(approach)>
```

Pela inspeção do traço acima, pode-se ver que sempre que o cruzamento fica vazio por um tempo igual ao especificado, o portão está aberto. Em alguns casos, o portão termina sua abertura exatamente no mesmo instante em que um novo trem sinaliza sua chegada à região R. Porém a abertura não é interrompida, pois o portão não recebe o sinal no mesmo instante, e sim um segundo depois, devido ao atraso de transmissão β. Valida-se assim a propriedade de utilização.

Além dos traços acima, é gerado um traço com as ações observáveis. As ações ocorrem em intervalos aleatórios, com a possibilidade de violações temporais. Desta forma, as duas propriedades podem ser validadas em conjunto.

```
<59 - in_r>
<68 - down>
<72 - in_r>
<100 - in>
<105 - in>
<138 - out>
<172 - in_r>
<186 - out>
<191 - down>
<212 - in_r>
<216 - in>
<253 - in>
<283 - out>
<310 - out>
<320 - up>
<352 - in_r>
<358 - down>
<369 - in_r>
<394 - in>
<406 - in>
<409 - out>
<442 - out>
<445 - up>
<450 - in_r>
<454 - down>
<487 - in>
<497 - in_r>
<511 - out>
<535 - in>
<539 - out>
<546 - up>
<587 - in_r>
<588 - down>
```

Por este traço pode-se ver que sempre que um trem entra no cruzamento, o portão está fechado no tempo correto. E sempre que o último trem sai, se não houver a chegada de outro trem, o portão é aberto, também no seu tempo correto.

5.3.3.2 Árvore de Execução

A árvore de execução para o problema do cruzamento rodovia-ferrovia apresenta 416 estados diferentes, num total de 768 estados analisados. Foram necessários 12 segundos

para a geração desta árvore na máquina descrita no início do capítulo. Não será apresentada neste trabalho, por motivos de espaço.

A inspeção da árvore mostra que a mesma apresenta *deadlocks*. Estes *deadlocks* ocorrem após a sinalização de erro para o controlador, pelo fato de um trem não entrar no cruzamento dentro do intervalo $[\varepsilon_1, \varepsilon_2]$ depois da sinalização de sua chegada na região R (`in_r`). Também ocorre *deadlock* quando o portão não se abre dentro do tempo estabelecido (γ_{up}) depois da saída do último trem no cruzamentos. Estas duas condições levam a especificação a um *deadlock* depois da ação `warning_control_center`. Já a falha do fechamento do portão no tempo estabelecido (γ_{down}) faz com que a especificação entre em *deadlock* depois da parada dos dois trens. Assim, depois de uma ação `siren`, ocorrem duas ações `stop_the_train` e a especificação passa para um estado de *deadlock*.

Esta inspeção da árvore de execução garante que todas as condições de erro levam a especificação a um estado seguro. O não fechamento do portão pára os trens. Já uma falha na abertura do portão não pára os trens, apenas emite um sinal de aviso (`warning_control_center`). Não é impedido que outro trem entre no cruzamento. Quando um trem não entrar no cruzamento no intervalo $[\varepsilon_1, \varepsilon_2]$ depois da sinalização de sua chegada na região R, a especificação não impede que outro trem entre no cruzamento. Somente é emitido um sinal de aviso (`warning_control_center`).

Nenhuma outra condição leva a especificação a *deadlock*. Pode-se considerar assim todas as propriedades desejadas validadas.

5.4 Algoritmo de Exclusão Mútua

O algoritmo de exclusão mútua foi proposto, originalmente, por [Fischer85] e descrito por Lamport [Lamport87], sendo um caso clássico no estudo dos métodos formais de verificação de sistemas dependentes de tempo [LSGL94, YPD94]. O objetivo deste algoritmo é garantir a exclusão mútua num sistema concorrente, consistindo de vários processos que utilizam uma variável compartilhada para o auxílio no acesso de uma seção crítica. Ou seja, somente um processo pode ter acesso à seção crítica em um determinado tempo.

5.4.1 Descrição Informal do Algoritmo

Assume-se que cada processo tem um identificador distinto. Uma descrição abstrata do algoritmo pode ser feita da seguinte forma:

```
Processo i:
start :    wait for x = 0
           x := i
           delay
           if x ≠ i then goto start
```

```
seção crítica
x := 0
```

Assim, cada processo i que solicita o acesso a região crítica deve primeiramente esperar até que a variável compartilhada x seja inicializada com o valor zero pelo último processo que utilizou a seção crítica, indicando que nenhum outro processo está utilizando a região crítica. Logo após, o processo atribui o valor do seu identificador à variável compartilhada. Desde que vários processos possam competir pela região crítica, o processo deverá esperar até que o valor da variável compartilhada estabilize, verificando em seguida se o valor desta ainda é o valor de seu identificador. O processo para o qual o valor do identificador for igual ao valor da variável compartilhada, ou seja, o último a inicializá-la, terá o direito de acessar a região crítica, sendo que os demais deverão esperar até que a variável seja reinicializada novamente, reiniciando o procedimento de escolha. Quando um processo termina a utilização da região crítica, o valor da variável compartilhada é retornado a zero. Para evitar a violação da exclusão mútua devido a possibilidade de uma diferença na velocidade dos processos, são adotadas restrições que estabelecem que todo processo deve esperar um tempo suficiente para que os outros processos verifiquem o novo valor da variável compartilhada.

5.4.2 Especificação formal do algoritmo em RT_LOTOS

A especificação formal do algoritmo de exclusão mútua, em RT-LOTOS, é apresentada a seguir.

