UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Verificação de Sistemas Dependentes do Tempo a partir de Especificações escritas em RT-LOTOS

Dissertação submetida à Universidade Federal de Santa Catarina
para obtenção do grau de
Mestre em Engenharia Elétrica

Ricardo Ferreira Martins

Florianópolis, 24 de Junho de 1996.

Verificação de Sistemas Depententes do Tempo a partir de Especificações escritas em RT-LOTOS

Ricardo Ferreira Martins

Esta dissertação foi julgada para obtenção do título de

Mestre em Engenharia

especialidade **Engenharia Elétrica**,

área de concentração **Controle, Automação e Informática Industrial**,

e aprovada em sua forma final pelo Curso de Pós-Graduação.

Prof. Jean-Marie Farines, Dr. Ing.
Orientador

Prof. Murilo Silva de Camargo, Dr.
Co-Orientador

Prof. Enio Valmor Kassick, Dr.
Coordenador

Banca Examinadora:

Prof. Daltro José Nunes, Dr.

Prof. Rosvelter Coelho da Costa, Dr.

Prof. Vitório Bruno Mazzola, Dr.

Aos meus pais Elisier e Lourdes,
aos meus irmãos Helton e Rogério,
e à Márcia.

O mundo é produto da vontade.
Então, o homem será, antes de mais nada,
o resultado de seu próprio projeto.

(Sartre)

AGRADECIMENTOS

Primeiramente, quero agradecer aos meus pais pelo amor, educação e apoio que venho recebendo durante toda a minha vida; aos meus irmãos pelo apoio e incentivo; e à Márcia, pelo carinho e compreensão, mesmo nos momentos difíceis.

Ao Prof. Hélio Leães Hey, por despertar em mim a vontade de pesquisar e procurar cada vez mais aprimorar o meu conhecimento.

Aos Professores Jean-Marie Farines e Murilo Silva de Camargo, pelo trabalho de orientação e apoio em todas as etapas deste trabalho.

Aos membros da banca examinadora, pela avaliação, críticas e comentários apontados durante a apresentação deste trabalho.

Ao Prof. Sérgio Yovine, pela contribuição dada através de sua tese de Doutorado e também pela ferramenta KRONOS.

Ao Laboratório INRIA, pela cooperação e fornecimento da ferramenta SYNTAX.

Aos meus amigos e companheiros desta jornada, engenheiros André Bittencourt Leal, Danielle Nishida, Emerson Raposo, Max Mauro Dias Santos e também os demais colegas de mestrado, que não foram citados apenas por motivos de espaço.

Aos colegas do Laboratório de Controle e Microinformática (LCMI) da Universidade Federal de Santa Catarina (UFSC), que ofereceram todo o suporte para a realização deste trabalho.

À UFSC e à CAPES, pelo suporte material e financeiro.

À Deus.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A álgebra de processos LOTOS é uma linguagem formal para especificação de sistemas, desenvolvida entre 1980 e 1988, e padronizada pela ISO desde 1988. Esta linguagem permite descrever a ordenação temporal de um conjunto de ações, percebidas por um observador externo, possibilitando a especificação de comportamentos complexos, combinando ações elementares com os operadores da linguagem. No entanto, LOTOS possui uma limitação, pois não permite especificar de forma explícita os sistemas onde o tempo intervém.

Neste trabalho, são apresentadas uma metodologia e ferramentas para a verificação de especificações de sistemas tempo real, escritas em RT-LOTOS, uma extensão temporizada da linguagem de especificação LOTOS. A partir do modelo em RT-LOTOS, é gerado um Grafo Temporizado correspondente, sobre o qual poderá ser realizada a verificação, usando fórmulas da Lógica Temporal Tempo-Real TCTL. O tradutor de RT-LOTOS para Grafos Temporizados, que foi construído para este efeito, é descrito neste trabalho; sua integração com ferramentas existentes que permitem a verificação é também mostrada. Finalmente, estudos de casos permitem analisar o uso desta abordagem e das ferramentas apresentadas.

ABSTRACT

The LOTOS process algebra is a formal language for systems specifications, developed between 1980 and 1988, and standardized by ISO since 1988. This language allows to describe the temporal ordering of a set of actions, perceived by an external observator, and let complex behaviours specification, combining elementary actions with the LOTOS operators. Nevertheless, LOTOS has a limitation, so it doesn't allow to specify in an explicit way systems with time dependencies.

In this work, we present a methodology and tools for verifying specifications of real-time systems, written in RT-LOTOS, a timed extension of the LOTOS specification language. From the model in RT-LOTOS, it is generated the related Timed Graph, over which the verification can be done, by using formulas written in the TCTL Real-Time Temporal Logic. The translator of RT-LOTOS specifications to Timed Graphs that was built is described in this work; his integration with existing tools that allow the verification is also shown. Finally, case-studies allow to analyze the use of this method and the tools presented.

CAPÍTULO 1

INTRODUÇÃO

As Técnicas de Descrição Formal (*FDT - Formal Description Techniques*) são cada vez mais aceitas e utilizadas para o desenvolvimento de sistemas complexos. Para Sistemas Distribuídos e Redes de Comunicação, em particular, as dificuldades provenientes do alto grau de paralelismo destas aplicações não encontram outra abordagem adequada para seu tratamento a não ser o uso de técnicas formais e das metodologias e ferramentas que lhes são associadas. [MCF96]

As linguagens de especificação padronizadas pela ISO (Estelle e LOTOS) e pelo ITU (SDL), nesta ultima década, surgiram destas necessidades e vêm sendo aplicadas, tanto no campo acadêmico como industrial, com relativo sucesso. Ferramentas para análise e projeto de sistemas a partir de especificações construídas numa dessas linguagens já são produtos comerciais, e problemas cada vez mais complexos estão sendo tratados.

Entretanto, algumas das limitações dessas técnicas padronizadas se tornaram também mais visíveis na medida do crescimento da complexidade e da diversidade das aplicações a serem desenvolvidas. Em particular, mostraram-se insuficientes para expressar as restrições temporais exigidas em diversas aplicações tais como: sistemas multimídias, sistemas de automação industrial e sistemas tempo-real embutidos (automóveis, aeronaves, etc.) entre outros.

A linguagem de especificação LOTOS, que se baseia nas álgebras de processos CCS e CSP, para a descrição comportamental, e na linguagem de tipos abstratos de dados ACT-One, tem despertado bastante interesse pelo seu poder de expressão que permite tratar a especificação em diferentes níveis de abstração e pelo seu suporte teórico que permite lhe associar técnicas e ferramentas de verificação baseadas em equivalências. Entretanto, sua expressividade apresenta limitações por não ter nenhuma característica que permite a representação explícita do tempo. Nesses últimos anos, varias extensões temporais desta linguagem foram propostas, além de existir no momento um esforço de um comitê de padronização na direção de um LOTOS Temporizado, incluindo contribuições oriundas destas propostas. Na espera deste novo padrão, será adotada neste trabalho a extensão chamada de RT-LOTOS, sendo que as propostas e desenvolvimentos de metodologias e ferramentas serão futuramente adaptadas a este padrão.

Neste trabalho, apresenta-se uma abordagem para verificação de especificações de sistemas tempo-real, escritas na linguagem RT-LOTOS, apresentando também a ferramenta desenvolvida

para este efeito e a utilização desta em alguns casos de estudo.

Esta dissertação está organizada da seguinte maneira:

No capítulo 2, é discutido o contexto global do trabalho desenvolvido, onde apresenta-se as características dos sistemas dependentes do tempo, os métodos formais de especificação destes sistemas, destacando a álgebra de processos RT-LOTOS, o método de verificação utilizado e a utilização do tradutor desenvolvido neste trabalho.

O capítulo 3 apresenta a forma de se representar uma especificação escrita em RT-LOTOS através de Grafos Temporizados, utilizado como formalismo para a modelagem do sistema a ser verificado, sendo que cada operador da linguagem possui sua representação neste formalismo. São apresentados alguns exemplos de pouca complexidade, visando apenas o melhor entendimento do método apresentado.

O capítulo 4 apresenta os princípios básicos para construir o tradutor RT-LOTOS / Grafos Temporizados, sendo que na etapa preliminar à geração do Grafo Temporizado, define-se uma extensão de Redes de Petri, com o intuito de simplificar o modelo manipulado. Como no capítulo 3, cada operador da linguagem possui sua representação neste formalismo intermediário. Os detalhes da implementação completa do tradutor também são discutidos.

O capítulo 5 apresenta algumas aplicações, visando testar e validar tanto o tradutor implementado como o método de verificação.

O capítulo 6 encerra esta dissertação, comentando os principais resultados obtidos e as perspectivas de trabalhos futuros.

CAPÍTULO 2

ESPECIFICAÇÃO E VERIFICAÇÃO DE SISTEMAS DEPENDENTES DO TEMPO

O projeto de sistemas dependentes do tempo é uma tarefa bastante complexa. Como em outros sistemas complexos, a utilização de métodos formais para a especificação e verificação é a solução a ser adotada, pois ela permite detectar erros de concepção, além de outras vantagens.

Neste capítulo, são apresentadas definições básicas sobre sistemas dependentes do tempo, destacando as grandes classes de sistemas informáticos. Em seguida, são apresentados os principais métodos formais para a especificação destes sistemas, destacando a álgebra de processos RT-LOTOS como sendo a abordagem utilizada neste trabalho. Ao final, descreve-se a maneira na qual a verificação é efetuada neste trabalho, baseada na representação do sistema através de Grafos Temporizados, e na representação das propriedades através de fórmulas escritas na lógica temporal TCTL.

2.1 SISTEMAS DEPENDENTES DO TEMPO

Sistemas dependentes do tempo [Camargo95] são aqueles onde o seu comportamento sofre influências do progresso do tempo num intervalo a ser considerado, ou seja, o tempo pode ser considerado como uma variável a ser manipulada pelo sistema ou um recurso a ser consumido pelo sistema. Os sistemas dependentes do tempo incluem três grandes classes de sistemas computacionais: sistemas tempo-real, protocolos de comunicação e aplicações multimídia.

Os sistemas tempo-real são sistemas que reagem a estímulos do ambiente, incluindo a passagem do tempo físico, dentro dos intervalos de tempo impostos pelo comportamento do seu ambiente. Existem requisitos gerais exigidos para estes sistemas, que permitem diferenciá-los e classificá-los, tais como: correção temporal ("*Timeliness*"), onde os sistemas reagem a estímulos do ambiente, dentro de intervalos de tempo especificados; relação de ordem ("*Ordeliness*"), que mantém as relações de ordem entre as entradas do sistema nas suas correspondentes saídas; atualidade ("*Freshness*"), estabelecendo que o sistema deve usar, ou fornecer, o dado que seja o mais válido possível, do ponto de vista do comportamento temporal; previsibilidade ("*Predictability*"), estabelecendo que o determinismo do comportamento funcional e temporal do sistema deve satisfazer as especificações do sistema; e coordenação ("*Coordination*"), onde

as ações do sistema no seu ambiente devem ser coordenadas de modo que uma não prejudique a outra.

Os protocolos de comunicação são normas e regras estabelecidas para a padronização da comunicação entre dispositivos, onde algumas características gerais são observadas, tais como: a concorrência, onde os nodos de comunicação executam estas regras de forma autônoma e paralela; a interação, denotando todas as formas de comunicação entre os nodos (comunicação síncrona, assíncrona, por difusão, por memória compartilhada, etc.); e a imprevisibilidade, onde mensagens podem ser perdidas e tempos de transferência de mensagem não podem ser garantidos.

As aplicações multimídia são sistemas que geram, processam, armazenam, restituem e apresentam informações de vários tipos e origens, chamadas de mídia, combinando assim várias fontes de informação, tais como: voz, gráficos, áudio de alta fidelidade e vídeo. As aplicações se estendem em áreas como automação de escritórios, sistemas de tele-conferência e aprendizado à distância, ambientes de cooperação científica e controle de tráfego aéreo.

2.2 MÉTODOS FORMAIS PARA ESPECIFICAÇÃO DE SISTEMAS DEPENDENTES DO TEMPO

As principais abordagens formais para o tratamento de sistemas dependentes do tempo são:

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

A teoria de Redes de Petri [Brams83] é um formalismo gráfico e matemático que se adapta bem a um grande número de aplicações, onde as noções de eventos e de evoluções simultâneas são importantes. Este formalismo foi um dos primeiros a serem apresentados na literatura, para tratar com concorrência, não-determinismo e conexões causais entre eventos. Entre as características das Redes de Petri, destacam-se a legibilidade dos modelos gerados e a capacidade de expressar as diversas relações de causalidades (paralelismo, seqüenciamento, não-determinismo e sincronização). No entanto, as Redes de Petri não estão aptas a tratar com a justiça (*fairness*) e com estruturas de dados, apesar da possibilidade de associar uma variável local de programa com o número de fichas contidas em um determinado lugar da rede.

Lógica temporal é uma classe das lógicas modais, em que os operadores modais são interpretados de uma maneira temporal. Lógicas temporais são compostas de operadores lógicos proposicionais que, combinados a um conjunto de operadores temporais, permitem que certas

propriedades sejam satisfeitas, tais como: a invariância, estabelecendo que certas condições serão sempre verdadeiras no sistema considerado; a eventualidade, onde existem propriedades que devem tornar-se verdadeiras em algum tempo futuro no sistema; e as precedências, que são asserções que estabelecem que um evento deve ocorrer antes de um outro. Porém, as lógicas temporais possuem como pontos negativos a complexidade e a falta de estruturas nas aplicações. As lógicas temporais tempo-real estendem as lógicas temporais clássicas, incorporando possibilidades de representação e tratamento de propriedades quantitativas, tais como: periodicidade, esgotamento do tempo (*deadlines*) e atrasos (*delays*). Algumas das lógicas temporais tempo-real mais conhecidas na literatura são: *Metric Temporal Logic* (MTL) [Koymans90]; *Real Time Temporal Logic* (RTTL) [Ostroff89]; e *Temporal Computation Tree Logic* (TCTL) [AH92].

A abordagem de álgebra de processos tem sido desenvolvida dentro do campo da teoria da concorrência, e teve seu início a partir das álgebras *Communicating Sequential Processes* (CSP) [Hoare85] e *Calculus of Communicating Systems* (CCS) [Milner80]. Os métodos baseados em álgebras de processos são significantes devido à forma elegante pela qual com poucos construtores pode-se obter uma linguagem capaz de expressar toda a complexidade de um sistema concorrente ou de um sistema distribuído.

A linguagem LOTOS [ISO88] é um padrão ISO desde 1988, baseada na álgebra de processos CCS, e enriquecida com o princípio de comunicação síncrona de *rendezvous* múltiplo do CSP e a representação de dados algébricos Act One. Entretanto, sua expressividade apresenta limitações quanto a modelagem de comportamentos tempo-real, pois impossibilita a especificação quantitativa do tempo preciso na qual um evento deve ocorrer e a ocorrência de maneira forçada de certos eventos. Nestes últimos anos, várias extensões temporais da álgebra de processos LOTOS foram propostas, onde as mais conhecidas são: RT-LOTOS [Camargo95], RT-LOTOS [CO94], Temporal LOTOS [Regan93], TIC-LOTOS [Azcorra90], T-LOTOS [BLT93] e ET-LOTOS [LL94].

A abordagem adotada neste trabalho para especificar sistemas dependentes do tempo é a álgebra de processos RT-LOTOS [Camargo95]. No Anexo A, é apresentada uma introdução rápida à linguagem LOTOS, mostrando principalmente a sintaxe da linguagem.

2.3 RT-LOTOS

2.3.1 Por quê escolher a extensão temporal RT-LOTOS?

Por ser uma extensão de LOTOS, RT-LOTOS possui todas as características desta linguagem,

citadas anteriormente. RT-LOTOS é uma linguagem que possibilita a associação de intervalos temporais às ações do sistema, sendo que a não ocorrência destas ações dentro de seus intervalos caracteriza uma violação temporal, cujo tratamento é efetuado a partir da definição de um novo operador, chamado de operador de "*Preempção Temporal*". Estes intervalos, da forma $[t_{min},t_{max}]$, determinam em que instante as diferentes ações podem ser oferecidas ao seu ambiente.

Conforme apresentado em [CF95], RT-LOTOS também permite estabelecer uma semântica baseada em Grafos Temporizados, possibilitando a utilização de técnicas de verificação para sistemas dependentes do tempo.

2.3.2 Sintaxe da linguagem

Os termos de RT-LOTOS são gerados pela sintaxe apresentada a seguir, que é uma extensão direta da sintaxe de LOTOS.

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

- O operador de inação representa um processo completamente inativo, onde nenhuma ação pode ser oferecida ao ambiente, e somente o tempo pode progredir.

- O operador de terminação com sucesso representa um processo onde a ação δ representa o fim bem sucedido do mesmo.

- O processo $[t_{min},t_{max}]a; E$ representa o oferecimento da ação a, dentro do intervalo especificado $[t_{min},t_{max}]$, sendo que se a ação a não pôde se realizar durante este intervalo, então ela não poderá o ser fora do mesmo, onde a ação especial a^* notifica uma eventual violação temporal. No entanto, uma prefixação da ação especial i não possui violação temporal, dado que esta ação deve, necessariamente, realizar-se no intervalo $[t_{min},t_{max}]$. Assim, se a ação a ocorrer dentro do intervalo, então $[t_{min},t_{max}]a; E$ passa a se comportar como E.

- O processo $E[]F$ representa a escolha entre E e F, decidida pela realização de uma primeira ação $a \in Act \cup \{i, \delta\}$, onde Act representa o conjunto das ações observáveis. Assim, a realização de uma violação temporal não decide uma escolha.

- O operador de ocultação transforma as ações observáveis, pertencentes ao conjunto L, em ações invisíveis e urgentes.

- O processo $E|[L]|F$ representa a composição paralela dos processos, E e F, onde cada um dos processos pode realizar, de modo independente, as ações $a \notin L$. A realização das ações $a \in L$ deve ser feita de maneira síncrona, ou seja, se ambos os processos oferecerem uma ação a ser sincronizada, então esta ação é realizada imediatamente. No caso de um dos processos não oferecer a ação sincronizável até a expiração do intervalo de tempo da ação oferecida, então ocorrerá uma violação temporal associada à ação cujo intervalo temporal expirou, dando seqüência a composição paralela do processo que segue a violação temporal com o processo que não ofereceu a ação sincronizável.

- A composição seqüencial de dois processos, E seguido de F, representa o início do processo F, após a terminação com sucesso do processo E.

- O processo E [> F representa a execução de E, podendo ser interrompido a qualquer instante pela realização de uma ação do processo F, onde somente a terminação com sucesso do processo E inibe a execução do processo F.

- O processo $E <L]\{a_1: Q_1, ..., a_n: Q_n\}$ representa o tratamento das violações temporais (a_1*,..., a_n*), ou seja, se ocorrer uma violação temporal, a_i*, durante a execução do processo E, então este processo é abandonado e o processo Q_i é iniciado no seu lugar.

- O processo $P[a_1, ..., a_n]$ representa uma instanciação do processo P, onde ocorre a troca dos nomes das ações.

2.3.3 Semântica operacional da linguagem

A semântica operacional é apresentada no Anexo B, no estilo SOS (*Structured Operational Semantics*) de Plotkin e inclui:

- um conjunto de regras de inferência para as ações clássicas de RT-LOTOS;
- um conjunto de regras de inferência para as violações temporais; e
- um conjunto de regras de inferência para a progressão do tempo.

Além disso, e em conformidade com a semântica padrão de LOTOS, todas as ocorrências de ações são consideradas atômicas e instantâneas.

A ação δ e as violações temporais a^*, sendo devidas a ocorrências de operações especificadas na linguagem mas não sendo diretamente manipuláveis nesta, não podem ser temporizadas. A temporização de ações é feita sobre um domínio de tempo D^w, que pode ser esparso ou denso, mas que deve ser enumerável.

A gramática da linguagem RT-LOTOS, utilizada no desenvolvimento do tradutor, é apresentada no Anexo B. A precedência dos operadores (binários e unitários) segue a forma apresentada em [BB87], acrescentando apenas o novo operador de tratamento de violações temporais, sendo assumido que este é o operador de menor precedência, ou seja:

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

2.4 VERIFICAÇÃO DE SISTEMAS DEPENDENTES DO TEMPO

2.4.1 O método

O método utilizado neste trabalho consiste na verificação de modelos (*Model Checking*) [ACD90], a partir do qual procura-se verificar se o modelo do comportamento do sistema satisfaz um conjunto de propriedades estabelecidas. Esta abordagem, dita dual, representa geralmente o comportamento do sistema a partir de um formalismo de descrição de seus estados, e as propriedades a partir de fórmulas de uma lógica modal, onde o conjunto de estados do sistema que satisfazem uma fórmula é definido como *conjunto característico de uma fórmula*.

Existem duas abordagens possíveis para a verificação: a abordagem numérica, que consiste em construir um modelo do grafo temporizado (Sistema de Transições Rotulado) e avaliar a fórmula, a partir de uma enumeração exaustiva de todos os estados do modelo; e a abordagem simbólica, que consiste em representar o conjunto característico através de predicados e avaliar a fórmula diretamente sobre o grafo temporizado, sem a necessidade de construir um modelo.

A abordagem numérica tem como vantagem a simplificação do problema de comparação e avaliação do conjunto de estados, porém, o seu uso é permitido somente para modelos de comportamento finito, além de permitir uma explosão combinatória durante a construção do modelo, de acordo com o número de composições paralelas do sistema.

Por outro lado, a abordagem simbólica evita a possibilidade de uma explosão combinatória, já

que não necessita construir o modelo do grafo temporizado, além de poder ser aplicada em sistemas onde o modelo possui um comportamento infinito. No entanto, é necessário um procedimento de decisão eficaz, que permita comparar os predicados.

Em [CF95], constatou-se que apesar de ser comum a utilização de um Sistema de Transições Rotulado para a representação de uma especificação escrita em álgebras de processos, no caso da linguagem RT-LOTOS, o uso deste formalismo não é adequado para a verificação de propriedades, pois implica na adição de um grande número de transições correspondentes à progressão do tempo e, conseqüentemente, num crescimento do número de estados do modelo do sistema. O uso de Grafos Temporizados para definir a semântica de RT-LOTOS diminui consideravelmente o espaço de estados a ser analisado, tornando viável o processo de verificação.

A verificação de modelos utilizada neste trabalho é baseada na abordagem simbólica, onde a ferramenta KRONOS [OY93], desenvolvida numa cooperação Indústria-Universidade pelo laboratório VERIMAG (França), implementa um algoritmo de verificação simbólica de modelos [HNSY92], sendo a lógica TCTL utilizada para representar as propriedades do sistema. O Anexo C apresenta os princípios básicos para a utilização desta ferramenta.

2.4.2 A lógica TCTL

As propriedades a serem verificadas sobre o modelo de um sistema podem ser descritas por fórmulas escritas através da lógica temporal tempo-real TCTL, construídas a partir de proposições e restrições atribuídas ao sistema.

A lógica TCTL (*Temporal Computation Tree Logic*) [AH92] é uma extensão da lógica CTL [CES86], estendendo os operadores temporais $\exists u$ (existe uma execução) e $\forall u$ (em todas execuções) de tal forma que sejam utilizadas restrições temporais, permitindo um tratamento temporal quantitativo.

As fórmulas TCTL seguem a seguinte gramática:

```
<fórmula> := <predicado> | <fórmula> ∧ <fórmula> | <fórmula> ∨ <fórmula> |
             ¬<fórmula> | <fórmula> ∃u_I <fórmula> | <fórmula> ∀u_I <fórmula>
<predicado> := init | enable(a) | after(a) | x ∈ I
```

sendo `init` o estado inicial do modelo do sistema, `x` um relógio pertencente ao conjunto de relógios do modelo do sistema, *enable*(a) o conjunto de estados do modelo do sistema onde a

ação a é possível de ocorrer, e *after*(a) o conjunto de estados do modelo do sistema alcançados após a ocorrência de uma ação a. O intervalo de valores positivos inteiros é chamado I.

Os operadores $\vee$, $\wedge$ e $\neg$ correspondem, respectivamente, aos operadores binários "*or*", "*and*" e ao operador unitário "*not*".

A fórmula <fórmula1> $\exists u_I$ <fórmula2> significa que existe uma execução do sistema com uma prefixação finita onde <fórmula2> é verdadeira no último estado, com um tempo $t \in I$, sendo que <fórmula1> é continuamente verdadeira nos estados anteriores.

A fórmula <fórmula1> $\forall u_I$ <fórmula2> se diferencia pelo fato de que todas as execuções do sistema são verdadeiras.

Através destas duas fórmulas, é possível obter algumas abreviações típicas, tais como:

true $\exists u_I$ <fórmula> denotada por $\exists \Diamond_I$ <fórmula>
true $\forall u_I$ <fórmula> denotada por $\forall \Diamond_I$ <fórmula>
$\neg \forall \Diamond_I \neg$<fórmula> denotada por $\exists_I$ <fórmula>
$\neg \exists \Diamond_I \neg$<fórmula> denotada por $\forall_I$ <fórmula>

Para tornar mais clara a maneira como a ferramenta KRONOS calcula, de forma iterativa, o conjunto característico de uma fórmula, o seguinte exemplo é apresentado: para verificar se é possível alcançar um estado que satisfaça uma fórmula, μ, a partir do conjunto de estados do modelo, a fórmula $\exists \Diamond_I \mu$ descreve esta propriedade. Assim, o conjunto característico, denotado por $\{\exists \Diamond_I \mu\}$, é:

$$\{\exists \Diamond_I \mu\} = \cup i K i_{[z:=0]}$$

onde:

$K0 = \{\mu \wedge z \in I\}$

$Ki+1 = Ki \cup (true \blacktriangleright Ki)$

O operador binário $\blacktriangleright$ retorna o conjunto de estados que satisfazem a fórmula, a partir dos *estados seguintes*, obtidos através da iteração anterior. A variável z representa um relógio, utilizado para marcar o tempo desde o início da execução. Assim, o algoritmo inicia com o conjunto de estados que satisfaz a fórmula $\{\mu \wedge z \in I\}$ e, em seguida, são calculados os demais estados da próxima iteração, sendo que o relógio z é inicializado a cada iteração.

2.4.3 Arquitetura do Processo de Verificação

O primeiro passo para a verificação de um sistema está na geração do Grafo Temporizado. A tradução das especificações RT-LOTOS para Grafos Temporizados é obtida através do tradutor implementado neste trabalho, sendo a verificação do grafo temporizado resultante feita a partir da lógica TCTL e através da ferramenta KRONOS. KRONOS utiliza como elementos de entrada: o grafo temporizado, que descreve o sistema; e fórmulas TCTL, que estabelecem as propriedades a serem verificadas. Finalmente, a ferramenta KRONOS retorna como resultado as condições em que o grafo temporizado satisfaz a fórmula. A Figura 2.1 ilustra o processo de verificação.

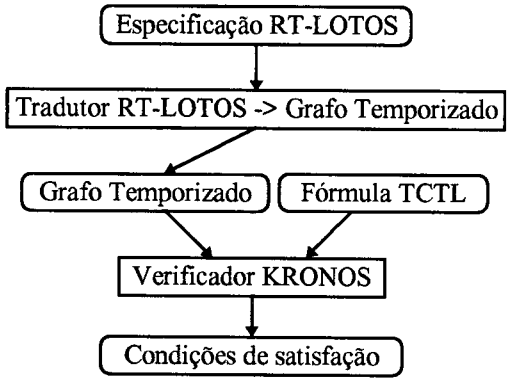


Figura 2.1 - O Processo de Verificação

2.5 CONCLUSÕES

Neste capítulo, foram apresentados alguns dos métodos formais existentes na literatura, para a especificação de sistemas dependentes do tempo, sendo que dentre estes foi escolhida a linguagem RT-LOTOS, pois possui características interessantes quanto a sua expressividade, adequando-se também à representação por Grafos Temporizados, de utilidade neste trabalho para a fase de verificação.

Foi apresentado também o método de verificação destes sistemas, que utiliza KRONOS como ferramenta auxiliar, onde as propriedades do sistema são descritas através da lógica temporal TCTL.

Em [Yovine93], foi desenvolvido um tradutor de especificações escritas através da álgebra de processos ATP [Nicollin92] para Grafos Temporizados, porém, a linguagem ATP possui limitações quanto a representação da ocorrência de uma ação em um intervalo de tempo e na representação e tratamento de violações temporais. E em [DOY94], foi apresentada uma

proposta de mapeamento para uma extensão de LOTOS (ET-LOTOS), entretanto, nenhum mecanismo de tradução automático foi desenvolvido nem implementado, deixando incompleta uma validação da abordagem para as extensões temporais de LOTOS.

Nos próximos capítulos, serão apresentados a metodologia utilizada na implementação do tradutor de especificações RT-LOTOS para Grafos Temporizados, e o teste do mesmo.

CAPÍTULO 3

REPRESENTAÇÃO DE RT-LOTOS POR GRAFOS TEMPORIZADOS FINITOS

Para se construir um grafo temporizado finito que represente uma especificação escrita em RT-LOTOS, é preciso mostrar como construí-lo através do tratamento de cada subexpressão contida na especificação, levando assim a definição de regras de translação para cada operador da linguagem. Em [CF95], foi apresentada uma maneira de representar tais operadores por grafos temporizados finitos, porém, a representação contida neste capítulo mostra-se mais interessante, do ponto de vista da implementação do tradutor e da verificação posterior do modelo do sistema.

Neste sentido, este capítulo apresenta um método de construção de grafos temporizados finitos, a partir de especificações escritas em RT-LOTOS, também chamado de método de construção guiado pela sintaxe. Este método é semelhante ao apresentado em [DOY94, Yovine93], que descreve, respectivamente, a construção de grafos temporizados finitos de termos escritos em ET-LOTOS [LL94] e ATP [Nicollin92].

A organização deste capítulo é feita da seguinte maneira: primeiramente, é apresentada a definição de Grafos Temporizados Finitos, definindo também uma extensão destes grafos, que tem como finalidade melhorar a maneira de expressar a relação entre relógios e vértices, e acrescentar uma ação especial, ε, descrita a seguir. Em seguida, é definida a forma de representar cada operador da linguagem RT-LOTOS em Grafos Temporizados. Ao final, são apresentados alguns exemplos ilustrativos.

3.1 GRAFOS TEMPORIZADOS FINITOS

Um Grafo Temporizado [AD91, NSY92] é uma representação de um autômato estendido com um conjunto de variáveis reais, chamadas de relógios, cujos valores crescem uniformemente com o passar do tempo.

Um relógio pode ser inicializado por um arco do grafo, e o valor de cada relógio é igual ao tempo decorrido desde a sua última inicialização. São associadas restrições sobre os relógios aos vértices e aos arcos do grafo. O sistema pode permanecer num vértice enquanto as restrições associadas a este vértice forem verificadas pelos valores dos relógios. Um arco pode ser disparado somente se

os valores correntes dos relógios satisfazem a condição associada ao arco.

Dentre as definições formais de Grafos Temporizados disponíveis na literatura [CF95, DOY94, Yovine93], optou-se pela definição apresentada em [Yovine93], pois esta se corresponde ao formalismo adotado na construção da ferramenta KRONOS [OY93], que será utilizada na fase de verificação do modelo de um sistema.

Definição 3.1 Seja A um vocabulário de ações, no qual a denota um elemento de A. Seja também $\psi(C)$ um conjunto de restrições temporais sobre o conjunto de relógios C.

Um Grafo Temporizado é uma quíntupla $<S, C, L, s_I, \delta>$, onde:

S $\Rightarrow$ conjunto dos vértices, onde cada vértice representa um ou mais estados do sistema.

C $\Rightarrow$ conjunto dos relógios

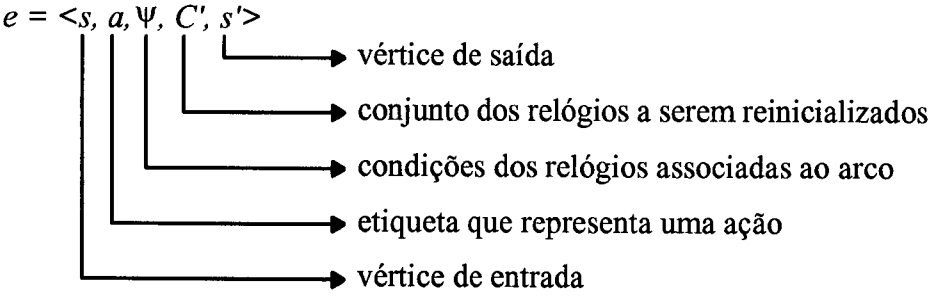
L $\Rightarrow$ conjunto dos arcos ($L \subseteq S \times A \times \Psi(C) \times 2^C \times S$)

$s_I \in S$ $\Rightarrow$ vértice inicial

$\delta: S \rightarrow \Psi(C)$ $\Rightarrow$ função que associa a cada vértice uma condição de atividade (condição de permanência no vértice).

□

Cada arco possui o seguinte formato:



Partindo do vértice inicial de um grafo temporizado, todos os relógios são inicializados.

Se o sistema está em um determinado vértice, s, então um arco, e, poderá ser disparado somente se os valores dos relógios satisfazem a condição Ψ, resultando na mudança para outro vértice, s', com a inicialização dos relógios pertencentes ao conjunto C'.

As condições dos relógios associadas aos arcos e as condições de atividade associadas aos vértices são combinações do tipo $x \sim c$, onde $x \in C$, $c \in$ N (números naturais) e $\sim$ é uma relação binária pertencente ao conjunto $\{<, \leq, >, \geq, =\}$.

3.1.1 Grafos Temporizados Estendidos

Esta extensão de Grafos Temporizados tem como objetivo facilitar a determinação dos relógios a serem inicializados pelos arcos, que depende de uma relação entre o conjunto de vértices e o conjunto de relógios do grafo. A definição formal de Grafos Temporizados Estendidos é dada a seguir [Yovine93].

Definição 3.2 Um Grafo Temporizado Estendido (GTE) é representado pela sêxtupla $<S, C, L, s_I, \delta, F>$, onde $<S, C, L, s_I, \delta>$ é um Grafo Temporizado, sendo $F \subseteq S \times C$ um conjunto de extensões, associando os relógios pertencentes ao conjunto C com os vértices do conjunto S.

Para todo $s \in S$, é definido:

$$F(s) = \{x \in C \mid (s, x) \in F\}$$

□

Assim, será possível obter o grafo temporizado de um processo P, denotado por $G[P]$, a partir do GTE, denotado por $\Phi[P]$. Ou seja, para $\Phi[P] = <S, C, L, s_I, \delta, F>$, definido como sendo o GTE de P, então o grafo temporizado de P será $G[P] = <S, C, L, s_I, \delta>$.

Outra definição importante é apresentada a seguir:

Definição 3.3 Seja $G = <S, C, L, s_I, \delta, F>$ um GTE, então:

$$Reach(G) = < \overline{S}, C, \overline{L}, s_I, \overline{\delta}, \overline{F} >$$

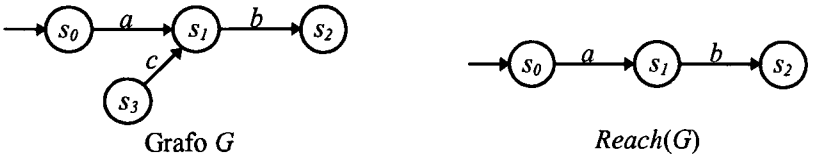
onde:

$s_I \in \overline{S}$
Se $s \in \overline{S}$ e $< s, a, \psi, C', s' > \in L$, então $s' \in \overline{S}$ $(\overline{S} \subseteq S)$
$\overline{L} = L \cap (\overline{S} \times A \times \psi(C) \times 2^C \times \overline{S})$
$\overline{\delta} = \delta \cap (\overline{S} \times \psi)$
$\overline{F} = F \cap (\overline{S} \times C)$

□

Reach(G) é um grafo temporizado estendido onde os vértices são unicamente os vértices acessíveis a partir do vértice inicial s_I do grafo G, ou seja, eliminando os vértices inacessíveis e os arcos que nunca serão disparados.

Para ilustrar a utilização do grafo *Reach*(G), a Figura 3.1 apresenta um simples exemplo, contendo o grafo G e o grafo *Reach*(G).



Grafo G *Reach*(G)

Figura 3.1 - Exemplo de um grafo *Reach*(G)

Neste exemplo, não existe uma seqüência de disparos possível, na qual o arco c seja disparado a partir do vértice inicial, s_0. Não existe também uma seqüência de disparos possível, na qual o vértice s_3 seja atingido. Assim, o grafo *Reach*(G) não possui o vértice s_3 e o arco c.

3.2 REPRESENTAÇÃO DOS OPERADORES RT-LOTOS EM GTE

As definições apresentadas a seguir demonstram como se constrói um GTE, $\Phi[P]$, a partir de expressões de comportamento, P, descritas em RT-LOTOS [Camargo95]. Como o Grafo Temporizado é construído a partir de tais expressões, é necessário saber tratar de cada sub-expressão, ou seja, de cada operador da linguagem.

Limitações:

A construção de tais grafos possui certas limitações, discutidas em [CF95, Garavel89]. É estabelecido que será possível construir um GTE, a partir de uma expressão P, somente se esta especificação levar a um sistema de transições finito, gerando assim um grafo com um número finito de vértices. Por exemplo, $P = Q \,|||\, i; P$ é um processo no qual não possui representação por um sistema de transições finito, apesar de ser uma expressão válida para RT-LOTOS. Em [Garavel89], observou-se que especificações escritas em LOTOS que apresentam uma recursividade contida em um dos operandos de um operador de composição paralela, ou contida no operando esquerdo de um operador de composição seqüencial ou de preempção, podem gerar um sistema de transições infinito, classificando estas recursividades como proibidas, do ponto de vista da geração do grafo.

O vocabulário de ações possíveis nos grafos temporizados obtidos a partir de RT-LOTOS é o seguinte:

$$A = Act \cup Act^* \cup \{i, \delta, \varepsilon\} \equiv Act^{i, *, \delta, \varepsilon}$$

Este vocabulário contém as ações possíveis em RT-LOTOS, além da introdução da ação especial ε. Esta ação é necessária para os métodos de construções composicionais [Garavel89], em particular, na representação do operador de composição paralela, onde expressa a habilitação das ações sincronizáveis. A ação ε não corresponde a uma ação observável, caracterizada também por

ser atômica, significando que ela não pode ser interrompida por uma ação concorrente. Este tipo de ação é utilizado de forma análoga às ε-transições da teoria de autômatos não-determinísticos [ASU86].

Para a representação dos operadores, serão considerados dois processos genéricos, P e Q, onde os seus GTE's são:

$$\Phi[P] \equiv <S_1, C_1, L_1, s_{I_1}, \delta_1, F_1> \text{ e } \Phi[Q] \equiv <S_2, C_2, L_2, s_{I_2}, \delta_2, F_2>$$

3.2.1 Inação

O GTE para o processo *stop* consiste apenas de um vértice, ilustrado pela Figura 3.2. Ele não possui nenhum arco, e a condição de atividade deste vértice é sempre verdadeira (*true*), possibilitando uma permanência indefinida.

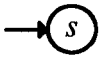


Figura 3.2 - GTE para o processo *stop*

$$\Phi[stop] = <\{s\}, 0, 0, s, \{(s, true)\}, 0>$$

3.2.2 Terminação com sucesso

O GTE para o processo *exit* consiste de dois vértices e um arco etiquetado com a ação δ, lembrando que esta ação não é urgente, caracterizada pela condição de atividade do vértice *so* (*true*), e pela condição do arco (*true*). Este fato dispensa a atribuição de um novo relógio.

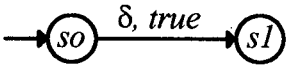


Figura 3.3 - GTE para o processo *exit*

$$\Phi[exit] = <\{so, s1\}, 0, e, so, \{(so, true), (s1, true)\}, 0>$$

onde:

$e = <so, \delta, true, 0, s1>$

3.2.3 Prefixação

A prefixação será representada de duas formas: $[t_1, t_2]i; P$, correspondente a prefixação de um

processo pela ação interna i; e $[t_1, t_2]a$; P, correspondente a prefixação de um processo por uma ação observável a. Esta diferenciação se deve ao fato de que, no instante $t = t_2$, a ação i torna-se urgente e incontrolável, enquanto que uma ação observável não ocorrerá mais, gerando uma violação temporal.

A representação da urgência de uma ação, em GTE, se faz através da condição de atividade associada ao vértice de entrada do arco, ou seja, quando o relógio t, associado ao vértice, atingir um valor $t_{lim} = t_2$, e se este vértice possuir como condição de atividade $t \leq t_{lim}$, o arco que possuir suas condições satisfeitas será, obrigatoriamente, disparado.

(i.) Para $[t_1, t_2]i$; P, com $t_2 \geq t_1$, tem-se a Figura 3.4.

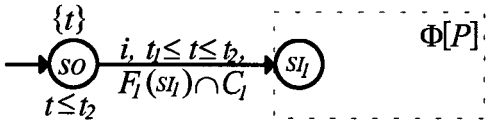


Figura 3.4 - GTE para a prefixação da ação interna *i*

A expressão $F_1(sI_1) \cap C_1$ estabelece o conjunto de relógios a serem inicializados pelo arco, que correspondem aos relógios relacionados ao vértice inicial do processo P. Assim, quando o relógio t, atingir o valor t_2, no vértice so, a ação i será disparada de maneira urgente e incontrolável, caso não tenha ocorrido durante o intervalo $[t_1, t_2)$, lembrando que i; P significa uma prefixação do tipo $[0, 0]i$; $P = [0]i$; P, e $[t]i$; $P = [t, t]i$; P. Portanto, a representação de $[t_1, t_2]i$; P, em GTE, é dada por:

$$\Phi[[t_1, t_2]i; P] = \langle S_1 \cup \{so\}, C_1 \cup \{t\}, L_1 \cup \{e\}, so, \delta_1 \cup \delta', F_1 \cup F' \rangle$$

onde:

$so \notin S_1$, $t \notin C_1$ e

$e = \langle so, i, t_1 \leq t \leq t_2, F_1(sI_1) \cap C_1, sI_1 \rangle$

$\delta' = \{(so, t \leq t_2)\}$

$F' = \{(so, t)\}$

(ii.) Para $[t_1, t_2]a$; P, com $t_2 \geq t_1$ e $a \in Act$, tem-se a Figura 3.5.

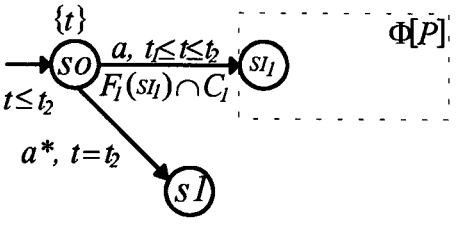


Figura 3.5 - GTE para a prefixação de uma ação observável temporizada

Neste caso, a partir do momento em que se tem $t = t_2$ e a ação a indisponível, a ação a^* passa a existir, tornando-se urgente e incontrolável. O vértice $s1$ não necessita de um relógio, pois a permanência no mesmo é permitida de maneira indefinida (*true*). A representação de $[t_1, t_2]a; P$, em GTE, é dada por:

$$\Phi[[t_1, t_2]a; P] = <S_1 \cup \{so, s1\}, C_1 \cup \{t\}, L_1 \cup L', so, \delta_1 \cup \delta', F_1 \cup F'>$$

onde:
$\{so, s1\} \not\subset S_1$, $t \notin C_1$ e:
$L' = \{<so, a, t_1 \leq t \leq t_2, F_1(s_{I_1}) \cap C_1, s_{I_1}>, <so, a^*, t = t_2, 0, s1>\}$
$\delta' = \{(so, t \leq t_2), (s1, true)\}$ e $F' = \{(so, t)\}$

Para uma prefixação do tipo $a; P$, ou seja, uma ação que pode acontecer em qualquer instante, a representação passa a ser da forma apresentada na Figura 3.6.

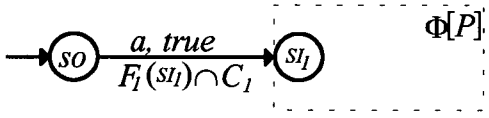


Figura 3.6 - GTE para a prefixação de uma ação observável de ocorrência indefinida

e a representação de $a; P$, em GTE, é dada por:

$$\Phi[a; P] = <S_1 \cup \{so\}, C_1, L_1 \cup \{e\}, so, \delta_1 \cup \delta', F_1>$$

onde:
$so \notin S_1$, $e = <so, a, true, F_1(s_{I_1}) \cap C_1, s_{I_1}>$ e $\delta' = \{(so, true)\}$

3.2.4 Escolha

O método de construção do GTE de P [] Q consiste em construir um grafo, PQ, a partir dos grafos P e Q, onde seus vértices são os pares $(s1, s2)$, com $s1 \in S_1$ e $s2 \in S_2$. A Figura 3.7 ilustra um exemplo. ($a_i, a_j \in Act^{i,\delta}$ e $a_i^*, a_j^* \in Act^*$)

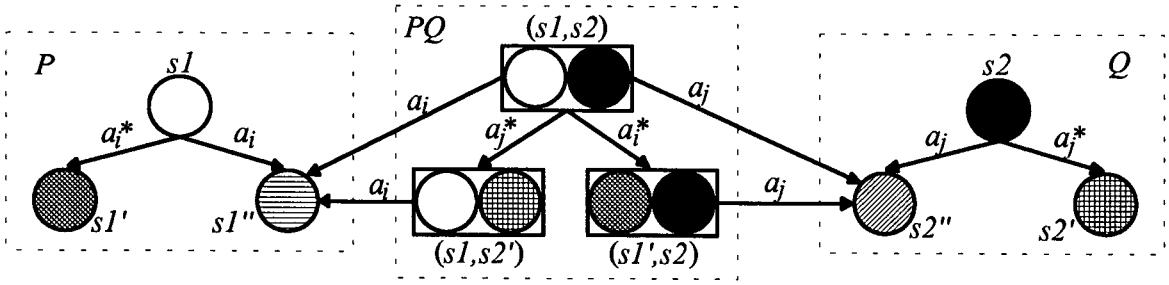


Figura 3.7 - GTE para o processo *P* [] *Q*

O vértice inicial do grafo PQ é o par $(s1, s2)$, que corresponde aos vértices iniciais dos grafos P e Q. Os arcos do grafo PQ são obtidos a partir dos arcos dos grafos P e Q, ou seja, a existência de um arco etiquetado com uma ação $a_i,a_j \in Act^{i,\delta}$ pertencente ao processo P ou Q, no qual o vértice de entrada deste arco é igual a um vértice contido num par do grafo PQ, implica num novo arco para o grafo PQ, que se difere apenas pelo vértice de entrada, resolvendo a escolha. Segundo a semântica de RT-LOTOS, apresentada no capítulo anterior, a violação temporal e a ação especial ε, oriunda de um operador de composição paralela, não resolvem a escolha. Portanto, a existência de um arco etiquetado com uma violação temporal ou com a ação especial ε, em qualquer um dos processos, com o vértice de entrada igual a um vértice contido num par do grafo PQ, implica também num novo arco para o grafo PQ, onde o vértice de entrada é o respectivo par do grafo PQ, e o vértice de saída é um novo par de vértices do grafo PQ, que expressa o fato da ocorrência deste tipo de ação num dos processos envolvidos na escolha.

Assim, o grafo resultante do operador P [] Q é obtido através do conjunto dos três grafos (P, Q e PQ), que são relacionados através das ações que finalizam o processo de escolha, sendo a representação em GTE dada por:

$$\Phi[P \,[]\, Q] = Reach(<S, C, L, s_I, \delta, F>)$$

onde:

$S = S_1 \cup S_2 \cup (S_1 \times S_2)$

$C = C_1 \cup C_2$

$s_I = (s_{I_1}, s_{I_2})$

$L = L_1 \cup L_2$

$\quad \cup \{<(s1, s2), a, \psi, C', s'> \mid <s1, a, \psi, C', s'> \in L_1 \wedge a \in Act^{i,\delta}\}$

$\quad \cup \{<(s1, s2), a, \psi, C', s'> \mid <s2, a, \psi, C', s'> \in L_2 \wedge a \in Act^{i,\delta}\}$

$\quad \cup \{<(s1, s2), a^*, \psi, 0, (s1', s2)> \mid <s1, a^*, \psi, C', s1'> \in L_1 \wedge a^* \in Act^{\varepsilon,*}\}$

$\quad \cup \{<(s1, s2), a^*, \psi, 0, (s1, s2')> \mid <s2, a^*, \psi, C', s2'> \in L_2 \wedge a^* \in Act^{\varepsilon,*}\}$

$F = F_1 \cup F_2 \cup \{((s1, s2), f) \mid (s1, f) \in F_1 \vee (s2, f) \in F_2\}, si \in Si, i = 1,2$

e para cada $s \in S$:

$$\delta[s]: \begin{cases} \delta i[s] & \text{se } s \in Si,\ i = 1,2 \\ \delta_1[s1] \wedge \delta_2[s2] & \text{se } s = (s1, s2) \in S_1 \times S_2 \end{cases}$$

Lembrando que a ação especial ε será adicionada a um determinado grafo somente através do operador de composição paralela. No entanto, os demais operadores devem considerar a possibilidade de seus operandos possuírem tal ação.

3.2.5 Ocultação

Este operador transforma as ações observáveis, pertencentes a um determinado conjunto (J), em ações invisíveis e urgentes (i). Assim, para os arcos com ações pertencentes ao conjunto J, a etiqueta do arco é substituída por i, e a condição dos relógios é alterada de modo a tornar possível o disparo do arco somente no primeiro instante do intervalo dado pela condição anterior. A condição de atividade do vértice de entrada deste arco também é alterada, sendo esta alteração feita apenas no relógio que torna urgente a ocorrência de tal ação.