```
specification MutEx [vo, v, v1, v2, v3, s1, s2, s3, start, end]
behaviour
       hide [vo, v, v1, v2, v3, s1, s2, s3] in
       (( P[vo, v, v1, s1, start, end] |||
          P[vo, v, v2, s2, start, end] |||
          P[vo, v, v3, s3, start, end])
       |[vo, v, v1, v2, v3, s1, s2, s3]|
       Shared_Var[vo, v, v1, v2, v3, s1, s2, s3])
where
       process P[vo, v, vi, si, start, end] :=
              i; v; [0, a]i; vi; [b, c]si; start; i; end; vo; stop
              < si ] {si : P[vo, v, vi, si, start, end]}
       endproc
       process Shared_Var[vo, v, v1, v2, v3, s1, s2, s3] :=
              v; ((Cmp[v1, v2, v3, s1, s2, s3]
              >>
              vo; Shared_Var[vo, v, v1, v2, v3, s1, s2, s3])
              [] Shared_Var[vo, v, v1, v2, v3, s1, s2, s3])
       where

              process Cmp[v1, v2, v3, s1, s2, s3] :=
                     v1; (s1; exit [] Cmp[v1, v2, v3, s1, s2, s3])
                     []
                     v2; (s2; exit [] Cmp[v1, v2, v3, s1, s3, s3])
                     []
                     v3; (s3; exit [] Cmp[v1, v2, v3, s1, s2, s3])
              endproc
       endproc
endspec
```

A especificação RT-LOTOS do algoritmo de exclusão mútua restringe-se, neste exemplo, a três processos, devido ao fato de não ser possível a representação da variável compartilhada de forma genérica. Este restrição se deve ao fato do simulador suportar somente RT-LOTOS Básico. A adição de novos processos concorrentes é feita de forma trivial.

Os processos concorrentes são representados pelo processo RT-LOTOS `P`. Cada processo, após verificar, por meio da ação `"v"`, indicando que a variável possui o valor zero, que a região crítica não está sendo utilizada, deve atribuir o valor de seu identificador `"vi"` à variável num limite de tempo definido pelo intervalo `[0, a]`. A espera pela estabilização do valor da variável ocorrerá dentro do limite `[b, c]`, sendo que se a ação `"si"` não ocorrer, indicando que a variável não possui o mesmo valor do identificador do processo, o processo terá que esperar até que a ação `"v"` fique disponível novamente para outra tentativa de acesso. A ação `"vo"` representa a reinicialização da variável compartilhada, executada quando um processo libera a região crítica.

O controle sobre o acesso à variável compartilhada é modelado pelo processo `Shared_Var`. Cada processo pode atualizar sua variável, sendo que durante a espera pela estabilização da variável, outro processo pode acessá-la. Isto é representado pelo processo `Cmp`. Se um processo conseguir o acesso à região crítica, a mesma é bloqueada até que o processo que a ocupa liberá-la, pela ação `"vo"`.

Toda as ações necessárias para o controle de acesso à variável compartilhada tiveram que ser ocultadas pelo operador hide, para tirarem proveito da propriedade da progressão máxima. Caso contrário, a exclusão mútua não seria mais garantida.

5.4.3 Validação da Especificação Através da Simulação

Para a simulação da especificação RT-LOTOS do problema da exclusão mútua, foram utilizados os seguintes valores para as constantes do problema: a = 2, b = 3 e c = 5.

As propriedades da especificação a serem validadas pela simulação são as seguintes:

- *Exclusão mútua*: depois de um processo conseguir o acesso à região crítica, nenhum outro processo pode entrar na mesma, antes que esta seja liberada.
- *Tempo de acesso*: esta propriedade foi apresentada em [LSGL94], e estabelece que, depois que o primeiro processo tente acessar a região crítica, testando a variável compartilhada, há um tempo máximo de 2c + 5a para que a região crítica seja acessado. Este limite de tempo é obtido pela análise das três fases mais importantes do algoritmo. A primeira estabelece que algum processo alcançará a região crítica em, no máximo, c + a. A segunda diz que, com a região crítica disponível, o primeiro evento importante será quando a variável compartilhada passar do valor zero para o valor do identificador de um processo, o que ocorrerá

até a. A última fase estabelece que a variável se tornará estável em c + 3a. Somando-se estes três tempos, chega-se ao limite acima.

A segunda propriedade não é mantida na especificação apresentada, devido à propriedade de progressão máxima introduzida pelo operador `hide`. Porém a ocultação das ações não pode ser dispensada, sob pena de introduzir *deadlocks* não desejados na especificação. Estes *deadlocks* são resultantes da não sincronização das ações, mesmo quando esta sincronização é possível.

Desta forma, como todas as ações irão ocorrer no primeiro instante possível, pode-se dizer que a região crítica será sempre acessada em a + b instantes de tempo. Os processos irão evoluir em paralelo, sendo que a decisão de quem irá acessar a região crítica se dá pela ordem de sincronização com o processo `Shared_Var`.

5.4.3.1 Traços de Simulação

Exclusão Mútua

A exclusão mútua implica que uma vez que um processo tem acesso à região crítica, nenhum outro terá. Os limites da região crítica são as ações `start` e `end`. Portanto, basta verificar se há a ocorrência de duas ações `start` sem a ocorrência de uma ação `end` entre eles. Para tal, serão gerados traços observando tais ações.

O primeiro traço foi gerado com as ações ocorrendo no seu limite inferior, permitindo a ocorrência de violações temporais. O resultado é o seguinte:

```
<12 - start>
<13 - end>
<16 - start>
<21 - end>
<31 - start>
<37 - end>
```

O segundo traço foi gerado com as ações ocorrendo no limite superior, sem violações temporais:

```
<7 - start>
<7 - end>
<16 - start>
<25 - end>
<30 - start>
<33 - end>
```

O terceiro traço também faz o disparo das ações no limite superior do intervalo associado, porém permitindo ocorrências de violações temporais:

```
<13 - start>
<18 - end>
<28 - start>
<32 - end>
<44 - start>
<50 - end>
```

Todos os traços apresentam a propriedade de exclusão mútua. Ou seja, depois da ocorrência da ação `start`, que representa a entrada de um processo na região crítica,

nenhum outro processo irá entrar na mesma, até que o processo libere a região crítica. Esta liberação é representada pela ação `end`. Pode-se assim considerar validada a propriedade da exclusão mútua.

Tempo de acesso

O tempo de acesso não ultrapassa o limite de 5 unidades de tempo (`a + b`), entre o momento que o primeiro processo tenta acessar a região crítica, até o momento em que um processo efetivamente entra na mesma. Para a validação desta propriedade, é gerado um traço que dispara as ações no limite superior do intervalo, permitindo violações temporais. Todas as ações são observadas. O traço gerado é:

```
<0 - i(i)>
<0 - i(v)>
<0 - i(i)>
<0 - i(i)>
<0 - i(v)>
<0 - i(v)>
<0 - 2>
<2 - i(i)>
<2 - i(i)>
<2 - i(i)>
<2 - i(v3)>
<2 - i(v2)>
<2 - i(v1)>
<2 - 3>
<5 - i(s1)>
<5 - i(EXIT)>
<5 - 2>
<7 - i(s3*)>
<7 - i(i)>
<7 - i(s2*)>
<7 - i(i)>
<7 - 6>
<13 - start>
<13 - i(i)>
<13 - 5>
<18 - end>
<18 - i(vo)>
<18 - i(v)>
<18 - i(v)>
<18 - 2>
<20 - i(i)>
<20 - i(v3)>
<20 - i(i)>
<20 - i(v2)>
<20 - 3>
<23 - i(s2)>
<23 - i(EXIT)>
<23 - 2>
<25 - i(s3*)>
<25 - i(i)>
<25 - 3>
<28 - start>
<28 - i(i)>
<28 - 4>
<32 - end>
<32 - i(vo)>
<32 - i(v)>
<32 - 2>
<34 - i(i)>
<34 - i(v3)>
<34 - 3>
<37 - i(s3)>
<37 - i(EXIT)>
<37 - 7>
<44 - start>
<44 - i(i)>
<44 - 6>
<50 - end>
<50 - i(vo)>
```

É possível ver que após o primeiro teste da variável compartilhada (`v`), a seção crítica é acessada (s1, s2 ou s3) após 5 unidades de tempo (`a + b`). O processo repete-se após cada liberação da região crítica, até que todos os processos tenham passado por ela.

Este traço representa o comportamento total da especificação. Depois destas ações, a especificação passa a estar num estado de *deadlock*, com todos os processo terminados.

5.4.3.2 Árvore de Execução

A árvore de execução completa da especificação apresentada acima contém 1161 estados diferentes, num total 2290 estados analisados, não sendo apresentada aqui por motivos de espaço. A geração desta árvore levou 7 minutos e 26 segundos para ser concluída na máquina descrita no início deste capítulo. A inspeção da mesma mostra que não há `deadlocks` sem que haja o término de todos os processos. Também é possível verificar que não há possibilidade de se percorrer um caminho que leve a duas ocorrências da ação `start` sem que haja uma ação `end` separando-as, o que valida a propriedade de exclusão mútua.

A propriedade do tempo de acesso à região crítica também pode ser validada pela árvore de execução, porém exige um exame minucioso da mesma, com as substituições dos tempos, sendo omitido neste trabalho.

A árvore gerada é completa, o que garante que o comportamento da especificação é finito.

5.5 Conclusões

Neste capítulo foram feitos estudos de caso para demonstrar o uso da ferramenta RTLS na validação de especificações. Pode-se concluir que a ferramenta mostra-se útil para uma validação inicial das características da ferramenta, complementando a verificação por predicados lógicos. Permite ainda a manipulação de especificações com comportamento infinito, o que não é possível para as técnicas de verificação que exigem a tradução da especificação para um formalismo subjacente como sistemas de transições rotuladas ou grafos temporizados.

A geração de traços necessita de uma combinação de opções de disparo e observação para validar determinadas características. A árvore de execução permite uma análise mais genérica, como a ausência de *deadlocks*, por exemplo. Porém sua utilização é um dificultada pelo seu rápido crescimento para especificações mais complexas.

Pode-se perceber também que o desempenho da ferramenta varia diretamente com a complexidade e o tamanho do texto da especificação. Especificações com grande grau de paralelismo independente de processos leva ao crescimento rápido da árvore de execução. E seu grande número de estados diferentes faz com que a geração da árvore de execução torne-se demorada. Até mesmo a geração dos traços é afetada pela complexidade das especificações, exigindo mais tempo para o cálculo das ações possíveis num estado da especificação e para a atualização da especificação.

Não é possível determinar com exatidão a quantidade máxima de estados que podem ser tratados pela ferramenta, já que isto depende diretamente do tamanho e da complexidade da especificação. Isto devido à abordagem utilizada, onde cada estado é uma representação interna do texto da especificação.

Conclusões

Neste trabalho foi apresentada uma ferramenta para simulação de especificações escritas em RT-LOTOS [Camargo95], uma extensão temporal da Técnica de Descrição Formal padrão LOTOS. O contexto deste trabalho abrange a utilização de métodos formais para a especificação de sistemas dependentes de tempo, e a utilização da ferramenta apresentada no auxílio à concepção de especificações e para a validação de propriedades da mesma.

A ferramenta proposta neste trabalho realiza a simulação de especificações RT-LOTOS através da transformação de uma representação interna da própria especificação. Desta forma, é eliminada a tradução da especificação para um modelo subjacente como, por exemplo, um grafo temporizado. Tal abordagem permite a simulação de especificações que apresentam comportamentos infinitos, não traduzíveis para um grafo temporizado finito. As transformações são feitas com base em funções definidas sobre os operadores da linguagem, constituindo o núcleo de simulação.

A ferramenta apresenta funcionalidades diversas. A simulação interativa permite o acompanhamento detalhado da evolução da especificação, possibilitando a identificação de pontos onde ocorrem erros, e a avaliação dos momentos exatos em que as ações tornam-se disponíveis. A simulação automática permite simular, de maneira rápida, como a especificação se comporta sob diversas circunstâncias. Estas circunstâncias são simuladas com diferentes opções de disparo das ações durante a simulação. Permite ainda o acompanhamento da evolução da especificação em diferentes níveis de detalhamento.

Já a árvore de execução permite que se teste se a especificação tem um comportamento finito ou mesmo se é livre de *deadlocks*. Porém, a explosão de estados em especificações complexas dificulta sua análise, podendo tornar sua utilização inviável, tanto pelo esgotamento da memória da máquina executando a ferramenta, quanto pela complexidade da análise. Também o desempenho da ferramenta é afetado diretamente pela complexidade e pelo tamanho da especificação.

A abordagem de simulação utilizada neste trabalho irá beneficiar uma futura extensão da ferramenta para tratar especificações utilizando RT-LOTOS completo. Isto porque, na tradução para o formalismo subjacente, a fase de *flattening* dos dados é necessária. Nesta fase, cada ação com uma variável associada é desdobrada numa escolha entre diversas ocorrências da ação, cada uma com um dos valores possíveis para a variável associado. Por exemplo, a expressão (`x?flag:boolean; B`) é desdobrada em (`x!true;`

`B [] x!false; B`). Este desdobramento pode tornar uma especificação intratável. Por exemplo, uma variável que possa assumir qualquer valor do conjunto dos números reais.

No nosso caso, para cada ação com dados associados, os valores são definidos interativamente pelo usuário, ou aleatoriamente na simulação automática. Todas as expressões são avaliadas posteriormente com estes valores, determinando assim o comportamento resultante da especificação.

Embora restringindo-se a RT-LOTOS básico, o estudo de casos ilustrou que a ferramenta é bastante útil para a concepção de sistemas

A ferramenta apresentada neste trabalho apresenta característica próximas àquelas desenvolvidas para outras extensões temporais de LOTOS existentes. Porém, o presente trabalho está inserido num contexto que envolve desde o desenvolvimento da linguagem até as ferramentas nela baseadas.

Durante a elaboração deste trabalho, várias foram as funcionalidades identificadas para o complemento da ferramenta. Entra as principais, pode-se citar:

- Definição de um *menu* de funções de distribuição de probabilidade para o momento da ocorrência das ações no seu intervalo de tempo. Neste *menu* está prevista a possibilidade de escolha e configuração das principais funções de distribuição de probabilidade.
- Permitir que o usuário defina probabilidades (ou distribuições) para a sincronização do ambiente com a especificação. Deste modo, pode-se calcular, por exemplo, a probabilidade da especificação falhar em decorrência de um falta do ambiente.