O GTE para o operador de Ocultação é dado por:

$$\Phi[hide\ J\ in\ P] = <S, C, L, s_I, \delta, F>$$

onde:

$S = S_1$, $C = C_1$, $s_I = s_{I_1}$, $F = F_1$ e

$$L = \{<s, a, \psi, C', s'> \mid <s, a, \psi, C', s'> \in L_1 \wedge a \in Act^{i,\delta,\varepsilon} \wedge a \notin J\}$$
$$\cup \{<s, i, Min(\psi), C', s'> \mid <s, a, \psi, C', s'> \in L_1 \wedge a \in J\}$$
$$\cup \{<s, a^*, \psi, C', s'> \mid <s, a^*, \psi, C', s'> \in L_1 \wedge a \notin J \wedge a^* \in Act^*\}$$
$$\cup \{<s, i, \psi, C', s'> \mid <s, a^*, \psi, C', s'> \in L_1 \wedge a \in J \wedge a^* \in Act^*\}$$

e para cada $s \in S$:

$$\delta[s]: \begin{cases} \delta_1'[s] & \text{se } <s, a, \psi, C', s'> \in L_1 \wedge a \in J \\ \delta_1[s] & \text{se } <s, a, \psi, C', s'> \in L_1 \wedge a \notin J \end{cases}$$

onde $Min(\psi)$ é definida como sendo a função que retorna a condição do relógio, relacionado ao arco, como sendo igual ao menor valor da condição anterior associada à transição, sendo que se a condição for *true*, ou seja, no caso de uma ação com um limite de ocorrência indefinido, a função retorna o valor 0. A condição de permanência, $\delta_1'[s]$, representa a alteração do tempo máximo permitido pelo relógio relacionado com o arco a ser ocultado, tornando o arco urgente no primeiro instante possível. As ações $a^* \in Act^*$, onde $a \in J$, serão fisicamente possíveis somente no caso de uma prefixação do tipo $[t, t]a; E$, pois em qualquer outra situação a ação a ocorrerá primeiro, obrigatoriamente.

3.2.6 Composição Paralela

O operador de composição paralela impõe urgência às ações a serem sincronizadas entre dois processos somente quando ambos estiverem prontos para isto. Assim, é utilizada uma ação

especial, etiquetada como ε. Neste operador, a ação especial ε representa a mudança do vértice de uma ação que deve ser sincronizada, mas que ainda não possui a condição dos relógios relacionada ao arco satisfeita pelo valor mínimo, para o vértice onde esta condição se torna satisfeita. Em outras palavras, ela representa o momento a partir do qual torna-se possível a ocorrência de uma ação, em um dos processos envolvidos na composição paralela.

Para arcos onde o limite mínimo da condição dos relógios seja igual a zero, não é necessário utilizar a ação ε, pois ela já estará habilitada a partir do vértice de entrada do arco. Este é o caso da ação δ, gerada pelo operador de terminação com sucesso.

A sincronização de uma ação pode ser ilustrada da seguinte forma:

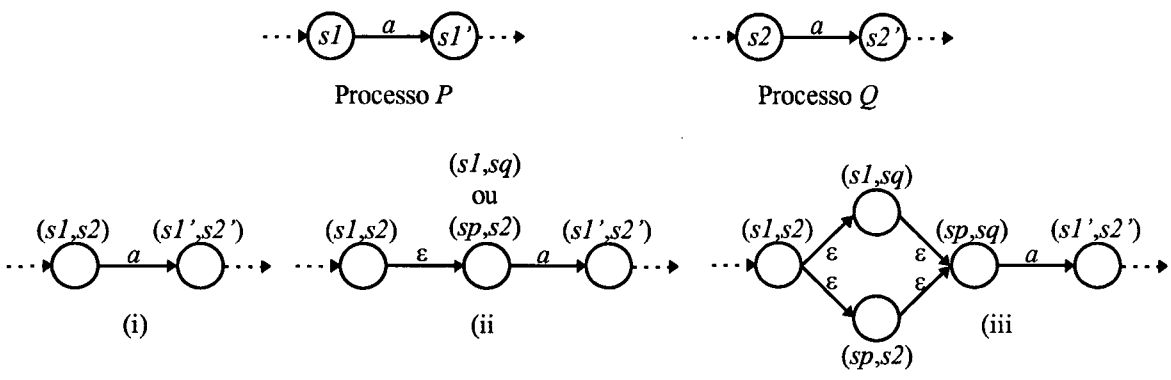


Figura 3.8 - Sincronização de uma ação através do operador de composição paralela: (i) ações com o limite mínimo igual a zero; (ii) uma das ações com o limite mínimo diferente de zero; (iii) ambas as ações com o limite mínimo diferente de zero.

Em (i), tem-se o caso da ação a ser sincronizada, *a*, com os valores do limite mínimo das condições dos relógios relativas aos arcos dos dois processos iguais a zero, como para uma ação δ. No entanto, quando uma ou ambas ações possuírem o limite mínimo diferente de zero, então a ação especial ε é utilizada, como em (ii) e (iii), para expressar a habilitação de uma ação em um dado processo, gerando novos vértices (*sp* e *sq*). Neste exemplo ilustrativo, a violação temporal não foi apresentada apenas para facilitar o entendimento.

O GTE para o operador $P|[J]|Q$ é dado por :

$$\Phi[P\,|[J]|\,Q] = Reach(<S, C, L, s_I, \delta, F>)$$

onde:

$S = (S_1 \times S_2) \cup (S_1 \times S_q) \cup (S_1 \times S_p)$

$C = C_1 \cup C_2$

$s_I = (s_{I1}, s_{I2})$

$L = \{<(s1, s2), a, \psi, C', (s1', s2)> \mid <s1, a, \psi, C', s1'> \in L_1 \wedge a \in Act^i \wedge a \notin J\}$

$\cup \{<(s1, s2), a, \psi, C', (s1, s2')> \mid <s2, a, \psi, C', s2'> \in L_2 \wedge a \in Act^i \wedge a \notin J\}$

$\cup \{<(s1,s2), \varepsilon, Min(\psi_1), 0,(sp, s2)> \mid <s1, a, \psi, C', s1'> \in L_1 \wedge a \in J \wedge Min(\psi_1) \neq 0\}$

$\cup \{<(s1,s2), \varepsilon, Min(\psi_2), 0,(s1,sq)> \mid <s2, a, \psi, C', s2'> \in L_2 \wedge a \in J \wedge Min(\psi_2) \neq 0\}$

$\cup \{<(sp,sq),a,0,C_1'\cup C_2',(s1',s2')> \mid <si,a,\psi i,Ci',si'> \in L_i \wedge a \in J \cup \{\delta\} \wedge Min(\psi i)\neq 0,\ i=1, 2\}$

$\cup \{<(s1,s2),a,0,C_1'\cup C_2',(s1',s2')> \mid <si,a,\psi i,Ci',si'> \in L_i \wedge a \in J \cup \{\delta\} \wedge Min(\psi i)=0,\ i=1, 2\}$

$\cup \{<(s1,sq),a,0,C_1'\cup C_2',(s1',s2')> \mid <si,a,\psi i,Ci',si'> \in L_i \wedge a \in J \cup \{\delta\} \wedge Min(\psi_1)=0 \wedge Min(\psi_2)\neq 0\}$

$\cup \{<(sp,s2),a,0,C_1' \cup C_2',(s1',s2')> \mid <si,a,\psi i,Ci',si'> \in L_i \wedge a \in J \cup \{\delta\} \wedge Min(\psi_1)\neq 0 \wedge Min(\psi_2)=0\}$

$\cup \{<(s1, s2), a^*, \psi_1, C_1', (s1', s2)> \mid <s1, a^*, \psi_1, C_1', s1'> \in L_1\}$

$\cup \{<(s1, s2), a^*, \psi_2, C_2', (s1, s2')> \mid <s2, a^*, \psi_2, C_2', s2'> \in L_2\}$

e para $s = (s1, s2) \in S$:

$F = \{((s1, s2), f) \mid (s1, f) \in F_1 \vee (s2, f) \in F_2\}$

$$\delta[s]: \begin{cases} \delta_1(s1) \wedge \delta_2(s2) \text{ se } < si, a, \psi i, Ci', si' > \notin Li \wedge a \in J,\ i = 1,2 \\ (\delta_1(s1) \wedge Clock(\psi_2) \leq Min(\psi_2)) \vee (\delta_2(s2) \wedge Clock(\psi_1) \leq Min(\psi_1)) \text{ caso contrario} \end{cases}$$

onde $Clock(\psi)$ é definida como sendo a função que retorna o relógio utilizado na condição ψ.

O GTE resultante possui como conjunto de vértices o produto cartesiano dos vértices de P e Q, unidos com o produto cartesiano dos vértices de P com os vértices que representam a habilitação das ações a serem sincronizadas em Q (S_q), e com o produto cartesiano dos vértices de Q com os vértices que representam a habilitação das ações a serem sincronizadas em P (S_p).

3.2.7 Composição Seqüencial

O GTE para o operador $P>>Q$ é dado por:

$$\Phi[P >> Q] = Reach(<S, C, L, s_I, \delta, F>)$$

onde:

$S = (S_1 \times s_{I2}) \cup S_2$

$C = C_1 \cup C_2 \cup \{t\},\ t \notin C_1 \cup C_2$

$s_I = (s_{I1}, s_{I2})$

$L = L_2 \cup L'$

$L' = \{<(s1, s_{I2}), a, \psi, C' \cup \{t\}, (s1', s_{I2})> \mid <s1, a, \psi, C', s1'> \in L_1 \wedge a \in Act^{i,\varepsilon,*} \wedge s1 \overset{\delta}{\not\rightarrow}\}$

$\cup \{<(s1, s_{I2}), i, true, F_2(s_{I2}) \cap C_2, s_{I2}> \mid <s1, \delta, \psi, C', s1'> \in L_1\}$

$F = F_2 \cup F'$

$$F': \begin{cases} ((s1, s_{I2}), f) \mid (s1, f) \in F_1 \wedge s1 \overset{\delta}{\not\rightarrow} \wedge s1 \in S_1 \\ ((s1, s_{I2}), t) \mid s1 \overset{\delta}{\rightarrow} \wedge s1 \in S_1 \end{cases}$$

e para cada $s \in S$:

$$\delta[s]: \begin{cases} \delta_2[s] & \text{se } s \in S_2 \\ \delta_1[s1] & \text{se } s = (s1, s_{I2}) \in (S_1 \times \{s_{I2}\}) \wedge s1 \overset{\delta}{\not\rightarrow} \\ (s, t \leq 0) & \text{se } s = (s1, s_{I2}) \in (S_1 \times \{s_{I2}\}) \wedge s1 \overset{\delta}{\rightarrow} \end{cases}$$

onde a notação $s1 \overset{\delta}{\not\rightarrow}$ significa que não existe uma ação δ que esteja pronta para se realizar, a partir do vértice $s1$; e a notação $s1 \overset{\delta}{\rightarrow}$ significa que existe uma ação δ que esteja pronta para se realizar, a partir do vértice $s1$.

A ocorrência da ação δ inicia o processo Q, lembrando que o progresso no tempo ocorrerá somente se a ação δ não for possível. Assim, δ se torna uma ação invisível e urgente, com o auxílio de um novo relógio que é associado aos vértices de saída das transições δ. Semelhante ao operador de ocultação, este relógio é adicionado ao conjunto dos relógios a serem inicializados de todos os arcos.

3.2.8 Preempção

O GTE para o operador $P[>Q$ é dado por:

$$\Phi[P\ [>\ Q] = Reach(<S, C, L, s_I, \delta, F>)$$

onde:

$S = S_1 \cup S_2 \cup (S_1 \times \{s_{I2}\})$

$C = C_1 \cup C_2$

$s_I = (s_{I1}, s_{I2})$

$L = L_1 \cup L_2 \cup L'$

$L' = \{<(s1, s_{I2}), a, \psi, C', (s1', s_{I2})> \mid <s1, a, \psi, C', s1'> \in L_1 \wedge a \in Act^{i,\varepsilon,*}\}$

$\quad \cup \{<(s1, s_{I2}), a, \psi, C', s2> \mid <s_{I2}, a, \psi, C', s2> \in L_2 \wedge a \in Act^{i,*}\}$

$\quad \cup \{<(s1, s_{I2}), \delta, \psi, C', s1'> \mid <s1, \delta, \psi, C', s1'> \in L_1\}$

$F = F_1 \cup F_2 \cup \{((s1, s2), f) \mid (s1, f) \in F_1 \vee (s_{I2}, f) \in F_2\}$

e para cada $s \in S$:

$$\delta[s]: \begin{cases} \delta_1[s] & \text{se } s \in S_1 \\ \delta_2[s] & \text{se } s \in S_2 \\ \delta_1[s1] \wedge \delta_2[s_{I2}] & \text{se } s = (s1, s_{I2}) \in (S_1 \times \{s_{I2}\}) \end{cases}$$

Assim, o processo P avança normalmente até que ocorra uma ação de Q, finalizando o processo P e dando continuidade ao processo Q, ou quando ocorrer a ação δ. Neste caso, a ocorrência da ação δ finaliza o operador de preempção.

3.2.9 Preempção Temporal

Este operador trata das violações temporais. Assim, se existir uma violação temporal e, se a mesma for tratada pelo operador, ou seja, pertencer ao conjunto J, então este processo é encerrado e um novo processo de tratamento é lançado, quando esta violação ocorrer.

O GTE para o operador $P <J]$ $\{Uj(aj: Qj)\}$ é dado por:

$$\Phi[P <J] \{Uj(aj: Qj)\}] = Reach(<S, C, L, s_I, \delta, F>), j=1, ..., k$$

onde:

$\Phi[P] \equiv <S_P, C_P, L_P, s_{IP}, \delta_P, F_P>$
$\Phi[Qj] \equiv <S_j, C_j, L_j, s_{Ij}, \delta_j, F_j>$
$S = S_P \cup (UjSj) \cup (S_P \times \{s_{I1}\} \times ... \times \{s_{Ik}\})$, $C = C_P \cup (UjCj)$,
$s_I = (s_{IP}, s_{I1}, ..., s_{Ik})$, $L = L_P \cup (UjLj) \cup L'$ e

$$L' = \{<(sp, s_{I1}, ..., s_{Ik}), a, \psi, C', (sp', s_{I1}, ..., s_{Ik})> \mid <sp, a, \psi, C', sp'> \in L_P \wedge a \in Act^{i,\varepsilon}\}$$
$$\cup \{<(sp, s_{I1}, ..., s_{Ik}), \delta, \psi, C', sp'> \mid <sp, \delta, \psi, C', sp'> \in L_P\}$$
$$\cup \{<(sp, s_{I1}, ..., s_{Ik}), i, \psi, Fj(s_{Ij}) \cap C_j, s_{Ij}> \mid <sp, aj^*, \psi, C', sp'> \in L_P \wedge aj \in J\}$$
$$\cup \{<(sp, s_{I1}, ..., s_{Ik}), b^*, \psi, C', (sp', s_{I1}, ..., s_{Ik})> \mid <sp, b^*, \psi, C', sp'> \in L_P \wedge b \notin J\}$$

$F = F_I \cup (UjFj) \cup \{((sp, s_{I1}, ..., s_{Ik}), f) \mid (sp, f) \in F_P \vee (s_{I1}, f) \in F_I \vee ... \vee (s_{Ik}, f) \in F_k\}$

e para cada $s \in S$:

$$\delta[s]: \begin{cases} \delta_P[s] & \text{se } s \in S_P \\ \delta_j[s] & \text{se } s \in S_j, j = 1, ..., k \\ \delta_P[sp] \wedge \delta_1[s_{I1}] \wedge ... \wedge \delta_k[s_{Ik}] & \text{se } s = (sp, s_{I1}, ..., s_{Ik}) \in (S_P \times \{s_{I1}\} \times ... \times \{s_{Ik}\}) \end{cases}$$

3.2.10 Instanciação de Processos

O operador de instanciação é definido a partir da função de troca de nomes de ações, representada por $\phi = [a1/a1', ..., an/an']$, em que para todo j, a ação aj' pode tornar-se aj.

O GTE para o operador $P[a1/a1', ..., an/an']$ é dado por:

$$\Phi[P[a1/a1', ..., an/an']] = <S, C, L, s_I, \delta, F>$$

onde:

$S = S_1$, $C = C_1$, $s_I = s_{I1}$, $\delta = \delta_1$, $F = F_1$ e

$L = \{<s, aj, \psi, C', s'> \mid <s, aj', \psi, C', s'> \in L_1 \wedge aj/aj' \in \phi \wedge \{aj, aj'\} \subseteq Act \cup Act^*\}$
$\cup \{<s, a, \psi, C', s'> \mid <s, a, \psi, C', s'> \in L_1 \wedge a \notin \{a1', a2', ..., an'\}\}$

3.3 ALGUNS EXEMPLOS ILUSTRATIVOS

Nesta seção, são apresentados alguns exemplos extraídos de [Camargo95], com o objetivo de ilustrar o mecanismo de translação apresentado na seção anterior.

3.3.1 Login Simplificado

Este exemplo pode ser descrito da seguinte forma: Para iniciar o procedimento, o sistema envia um *prompt* de *Login*, denominado *p*. Após este evento, o usuário tem *l* unidades de tempo para entrar com o nome válido, denominado *v*, para terminar a tarefa e iniciar a fase sessão *S*. Quando o usuário fornece uma resposta inválida, denominada *n*, ou quando o tempo *l* expira, um novo procedimento de *Login* é lançado.

A representação em RT-LOTOS é dada por:

Login(*p*, *n*, *v*) := *p*; (*v*; *S* [] [0, *l*]*n*; *Login*(*p*, *n*, *v*)) <*n*] {*n*: *Login*(*p*, *n*, *v*)}

Para a construção do GTE, é necessário dividir o processo em vários sub-processos, afim de tratar cada um dos operadores separadamente. Assim:

Login := *p*; Q_1

O GTE para *L*ogin é dado por:

$\Phi[Login] = <S, C, L, s_I, \delta, F>$ e $\Phi[Q_1] = <S_1, C_1, L_1, s_{I1}, \delta_1, F_1>$

O primeiro operador é uma prefixação do processo Q_1 pela ação *p*, de execução indefinida.

Graficamente, tem-se a Figura 3.9.

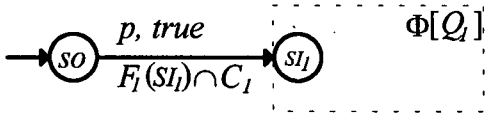


Figura 3.9 - GTE da prefixação da ação *p*

onde:

$S = S_1 \cup \{so\}$, $C = C_1$, $L = L_1 \cup \{<so, p, true, F_1(si_1) \cap C_1, si_1>\}$
$si = so$, $F = F_1$ e $\delta = \delta_1 \cup \{(so, true)\}$

Ou seja, este GTE possui como vértice inicial *so*, sendo que não foi necessário associar nenhum relógio e nenhuma condição de atividade a *so*, pois a ação *p* pode ocorrer em qualquer instante. O conjunto $F_1(si_1) \cap C_1$ representa o conjunto dos relógios associados ao vértice si_1, e que devem ser inicializados quando a ação *p* ocorrer.

Mas, $Q_1 := (Q_2 [] Q_3) <n] \{n: Login\}$

sendo que:

$Q_2 := v; S$, onde os GTE's são dados por:

$$\Phi[Q_2] = <S_2, C_2, L_2, si_2, \delta_2, F_2> \text{ e } \Phi[S] = <S_s, C_s, L_s, si_s, \delta_s, F_s>$$

De maneira análoga a Q_1, Q_2 pode ser representado graficamente através da Figura 3.10.

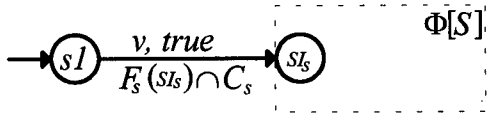


Figura 3.10 - GTE da prefixação da ação *v*

onde:

$S_2 = S_s \cup \{s1\}$, $C_2 = C_s$, $L_2 = L_s \cup \{<s1, v, true, Fs(si_s) \cap C_s, si_s>\}$
$si_2 = s1$, $F_2 = F_s$ e $\delta_2 = \delta_s \cup \{(s1, true)\}$

$Q_3 := [0, l]n; Login$

Neste tipo de prefixação, um novo relógio é associado ao vértice inicial, e as condições para a ocorrência das ações *n* e *n** são baseadas neste relógio. É criado um vértice "poço" (*sink*), no qual o arco da ação *n** está direcionado.

Graficamente, tem-se a Figura 3.11.

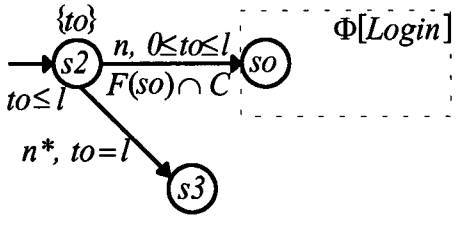


Figura 3.11 - GTE da prefixação da ação *n*

onde:

$S_3 = S \cup \{s2, s3\}$, $C_3 = C \cup \{to\}$, $s_{I3} = s2$,
$L_3 = L \cup \{<s2, n, 0 \leq to \leq l, F(so) \cap C, so>, <s2, n^*, to = l, 0, s3>\}$,
$F_3 = F \cup \{(s2, to)\}$ e $\delta_3 = \delta \cup \{(s2, to \leq l), (s3, true)\}$

$Q_4 := Q_2 \; [] \; Q_3$

Para o operador de escolha, novos vértices são criados a partir dos GTE's de Q_2 e Q_3, ou seja, pelo produto cartesiano S_2 x S_3. Porém, todos estes novos vértices não serão obrigatoriamente utilizados.

Graficamente, tem-se a Figura 3.12.

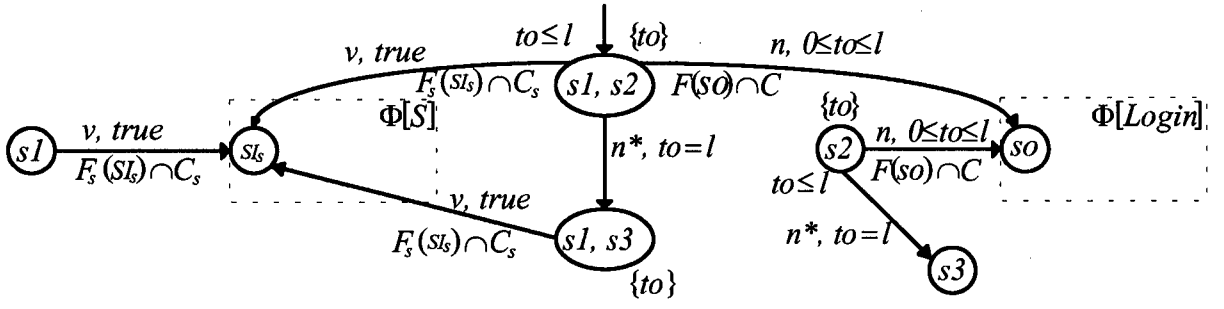


Figura 3.12 - GTE parcial do processo Q_4

O grafo correspondente a Q_4 é obtido através do GTE $Reach(Q_2 \; [] \; Q_3)$. Ou seja, a partir do grafo ilustrado acima, procura-se o grafo dos vértices acessíveis partindo-se do vértice inicial. Os vértices $(s1)$, $(s2)$ e $(s3)$ não são acessíveis a partir do vértice inicial $(s1, s2)$, pois não existe nenhuma seqüência de ações na qual estes vértices são atingidos. Assim, estes vértices, bem como os arcos onde o vértice de entrada for igual a um destes, são eliminados, e o GTE acessível é dado pela Figura 3.13.

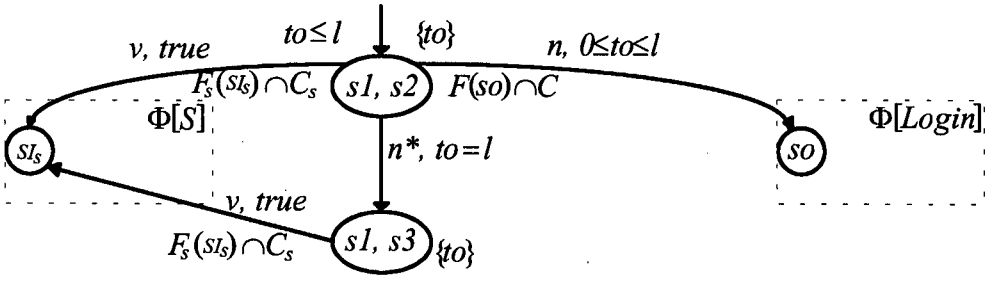


Figura 3.13 - GTE do processo Q_4

onde:

$S_4 = S \cup S_s \cup \{(s1, s2), (s1, s3)\}$, $C_4 = C \cup C_s \cup \{to\}$, $s_{I4} = (s1, s2)$,

$L_4 = L \cup L_s \cup \{<(s1, s2), n, 0 \leq to \leq l, F(so) \cap C, so>, <(s1, s2), n^*, to = l, 0, (s1, s3)>,$
$<(s1, s2), v, true, F_s(s_{I_s}) \cap C_s, s_{I_s}>, <(s1, s3), v, true, F_s(s_{I_s}) \cap C_s, s_{I_s}>\},$
$F_4 = F \cup F_s \cup \{((s1, s2), to), ((s1, s3), to)\}$
e $\delta_4 = \delta \cup \delta_s \cup \{((s1, s2), to \leq l), ((s1,s3), true)\}$

$Q_1 := Q_4 <n]\ \{n: Login\}$

Agora, é tratada a violação temporal n^*, através do operador de tratamento de violações temporais, onde a ação n^* torna-se invisível. Isto é feito para que seja iniciada uma nova execução do processo *Login*, desde que uma resposta válida não seja fornecida em um limite de tempo, l.

Graficamente, tem-se a Figura 3.14.

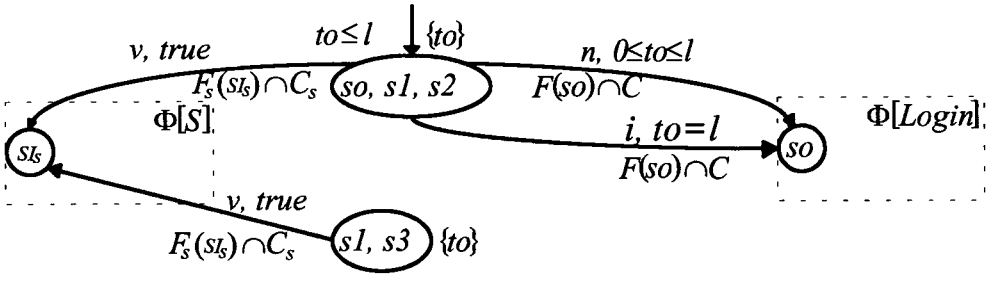


Figura 3.14 - GTE parcial do tratamento da violação temporal do processo Q_4

O vértice $(s1, s3)$ não é acessível a partir de $(so, s1, s2)$, e o GTE acessível é dado na Figura 3.15.

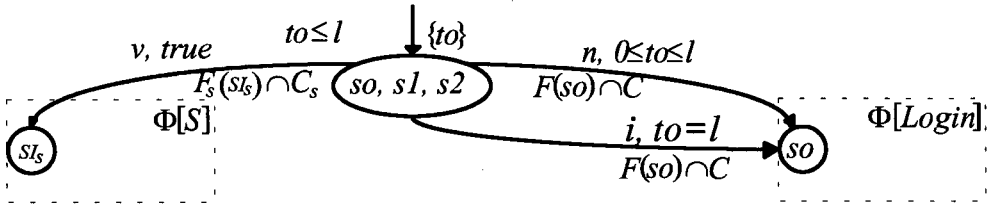


Figura 3.15 - GTE do tratamento da violação temporal do processo Q_4 (Q_1)

onde:

$S_1 = S \cup S_s \cup \{(so, s1, s2)\}$, $C_1 = C \cup C_s \cup \{to\}$, $s_{I_1} = (so, s1, s2)$,
$L_1 = L \cup L_s \cup \{<(so, s1, s2), n, 0 \leq to \leq l, F(so) \cap C, so>,$
$<(so, s1, s2), i, to = l, F(so) \cap C, so>,$
$<(so, s1, s2), v, true, F_s(s_{I_s}) \cap C_s, s_{IS}>\},$
$F_1 = F \cup F_s \cup \{((so, s1, s2), to)\}$ e $\delta_1 = \delta \cup \delta_s \cup \{((so, s1, s2), to \leq l)\}$

Como $Login := p; Q_1$, tem-se:

$S = S_s \cup \{so, (so, s1, s2)\}$, $C = C_s \cup \{to\}$, $s_I = so$,
$L = L_s \cup \{<so, p, true, \{to\}, (so, s1, s2)>,$
$<(so, s1, s2), n, 0 \leq to \leq l, 0, so>,$
$<(so, s1, s2), i, to = l, 0, so>,$
$<(so, s1, s2), v, true, F_s(s_{I_s}) \cap C_s, s_{I_s}>\}$,
$F = F_s \cup \{((so, s1, s2), to)\}$ e $\delta = \delta_s \cup \{(so, true), ((so, s1, s2), to \leq l)\}$

e o GTE resultante é dado pela Figura 3.16.

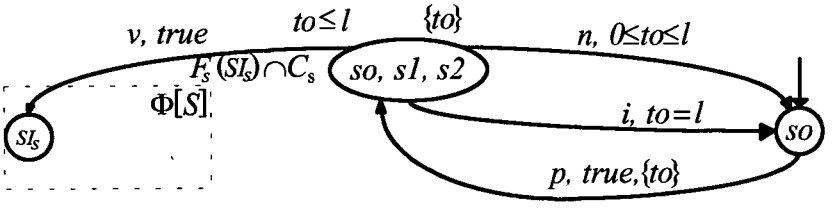


Figura 3.16 - GTE do processo de *Login* Simplificado

3.3.2 Login Completo

Neste exemplo, o sistema limita a duração global do procedimento de *Login* em um tempo máximo, *d*, e força uma exceção, *E*, a realizar-se após este tempo.

A especificação em RT-LOTOS do procedimento *Login* completo é apresentada a seguir:

Complete_Login := ($Login_2(p, n, v)$ [> [*d*]*i*; *E*) >> *S*
where
process $Login_2(p, n, v)$:= *p*; (*v*; *exit* [] [0, *l*]*n*; $Login_2(p, n, v)$) <*n*] {*n*: $Login_2(p, n, v)$}

O GTE do processo $Login_2$ pode ser obtido a partir do exemplo anterior, sendo que a única diferença é a maneira na qual o processo *S* é acionado após a realização da ação *v*. Assim:

$$\Phi[Login_2] = \Phi[Q_1] = <S_1, C_1, L_1, s_{I_1}, \delta_1, F_1>$$

Graficamente, tem-se a Figura 3.17.

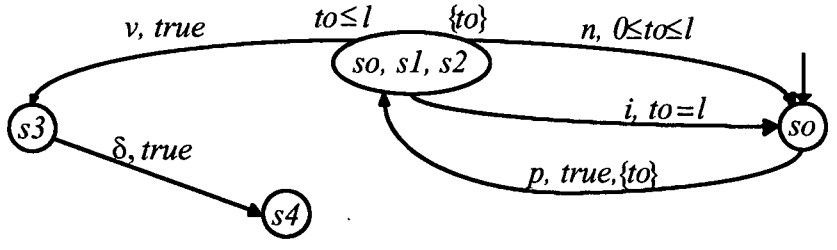


Figura 3.17 - GTE do processo $Login_2$

onde:

$S_1 = \{so, (so, s1, s2), s3, s4\}$, $C_1 = \{to\}$,
$si_1 = so$,
$L_1 = \{ <so, p, true, \{to\}, (so, s1, s2)>, <(so, s1, s2), n, 0 \leq to \leq l, 0, so>,$
$<(so, s1, s2), i, to = l, 0, so>, <(so, s1, s2), v, true, 0, s3>,$
$<s3, \delta, true, 0, s4>\}$,
$F_1 = \{((so, s1, s2), to)\}$ e $\delta_1 = \{(so, true), ((so, s1, s2), to \leq l), (s3, true), (s4, true)\}$

De maneira simplificada, tem-se:

$Complete\_Login := (Q_1 [> Q_2) >> S$

onde:

$Q_2 := [d]i; E$ e $\Phi[Q_2] = <S_2, C_2, L_2, si_2, \delta_2, F_2>$

O processo Q_2 apresenta uma prefixação do processo E pela ação i, com um retardo no tempo d. A ação $[d]i$ é equivalente à ação $[d, d]i$, e a condição do arco $d \leq t1 \leq d$ é equivalente à $t1=d$, onde $t1$ é o novo relógio associado ao vértice inicial.

Graficamente, tem-se a Figura 3.18.

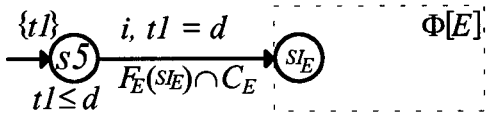


Figura 3.18 - GTE da prefixação da ação *i*

onde:

$S_2 = S_E \cup \{s5\}$, $C_2 = C_E \cup \{t1\}$,
$si_2 = s5$,
$L_2 = L_E \cup \{<s5, i, t1 = d, F_E(si_E) \cap C_E, si_E>\}$,
$F_2 = F_E \cup \{(s5, t1)\}$
e $\delta_2 = \delta_E \cup \{(s5, t1 \leq d)\}$

$Q_3 := Q_1 [> Q_2$ e $\Phi[Q_3] = <S_3, C_3, L_3, si_3, \delta_3, F_3>$

Para o operador de preempção, cada vértice de Q_1 possuirá um arco para o vértice inicial de E, a não ser que o vértice seja um vértice de entrada de um arco etiquetado com a ação δ.

Graficamente, tem-se a Figura 3.19.

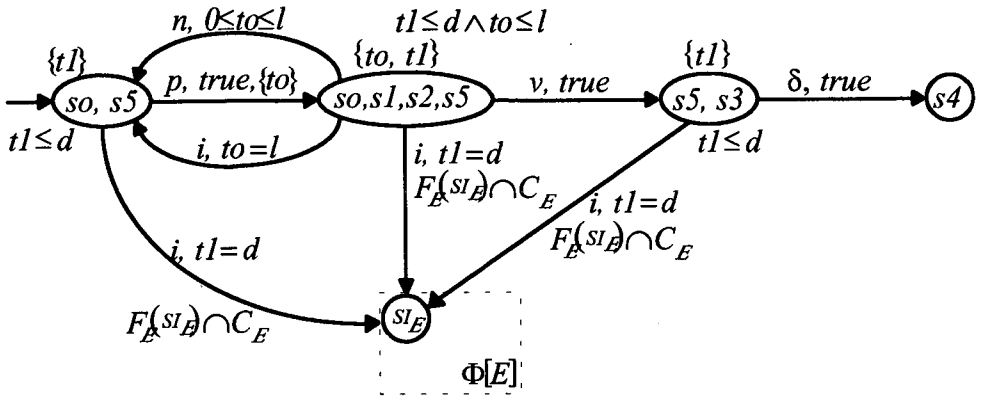


Figura 3.19 - GTE do processo Q_3

onde:

$S_3 = S_E \cup \{(so, s5), (so, s1, s2, s5), (s3, s5), s4\}$, $C_3 = C_E \cup \{to, t1\}$, $sI_3 = (so, s5)$,
$L_3 = L_E \cup \{<(so, s5), p, true, \{to\}, (so, s1, s2, s5)>, <(so, s5), i, t1 = d, F_E(sI_E) \cap C_E, sI_E>,$
$<(so, s1, s2, s5), n, 0 \le to \le l, 0, (so, s5)>, <(so, s1, s2, s5), i, to = l, 0, (so, s5)>,$
$<(so, s1, s2, s5), i, t1 = d, F_E(sI_E) \cap C_E, sI_E>, <(so, s1, s2, s5), v, true, 0, (s3, s5)>,$
$<(s3, s5), \delta, true, 0, s4>, <(s3, s5), i, t1 = d, F_E(sI_E) \cap C_E, sI_E>\},$
$F_3 = F_E \cup \{((so, s5), t1), ((so, s1, s2, s5), to), ((so, s1, s2, s5), t1), ((s3, s5), t1)\}$
e $\delta_3 = \delta_E \cup \{((so, s5), t1 \le d), ((so, s1, s2, s5), t1 \le d \wedge to \le l), ((s3, s5), t1 \le d), (s4, true)\}$

Assim:

Complete_Login := $Q_3 >> S$ e $\Phi[Q_3 >> S] = <S, C, L, sI, \delta, F>$

e o GTE resultante é dado pela Figura 3.20.

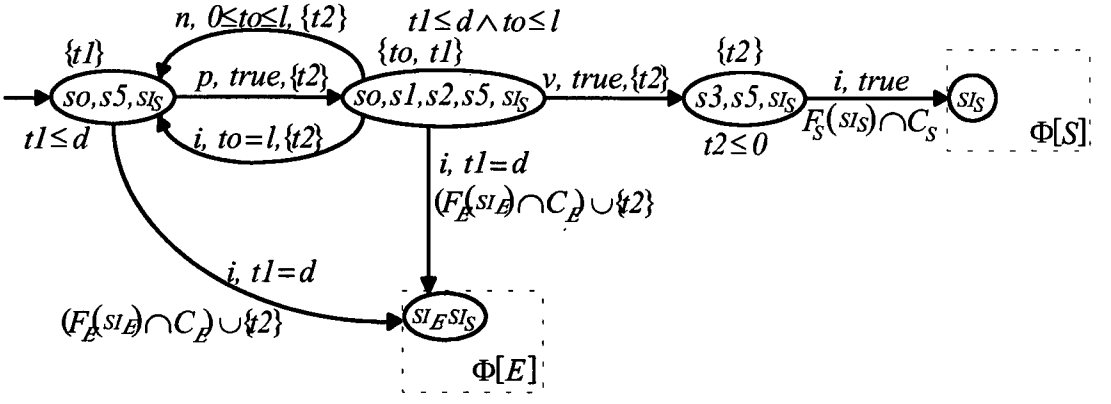


Figura 3.20 - GTE do processo de *Login* Completo

onde:

$S = S_S \cup (S_E \times \{sI_S\}) \cup \{(so, s5, sI_S), (so, s1, s2, s5, sI_S), (s3, s5, sI_S)\}$,

$C = C_E \cup C_S \cup \{to, t1, t2\}, sI = (so, s5, sI_S),$

$L = L_S \cup \{<(so, s5, sI_S), p, true, \{to, t2\}, (so, s1, s2, s5, sI_S)>,$

$<(so, s5, sI_S), i, t1 = d, (F_E(sI_E) \cap C_E) \cup \{t2\}, (sI_E, sI_S)>,$

$<(so, s1, s2, s5, sI_S), n, 0 \le to \le l, \{t2\}, (so, s5, sI_S)>,$

$<(so, s1, s2, s5, sI_S), i, to = l, \{t2\}, (so, s5, sI_S)>,$

$<(so, s1, s2, s5, sI_S), i, t1 = d, (F_E(sI_E) \cap C_E) \cup \{t2\}, (sI_E, sI_S)>,$

$<(so, s1, s2, s5, sI_S), v, true, \{t2\}, (s3, s5, sI_S)>,$

$<(s3, s5, sI_S), i, true, F_S(sI_S) \cap C_S, sI_S>\} \cup$

$\{<(s, sI_S), a, \Psi, C' \cup \{t2\}, (s', sI_S)> \mid <s, a, \Psi, C', s'> \in L_E \wedge a \in Act^{i,*} \wedge s1 \overset{\delta}{\not\rightarrow}\} \cup$

$\{<(s, sI_S), i, true, F_S(sI_S) \cap C_S, sI_S)> \mid <s, \delta, \Psi, C', s'> \in L_E\},$

$F = F_S \cup \{((so, s5, sI_S), t1), ((so, s1, s2, s5, sI_S), to), ((so, s1, s2, s5, sI_S), t1), ((s3, s5, sI_S), t2)\}$

$\cup \{((so, sI_S), f) \mid (s, f) \in F_E(s)\}$

e $\delta = \delta_S \cup \{((so, s5, sI_S), t1 \le d), ((so, s1, s2, s5, sI_S), t1 \le d \wedge to \le l), ((s3, s5, sI_S), t2 \le 0)\} \cup$

$\{((so, sI_S), f) \mid (s, f) \in \delta_E(s)\}$

Um novo relógio ($t2$) foi criado para expressar a urgência das ações δ, que provocam o início do processo S. O processo E foi modificado de tal forma que se ele possuir algum arco etiquetado com a ação δ, este será invisível e urgente, direcionado para o vértice inicial de S.

3.3.3 Um processo com dois relógios

Seja um processo P, resultante da composição paralela de dois processos elementares.

$P := (a; [0, 3]b; E) \,|[b]|\, (c; [0, 5]b; F)$

$:= Q_1 \,|[b]|\, Q_2$

$Q_1 := a; [0, 3]b; E$ e $\Phi[Q_1] = <S_1, C_1, L_1, sI_1, \delta_1, F_1>$

Graficamente, tem-se a Figura 3.21.

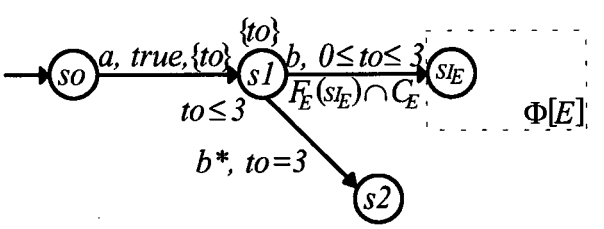


Figura 3.21 - GTE do primeiro processo (Q_1)

onde:

$S_1 = S_E \cup \{so, s1, s2\}, C_1 = C_E \cup \{to\}, sI_1 = so,$

$L_1 = L_E \cup \{<so, a, true, \{to\}, s1>, <s1, b, 0 \le to \le 3, F_E(SI_E) \cap C_E, SI_E>, <s1, b^*, to = 3, 0, s2>\}$,
$F_1 = F_E \cup \{(s1, to)\}$ e $\delta_1 = \delta_E \cup \{(so, true), (s1, to \le 3), (s2, true)\}$

$Q_2 := c; [0, 5]b; F$ e $\Phi[Q_2] = <S_2, C_2, L_2, SI_2, \delta_2, F_2>$

Graficamente, tem-se a Figura 3.22.

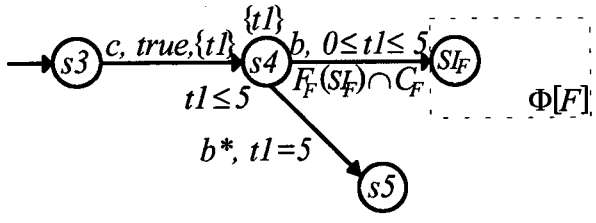


Figura 3.22 - GTE do segundo processo (Q_2)

onde:

$S_2 = S_F \cup \{s3, s4, s5\}$, $C_2 = C_F \cup \{t1\}$,
$SI_2 = s3$,
$L_2 = L_F \cup \{<s3, c, true, \{t1\}, s4>, <s4, b, 0 \le t1 \le 5, F_F(SI_F) \cap C_F, SI_F>, <s4, b^*, t1 = 5, 0, s5>\}$,
$F_2 = F_F \cup \{(s4, t1)\}$ e $\delta_2 = \delta_F \cup \{(s3, true), (s4, t1 \le 5), (s5, true)\}$

O próximo passo será aplicar o operador de composição paralela em Q_1 e Q_2.

Assim, o GTE resultante é dado pela Figura 3.23.

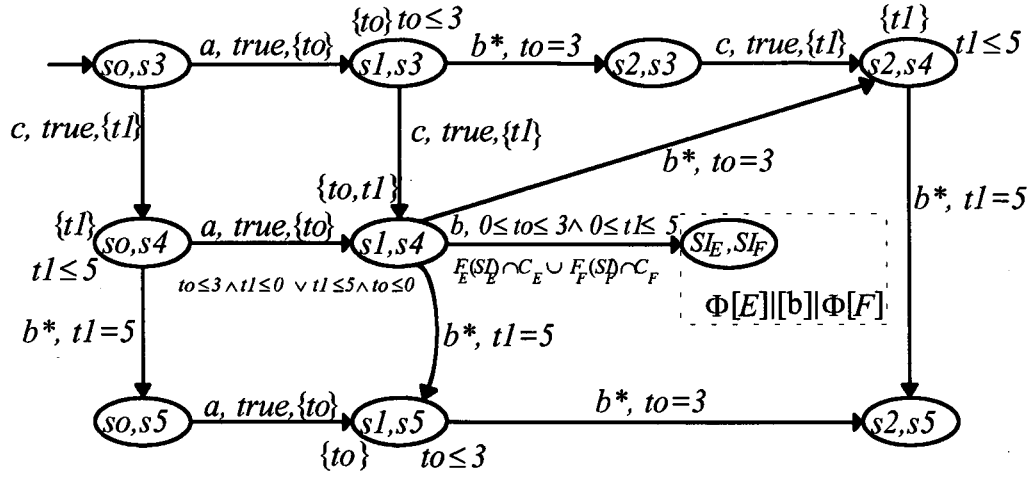


Figura 3.23 - GTE da composição paralela de Q_1 e Q_2

onde:

$S = (S_E \times S_F) \cup \{(so, s3), (so, s4), (so, s5), (s1, s3), (s1, s4), (s1, s5), (s2, s3), (s2, s4), (s2, s5)\}$,
$C = C_E \cup C_F \cup \{to, t1\}$, $SI = (so, s3)$,

L = { $<(so, s3), a, true, \{to\}, (s1, s3)>$, $<(so, s3), c, true, \{t1\}, (so, s4)>$,
$<(so, s4), a, true, \{to\}, (s1, s4)>$, $<(so, s4), b^*, t1 = 5, 0, (so, s5)>$,
$<(so, s5), a, true, \{to\}, (s1, s5)>$,$<(s1, s3), c, true, \{t1\}, (s1, s4)>$,
$<(s1, s3), b^*, to = 3, 0, (s2, s3)>$, $<(s1, s4), b^*, t1 = 5, 0, (s1, s5)>$,
$<(s1, s4), b, 0 \leq to \leq 3 \wedge 0 \leq t1 \leq 5, F_E(s1_E) \cap C_E \cup F_F(s1_F) \cap C_F, (s1_E, s1_F)>$,
$<(s1, s4), b^*, to = 3, 0, (s2, s4)>$, $<(s1, s5), b^*, to = 3, 0, (s2, s5)>$,
$<(s2, s3), c, true, \{t1\}, (s2, s4)>$, $<(s2, s4), b^*, t1 = 5, 0, (s2, s5)>$},
$\cup L_E' \cup L_F'$

F = $\{((so, s4), t1), ((s1, s3), to), ((s1, s4), to), ((s1, s4), t1), ((s1, s5), to), ((s2, s4), t1)\}$
$\cup F_E' \cup F_F'$

e δ = $\{((so, s3), true), ((so, s4), t1 \leq 5), ((so, s5), true), ((s1, s3), to \leq 3)$,
$((s1, s4), to \leq 3 \wedge t1 \leq 0 \vee t1 \leq 5 \wedge to \leq 0), ((s1, s5), to \leq 3), ((s2, s3), true)$,
$((s2, s4), t1 \leq 5), ((s2, s5), true)\} \cup \delta_E' \cup \delta_F'$

Os arcos pertencentes ao conjunto $L_E' \cup L_F'$ são os arcos dos processos E e F, porém com a condição do operador de composição paralela. O mesmo ocorre para $F_E' \cup F_F'$ e $\delta_E' \cup \delta_F'$.

Pode-se notar que apesar dos arcos etiquetados com b^*, a partir do vértice $(s1, s4)$, serem representados, estes nunca ocorrerão, pois neste exemplo temos como limites mínimos o valor 0, provocando a urgência imediata de b.

3.4 CONCLUSÕES

Neste capítulo, foi apresentado um método de construção de grafos temporizados finitos a partir de especificações escritas em RT-LOTOS, através das definições de regras de translação estabelecidas para cada um dos operadores da linguagem. Apesar da abordagem apresentada em [CF95] possibilitar a geração de tal grafo, adotou-se uma outra definição formal de grafos temporizados, visando a obtenção de um formato adequado à implementação do tradutor e verificação do modelo, conforme pôde ser verificado nos exemplos ilustrativos anteriores.

Para que tal mapeamento tenha êxito, é necessário que a especificação possua um comportamento no qual gere um sistema de transições finito.

Com a representação de uma especificação em grafos temporizados, será possível verificar as suas propriedades, através da ferramenta de verificação de sistemas Tempo-Real, KRONOS [OY93].

CAPÍTULO 4

PRINCÍPIOS BÁSICOS PARA A CONSTRUÇÃO DO TRADUTOR DE RT-LOTOS PARA GRAFOS TEMPORIZADOS

Conforme discutido em [Yovine93], a implementação direta das regras de tradução de especificações escritas em RT-LOTOS para Grafos Temporizados, apresentada no capítulo anterior, apesar de realizável, não é a melhor solução do ponto de vista prático, pois esta abordagem compromete o desempenho da compilação. Assim, é proposto em [Yovine93] que esta tradução seja realizada em duas etapas: uma etapa preliminar, tendo o objetivo de simplificar o modelo a ser manipulado, para que em seguida seja construído o Grafo Temporizado.

A organização deste capítulo é feita da seguinte maneira: primeiramente, é apresentada a definição de uma extensão de Redes de Petri, que será utilizada na etapa preliminar do processo de tradução como modelo intermediário para a construção do Grafo Temporizado. Em seguida, é definida de forma similar ao capítulo anterior a maneira de representar cada operador da linguagem através desta extensão, sendo que uma aplicação é feita sobre um dos exemplos ilustrativos do capítulo anterior. A seguir, é descrito o método de construção do Grafo Temporizado a partir da Rede de Petri Estendida, correspondente à segunda etapa do processo de tradução. Ao final, é discutida a forma de implementar a ferramenta, bem como os algoritmos e ferramentas auxiliares utilizados.

4.1 UMA REDE DE PETRI ESTENDIDA

A etapa preliminar da tradução consiste na utilização de uma extensão de Redes de Petri (RdP) como representação intermediária. Esta extensão é baseada na proposta feita em [Yovine93], na qual acrescenta-se condições para tratar de intervalos específicos no tempo, necessários para expressar os termos de RT-LOTOS.