- Geração de gráficos de ocorrência de ações no tempo. Esta funcionalidade está baseada na apresentada em [CoOl95] para uma outra extensão temporizada de LOTOS homônima à nossa. Tal gráfico permite identificar tempos máximos e mínimos com maior facilidade do que a inspeção de listagens de momentos de ocorrência das ações.
- Inclusão de algumas verificações durante a geração da árvore de execução. Tais verificações podem incluir análise de alcançabilidade, probabilidade de falhas, avaliação de desempenho, entre outras.
- Composição da especificação com processos teste que expressem propriedades na forma de uma especificação RT-LOTOS.
- Extensão da ferramenta para o tratamento de especificações descritas em RT-LOTOS completa.

- Definição de metodologias para a análise de sistemas dependentes de tempo desde a concepção do problema, passando pela simulação e depuração da especificação, até a verificação de propriedades.

Recentemente houve a divulgação de uma *Draft Version* da linguagem E-LOTOS (*Enhancement to LOTOS*) [ISO97] que incorpora elementos necessários para representação de processos dependentes do tempo. E-LOTOS está em processo de padronização por um comitê especial da ISO e já se encontra em *status* de *Draft Proposal* . Assim, todo o trabalho desenvolvido na implementação da ferramenta de simulação apresentada neste trabalho é de interesse para a adaptação das funcionalidades de simulação para o futuro padrão.

Referências Bibliográficas

[AlCo90] Alur, R.; Courcoubetis, C.: *Model Checking for Real-Time Systems.* In Proceedings of the 5th IEEE Symposium on Logic in Computer Science, 1990.

[AlHe92] Alur, R.; Henzinger, T. A.: *Logics and Models for Real Time: A Survey.* Proceedings of the REX Workshop, Mook, The Netherlands, June, 1991, Lecture Notes in Computer Science No.600, Springer-Verlag, 1992.

[Audsley90] Audsley, N.; Bhattacharyya, A.; Burns, A.; Fohler, G.; Kantz, H.; Kopetz, H.; McDermid, J. A.; Shütz, W.; Zainlinger, R.: *Timeliness - Summary and Conclusions.* Vol. 2 Specification and Design for Dependability, in First Year Report of the Predictably Dependable Computing Systems ESPRIT Project, Toulouse, 1990.

[Azcorra90] Azcorra, A. S.: *Formal Modeling of Synchronous Systems.* Ph.D. Thesis, Dpto. de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, Madrid, Espanha, 1990.

[BBBC93] Blair, G.; Blair, L.; Bowman, H.e Chetwynd, A.: *Formal Support for the Specification and Construction of Distributed Multimedia Systems (The Tempo Project) - Final Project Deliverable.* Internal Report MPG-93-23, Lancaster University, 1993.

[BoBr87] Bolognesi, T; Brinksma, E: *"Introduction to the ISO specification language LOTOS.* Computer Networks and ISDN Systems, North-Holland, num. 14, 1987.

[BoDe88] Boullier, P.; Deschamp, P.: *Le Système SYNTAX: Manuel d'Utilisation et de Mise en Oeuvre sous UNIX.* Project Languages et Traducteurs, INRIA, Setembro, 1988.

[BoDe89] Boullier, P.; Deschamp, P.: *SYNTAX: User Commands and C Library Functions.* Project Languages et Traducteurs, INRIA, Junho, 1989.

[Brams83] Brams, G. W.: *Réseaux de Petri: Théorie et Pratique.* Ed. Masson, Tomos 1 et 2, Paris, 1983.

[Camargo95] Camargo, M. S. de: *Tornando a Linguagem LOTOS Apta para Especificar Sistemas Dependentes do Tempo.* Tese de Doutorado LCMI/UFSC, Florianópolis, 1995.

[CaFa95] Camargo, M. S. de; Farines, J.-M.: *Uma abordagem para especificação e verificação de sistemas dependentes de tempo.* IX SBES, 1995.

[CoOl95] Courtiat, J. P.; Oliveira, R. D. de: *On RT-LOTOS and its application to the formal design of multimedia protocols.* In Annals of Telecommunications, vol 50, no. 11-1, p. 888-906, 1995.

[CoRo83] Coolahan, J. E. e Roussopoulus, N.: *Timing Requiriments for Time-driven Systems Using Augmented Petri Nets.* IEEE Transactions on Software Engineering, p. 603-616, 1983.

[DOY94] Daws, C.; Olivero, A.; Yovine, S.: *Verifying ET-LOTOS programs with KRONOS.* In Proceedings of the FORTE'94, Berne, Switzerland, outubro, 1994.

[Eertink92] Eertink, H.: *Executing LOTOS specifications: the SMILE Tool,* Telematics Research Centre, Enschede, Netherlands, 1992.

[Eijk89] Eijk, P. van: *The Design of a Simulator Tool.* In The Formal Description Technique LOTOS, P. H. J. van Eijk, C. A. Vissers and M. Diaz (editors), Elsevier Science Publishers B. V. (North-Holland), 1989.

[Ernb91] Ernb, P.; Fredlund, L.; Hansson, H.; Jonsson, B.; Orava, F.; Pehrson, B.: *Guidelines for Specification and Verification of Communication Protocols.* Swedish Institute of Computer Science (SICS), report no. 1, 1991.

[Fischer85] Fischer, M.: Re: Where are you? E-Mail message to Leslie Lamport. Arpanet message number 850625227.AA07636@YALE-BULLDOG.YALE.ARPA (47 lines), junho, 1985.

[Garavel89] Garavel, H.: *Compilation et Vérification de Programmes LOTOS.* Thèse de Docteur de l'Université Joseph Fourier - Grenoble I, França, Novembro, 1989.

[Garavel94] Garavel, H.: *CAESAR Reference Manual,* Laboratoire de Génie Informatique, Institut I.M.A.G., Grenoble, France, 1989.

[HeLy94] Heitmeyer, C; Lynch, N.: *The Generalized Railroad Crossing: A case study Informal Verification of Real-Time Systems.* Laboratory for Computer Science and Massachusetts Institute of Technology, novembro, 1994.

[HeRe90] Hennessy, M.; Regan, T.: *A Temporal Process Algebra.* FORTE'90, Madrid, Spain, 1990.

[Hoare85] Hoare, C. A. R.: *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[ISO88] Information Processing Systems - Open Systems Interconnection - *LOTOS - A formal description technique based on the temporal ordering of observational behavior.* IS8807, 1988.

[ISO97] Jeffrey, Alan e Leduc, Guy: *E-LOTOS Core Language.* Draft Proposal, janeiro, 1997.

[Kirkwood94] Kirkwood, C. E.: *Verification of LOTOS Specifications using Term Rewriting Techniques.* Submitted for the Degree of Doctor of Philosophy. Department of Computing Science, University of Glasgow, junho 1994.

[Koymans90] Koymans, R.: *Specifying real-time properties with metric temporal logic.* Real-Time Systems, 2(4), p. 255-299, novembro 1990.

[Lamport87] Lamport, L.: *A Fast Mutual Exclusion Algorithm.* ACM Transactions on Computer Systems, Vol. 5, no. 5, fevereiro, 1987.

[LeLé94] Leduc, G.; Léonard, L.: *A Formal Definition of Time in LOTOS.* Research Report, Université de Liège, Institut d'Electricité Montefiori, Bélgica, 1994.

[LLD94] Leduc, G.; Léonard, L.; Danthine, A.: *The Tick-Tock case study for the assessment of timed FDTs.* In the ISO95 transport service with multimidia support on HSLAN's and B-ISDN, 1994.

[LSGL94] Luchangco, V.; Söylemez, E.; Garland, S.; Lynch, N.: *Verifying Timing Properties of Concurrent Algorithms.* In Proc. FORTE'94, Berna, Suíça, outubro, 1994.

[Martins96] Martins, R. F.: *Verificação de Sistemas Dependentes de Tempo a partir de Especificações escritas em RT-LOTOS.* Dissertação de Mestrado LCMI/UFSC, Florianópolis, 1996.

[MeFa76] Merlin, P. e Farber, D. J.: *Recoverability of Communication Protocols - Implications of a Theoretical Study.* IEEE Transactions on Communications, Vol. Com-24, p. 1036-1043, 1976.

[Meyer90] Meyer, B.: *Introduction to the Theory of Programming Languages.* International Series in Computer Science, Prentice Hall, 1990.

[Milner80] Milner, R.: *A Calculus of Communicating Systems.* Lectures and Notes in Comp. Science, No. 92, Springer Verlag, 1980.

[MoTo89] Moller, F.; Tofts, C.: *A Temporal Calculus of Communicating Systems.* Research Report ECS-LFCS-89-104, University of Edimburgh, 1989.

[Ostroff89] Ostroff, J. S: *Temporal Logic for Real-Time Systems.* Research Studies Press LTD, 1989.

[OlYo93] Olivero, A.; Yovine, S.: *KRONOS: A Tool for Verifying Real-Time Systems - User's Guide and Reference Manual.* Montbonnot Soint Martin, França, agosto 1993.

[Regan93] Regan, T.: *Multimidia in Temporal LOTOS.* In IFIP - Protocol Specification, Testing and Verification, XIII (C-16), Dantine, A.; Leduc, G.; Wolper, P. (editors), Elsevier Science Publishers B. V. (North-Holland), p. 127-143, 1993.

[Tretmans89] Tretmans, J.: *HIPPO: A LOTOS Simulator.* In The Formal Description Technique LOTOS, P. H. J. van Eijk, C. A. Vissers and M. Diaz (editors), Elsevier Science Publishers B. V. (North-Holland), 1989.

[Walter83] Walter, B.: *Timed Petri-Nets for Modelling and Analyzing Protocols with Real-Time Characteristics.* The 3rd IFIP Workshop on Protocol Specification, Testing and Verification, North-Holland, p. 149-159, 1983

[Yovine93] Yovine, S.: *Méthodes et Outils pour la Vérification Symbolique de Systèmes Temporisés.* Thèse de Docteur de l'Institut National Polytechnique de Grenoble, Grenoble, França, maio, 1993.

[YPD94] Yi, W.; Petterson, P.; Daniels, M.: *Automatic Verification of Real-Time Communicating Systems by Constraint-Solving.* In Proc. FORTE'94, Berna, Suíça, outubro, 1994.

Anexo A

A Gramática da Linguagem RT-LOTOS

Este anexo apresenta a gramática atribuída à linguagem RT-LOTOS, utilizada no desenvolvimento do simulador RTLS.