Definição 4.1 A Rede de Petri Estendida é definida pela quadrupla $N = <\rho, \tau, MI, \varphi>$, onde:

ρ $\Rightarrow$ Conjunto dos lugares

τ $\Rightarrow$ Conjunto das transições ($\tau \subseteq \rho \times \rho \times Act^{i,\delta,*,\varepsilon} \times \rho$),

$M_I \subseteq \rho$ $\Rightarrow$ Subconjunto de lugares, correspondendo a marcação inicial

$\varphi: \tau \rightarrow (N \times N)$ $\Rightarrow$ Função que associa a cada transição um intervalo no tempo

□

Seja T = <ρ1, ρ2, α, ρ3> uma transição.

- ρ1 é o conjunto de lugares de entrada obrigatórios de T, denotado por '•T', sendo que uma transição será disparada se todos os lugares de entrada obrigatórios estiverem marcados.
- ρ2 é o conjunto de lugares de entrada opcionais de T, denotado por 'oT'.
- ρ3 é o conjunto de lugares de saída de T, denotado por 'T•'.
- $\alpha \in Act^{i,\delta,*,\varepsilon}$ é a etiqueta de T, denotada por *label*(T).

A defínção anterior representa uma variante do modelo de [Merlin74]. A semântica de redes de Petri [Brams83] é dada por um sistema de transições entre marcações, onde uma marcação é definida como sendo um conjunto de lugares marcados com fichas.

Seja uma transição T qualquer, representada pela Figura 4.1.

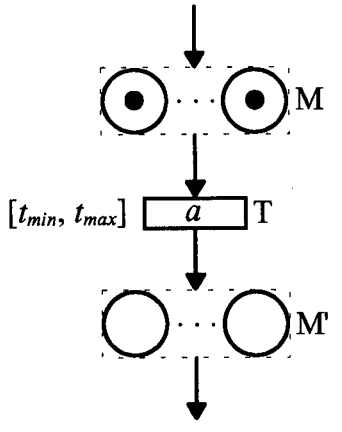


Figura 4.1 - Representação de uma transição da Rede de Petri Estendida

onde $\varphi(T) = [t_{min}, t_{max}]$.

A política de disparo adotada nesta extensão de redes de Petri é definida da seguinte forma: sendo a marcação atual da rede igual a M, e sendo uma transição T onde o seu conjunto de lugares de entrada obrigatórios está contido em M, e ainda que o tempo de habilitação desta transição, t_{enable}, esteja dentro do intervalo estabelecido pela função φ(T), então será possível o disparo da transição T. Após o disparo da transição T, a marcação M' é obtida retirando-se da marcação M os lugares obrigatórios e os lugares opcionais da transição T, adicionando-se também os lugares de saída da transição T. A expressão $M \xrightarrow{T} M'$ indica que o sistema pode passar da marcação M para a marcação M', através do disparo da transição T.

Assim, a relação de transição é formalmente definida através da regra abaixo:

$$\frac{\bullet T \subseteq M,\ \varphi(T) = [t_{min}, t_{max}],\ t_{min} \leq t_{enable} \leq t_{max},\ M' = M - (\bullet T \cup \circ T) \cup T\bullet}{M \xrightarrow{T} M'}$$

Os lugares de entrada obrigatórios serão representados graficamente por uma seta contínua, e os lugares de entrada opcionais por uma seta pontilhada. Será considerado que a rede definida é binária (safe), ou seja, cada lugar possuirá a qualquer instante, no máximo, uma ficha.

4.2 REPRESENTAÇÃO DOS OPERADORES RT-LOTOS EM REDES DE PETRI

As definições apresentadas a seguir demonstram como se constrói uma Rede de Petri Estendida, denotada por $\eta[P]$, a partir de expressões de comportamento, P, descritas em RT-LOTOS.

Sejam:

$$\eta[P] \equiv <\rho_1, \tau_1, M_{I1}, \varphi_1> \text{ e } \eta[Q] \equiv <\rho_2, \tau_2, M_{I2}, \varphi_2>$$

4.2.1 Inação

A rede para o processo *stop* é simplesmente uma rede com um lugar de entrada e sem nenhuma transição.

$$\eta[stop] = <\{p1\}, 0, \{p1\}, 0>$$

4.2.2 Terminação com sucesso

A rede para o processo *exit* é mostrada pela Figura 4.2.

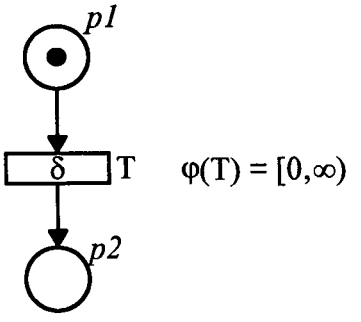


Figura 4.2 - RdP para o processo *exit*

onde:

$$\eta[exit] = <\{p1, p2\}, \{T\}, \{p1\}, \{(T, [0, \infty))\}> \text{ e } T = <\{p1\}, 0, \delta, \{p2\}>$$

A rede possui apenas a transição associada à ação δ, sendo que ela poderá ser disparada em qualquer instante no tempo pertencente ao intervalo associado à esta transição ([0, ∞)), ou seja, em qualquer momento, desde que o lugar *p1* esteja marcado.

4.2.3 Prefixação

A prefixação será apresentada de forma similar à utilizada na representação em GTE. Assim, para uma prefixação do tipo $[t_1, t_2]i; P$, com $t_2 \geq t_1$, tem-se a Figura 4.3.

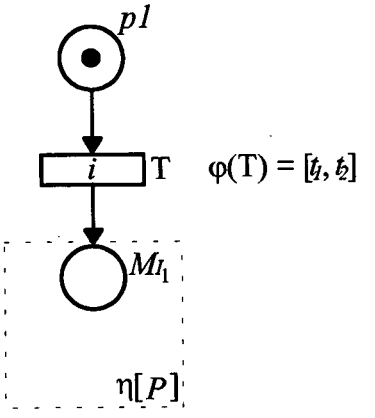


Figura 4.3 - RdP para a prefixação da ação *i*

onde:

$$\eta[[t_1, t_2]i; P] = <\rho_1 \cup \{p1\}, \tau_1 \cup \{T\}, \{p1\}, \varphi_1 \cup \{(T, [t_1, t_2])\}> \text{ e } T = <\{p1\}, 0, i, M_{I1}>$$

A transição T poderá ser disparada em qualquer instante pertencente ao intervalo $[t_1, t_2]$, permitindo assim o progresso do processo *P*. Uma prefixação do tipo $[t]i; P$ é tratada da mesma forma apresentada acima, já que $[t]i; P = [t, t]i; P$, sendo que a única diferença está no intervalo associado à transição T, que seria $\varphi(T) = [t, t]$.

Para uma prefixação do tipo $[t_1, t_2]a; P$, com $t_2 \geq t_1$ e $a \in Act$, tem-se a Figura 4.4.

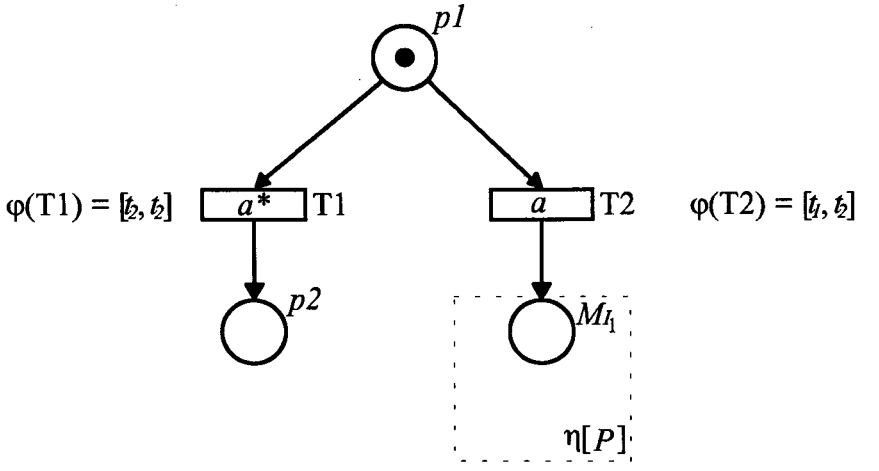


Figura 4.4 - RdP para a prefixação de uma ação observável temporizada

onde:

$\eta[[t_1, t_2]a; P] = <\rho_1 \cup \{p1, p2\}, \tau_1 \cup \{T1, T2\}, \{p1\}, \varphi_1 \cup \{(T1, [t_2, t_2]), (T2, [t_1, t_2])\}>$,

T1 = <{*p1*}, 0, *a**, {*p2*}> e T2 = <{*p1*}, 0, *a*, M_{I1}>

Esta prefixação trata agora da violação temporal. Assim, a transição T1 será disparada no instante t_2, desde que a ação *a* não se encontre disponível. Da mesma forma que anteriormente, uma prefixação do tipo [*t*]*a*; *P* alteraria somente o intervalo associado à transição T2.

Para uma prefixação do tipo *a*; *P*, tem-se a Figura 4.5. (*a* ∈ *Act*)

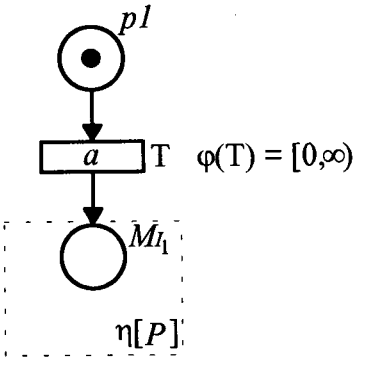


Figura 4.5 - RdP para a prefixação de uma ação observável de ocorrência indefinida

onde:

$$\eta[a; P] = <\rho_1 \cup \{p1\}, \tau_1 \cup \{T\}, \{p1\}, \varphi_1 \cup \{(T, [0, \infty))\}> \text{ e } T = <\{p1\}, 0, a, M_{I1}>$$

A transição T é disparada em qualquer instante no intervalo [0, ∞), pois as ações observáveis não temporizadas são consideradas não urgentes.

4.2.4 Escolha

A Figura 4.6 ilustra como se estabelece a relação entre duas redes, *P* e *Q*, quando o operador de escolha é utilizado, onde *a*, *b* ∈ $Act^{i,\delta}$ e *a**, *b** ∈ $Act^{*,\varepsilon}$.

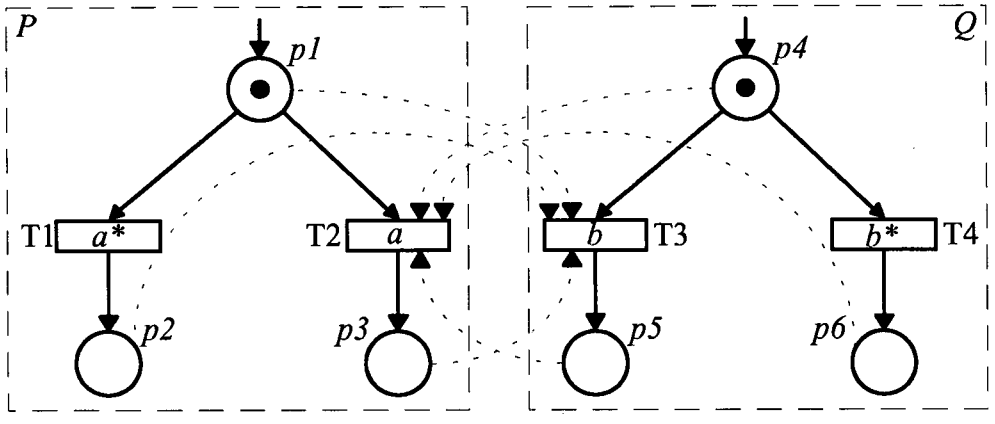


Figura 4.6 - RdP para o processo *P* [] *Q*

O processo de escolha utiliza-se de lugares de entrada opcionais, representados pelas setas pontilhadas, para resolver o não-determinismo existente entre os processos. Portanto, se a transição de um dos processos for disparada, e com sua etiqueta pertencendo ao conjunto $Act^{i, \delta}$, então a escolha é resolvida e a ficha do outro processo é retirada por meio dos novos lugares de entrada opcionais, impedindo o seu progresso. Para as transições com etiquetas pertencentes ao conjunto $Act^{*, \varepsilon}$, não são adicionados mais lugares opcionais, pois estas transições não resolvem a escolha.

Assim, a rede para o operador de escolha é dada por:

$$\eta[P \,[]\, Q] = <\rho_1 \cup \rho_2, \tau_1' \cup \tau_2', M_{I1} \cup M_{I2}, \varphi>$$

onde:

$$\tau_1' = \{<\bullet T, oT \cup \rho_2, label(T), T\bullet> \mid T \in \tau_1 \wedge label(T) \in Act^{i,\delta}\} \cup \{T \in \tau_1 \mid label(T) \in Act^{*,\varepsilon}\}$$

$$\tau_2' = \{<\bullet T, oT \cup \rho_1, label(T), T\bullet> \mid T \in \tau_2 \wedge label(T) \in Act^{i,\delta}\} \cup \{T \in \tau_2 \mid label(T) \in Act^{*,\varepsilon}\}$$

$$\varphi(T): \begin{cases} \varphi i(T) & \text{se } T \in \tau i \wedge label(T) \in Act^{*,\varepsilon} \\ \varphi i(T') & \text{se } T =< \bullet T', \ \circ T' \cup \rho \bar{i}, label(T'), T\bullet' > \ \wedge \ T' \in \tau i \wedge label(T') \in Act^{i,\delta} \\ & \text{onde } i = 1,2 \text{ e } \bar{i} = 2,1 \end{cases}$$

4.2.5 Ocultação

A rede para o operador de ocultação é dada por:

$$\eta[hide\ J\ in\ P] = <\rho_1, \tau, M_{I1}, \varphi>$$

onde:

$$\tau = \{<\bullet T, oT, i, T\bullet> \mid T \in \tau_1 \wedge label(T) \in J\} \cup \{<\bullet T, oT, i, T\bullet> \mid T \in \tau_1 \wedge label(T) = a^* \wedge a^* \in Act^* \wedge a \in J\} \cup \{T \in \tau_1 \mid label(T) \notin J \vee (label(T) = a^* \wedge a^* \in Act^* \wedge a \notin J)\}$$

$$\varphi(T): \begin{cases} \varphi_1(T) & \text{se } T \in \tau_1 \wedge (label(T) \notin J \vee (label(T) = a^* \wedge a^* \in Act^* \wedge a \notin J)) \\ [t_1, t_1] & \text{se } T =< \bullet T', \circ T', i, T\bullet' > \wedge T' \in \tau_1 \wedge label(T') \in J \wedge \varphi(T') = [t_1, t_2] \end{cases}$$

As ações espontâneas, ε, são tratadas da mesma maneira que as ações que não pertencem ao conjunto J. Pode-se observar que as ações pertencentes ao conjunto J se tornam urgentes, sendo

que os intervalos associados às suas transições são obtidos através do limite inferior dos intervalos anteriores.

4.2.6 Composição Paralela

Como foi dito no capítulo anterior, tornou-se necessária a definição de uma ação espontânea, representada por ε, com o objetivo de representar o momento em que cada processo apresente as ações a serem sincronizadas disponíveis, sendo que uma análise abstrata pode ser feita para a verificação de que o funcionamento do operador não se altera. Portanto, a rede para o operador de composição paralela é dada por:

$$\eta[P \,|[J]|\, Q] = <\rho_1 \cup \rho_2 \cup \rho_s \cup \rho_t,\ \tau,\ M_{I1} \cup M_{I2},\ \varphi>$$

onde:

$$\begin{aligned}
\tau = {} & \{T \in (\tau_1 \cup \tau_2) \mid label(T) \notin J \cup \{\delta\} \vee (label(T) = a^* \wedge a^* \in Act^* \wedge a \notin J \cup \{\delta\})\} \cup \\
& \{T \in (\tau_1 \cup \tau_2) \mid label(T) = a^* \wedge a^* \in Act^* \wedge a \in J \wedge min(\varphi(T)) = 0\} \cup \\
& \{<\bullet T1,\ 0,\ \varepsilon,\ \{ps\}> \mid T1 \in \tau_1 \wedge label(T1) \in J \cup \{\delta\} \wedge min(\varphi_1(T1)) \neq 0\} \cup \\
& \{<\bullet T2,\ 0,\ \varepsilon,\ \{pt\}> \mid T2 \in \tau_2 \wedge label(T2) \in J \cup \{\delta\} \wedge min(\varphi_2(T2)) \neq 0\} \cup \\
& \{<\{ps\}, oT1, label(T1), T1\bullet> \mid T1 \in \tau_1 \wedge label(T1) = a^* \wedge a^* \in Act^* \wedge a \in J \wedge min(\varphi_1(T1)) \neq 0\} \cup \\
& \{<\{pt\}, oT2, label(T2), T2\bullet> \mid T2 \in \tau_2 \wedge label(T2) = a^* \wedge a^* \in Act^* \wedge a \in J \wedge min(\varphi_2(T2)) \neq 0\} \cup \\
& \{<\{ps, pt\},\ oT1 \cup oT2,\ label(T1),\ T1\bullet \cup T2\bullet> \mid T1 \in \tau_1 \wedge T2 \in \tau_2 \wedge \\
& \quad label(T1) = label(T2) \in J \wedge min(\varphi_1(T1)) \neq 0 \wedge min(\varphi_2(T2)) \neq 0\} \cup \\
& \{<\{ps\} \cup \bullet T2,\ oT1 \cup oT2,\ label(T1),\ T1\bullet \cup T2\bullet> \mid T1 \in \tau_1 \wedge T2 \in \tau_2 \wedge \\
& \quad label(T1) = label(T2) \in J \wedge min(\varphi_1(T1)) \neq 0 \wedge min(\varphi_2(T2)) = 0\} \cup \\
& \{<\{pt\} \cup \bullet T1,\ oT1 \cup oT2,\ label(T1),\ T1\bullet \cup T2\bullet> \mid T1 \in \tau_1 \wedge T2 \in \tau_2 \wedge \\
& \quad label(T1) = label(T2) \in J \wedge min(\varphi_1(T1)) = 0 \wedge min(\varphi_2(T2)) \neq 0\} \cup \\
& \{<\bullet T1 \cup \bullet T2,\ oT1 \cup oT2,\ label(T1),\ T1\bullet \cup T2\bullet> \mid T1 \in \tau_1 \wedge T2 \in \tau_2 \wedge \\
& \quad label(T1) = label(T2) \in J \cup \{\delta\} \wedge min(\varphi_1(T1)) = 0 \wedge min(\varphi_2(T2)) = 0\}
\end{aligned}$$

$$\varphi(T): \begin{cases}
\varphi i(T) & \text{se } T \in \tau i \wedge (label(T) \notin J \cup \{\delta\} \vee (label(T) = a^* \wedge a^* \in Act^* \wedge a \notin J)) \\
\varphi i(T) & \text{se } T \in \tau i \wedge label(T) = a^* \wedge a^* \in Act^* \wedge a \in J \wedge min(\varphi(T)) = 0 \\
[t_1,\ t_1] & \text{se } label(T) = \varepsilon \wedge T' \in \tau i \wedge \bullet T = \bullet T' \wedge label(T') \in J \wedge \varphi(T) = [t_1,\ t_2] \\
[0,\ 0] & \text{se } label(T) \in J \cup \{\delta\} \\
[t, t] & \text{se } T' \in \tau i \wedge label(T') = label(T) = a^* \wedge a^* \in Act^* \wedge a \in J \wedge min(\varphi(T')) \neq 0 \wedge \\
 & \quad T'' \in \tau i \wedge label(T'') = a \wedge \bullet T' = \bullet T'' \wedge \varphi(T) = [t_1,\ t_2] \wedge t = t_2 - t_1, \text{ onde } i = 1,2
\end{cases}$$

Serão criados dois novos lugares para cada ação a ser sincronizada, ou seja, $\{ps\}$ e $\{pt\}$ representam os pares de novos lugares, sendo que ρs e ρt representam os conjuntos destes novos lugares criados por cada ação a ser sincronizada. O sincronismo de uma ação entre dois processos

pode ser visto graficamente da seguinte forma:

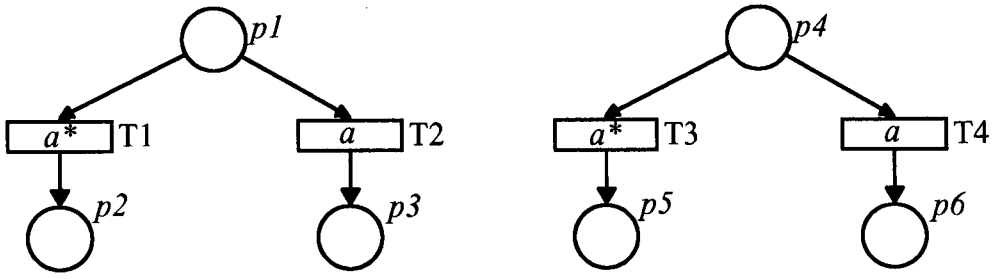


Figura 4.7 - Ações sincronizáveis disponíveis nos processos

Na Figura 4.7, a ação *a* estará disponível nos dois processos nas marcações *p1* e *p4*. Neste exemplo, será utilizada uma prefixação do tipo $[t_1, t_2]a$, para o primeiro processo, e $[t_3, t_4]a$, para o segundo processo.

Portanto, o sincronismo resulta na rede da Figura 4.8.

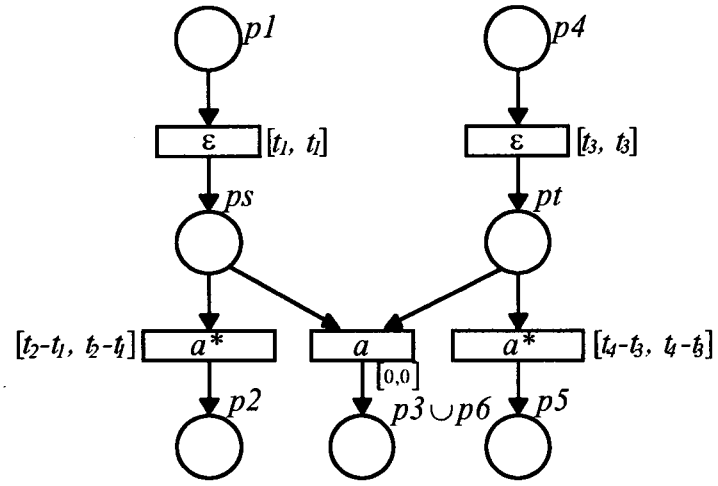


Figura 4.8 - Sincronização dos processos

Observa-se que a condição necessária para que a ação *a* ocorra é que $t_1 + t_3 \leq t_2$ e $t_1 + t_3 \leq t_4$, ou seja, a ação *a* ocorrerá somente se nenhuma das violações temporais ocorrer. A ocorrência de uma ação a ser sincronizada será instantânea, a partir do momento no qual os dois processos estejam dispostos a executá-la. Para as ações $a \in J \cup \{\delta\}$, onde o limite inferior do intervalo de ocorrência é igual a zero, não há necessidade de acrescentar a ação ε, pois as ações estarão disponíveis instantaneamente.

4.2.7 Composição Seqüencial

A Figura 4.9 ilustra como é obtida a rede para o operador de composição seqüencial:

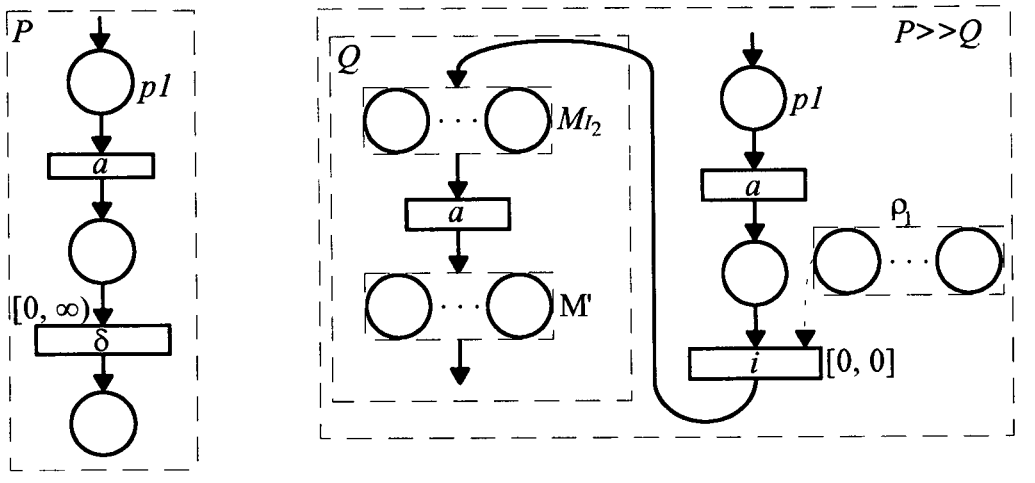


Figura 4.9 - RdP para o processo *P* >> *Q*

Assim, todas as ações δ do primeiro processo são direcionadas para o início do processo de composição seqüencial, encerrando o processo anterior através da adição de todos os seus lugares ao conjunto de lugares de entrada opcionais destas transições. Observa-se que a ação δ torna-se invisível e urgente.