```
<spec>          ::=  specification <func> <spec_def>
<func>          ::=  <name> <list>
<spec_def>      ::=  behaviour <body> <processes> endspec
<name>          ::=  string
<list>          ::=  [ <actions> ]
<body>          ::=  <body> <time_op> | <term1>
<processes>     ::=  where <procs> | ε
<actions>       ::=  <name> | <name> , <action> | ε
<time_op>       ::=  < <actions> ] { <rec_proc> }
<term1>         ::=  <hide_op> <term1> | <term2>
<procs>         ::=  <proc> | <proc> <procs>
<proc>          ::=  process <func> <proc_def>
<proc_def>      ::=  := <body> <processes> endproc
<action>        ::=  <name> | <name> , <action>
<rec_proc>      ::=  <name> : <body> |
                     <name> : <body> , <rec_proc>
<hide_op>       ::=  hide <list> in
<term2>         ::=  <term2> <seq_op> <term3> | <term3>
<seq_op>        ::=  >>
<term3>         ::=  <term3> <preemp_op> <term4> | <term4>
<preemp_op>     ::=  [>
<term4>         ::=  <term4> <par_op> <term5> | <term5>
<par_op>        ::=  | <list> |
<term5>         ::=  <term5> <choice_op> <term6> | <term6>
<choice_op>     ::=  []
<term6>         ::=  <act> <prefix_op> <term6> | <factor>
<act>           ::=  <name> | [ integer , integer ] <name> |
                     [ integer ] <name>
<prefix_op>     ::=  ;
<factor>        ::=  <nullary_op> | ( <body> ) | <act> | <proc_inst>
<nullary_op>    ::=  stop | exit
<proc_inst>     ::=  <func>
```

A representação acima diferencia os elementos não-terminais da gramática colocando-os entre os sinais de < e >. Os elementos terminais estão destacados em negrito. O terminal **`string`** denota qualquer seqüência de caracteres formada pelas letras do alfabeto e pelo caracter "_". Já o terminal **`integer`** representa um número pertencente ao conjunto dos números naturais.

Anexo B

O Sistema SINTAX

Neste anexo, serão apresentados os conceitos básicos do Sistema SINTAX [BoDe88, BoDe89]. Esta ferramenta permite a produção de compiladores, sendo implementada sobre a plataforma UNIX.

B.1 Introdução

O Sistema SINTAX agrupa um conjunto de ferramentas que facilitam a concepção e a realização de compiladores. Os objetivos são os mesmos apresentados pelos utilitários LEX e YACC, do sistema UNIX. Porém, o sistema SINTAX possui uma arquitetura mais estruturada, particularmente no tratamento de erros.

A ferramenta SINTAX compreende principalmente os seguintes módulos:

- BNF: construtor da forma interna das definições sintáticas;
- LECL: construtor lexicográfico;
- CSYNT e PRIOR: construtor sintático;
- SEMACT e SEMAT: construtor semântico;
- RECOR: módulo de tratamento de erros;
- TABLES_C: módulo de produção de tabelas de análise em linguagem C;
- ferramentas que realizam e auxiliam a análise do texto fonte.

A seguir é apresentada uma explanação sucinta de cada módulo.

B.2 O Processador da Gramática - BNF

O módulo BNF é o processador de base, que transforma as definições sintáticas para uma forma interna, utilizada pelo sistema. Ele trata de uma gramática independente de contexto, efetuando todas as verificações simples de coerência, produzindo a forma interna desta gramática (tabelas), que será utilizada por outros processadores, tais como CSYNT e LECL.

A gramática de entrada é escrita em uma linguagem próxima à "Forma de Backus-Naur". Os terminais e não-terminais são diferenciados ao nível léxico, sendo que cada regra deve ser única. Um não-terminal deve ser produtivo, ou seja, deve conduzir à uma cadeia terminal (eventualmente vazia), e um não-terminal não deve derivar dele mesmo por produções simples ($N \rightarrow^{+} N$), mas a recursividade é permitida.

BNF aceita gramáticas ambíguas, com a condição de que estas ambigüidades sejam tratadas, adotando-se níveis de prioridade (através de CSYNT e PRIO). É possível também associar predicados e ações, programados pelo autor, que permitem o tratamento de linguagens não-determinísticas.

Como exemplo, a regra <action> da gramática da linguagem RT_LOTOS, apresentada no anexo A, deve ser representada da seguinte forma:

```
<ACTION>    = %IDENT "," <ACTION>
            | %IDENT
            ;
```

Cada regra da gramática é composta de duas partes, separadas pelo sinal "=". O lado esquerdo deverá ser sempre um não-terminal a ser definido, delimitado pelos caracteres "<" e ">". O lado direito é composto de símbolos terminais e não-terminais, sendo que o caracter "|" representa uma opção do não-terminal que está sendo definido. Cada regra deve ser finalizada com um caracter ";", sendo que as regras onde o lado direito é vazio são escritas de forma natural. Assim, a gramática é representada por uma lista de regras, também chamadas de produções, onde o primeiro não-terminal definido é assumido como sendo o axioma da gramática.

Os símbolos terminais são classificados em dois tipos:

- os símbolos terminais propriamente ditos, que são as palavras-chave, símbolos especiais e outros, escritos exatamente como eles devem aparecer no texto a ser analisado, delimitados por espaços em branco ou finais de linha. Sua identificação deve ser feita através de aspas, ou iniciados pelo caracter "#".
- os terminais genéricos, que são os identificadores que representam um conjunto teoricamente infinito de possibilidades de ocorrência, iniciados pelo caracter "%". Estes terminais devem ser obrigatoriamente definidos ao nível léxico.

Comentários podem ser incluídos na definição da gramática. Para isto, basta iniciar cada linha de comentário com o caracter "*".

Também é possível associar a semântica a cada produção da gramática, mas o módulo BNF tratará somente da interpretação das regras sintáticas, sendo o tratamento da semântica feito através de módulos específicos (SEMAT, SEMACT, TABC, YAX, etc.).

BNF gera dois arquivos de resultados: o arquivo *name*.bt, que contém a forma interna da gramática (tabelas); e o arquivo *name*.bn.l, que contém uma listagem de toda a análise feita pelo módulo, informando a ocorrência de possíveis erros.

B.3 O Construtor do Analisador Léxico - LECL

O módulo LECL é o construtor léxico do sistema SINTAX, onde a descrição das unidades léxicas da linguagem são feitas sob a forma de expressões regulares. LECL utiliza também as informações contidas nas tabelas produzidas pelo módulo BNF (*name*.bt), sendo o resultado um autômato de estados finitos capaz de efetuar o reconhecimento dos terminais do nível sintático.

O texto que descreve a estrutura do analisador léxico deve ser armazenado no arquivo *name*.lecl, contendo as expressões regulares da linguagem. No entanto, existem alguns passos preliminares à declaração destas expressões, que podem ser seguidos opcionalmente, com o objetivo de simplificar a montagem da estrutura.

A primeira simplificação possível é a definição de classes, que representam um conjunto de caracteres a ser identificado em um dado momento. Esta definição é iniciada pela palavra-chave *Classes*. Existem algumas classes pré-definidas pelo sistema SINTAX, como por exemplo a classe *LETTER*, que representa todas as letras do alfabeto, maiúsculas e minúsculas. Alguns exemplos de classes são:

```
Classes

    DIGIT       = "0".."9"
    LOWER       = "a".."z"
    keyword_one       = ONE
```

A classe `DIGIT` define os dígitos de 0 a 9. A classe `LOWER` representa as letras do alfabeto, porém somente minúsculas. A classe `keyword_one` define a palavra reservada "one", em qualquer combinação de letras maiúsculas ou minúsculas.

Outra forma de simplificar a estrutura do texto é utilizando abreviações, que são iniciadas com a palavra-chave *Abbreviations*. Como exemplo, um número natural pode ser definido como:

```
Abbreviations

    NATURAL = DIGIT { DIGIT };
```

Declarado desta forma, um número natural é um dígito seguido de qualquer quantidade de dígitos.

A declaração das expressões regulares é iniciada pela palavra-chave *Tokens*. É bom lembrar que todos os terminais genéricos devem ser definidos. Para o terminal da seção anterior, a seguinte definição é válida:

```
Tokens
    %IDENT      = LETTER {["_"] (LETTER | DIGIT)};
```

Em alguns casos, pode haver conflito entre duas ou mais expressões regulares. O exemplo a seguir mostra tal conflito:

```
Tokens
    %IDENT      = LETTER {["_"] (LETTER | DIGIT)};
    BEHAVIOUR   = B E H A V I O U R;
```

Neste caso, a palavra reservada *behaviour* pode ser reconhecida tanto pela primeira expressão regular como pela segunda. Este conflito é chamado de "conflito de reduções entre expressões". Para resolver tais situações, é possível definir qual expressão possui maior prioridade, bastando incluir a esta expressão a declaração de prioridade.

Outro problema seria o reconhecimento de apenas uma letra do identificador. Neste caso, o conflito é chamado de "conflito de redução e avanço entre expressões". Este conflito pode ser resolvido utilizando restrições de contexto, que estabelecem o que pode suceder uma expressão.

O texto de definições pode conter comentários, identificados através dos caracteres "—", que marcam o início do mesmo. Linhas que começam por estes caracteres são consideradas comentários até seu final.

LECL gera dois arquivos de resultados: o arquivo *name*.st, que contém as tabelas com todos os dados utilizados na análise léxica; e o arquivo *name*.lecl.l, que contém uma listagem completa da análise feita sobre as definições, relatando os possíveis erros.

B.4 Os Construtores do Analisador Sintático - CSYNT e PRIO

O módulo CSYNT é responsável pela geração do analisador sintático. Ele utiliza as tabelas geradas pelo módulo BNF, contendo a forma interna da gramática, e gera um analisador baseado em autômatos de pilha.

A classe de gramática aceita pelo sistema é a LARL(1). Os possíveis conflitos entre as regras da gramática podem ser tratados através do módulo PRIO, porém é aconselhável procurar contornar tais conflitos na gramática da linguagem, para tornar o analisador mais simples.

Os conflitos possíveis são:

- conflito Reduce / Reduce: indica que existem várias possibilidades de reconhecimento simultâneo dos lados da direita das regras envolvidas (reduce), e
- conflito Shift / Reduce: indica que é possível realizar um avanço para o símbolo seguinte (shift) ou uma redução da regra (reduce).

O módulo PRIO também utiliza as tabelas geradas pelo módulo BNF, sendo que a descrição da gramática deverá conter a prioridade dos operadores e das regras conflitantes. A prioridade dos operadores é definida através das palavras-chave *%left* e *%right*, que indicam, respectivamente, que um dado operador possui associatividade pela esquerda ou pela direita, indicando também a prioridade entre os mesmos. A prioridade das regras é definida após a descrição de cada regra, onde a palavra-chave *%prec* indica qual a sua associatividade e sua prioridade, relacionando-a com um operador.

PRIO gera como resultado o arquivo *name*.dt, contendo as tabelas internas de tratamento de ambigüidades, utilizadas pelo módulo CSYNT.

CSYNT gera como resultados: o arquivo *name*.pt, que contém as tabelas para a análise sintática; e o arquivo *name*.la.l, que contém uma listagem completa da análise feita sobre as definições, relatando possíveis erros.

B.5 Os Construtores do Analisador Semântico - SEMACT e SEMAT

Os módulos SEMACT e SEMAT representam duas formas de tratamento da semântica da linguagem. Eles tratam também da sintaxe da linguagem, através das funcionalidades herdades do módulo BNF.

No módulo SEMACT, o tratamento semântico é efetuado através de ações semânticas associadas à análise sintática, ou seja, ações são associadas às regras da gramática, sendo executadas após o reconhecimento do lado direito das mesmas.

No módulo SEMAT, o tratamento semântico é efetuado através da utilização de uma árvore abstrata, construída na análise sintática, onde é aplicado sobre esta árvore um programa de avaliação dos atributos semânticos.

Neste trabalho, foi utilizado o módulo SEMAT. Portanto, uma descrição mais detalhada deste módulo será dada.

Para se obter a construção da árvore abstrata, basta modificar as definições das regras da gramática, colocando no final das regras um identificador do nó, delimitado por aspas.

Como exemplo, sejam as seguintes regras:

```
<OBJECT_LIST>    = <OBJECT_LIST> , <OBJECT> ;
<OBJECT_LIST>    = <OBJECT>                 ; "OBJ_S"
<OBJECT>         = ...                      ; "OBJ"
```

Seja ainda um texto contendo uma lista de três objetos. Então, a árvore abstrata gerada seria da forma mostrada na figura D.1

Assim, a árvore abstrata não possui nenhum nó contendo informações desnecessárias à análise semântica, tais como: palavras-chave, delimitadores ou separadores (como ",").

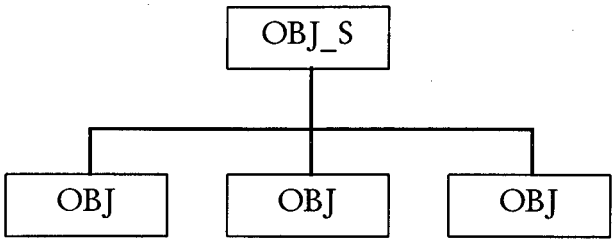


Figura D.1 - Exemplo de uma árvore abstrata

Os atributos semânticos são associados aos nós da árvore abstrata. Um passo semântico consiste de um caminho nesta árvore abstrata, no sentido de cima para baixo, da esquerda para a direita, aproveitando-se das passagens sobre os nós para avaliar os atributos, onde vários passos podem ser realizados desta maneira.

Cada nó é visitado duas vezes:

- na entrada de sua sub-árvore: no momento desta passagem, os atributos "herdados" do nó são avaliados, sendo que esta visita é chamada "hereditária"; e
- na saída da sub-árvore: desta vez, os atributos "sintetizados" são avaliados, sendo que esta visita é chamada de "síntese".

A partir de um nó, é possível acessar os nós vizinhos (se eles existirem). Ou seja, os pais, os irmãos e os filhos deste nó. É possível, portanto, calcular um atributo do nó em função dos atributos já calculados dos nós vizinhos.

As ferramentas necessárias à realização da análise semântica trabalham com a estrutura dos nós e com ponteiros, que permitem percorrer toda a árvore abstrata (sxatc e sxat_mngr). Todos os nós da árvore possuem a mesma estrutura.

Portanto, a partir das regras da gramática, o módulo SEMAT gera um arquivo *name*.att, contendo as tabelas internas da árvore abstrata, gerando também o arquivo *name*.bt, através do módulo BNF, e o arquivo *name*.bn.l, que contém uma listagem da análise feita pelo módulo, relatando possíveis erros.

B.6 O Tratamento de Erros - RECOR

O tratamento de erros efetuado pelo sistema SYNTAX é dividido em duas partes:

- a tentativa de correção local do programa fonte, caso exista algum erro; e
- se a tentativa anterior falhar, procura-se recuperar os erros detectados, de maneira global.

O tratamento de erros é feito somente nos níveis léxico e sintático. O reconhecimento dos erros de nível semântico é feito separadamente e sem nenhuma possibilidade de correção. A correção se baseia nos elementos léxicos da linguagem e nas regras gramaticais.

B.7 A Produção de Tabelas em Linguagem C - TABLES_C

O módulo TABLES_C processa as formas internas de análise léxica, análise sintática, análise semântica e análise de erros, gerando um programa escrito em linguagem C, contendo estruturas de dados com todas as informações necessárias ao sistema de execução para o tratamento da linguagem em questão.

Portanto, o sistema SYNTAX não produz diretamente um programa executável, mas um conjunto de dados escritos na linguagem C, que serão compilados e ligados com os módulos do sistema de execução.

B.8 Realização do Compilador

A realização do compilador é obtida através da utilização de um compilador C, em particular, do compilador GCC da GNU. Assim, o programa de avaliação de atributos semânticos é compilado, juntamente com os módulos do sistema de execução (sxparser, sxscanner, sxatc, etc), necessários para as análises léxica, sintática e semântica, incluindo também as estruturas de dados produzidas pelo módulo TABLES_C e uma biblioteca do sistema (lxba.a).