A rede para o operador de composição seqüencial é dada por:

$$\eta[P >> Q] = <\rho_1 \cup \rho_2, \tau, M_{I_1}, \varphi>$$

onde:

$$\tau = \tau_2 \cup \{T \in \tau_1 \mid label(T) \in Act^{i,*,\varepsilon}\}$$
$$\cup \{<\bullet T1, oT1 \cup \rho_1, i, M_{I_2}> \mid T1 \in \tau_1 \wedge label(T1) = \delta\}$$

$$\varphi(T): \begin{cases} \varphi_1(T) & \text{se } T \in \tau_1 \wedge label(T) \in Act^{i,*,\varepsilon} \\ \varphi_2(T) & \text{se } T \in \tau_2 \\ [0, 0] & \text{se } T \in \tau_1 \wedge label(T) = \delta \end{cases}$$

4.2.8 Preempção

Da mesma forma anterior, a Figura 4.10 ilustra a geração da rede para o operador de Preempção, obtida através da relação entre os processos, *P* e *Q*.

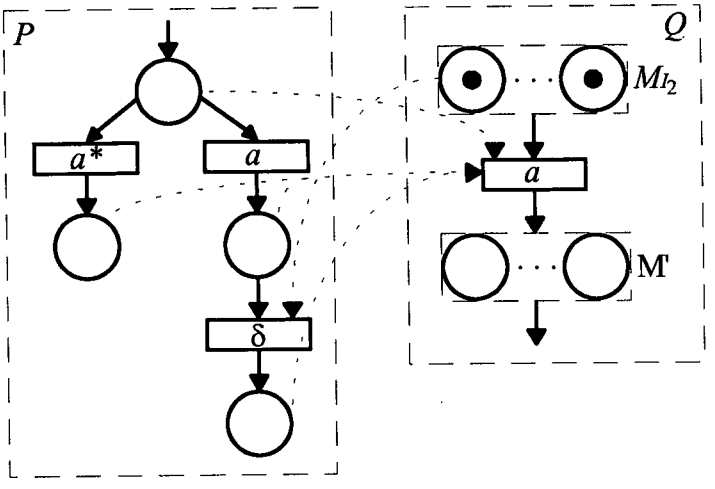


Figura 4.10 - RdP para o processo *P* [> *Q*

Ou seja, as transições do segundo processo possuem como lugares opcionais todos os lugares do primeiro processo, para que caso o segundo processo inicie, o primeiro seja finalizado. O mesmo pode ser dito para o primeiro processo com relação a ação δ, onde a sua ocorrência finaliza o tratamento do operador de preempção, pois suas fichas são retiradas através da adição da marcação inicial do segundo processo ao conjunto de lugares de entrada opcionais das transições que possuem esta ação.

A rede para o operador de preempção é dada por:

$$\eta[P\ [>\ Q] = <\rho_1 \cup \rho_2, \tau, M_{I_1} \cup M_{I_2}, \varphi>$$

onde:

$$\begin{aligned}\tau = {} & \{T \in \tau_1 \mid label(T) \neq \delta\} \\ & \cup \{T \in \tau_2 \mid \bullet T \neq M_{I_2}\} \\ & \cup \{<\bullet T, oT \cup \rho_1, label(T), T\bullet> \mid T \in \tau_2 \wedge \bullet T = M_{I_2}\} \\ & \cup \{<\bullet T, oT \cup M_{I_2}, \delta, T\bullet> \mid T \in \tau_1 \wedge label(T) = \delta\}\end{aligned}$$

$$\varphi(T): \begin{cases} \varphi_1(T) & \text{se } T \in \tau_1 \wedge label(T) \neq \delta \\ \varphi_2(T) & \text{se } T \in \tau_2 \wedge \bullet T \neq M_{I2} \\ \varphi_1(T') & \text{se } T =< \bullet T', \circ T' \cup M_{I2}, \delta, T'\bullet > \wedge T' \in \tau 1 \wedge label(T') = \delta \\ \varphi_2(T') & \text{se } T =< \bullet T', \circ T' \cup \rho_1, label(T'), T'\bullet > \wedge T' \in \tau_2 \wedge \bullet T' = M_{I2} \end{cases}$$

4.2.9 Preempção Temporal

A Figura 4.11 ilustra como é obtido a rede para o operador de Preempção Temporal:

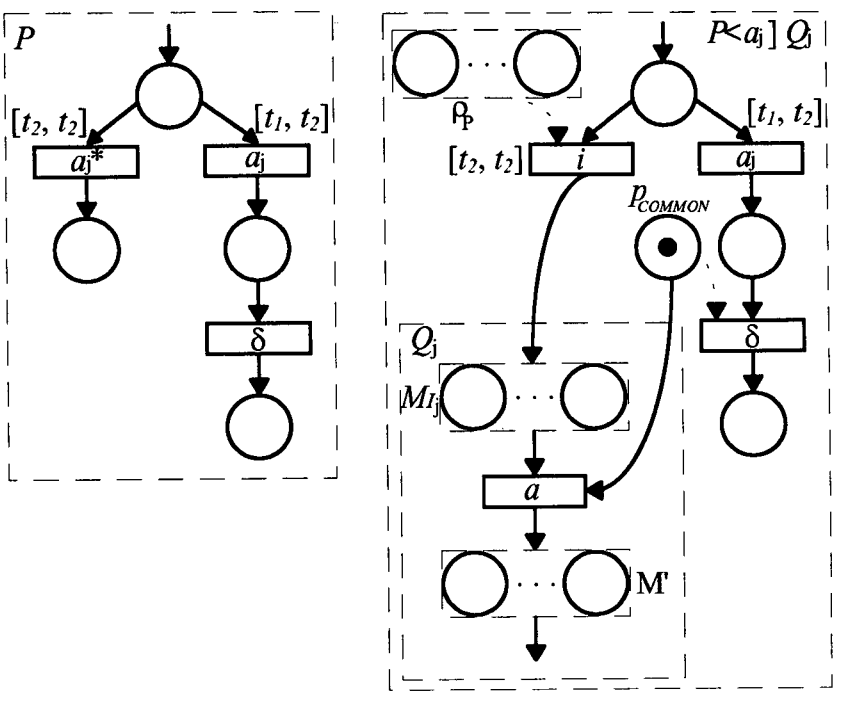


Figura 4.11 - RdP para o processo *P* <*J*] {U$_j$(*a*$_j$: *Q*j)}

Neste operador, é criado um novo lugar (p_{COMMON}) que representa o lugar de habilitação do tratamento de violações temporais. Assim, se no processo a ser tratado ocorrer uma transição com a etiqueta igual a δ, encerra-se este tratamento através da retirada da ficha existente em p_{COMMON}, seguindo a semântica da linguagem definida em [Camargo95]. Da mesma maneira, se ocorrer uma violação temporal, a^*, onde $a \in J$, então é lançado o processo de tratamento, encerrando o processo tratado.

A rede para este operador é dada por:

Seja agora $\eta[P] = <\rho_p, \tau_p, M_{I_p}, \varphi_p>$ e $\eta[Q_j] = <\rho_j, \tau_j, M_{I_j}, \varphi j>$, j = 1, ..., k

$\eta[P <J] \{U_j(a_j: Qj)\}] = <\rho_p \cup \rho_1 \cup \rho_2 \cup ... \cup \rho_j \cup \{p_{COMMON}\}, \tau, M_{I_p} \cup \{p_{COMMON}\}, \varphi>$

onde:

$\tau = \{T \in \tau_p \mid label(T) \neq \delta \vee (a_j^* \in Act^* \wedge (label(T) \neq a_j^* \vee (label(T) = a_j^* \wedge a_j \notin J)))\}$

$\cup \{T \in \tau_j \mid \bullet T \not\subset M_{I_j}\}$

$\cup \{<\bullet T \cup \{p_{COMMON}\}, oT, label(T), T\bullet> \mid T \in \tau_j \wedge \bullet T \subseteq M_{I_j}\}$

$\cup \{<\bullet T, oT \cup \rho_p, i, M_{I_j}> \mid T \in \tau_p \wedge label(T) = a_j^* \wedge a_j^* \in Act^* \wedge a_j \in J\}$

$\cup \{<\bullet T, oT \cup \{p_{COMMON}\}, label(T), T\bullet> \mid T \in \tau_p \wedge label(T) = \delta\}$

$\varphi = \varphi_P \cup \varphi_1 \cup ... \cup \varphi_j$

4.2.10 Instanciação de Processos

O operador de instanciação é definido a partir da função de troca de nomes de ações, representada por $\phi = [a1/a1', ..., an/an']$, em que para todo j, a função aj' pode tornar-se aj. Assim, a rede para este operador é dada por:

$$\eta[P\ [a1/a1', ..., an/an']] = <\rho_1, \tau, M_{I1}, \varphi>$$

onde:

$$\tau = \{<\bullet T, oT, aj, T\bullet> \mid T \in \tau_1 \wedge label(T) = aj' \wedge aj/aj' \in \phi \wedge \{aj, aj'\} \subseteq Act \cup Act^*, j = 1, ..., n\} \cup \{T \in \tau_1 \mid label(T) \notin \{a1', a1', ..., an'\}\}$$

$$\varphi(T): \begin{cases} \varphi(T) & \text{se } T \in \tau_1 \wedge label(T) \notin \{a1', a2', ..., an'\} \\ \varphi(T') & \text{se } T = <\bullet T', \circ T', aj, T'\bullet> \wedge T' \in \tau_1 \wedge label(T') = aj' \wedge aj / aj' \in \phi \wedge \{aj, aj'\} \subseteq Act \cup Act^* \end{cases}$$

4.3 UM EXEMPLO ILUSTRATIVO

Nesta seção, é apresentado um dos exemplos do capítulo anterior (Login Simplificado), para tornar mais claro o entendimento do método descrito acima. A especificação do exemplo é dada abaixo:

Login$(p, n, v) := p$; $(v; S\ [] [0, l]n; Login(p, n, v)) <n]$ $\{n: Login(p, n, v)\}$

Para a construção da rede, o processo é dividido da mesma maneira feita para a construção do GTE, afim de tratar de cada um dos operadores. Assim:

$Login := p; Q_l$

A rede para *Login* é dada por:

$\eta[Login] = <\rho, \tau, M_I, \varphi>$ e

$\eta[Q_l] = <\rho_1, \tau_1, M_{I1}, \varphi_1>$

Graficamente, tem-se a Figura 4.12.

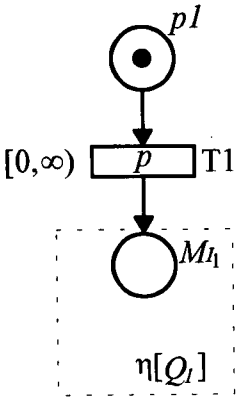


Figura 4.12 - RdP da prefixação da ação *p*

onde:

$\eta[p; Q_1] = <\rho_1 \cup \{p1\}, \tau_1 \cup \{T1\}, \{p1\}, \varphi_1 \cup \{(T1, [0, \infty))\}>$ e $T1 = <\{p1\}, 0, p, M_{I1}>$

Mas, $Q_1 := (Q_2 [] Q_3) <n] \{n: Login\}$

onde:

$Q_2 := v; S$

$\eta[Q_2] = <\rho_2, \tau_2, M_{I2}, \varphi_2>$

e $\eta[S] = <\rho_s, \tau_s, M_{Is}, \varphi_s>$

Graficamente, tem-se a figura 4.13.

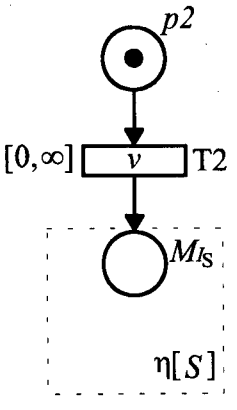


Figura 4.13 - RdP da prefixação da ação *v*

onde:

$\eta[Q_2] = <\rho_s \cup \{p2\}, \tau_s \cup \{T2\}, \{p2\}, \varphi_s \cup \{(T2, [0, \infty))\}>$ e

$T2 = <\{p2\}, 0, v, M_{Is}>$

$Q_3 := [0, l]n$; *Login*

Graficamente, tem-se a Figura 4.14.

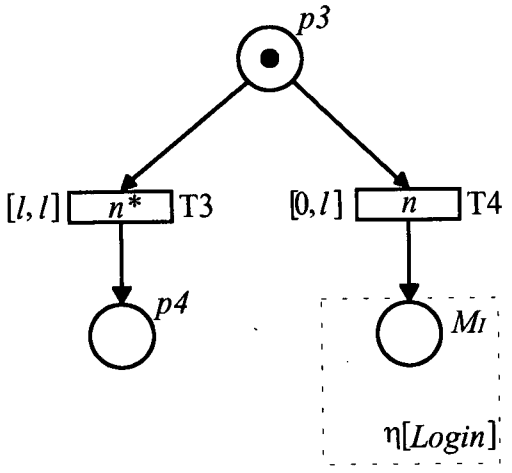


Figura 4.14 - RdP da prefixação da ação *n*

onde:

$\eta[Q_3] = <\rho \cup \{p3, p4\}, \tau \cup \{T3, T4\}, \{p3\}, \varphi \cup \{(T3,[l, l]),(T4,[0, l])\}>$,

$T3 = <\{p3\}, 0, n^*, \{p4\}>$ e $T4 = <\{p3\}, 0, n, Ml>$

$Q_4 := Q_2 [] Q_3$

A rede para o processo Q_4 é obtida através da adição de novos lugares de entrada opcionais às transições de Q_2 e Q_3 (setas pontilhadas), permitindo assim o encerramento de um processo quando o outro prosseguir, seguindo a semântica do operador de escolha. Portanto, graficamente tem-se a Figura 4.15:

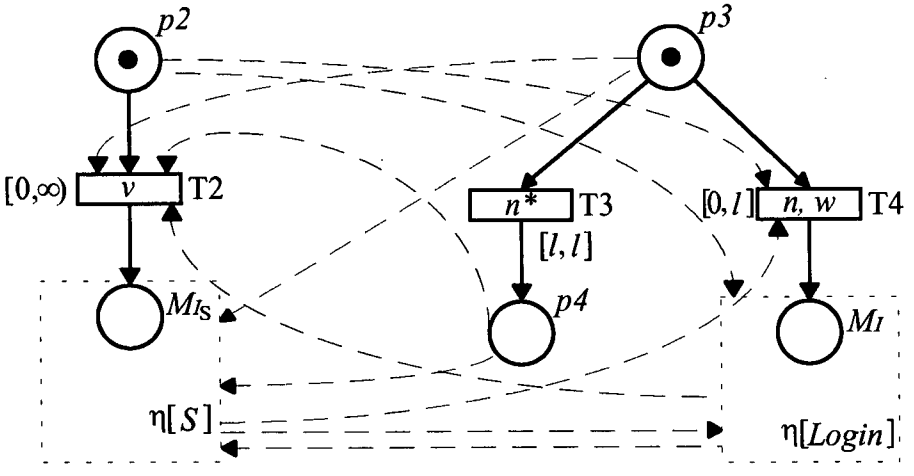


Figura 4.15 - RdP do processo Q_4

onde:

$\eta[Q_4] = <\rho \cup \rho_s \cup \{p2, p3, p4\}, \tau_4, \{p2, p3\}, \varphi_4 \cup \varphi_4' \cup \varphi_4''>$ e

$\tau_4 =$ { T2, T3, T4} $\cup$

$\{T \in \tau_s \mid label(T) \in Act^{*,\varepsilon}\} \cup$

$\{<\bullet T, oT \cup \rho \cup \{p3, p4\}, label(T), T\bullet> \mid T \in \tau_s \wedge label(T) \in Act^{i,\delta}\} \cup$

$\{T \in \tau \mid label(T) \in Act^{*,\varepsilon}\} \cup$

$\{<\bullet T, oT \cup \rho_s \cup \{p2\}, label(T), T\bullet> \mid T \in \tau \wedge label(T) \in Act^{i,\delta}\},$

$T2 = <\{p2\}, \{p3, p4\} \cup \rho, v, M_{is}>,$

$T3 = <\{p3\}, 0, n^*, \{p4\}>$ e

$T4 = <\{p3\}, \{p2\} \cup \rho_s, n, M_I>$

$\varphi_4 = \{(<T2, [0, \infty)), (T3, [l, l]), (T4, [0, l])\}$

$$\varphi_4'(T): \begin{cases} \varphi_s(T) & \text{se } T \in \tau_s \wedge label(T) \in Act^{*,\varepsilon} \\ \varphi_s(T') & \text{se } T =< \bullet T', \circ T' \cup \rho \cup \{p3, p4\}, label(T'), T'\bullet > \wedge T' \in \tau_s \wedge label(T') \in Act^{i,\delta} \end{cases}$$

$$\varphi_4''(T): \begin{cases} \varphi(T) & \text{se } T \in \tau \wedge label(T) \in Act^{*,\varepsilon} \\ \varphi[T'] & \text{se } T =< \bullet T', \circ T' \cup \rho_s \cup \{p2\}, label(T'), T'\bullet > \wedge T' \in \tau \wedge label(T') \in Act^{i,\delta} \end{cases}$$

$Q_1 := Q_4 <n] \{n: Login\}$

Para se obter a rede para o processo Q_1 , a partir de Q_4, basta redirecionar a transição etiquetada com a ação n^* para o início do processo *Login*, tratando assim da violação temporal da ação n. Portanto, a rede para o processo Q_1 é dada pela Figura 4.16:

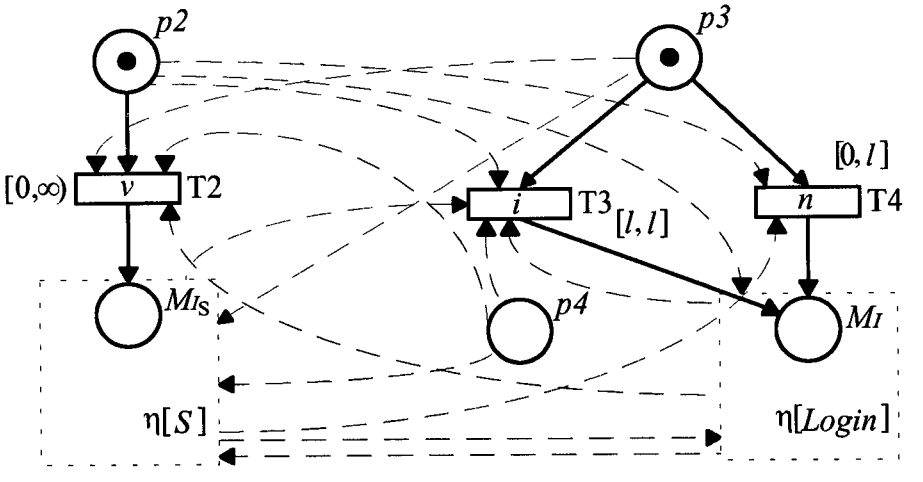


Figura 4.16 - RdP do processo Q_1

onde:

$\eta[Q_1] = <\rho \cup \rho_s \cup \{p2, p3, p4\}, \tau_1, \{p2, p3\}, \varphi_1 \cup \varphi_1' \cup \varphi_1''>$ e

$\tau_1 =$ $\{T2, T3, T4\} \cup$

$\{T \in \tau_s \mid label(T) \in Act^{*,\varepsilon} \wedge label(T) \neq n^*\} \cup$

$\{<\bullet T, \rho \cup \rho_s \cup \{p2, p3, p4\}, i, M_I> \mid T \in \tau_s \wedge label(T) = n^*\} \cup$

$\{<\bullet T, oT \cup \rho \cup \{p3, p4\}, label(T), T\bullet> \mid T \in \tau_s \wedge label(T) \in Act^{i,\delta}\} \cup$

$\{T \in \tau \mid label(T) \in Act^{*,\varepsilon} \wedge label(T) \neq n^*\} \cup$

$\{<\bullet T, \rho \cup \rho_s \cup \{p2, p3, p4\}, i, M_I> \mid T \in \tau \wedge label(T) = n^*\}$

$\{<\bullet T, oT \cup \rho_s \cup \{p2\}, label(T), T\bullet> \mid T \in \tau \wedge label(T) \in Act^{i,\delta}\},$

$T2 = <\{p2\}, \{p3, p4\} \cup \rho, v, M_{Is}>,$

$T3 = <\{p3\}, \rho \cup \rho s \cup \{p2, p4\}, i, M_I>$ e

$T4 = <\{p3\}, \{p2\} \cup \rho s, n, M_I>$

$\varphi 1 = \{(T2, [0, \infty)), (T3, [l, l]), (T4, [0, l])\}$

$$\varphi_1'(T): \begin{cases} \varphi_s(T) & \text{se } T \in \tau_s \wedge label(T) \in Act^{*,\varepsilon} \wedge label(T) \neq n^* \\ \varphi_s(T') & \text{se } T =< \bullet T', \rho \cup \rho_s \cup \{p2, p3, p4\}, i, T\bullet > \wedge T' \in \tau_s \wedge label(T') = n^* \\ \varphi_s(T') & \text{se } T =< \bullet T', oT' \cup \rho \cup \{p3, p4\}, label(T'), T'\bullet > \wedge T' \in \tau_s \wedge label(T') \in Act^{i,\delta} \end{cases}$$

$$\varphi_1''(T): \begin{cases} \varphi(T) & \text{se } T \in \tau \wedge label(T) \in Act^{*,\varepsilon} \wedge label(T) \neq n^* \\ \varphi(T') & \text{se } T =< \bullet T', \rho \cup \rho_s \cup \{p2, p3, p4\}, i, T'\bullet > \wedge T' \in \tau \wedge label(T') = n^* \\ \varphi(T') & \text{se } T =< \bullet T', oT' \cup \rho_s \cup \{p2\}, label(T'), T'\bullet > \wedge T' \in \tau \wedge label(T') \in Act^{i,\delta} \end{cases}$$

Resta agora acrescentar apenas a prefixação da ação *p*. Assim, a rede resultante que representa o processo de *Login* é dada pela Figura 4.17:

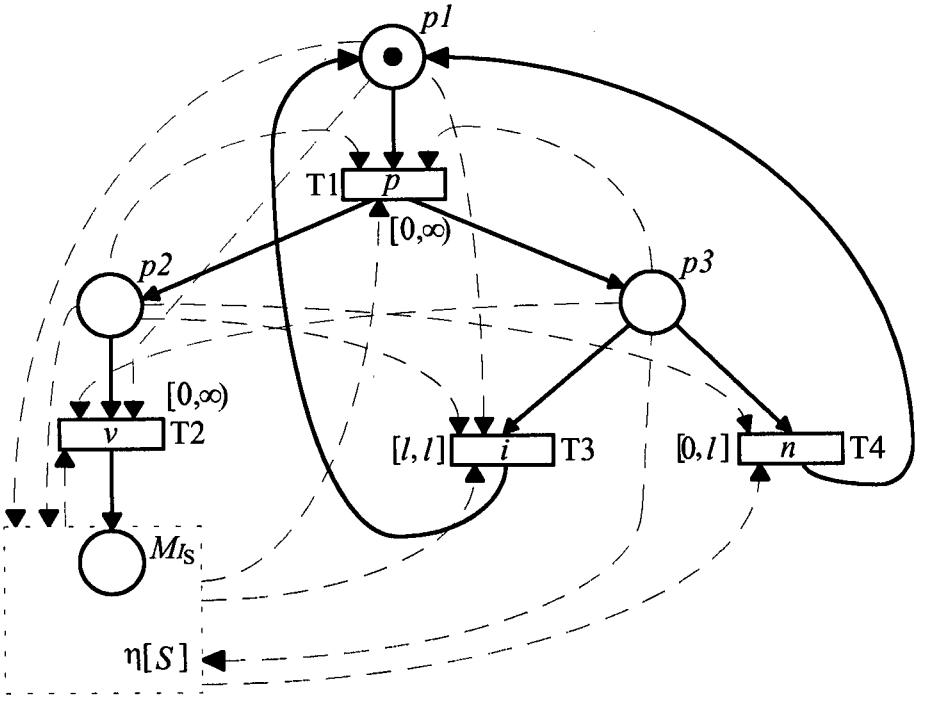


Figura 4.17 - RdP do processo *Login* Simplificado

onde:

$\eta[Login] = <\rho_s \cup \{p1, p2, p3\}, \tau, \{p1\}, \varphi \cup \varphi'>$ e

$\tau =$ $\{T1, T2, T3, T4\} \cup$

$\{T \in \tau_s \mid label(T) \in Act^{*,\varepsilon} \wedge label(T) \neq n^*\} \cup$

$\{<\bullet T, \rho_s \cup \{p1, p2, p3\}, i, \{p1\}> \mid T \in \tau_s \wedge label(T) = n^*\} \cup$

$\{<\bullet T, oT \cup \rho_s \cup \{p1, p2, p3\}, label(T), T\bullet> \mid T \in \tau_s \wedge label(T) \in Act^{i,\delta}\},$

$T1 = <\{p1\}, \rho_s \cup \{p2, p3\}, p, \{p2, p3\}>$, $T2 = <\{p2\}, \{p1, p3\} \cup \rho s, v, M_{Is}>$,

$T3 = <\{p3\}, \rho_s \cup \{p1, p2\}, i, \{p1\}>$ e $T4 = <\{p3\}, \{p2\} \cup \rho s, n, \{p1\}>$

$\varphi =$ $\{(T1, [0, \infty)), (T2, [0, \infty)), (T3, [0,0]), (T4, [0, l])\}$

$$\varphi'(T): \begin{cases} \varphi_s(T) & \text{se } T \in \tau_s \wedge label(T) \in Act^{*,\varepsilon} \wedge label(T) \neq n^* \\ \varphi_s(T') & \text{se } T =< \bullet T', \rho_s \cup \{p1, p2, p3\}, i, T\bullet > \wedge T' \in \tau_s \wedge label(T') = n^* \\ \varphi_s(T') & \text{se } T =< \bullet T', \rho_s \cup \{p1, p2, p3\}, label(T'), T\bullet > \wedge T' \in \tau_s \wedge label(T') \in Act^{i,\delta} \end{cases}$$

A partir da rede apresentada acima, faz-se as seguintes observações:

→ As marcações acessíveis da rede são: $\{p1\}$, $\{p2, p3\}$ e M_{is}.

→ A partir da marcação inicial, $\{p1\}$, T1 é a única transição possível, que representa o início do procedimento, onde o sistema envia um *prompt* de *Login*, *p*.

→ Após o *prompt* de *Login* (marcação $\{p2, p3\}$), existem três transições possíveis: a transição T2, que representa a entrada de um nome válido de usuário, dentro do limite de tempo (*l*); a transição T3, que representa o final do limite de tempo para a entrada de um nome; e a transição T4, que representa a entrada de um nome inválido.

→ Caso as transições T3 e T4 sejam disparadas, $\{p1\}$ será a próxima marcação da rede, lançando novamente o procedimento.

→ O disparo da transição T2 leva à marcação M_{is}, dando início a sessão *S*, após a entrada de um nome válido.

Através das observações acima, mostra-se que o processo de *Login* Simplificado é bem representado pela rede obtida.

4.4 DIRETIVAS PARA A CONSTRUÇÃO DO GRAFO TEMPORIZADO A PARTIR DA REDE DE PETRI ESTENDIDA

Nesta seção, é apresentada a maneira de se obter o Grafo Temporizado de uma especificação a partir da Rede de Petri estendida.

Seja $N = <\rho, \tau, M_I, \varphi>$ a Rede de Petri de uma especificação descrita em RT-LOTOS. O Grafo Temporizado $G = < S, H, E, S_I, \delta>$, construído a partir de N é tal que:

- O conjunto de vértices, S, é o conjunto das marcações acessíveis, M, a partir da marcação inicial, M_I. Cada marcação representa um vértice do modelo.
- O vértice inicial, S_I, é a marcação inicial, M_I.
- O conjunto de relógios, H, é o conjunto de relógios associados às transições $T \in \tau$ (*clock*(T)), pois para cada transição T é associado um relógio, desde que $\varphi(T) = [t_1, t_2]$ e $t_1 = t_2$, ou ainda que $\varphi(T) \neq [0, \infty)$.
- A condição de atividade de uma marcação é a conjunção das condições dos relógios, na forma *clock*(T) $\leq t_2$, para todas as transições, T, acessíveis a partir desta marcação, tais que $\varphi(T) = [t_1, t_2]$ e $t_1 = t_2$, ou que $\varphi(T) \neq [0, \infty)$.Se a marcação não possui tais transições, então a condição de atividade é *true*, significando que a permanência naquele estado é permitida de maneira indefinida.
- Seja $e = <M, label(T), \psi, H', M'>$ o arco que leva M para M', produzido pela transição T:
  - ⇒ Os vértices inicial e final são, respectivamente, as marcações M e M'.
  - ⇒ A ação de e é *label*(T).
  - ⇒ A condição de e (ψ) é:
    - ♦ Se $\varphi(T) = [t_1, t_2]$ e $t_1 = t_2$, então o fator é *clock*(T) = t_2, indicando que o arco será disparado somente no instante em que o relógio T possuir o valor t_2.
    - ♦ Se $\varphi(T) = [0, \infty)$, então o fator é true, ou seja, o arco pode ser disparado a qualquer momento.
    - ♦ Se $\varphi(T) = [t_1, t_2]$ e $t_1 < t_2$, então o fator é $t_1 \leq$ *clock*(T) $\leq t_2$, indicando um intervalo onde o arco pode ser disparado.
  - ⇒ O conjunto dos relógios a serem inicializados (H') é dado por:
    $H' = \{clock(T') \mid {}^\bullet T' \subseteq M' \wedge {}^\bullet T' \not\subseteq M\}$
    Representando o conjunto dos relógios das transições que são habilitadas a partir de M', e que não são habilitadas a partir de M.

É possível se fazer uma otimização sobre o número de relógios do Grafo Temporizado. Esta é feita sobre os relógios das transições que possuem $\varphi(T) = [t_1, t_2]$ e $t_1 = t_2 = 0$, indicando uma ação imediata. Assim, para todas as transições que satisfazem esta condição, será adotado o mesmo relógio, pois ele sempre será utilizado somente na representação de ações imediatas.

Para exemplificar a utilização das considerações apresentadas acima, faz-se a relação entre o resultado obtido no capítulo anterior, ou seja, o Grafo Temporizado do procedimento de *Login* Simplificado, com a rede de Petri estendida, apresentados na Figura 4.18:

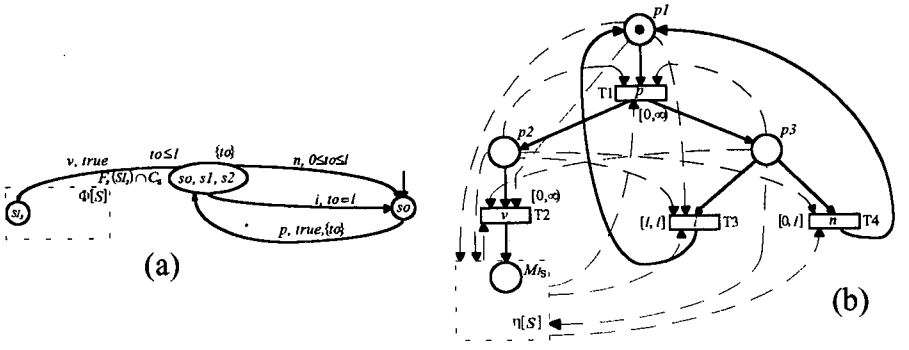


Figura 4.18 - Resultados do processo de *Login* Simplificado: (a) Grafo Temporizado; (b) Rede de Petri

Assim, é fácil de se observar:

→ Como cada marcação acessível da rede é relacionada a um vértice do grafo, tem-se:

Marcações da Rede	Vértices do Grafo
$\{p1\}$	(so)
$\{p2, p3\}$	$(so, s1, s2)$
M_{I_S}	SI_S

→ O vértice inicial do grafo, *so*, relaciona-se com a marcação inicial da rede, $\{p1\}$.

→ O conjunto de relógios, *H*, é constituído de dois relógios (t_0 e t_1), associados às transições T3 e T4 da rede.

→ As condições de atividades associadas às marcações são:

Marcação da Rede	Condição de Atividade
$\{p1\}$	*true*
$\{p2, p3\}$	$t_0 \leq l \wedge t_1 \leq l$

O Grafo Temporizado da Figura 4.18 possui apenas um relógio. Observa-se então que uma simplificação pode ser feita sobre os relógios associados às transições, pois possuem as mesmas características com relação às condições de atividade.

→ São associados arcos a todas as transições da rede, pois estas podem ser disparadas através das marcações acessíveis. Assim, os arcos são:

$$\text{T1} \Rightarrow <\{p1\}, p, true, \{t_0, t_1\}, \{p2, p3\}>$$
$$\text{T2} \Rightarrow <\{p2, p3\}, v, true, H_{MI_S}, M_{I_S}>$$
$$\text{T3} \Rightarrow <\{p2, p3\}, i, t_0 = l, 0, \{p1\}>$$
$$\text{T4} \Rightarrow <\{p2, p3\}, n, 0 \leq t_1 \leq l, 0, \{p1\}>$$

que correspondem aos arcos do Grafo Temporizado, onde H_{MI_S} representa o conjunto dos relógios das transições habilitadas a partir da marcação M_{i_S}, e que não são habilitados a partir da marcação $\{p2, p3\}$. Portanto, observa-se que o Grafo Temporizado que poderá ser gerado a partir das considerações feitas sobre a rede de Petri estendida, será o mesmo que o Grafo Temporizado obtido diretamente no capítulo anterior.

4.5 IMPLEMENTAÇÃO DA FERRAMENTA DE TRADUÇÃO RT-LOTOS / GRAFO TEMPORIZADO

Nesta seção, são apresentados os passos de desenvolvimento do compilador elaborado neste trabalho, que implementa o método de tradução de uma especificação escrita em RT-LOTOS para grafos temporizados, descritos nas seções anteriores. A arquitetura deste compilador pode ser ilustrada através da Figura 4.19.

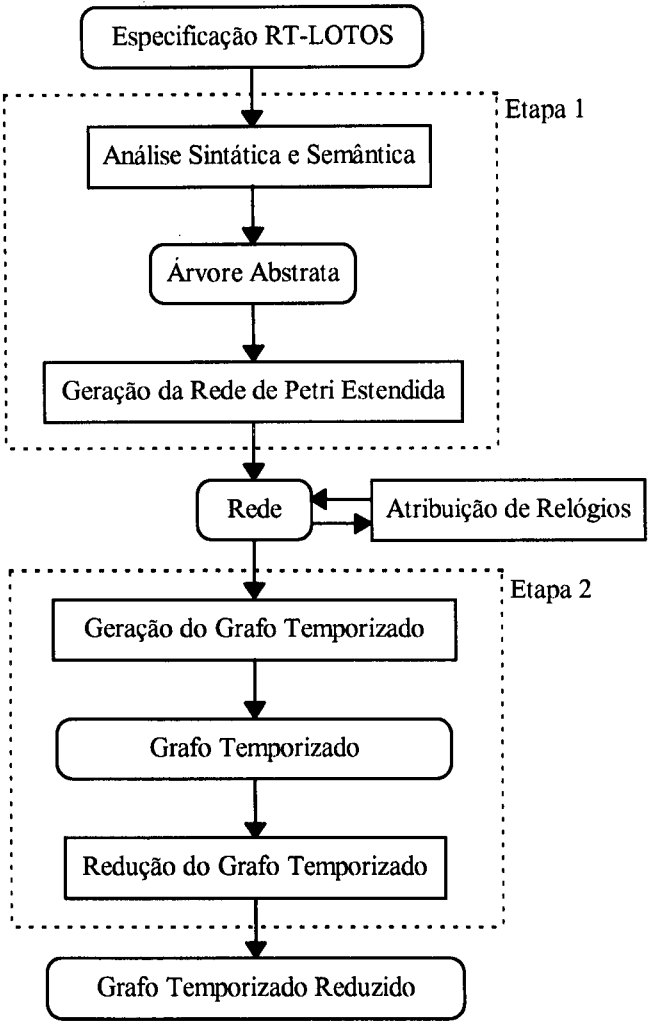


Figura 4.19 - Tradutor RT-LOTOS / Grafo Temporizado

Os resultados gerados pelo compilador são: o Grafo Temporizado finito que representa a especificação, a Rede de Petri correspondente e o relato de possíveis erros, que podem ser tratados pelo compilador, dependendo do nível no qual este erro pertence.

4.5.1 Análise Sintática e Semântica

Esta é a primeira fase do compilador, que consiste basicamente na verificação da especificação com relação à gramática da linguagem, apresentada no Anexo B.

É utilizada como ferramenta auxiliar o sistema SYNTAX [BD88, BD89], descrito com mais detalhes no Anexo D. Com esta ferramenta, e utilizando a teoria de Sintaxe Abstrata [Meyer90], é construída a árvore abstrata da especificação.

SYNTAX 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 escrito em C que é o "esqueleto" do programa de análise semântica, que também é base para a construção da Rede de Petri Estendida.

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

4.5.2 Geração da Rede de Petri Estendida

O compilador gera a Rede de Petri a partir da Árvore Abstrata da especificação, que contém apenas as informações essenciais à montagem. O algoritmo de geração percorre esta árvore, a partir da raiz, procurando sempre o nó filho localizado mais à esquerda. Cada nó é visitado duas vezes: uma no momento em que ele é acessado a partir do seu nó pai, e outra quando é finalizada o acesso aos seus nós filhos.

Desta forma, o compilador identifica cada nó (operando ou operador), e utiliza duas pilhas do tipo *LIFO* (*Last Input is First Output*), que armazenam a ordem de execução dos operadores e os operandos, colocados na forma de resultados (redes), para serem consultados posteriormente a cada segunda visita de um nó contendo um operador.

Para exemplificar, a seguinte especificação é analisada:

Exemplo := *a*; *stop* [] *b*; *Exemplo*

A sua árvore abstrata seria da forma apresentada na Figura 4.20:

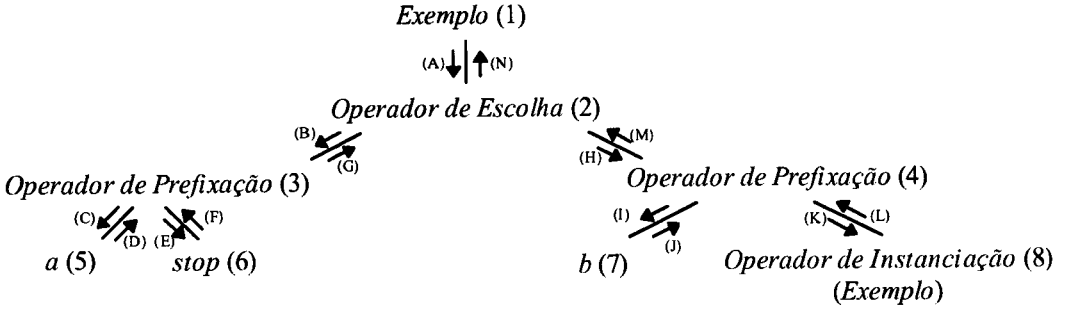


Figura 4.20 - Árvore Abstrata da especificação *Exemplo*

Partindo do nó raiz (1), o algoritmo visita o nó (2) pela primeira vez (A), armazenando na pilha de operadores o operador de escolha. Em seguida, visita-se o nó filho mais a esquerda do nó (2), ou seja, o nó (3), pela primeira vez (B), armazenando novamente na pilha de operadores o operador de prefixação. Na visita ao nó (5), gera-se a rede equivalente ao operando (C), armazenando-o em seguida na pilha de resultados (D). Da mesma forma, faz-se a visita ao nó (6), onde o seu resultado é também armazenado na pilha correspondente (E). Na segunda visita ao nó (3), através de (F), aplica-se o operador de prefixação sobre os dois operandos contido no topo da pilha de resultados, sendo a rede resultante armazenada na pilha de resultados, e o operador utilizado retirado da pilha de operadores. Desta maneira, o tradutor prossegue até retornar ao nó raiz (1), através de (N), sendo que o resultado final se encontra no topo da pilha de resultados.

4.5.3 Atribuição de Relógios

A atribuição de relógios é feita seguindo as condições colocadas na seção 4.4, ou seja, apenas as transições onde $\varphi(T) = [t_1, t_2]$, $t_2 \neq \infty$, $t_1 \neq t_2$ ou $t_1 = t_2 \neq 0$, serão associadas a relógios distintos.

O algoritmo faz uma otimização sobre todas as transições que apresentarem $t_1 = t_2 = 0$, pois para tais transições é associado apenas um único relógio (global), que representará sempre o mesmo comportamento durante o percorrer da rede, ou seja, ações de ocorrência imediata.

A associação de condições de atividade aos vértices do Grafo Temporizado também envolvem o conjunto dos relógios atribuídos às transições da rede. No entanto, esta associação é efetuada na fase de geração do Grafo Temporizado.

4.5.4 Geração do Grafo Temporizado

Com a conclusão da geração da rede de Petri e atribuição de relógios às transições, resta realizar o mapeamento da rede para grafos temporizados. O algoritmo que permite simular a rede de Petri estendida e gerar o Grafo Temporizado, apresentado em [Yovine93], pode ser descrito, resumidamente, da seguinte maneira:

```
begin
     S+ := 0
     S- := {MI}
     E  := 0
     while S- ≠ 0
          let M ∈ S-
          S+ := S+ ∪ {M}
          S- := S- - {M}
          build_condition(M)
          forall T, M' : <M, T, M'>
               e := build_arc(M, T, M')
               E := E ∪ {e}
               if M' ∉ S+ ∪ S-
               then S- := S- ∪ {M'}
          end
     end
end
```

O algoritmo inicia a montagem a partir da marcação inicial, M_I, onde S^+ representa o conjunto de marcações acessíveis visitadas, e S^- o conjunto de marcações acessíveis que ainda não foram visitadas. Assim, o algoritmo grava todas as marcações acessíveis a partir de M_I, finalizando a tarefa somente quando o conjunto de marcações a serem visitadas, S^-, ficar vazio.

Para cada marcação, são identificadas todas as transições sensibilizadas pela mesma, gerando assim os arcos que partem do vértice relacionado à marcação. São analisadas também as novas marcações geradas pelas transições, verificando se estas marcações já foram visitadas, através do conjunto S^+, armazenando as marcações que ainda não foram visitadas no conjunto S^-.

As condições relacionadas aos vértices são obtidas a partir das transições disparadas pela marcação analisada, M, observando os relógios e os limites associados a cada transição, colocando assim as restrições necessárias para cada relógio.

Os relógios a serem inicializados por cada arco são obtidos a partir das marcações finais geradas pelas transições disparadas pela marcação *M*, analisando todas as transições sensibilizadas por estas marcações.

4.5.5 Redução do Grafo Temporizado

Esta fase tem como objetivo a eliminação de vértices repetidos, gerados pelo método apresentado, já que no momento da geração, ocorre a identificação distinta de estados com as mesmas características, ou seja, condição de atividade e arcos. Mas, para que dois estados sejam semelhantes, é necessário que a relação deles com o conjunto de relógios seja a mesma.

4.6 CONCLUSÕES

Neste capítulo, foi apresentado o método utilizado para a implementação do compilador que constrói um Grafo Temporizado a partir de uma especificação escrita em RT-LOTOS. Foi apresentada uma etapa intermediária à construção do Grafo Temporizado, visando o melhor desempenho da compilação.

Assim, foi utilizada como etapa intermediária a construção de uma Rede de Petri com algumas extensões, sendo que o Grafo Temporizado é obtido simplesmente percorrendo as marcações desta rede. Esta rede permite representar uma especificação de maneira mais compacta, simplificando todo o mecanismo de mapeamento.

O tradutor RT-LOTOS / Grafos Temporizados possui um programa fonte escrito em linguagem C, com aproximadamente 9000 linhas de código. O funcionamento do compilador será demonstrado no próximo capítulo, através de exemplos de aplicações.

CAPÍTULO 5

VALIDANDO A METODOLOGIA E A FERRAMENTA DE VERIFICAÇÃO

Neste capítulo, são apresentadas algumas aplicações para ilustrar o método de especificação e verificação descrito nos capítulos anteriores, testando também a ferramenta desenvolvida para realizar a tradução de RT-LOTOS para Grafos Temporizados. Nos dois primeiros exemplos, procura-se, em particular, verificar o funcionamento deste tradutor, a partir de uma comparação com os resultados obtidos nas verificações realizadas em [DOY94, Yovine93], onde foi utilizada uma metodologia semelhante, para especificações escritas, respectivamente, em ET-LOTOS [LL94] e ATP [Nicollin92]. Serão discutidas no final deste capítulo as características da ferramenta e da metodologia, destacando suas vantagens e limitações. As verificações das restrições temporais são feitas utilizando a ferramenta KRONOS [OY93], descrita no Anexo B.

5.1 PROTOCOLO TICK-TOCK

O exemplo a seguir foi descrito em [LLD94] e verificado também em [DOY94]. Trata-se de um protocolo de comunicação, composto de três entidades principais: *sender, receiver, service*. A verificação será feita somente sobre a especificação de *service*, com o objetivo de simplificar o problema, já que ela possui grande parte das restrições temporais essenciais ao protocolo. A especificação proposta neste trabalho seguirá a mesma abordagem encontrada em [DOY94], adotando-se a linguagem RT-LOTOS.

5.1.1 Descrição informal da entidade *service*

A entidade *service* interage com *sender* e *receiver* através de seus respectivos pontos de comunicação, Ss-SAP e Sr-SAP. A Figura 5.1 ilustra de forma global o protocolo.

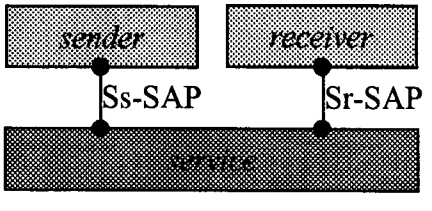


Figura 5.1 - Componentes do protocolo TICK-TOCK

A entidade *service* transmite os dados da entidade *sender* para a entidade *receiver*, sendo assumido que as trocas de dados nos SAPs são feitas de forma instantânea e atômica; uma célula de dados é adotada como parâmetro de comunicação.

A especificação da entidade *service* deve satisfazer as seguintes restrições:

- Isocronismo: O comportamento da entidade deve ser isocrono, ou seja, uma célula enviada pela entidade *sender* será aceita somente num instante preciso, com um período de π unidades de tempo entre os envios. Apenas uma célula poderá ser reconhecida a cada instante.
- Atrasos de transmissão: A entidade *service* sempre colocará uma célula a disposição da entidade *receiver* com um atraso entre τ_{min} e τ_{max} unidades de tempo, após a sua emissão.
- Espaços entre as entregas: Existirá sempre um atraso de, pelo menos, α unidades de tempo entre duas ofertas sucessivas de células para Sr-SAP.
- Aceitação Imediata: Uma célula oferecida pela entidade *service* à entidade *receiver* deverá ser aceita imediatamente, caso contrário, a entidade perderá esta célula.
- Transmissão sem falhas: Durante a transmissão feita pela entidade *service*, nenhuma célula será perdida. Assim, a única possibilidade de perda de células é a descrita na restrição anterior.

5.1.2 Especificação formal da Entidade *service* em RT-LOTOS

A especificação da entidade *service* seguirá a abordagem "por restrição", que segue o seguinte princípio: para que o sistema atenda a todas as restrições citadas acima, serão elaborados processos que representem exatamente o comportamento exigido por cada restrição, sendo que o comportamento global do sistema será o resultado da composição do comportamento de todos estes processos, que irão avançar de maneira paralela.

A entidade *service*

O comportamento do processo que representa a entidade *service* é a composição paralela dos processos que representam as restrições, apresentados a seguir. Introduz-se a ação interna DELIVER, que representa o momento onde uma célula fica disponível para a entidade *receiver*. DELIVER, sendo uma ação interna do processo, é escondida através do operador HIDE.

Assim, a especificação da entidade *service* é dada por:

```
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]
      )
   )
endspec
```

Isocronismo

O comportamento desta restrição é dado pelo processo denominado `Isochronous`.

```
process Isochronous[Ss_SAP]:=
   [0]Ss_SAP; [π]i; Isochronous[Ss_SAP]
   <Ss_SAP]
   {Ss_SAP: [π]i; Isochronous[Ss_SAP]}
endproc
```

Assim, Ss_SAP é o ponto de comunicação entre as entidades *sender* e *service*. A prefixação [0]Ss_SAP representa a necessidade da ocorrência de maneira instantânea do envio de uma célula, sendo que novas oportunidades serão apresentadas, de forma periódica, caso ocorra ou não o envio de uma célula. O operador de tratamento de violações temporais é utilizado para tratar do caso da não ocorrência pontual do envio, lançando o processo novamente após um atraso de π unidades de tempo, representado pela prefixação [π]i.

Atrasos de Transmissão

O comportamento desta restrição é dado pelo processo denominado `Trans_Dels`.

```
process Trans_Dels[Ss_SAP, DELIVER]:=
   Ss_SAP; [τmin, τmax]i; DELIVER; Trans_Dels[Ss_SAP, DELIVER]
endproc
```

O processo acima restringe o número de células enviadas pela entidade *sender*, pois não há uma

forma de representar a recepção de infinitas células através de um Grafo Temporizado finito. A prefixação $[\tau_{min}, \tau_{max}]$i representa o atraso de transmissão.

Espaços entre as Entregas

O comportamento desta restrição é dado pelo processo denominado `Spacing_Deliveries`.

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

O atraso mínimo entre as entregas é representado pela prefixação [α]i.

Aceitação sem falhas

O comportamento desta restrição é dado pelo processo denominado `Imm_Accept`.

```
process Imm_Accept[DELIVER, Sr_SAP]:=
  DELIVER;
  ( [0]Sr_SAP; Imm_Accept[DELIVER, Sr_SAP]
    <Sr_SAP]
    {Sr_SAP: Imm_Accept[DELIVER, Sr_SAP]}
  )
endproc
```

Assim que uma célula estiver disponível, representada pela ação DELIVER, a entidade *receiver* deverá reconhece-la imediatamente através da prefixação [0]Sr_SAP. Caso o reconhecimento não ocorra, então *receiver* perderá esta célula. Esta possibilidade é representada pelo operador de tratamento de violações temporais, que lança novamente o processo `Imm_Accept`.

5.1.3 Verificação de propriedades da Entidade *service*

A verificação das restrições temporais do sistema, utilizando a ferramenta KRONOS, é feita de forma a verificar se estas restrições, que serão expressas através da lógica temporal tempo-real TCTL, onde a sua sintaxe é apresentada no Anexo B, são verdadeiras para todo o Grafo Temporizado gerado a partir da especificação de *service*.

A seguir, são apresentadas as fórmulas que representam as restrições que precisam ser verificadas pela especificação. [DOY94]

Isocronismo

Esta restrição é dividida em três fórmulas:

$$\texttt{init} \Rightarrow \forall\ (\texttt{enable(Ss\_SAP)} \Rightarrow \forall\Diamond_{=\pi}(\texttt{enable(Ss\_SAP)})) \quad (1)$$

$$\texttt{init} \Rightarrow \forall\ (\texttt{enable(Ss\_SAP)} \Rightarrow \neg\exists\Diamond_{(0,\pi)}(\texttt{enable(Ss\_SAP)})) \quad (2)$$

$$\neg(\texttt{enable(Ss\_SAP)}\ \exists\mathcal{U}_{>0}\ \texttt{enable(Ss\_SAP)}) \quad (3)$$

A primeira fórmula estabelece que uma oferta em Ss_SAP sempre será seguida de outra oferta, após π unidades de tempo. A segunda diz que uma oferta em Ss_SAP nunca seguirá a oferta anterior antes de π unidades de tempo. E a última fórmula estabelece que não existe nenhuma possibilidade onde a oferta em Ss_SAP tenha duração maior que zero.

Atrasos de Transmissão

Esta restrição é representada pela seguinte fórmula:

$$\texttt{init} \Rightarrow \forall\ (\texttt{after(Ss\_SAP)} \Rightarrow \forall\Diamond_{[\tau min,\tau max]}(\texttt{enable(Sr\_SAP)})) \quad (4)$$

estabelecendo que uma oferta em Sr_SAP sempre ocorrerá dentro do intervalo $[\tau_{min}, \tau_{max}]$, após a emissão da célula pela entidade *service*.

Espaços entre as Entregas

A fórmula:

$$\texttt{init} \Rightarrow \forall\ (\texttt{enable(Sr\_SAP)} \Rightarrow \neg\exists\Diamond_{(0,\alpha)}(\texttt{enable(Sr\_SAP)})) \quad (5)$$

estabelece que o espaço de tempo entre as ofertas em Sr_SAP não deve ser inferior a α unidades de tempo.

Aceitação Imediata

A restrição feita sobre a duração da oferta em Sr_SAP é dada pela fórmula:

$$\neg(\texttt{enable(Sr\_SAP)}\ \exists\mathcal{U}_{>0}\ \texttt{enable(Sr\_SAP)}) \quad (6)$$

onde esta oferta deve ter um comportamento instantâneo.

Resultados

O Grafo Temporizado gerado a partir da especificação anterior possui 44 vértices, 124 arcos e 5 relógios, lembrando que o processo `Trans_Dels` possui uma capacidade de apenas uma célula enviada através de *sender*.

Os valores atribuídos as constantes do sistema são os mesmos valores utilizados em [DOY94], sendo que foram verificados quatro conjuntos de valores, com o objetivo de observar a influências das constantes sobre a validação do sistema.

A ferramenta KRONOS retorna as condições para as quais as fórmulas são satisfeitas pelo Grafo Temporizado, além de uma análise de desempenho da verificação, que mostra o número total de iterações e o tempo de utilização do sistema para a realização da verificação. Os resultados da análise de desempenho, obtidos na verificação da entidade *service*, são apresentados na Tabela 5.1.

Fórmula	$\tau_{min} = 50$ $\alpha = 90$ $\tau_{max} = 80$ $\pi = 100$		$\tau_{min} = 10500$ $\alpha = 12000$ $\tau_{max} = 13666$ $\pi = 15000$		$\tau_{min} = 13$ $\alpha = 1050$ $\tau_{max} = 813$ $\pi = 1000$		$\tau_{min} = 75$ $\alpha = 50$ $\tau_{max} = 120$ $\pi = 100$	
	Iterações	Tempo (seg.)	Iterações	Tempo (seg.)	Iterações	Tempo (seg.)	Iterações	Tempo (seg.)
1	11	3,95	11	3,92	47	8,70	22	9,73
2	9	0,88	9	0,83	8	0,68	9	1,00
3	1	0,05	1	0,05	1	0,05	1	0,05
4	17	2,63	11	3,63	27	4,08	13	6,92
5	8	1,35	8	1,28	10	1,62	8	1,40
6	1	0,17	1	0,17	1	0,18	1	0,20

Tabela 5.1 - Resultados da verificação da entidade *service* (KRONOS)

Os dados da análise de desempenho estão diretamente relacionados com o tamanho do Grafo Temporizado e com a complexidade das fórmulas relativas às propriedades a serem verificadas. Como exemplo, para uma fórmula μ, onde deseja-se verificar a propriedade $\forall\Diamond_{I}\mu$, o número de iterações e, conseqüentemente, o tempo de utilização do sistema serão maiores do que numa verificação da fórmula $\exists\Diamond_{I}\mu$, pois a complexidade desta envolve um número menor de iterações.

Para os dois primeiros conjuntos de valores da Tabela 5.1, verificou-se que a especificação do sistema satisfaz todas as fórmulas TCTL, ou seja, segundo os resultados da ferramenta KRONOS, o Grafo Temporizado que representa a entidade *service* satisfaz todas as propriedades estabelecidas pelas fórmulas, pois o conjunto característico das fórmulas é igual ao conjunto dos vértices do Grafo Temporizado, validando a especificação.

Para o terceiro conjunto de valores, as fórmulas (1) e (4) não foram satisfeitas pela especificação do sistema. Isto se deve ao fato que numa emissão de duas células consecutivas separadas por π unidades de tempo, é impossível satisfazer a fórmula (4) com $\alpha > \pi$, pois se o sistema entregar a primeira célula em τ_{max}, a segunda célula seria entregue depois de $\tau_{max} + \alpha$ unidades de tempo após a primeira, satisfazendo a fórmula (5); entretanto, a fórmula (4) estabelece que esta entrega deveria ser antes de $\pi + \tau_{max}$, o que não é possível. Por conseqüência, a fórmula (1) não é satisfeita, pois esta estabelece a separação entre duas emissões.

No último conjunto de valores, a fórmula (1) não é satisfeita pela especificação do sistema. Neste

caso, é observado que se $\pi < \tau_{max}$, a capacidade de recepção de células não é liberada, e o sistema não pode aceitar uma nova célula em Ss_SAP.

Os resultados obtidos nesta verificação estão de acordo com os resultados obtidos em [DOY94]. Do ponto de vista da verificação, a partir da observação de que os dois primeiros conjuntos apresentados na Tabela 5.1 constituem-se de valores válidos para a verificação da especificação, conclui-se que a especificação que representa a entidade *service* possui algumas restrições quanto aos valores das constantes utilizadas ($\alpha \leq \pi$ e $\pi \geq \tau_{max}$).

Os resultados obtidos em [DOY94], com relação ao Grafo Temporizado, foram melhores (24 vértices, 64 arcos e 5 relógios), mas não ficaram distantes dos resultados obtidos pelo tradutor de especificacões escritas em RT-LOTOS. Esta diferença é devida a necessidade de melhorar o tradutor, com relação à fase de redução do Grafo Temporizado, pois o algoritmo implementado não identifica na totalidade todos os vértices semelhantes. Este exemplo permite destacar esta deficiência pelo fato de que, em [DOY94], o Grafo Temporizado foi gerado manualmente, eliminando qualquer vértice semelhante.

5.2 SISTEMA TELEFÔNICO

O exemplo a seguir foi descrito e verificado em [Yovine93]. Este sistema é uma simplificação de um sistema telefônico real, no entanto, apresenta 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 a Central Telefônica.

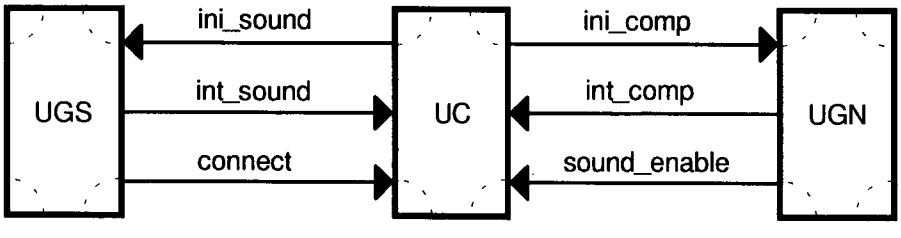


Figura 5.2 - Central Telefônica

A comunicação entre a UGS e a UC se faz 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 esta pronta para provocar o toque do telefone chamado.
int_comp → indica a interrupção da composição do número.

Por outro lado, a comunicação entre um terminal e a central telefônica ocorre da forma apresentada na Figura 5.3:

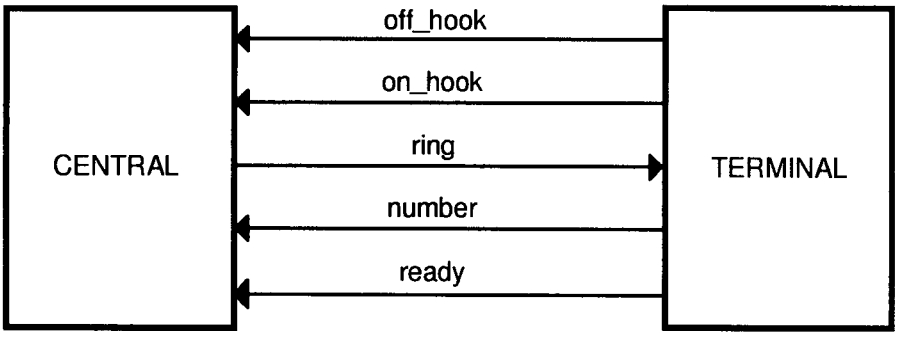


Figura 5.3 - Comunicação Central Telefônica - Terminal

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

off_hook → indica a retirada do telefone do gancho, para a solicitação de uma conexão.
on_hook → indica o retorno do telefone ao gancho.
ring → indica a 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* (seg.), 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 até *tp* (seg.), logo após o tom de discar.
  ⇒ O tempo entre os algarismos não deve ultrapassar o limite de *ts* (seg.).
  ⇒ A composição de um número não deve ultrapassar o limite de *tc* (seg.).

  A composição de um número é interrompida se as restrições acima não forem respeitadas ou se o chamador retornar o telefone ao gancho.

- A função da Unidade de Gestão do Som consiste em provocar um toque, de duração *tn* (seg.), no terminal chamado. Se este não responder até *tr* (seg.), a chamada será cancelada, caso contrário, faz-se a sinalização de conexão para a Unidade Central.

5.2.2 Especificação formal da Central Telefônica em RT-LOTOS

1) A especificação formal da Unidade Central, escrita em RT-LOTOS, é apresentada abaixo, sendo que a declaração das ações foi omitida ([...]), visando uma melhor legibilidade.

```
process UC[...]:=
  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
```

A ação `sync_end` é utilizada para sincronizar a inicialização das unidades da Central Telefônica, sendo assim uma ação interna do sistema. A ação `int_comm` representa a interrupção da comunicação feita pela Unidade Central quando o tempo de conexão atingir o limite estabelecido pelo dispositivo de *timeout* ([*tm*]i), inicializando a Central Telefônica.

2) A especificação formal da Unidade de Gestão do Número, escrita em RT-LOTOS, é:

```
process UGN[...]:=
  ini_comp;
    (
      ( ( ( number1; [0,ts]number2; [0,ts]number3; [0,ts]number4;
            [0]interrupt; stop
            [>
            ( [tc]i; int_comp; exit
              []
              on_hook; int_comp; exit
          ) ) <number2, number3, number4, interrupt]
            { number2: int_comp; exit,
              number3: int_comp; exit,
              number4: int_comp; exit,
              interrupt: sound_enable; exit}
        )>>sync_end; UGN[...]
      ) []
      [tp]i; int_comp; sync_end; UGN[...]
  )
endproc
```

A ação `interrupt` também é uma ação interna, que representa a possibilidade de interrupção do início da chamada, após finalizar a composição do número.

Para cada digito do número chamado, existe a restrição sobre o tempo máximo (`ts`) para que o mesmo seja discado. A expressão deste comportamento é facilitada pelo operador de tratamento de violações temporais. Existe também uma restrição sobre o tempo máximo (`tc`) para a composição do número completo, expressa através de um mecanismo de *watchdog*, que é modelado utilizando o operador de preempção (P [> [t]i; Q).

Um mecanismo de *timeout* (P [] [t]i; Q) checa se o primeiro digito será discado antes do limite de tempo permitido (`tp`), caso contrário, a composição do número é cancelada.

3) A especificação formal da Unidade de Gestão do Som, escrita em RT-LOTOS, é:

```
process UGS[...]:=
   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; Make_new_sound[ring]
   endproc
endproc
```

O processo `Make_new_sound` representa o toque da campainha, efetuado em intervalos de *tn* segundos. Assim, um mecanismo de *watchdog* é utilizado para encerrar o toque da campainha, onde existe três possibilidades de encerramento: quando o terminal do número chamado for retirado do gancho (`ready`); quando o terminal chamador desistir da ligação, colocando o fone no gancho (`on_hook`); ou quando o terminal chamado não responder até o limite de tempo permitido (*tr*).

4) A especificação formal da Central Telefônica é o resultado da composição paralela das unidades UC, UGN e UGS, na qual as ações `sync_end` e `interrupt` são ocultadas, através do operador HIDE, pois estas são ações internas do processo.

```
specification Central[...]
behaviour
   ( hide [sync_end, interrupt] in
      ( UC[...]
        |[ini_comp, int_comp, sound_enable, sync_end]|
        UGN[...]
   ) ) |[ini_sound, int_sound, connect]|
   UGS[...]
endspec
```

5.2.3 Verificação de propriedades da Central Telefônica

As propriedades a serem verificadas são dadas pelas seguintes fórmulas, propostas em [Yovine93]:

$\texttt{connect} \Rightarrow \forall\Diamond_{\leq tm}\ \texttt{free}$ (1)

$\texttt{free} \Rightarrow \forall\ (\texttt{comm} \Rightarrow \forall\Diamond_{\leq tc+tr+tm}\ \texttt{free})$ (2)

$\texttt{comm} \Rightarrow \forall\Diamond_{\leq tp}\ (\texttt{number1} \vee \texttt{free})$ (3)

$\texttt{comm} \Rightarrow \forall\Diamond_{\leq tc}\ (\texttt{number4} \vee \texttt{free})$ (4)

$\texttt{ring} \Rightarrow \forall\Diamond_{\leq tr}\ (\texttt{connect} \vee \texttt{free})$ (5)

A fórmula (1) estabelece que o usuário deverá finalizar a conversação até t_m segundos, caso contrário, a Central Telefônica se encarregará de encerrar a conexão, tornando-se disponível. A fórmula (2) diz que após a demanda de comunicação, a Central Telefônica estará livre no mais tardar depois de $t_c + t_r + t_m$ segundos. As fórmula (3) e (4) estabelecem que o primeiro digito deverá ser obtido em até t_p e que o número completo deverá ser obtido em até t_c segundos, caso contrário, a Central encerra a composição do número e torna-se disponível novamente. A fórmula (5) diz que se o número chamado não responder em até t_r segundos, após o início da chamada, então a Central Telefônica encerra esta demanda, tornando-se livre novamente.

Resultados

O Grafo Temporizado gerado pela ferramenta possui 24 vértices, 51 arcos e 12 relógios. O resultado da verificação é apresentado na Tabela 5.2.

Fórmula	1	2	3	4	5
Iterações	14	16	10	7	13
Tempo (seg.)	1,30	1,53	0,62	0,63	0,70

para: $t_m = 180$ seg., $t_c = 40$ seg., $t_p = 20$ seg., $t_s = 10$ seg., $t_r = 60$ seg., $t_n = 3$ seg.

Tabela 5.2 - Resultados da verificação da Central Telefônica (KRONOS)

A ferramenta KRONOS retornou que os conjuntos característicos de todas as fórmulas acima são iguais ao conjunto de vértices do Grafo Temporizado, que representa a Central Telefônica. Em outras palavras, a especificação da Central Telefônica atende todas as propriedades estabelecidas.

Em [Yovine93], o Grafo Temporizado foi construído através de um tradutor de especificações escritas em ATP [Nicollin92], possuindo 36 vértices, 52 arcos e 6 relógios. A verificação deste Grafo Temporizado também foi realizada através da ferramenta KRONOS, e com as mesmas fórmulas apresentadas anteriormente. A principal diferença entre as duas verificações está nos valores da análise de desempenho, e em particular, com relação as fórmulas (2) e (5). O número de iterações necessárias em [Yovine93] para obter o conjunto característico das fórmulas (2) e (5) foram, respectivamente, 53 e 48, enquanto que foram necessárias apenas 16 iterações para a

fórmula (2) e 13 iterações para a fórmula (5), segundo os valores obtidos através da verificação destas propriedades sobre o Grafo Temporizado gerado a partir da especificação escrita em RT-LOTOS, e apresentados na Tabela 5.2. Este fato é atribuído à diferença entre o tamanho dos Grafos Temporizados, pois o tamanho deste afeta o desempenho da ferramenta KRONOS, já dito anteriormente. O motivo pelo qual o Grafo Temporizado, gerado a partir da especificação em ATP, é maior que o Grafo Temporizado, gerado a partir da especificação em RT-LOTOS, está relacionado a representação das ações temporizadas, pois em [Yovine93] foram utilizados vários dispositivos de *timeout* e *watchdog* para suprir a falta deste tipo de representação, que são operadores complexos, comparados à representação de uma ação temporizada.

Por outro lado, o Grafo Temporizado gerado a partir da especificação escrita em ATP possui apenas 6 relógios, sendo que para a especificação em RT-LOTOS foram 12 relógios. Isto se deve ao fato de que em [Yovine93], a otimização do número de relógios é efetuada de uma maneira mais eficiente, identificando os relógios comuns durante a representação de alguns operadores da linguagem ATP. No entanto, este fato não se mostra comprometedor neste exemplo; entretanto, num exemplo mais complexo, o desempenho da verificação pode ser comprometido.

5.3 ALGORITMO DE EXCLUSÃO MÚTUA

O algoritmo de exclusão mútua foi proposto, originalmente, por Fischer [Fischer85] e descrito por Lamport [Lamport87], sendo um caso clássico no estudo dos métodos formais de verificação de sistemas dependentes do 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 a uma seção crítica, ou seja, somente um processo pode ter acesso à seção crítica em um determinado tempo.

5.3.1 Descrição 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 é ainda 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 a 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.

5.3.2 Especificação formal do algoritmo em RT-LOTOS

Como o tradutor desenvolvido trata de especificações escritas em RT-LOTOS Básico, não é possível especificar o comportamento da variável compartilhada de forma genérica, sendo que para tal seria necessário utilizar a componente de RT-LOTOS que trata de tipos de dados. Assim, a especificação do algoritmo é apresentada a seguir, para um número de processos limitado (3), mas observa-se que o acréscimo de processos pode ser feito de uma maneira trivial. O comportamento de cada processo é representado pela seguinte especificação:

```
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
```

Onde o Grafo Temporizado que representa o comportamento de um processo pode ser ilustrado através da Figura 5.4.

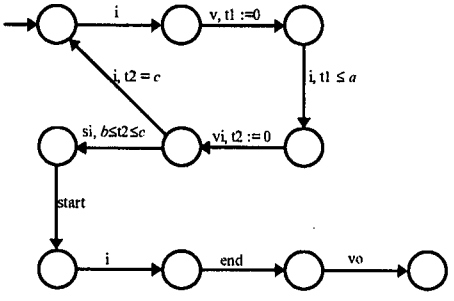


Figura 5.4 - Processo P

Assim, após um processo verificar, por meio da ação "v", indicando que a variável possui o valor zero, que a região crítica não está sendo utilizada, o processo 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 comportamento da variável compartilhada é representado pela seguinte especificação:

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

A Figura 5.5 ilustra o Grafo Temporizado que representa o comportamento da variável compartilhada (Shared_Var), onde é possível observar que não existem restrições temporais associadas aos seus arcos. Isto se deve ao fato da especificação P, que representa o comportamento dos processos que necessitam acessar a região crítica, estabelecer as restrições sobre o acesso à variável compartilhada.

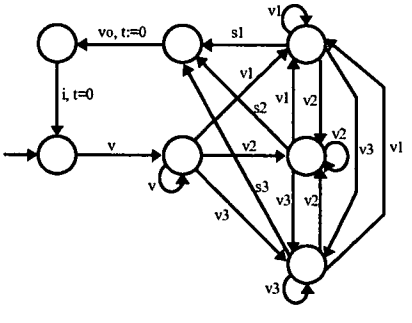


Figura 5.5 - Processo Shared_Var

O comportamento do sistema é dado pela composição paralela dos processos com a variável compartilhada, mostrado na especificação a seguir:

```
specification Mutual_Exclusion_Algorithm[vo,v,v1,v2,v3,s1,s2,s3,start,end]
behaviour
   (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]
endspec
```

5.3.3 Verificação do algoritmo

A primeira verificação feita consistiu em garantir a propriedade de exclusão mútua. Esta propriedade estabelece que após o acesso a região crítica por um processo, nenhum outro processo poderá acessá-la até que esta se torne disponível. Esta verificação não envolve nenhuma restrição temporal, mas consiste na verificação clássica do problema. A propriedade é estabelecida pela seguinte fórmula TCTL:

```
init => (after(start) => ((∀ (¬enable(vo)) ∀u (∀◊(¬enable(start))))) (1)
```

Visando ainda verificar propriedades que envolvam limites temporais, foi considerada também a propriedade apresentada em [LSGL94], estabelecendo que a partir do momento em que qualquer processo tente acessar a região crítica, existe um limite máximo, $2c + 5a$, para que a região crítica seja acessada por qualquer processo. Este limite é estabelecido através da 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; e a última fase estabelece que a variável se tornará estável em até $c + 3a$, totalizando o limite apresentado acima.

Esta propriedade é representada pela seguinte fórmula TCTL:

```
after(v) => (∀◊ {<=2c + 5a} (enable(start)))          (2)
```

Resultados

O Grafo Temporizado gerado pela ferramenta possui 1164 vértices, 2794 arcos e 10 relógios. Os resultados da verificação são apresentados na Tabela 5.3.

Fórmula	1	2
Iterações	36	43
Tempo (seg.)	25,73	34,75

para: $a = 4$, $b = 5$, $c = 10$

Tabela 5.3 - Resultados da verificação do algoritmo de Exclusão Mútua (KRONOS)

A ferramenta KRONOS permitiu obter como resultado a validação das duas propriedades, definidas anteriormente, para todos os vértices do Grafo Temporizado, desde que os valores das constantes a e b sejam tais que $a < b$, pois caso contrário, a exclusão mútua não é garantida. Por exemplo, com os valores $a = 6$, $b = 4$ e $c = 10$, apenas a fórmula (2) é verificada com sucesso, onde a fórmula (1) que representa a propriedade de exclusão mútua não é mais verificada.

Era esperado um número elevado de vértices do Grafo Temporizado, pois somente o entrelaçamento dos processos produz 1000 vértices (10^3). Assim, o número de iterações também é elevado, devido ao número de caminhos possíveis.

5.4 CRUZAMENTO RODOVIA-FERROVIA

O problema descrito a seguir é o clássico Cruzamento Rodovia-Ferrovia (RailRoad Crossing) [AH92]. Em [HL94], foi proposto uma forma generalizada deste problema, ou seja, o caso de um cruzamento onde um número genérico de trens, n, viaja pelo mesmo. Como a linguagem RT-LOTOS Básico possui algumas limitações quanto a representação de problemas genéricos, como no exemplo anterior, foram feitas algumas considerações, com relação ao número de trens, para tornar possível uma especificação do sistema que permita validar a metodologia e a ferramenta.

5.4.1 Descrição do sistema

O problema a considerar 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, correspondente ao local onde o sensor está localizado até a saída do cruzamento I.

O sistema proposto deve satisfazer as seguintes propriedades:

- Segurança: O portão deve ser fechado quando algum 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 adotadas as seguintes notações:

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} → limite superior no tempo para fechar o portão completamente.
γ_{up} → limite superior no tempo para abrir o portão completamente.
β → constante utilizada para representar o atraso do sinal enviado pelo sensor.

Sendo que algumas restrições são atribuídas a estes valores:

$\varepsilon_1 \leq \varepsilon_2$ → 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.

5.4.2 Especificação formal do sistema em RT-LOTOS

1) O comportamento de cada trem é representado pela seguinte especificação:

```
process Train[...]:=
  TrainMotion[...] [> Emergency[...]
where
  process TrainMotion[...]:=
    i; approach; [ε1, ε2]in; out; leaving; TrainMotion[...]
    <in]{in: warning_control_center; stop}
  endproc
  process Emergency[...]:=
    stop_the_train; stop
  endproc
endproc
```

Assim, após um trem alcançar o sensor localizado no início da região R, representado pela ação `i`, 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).

2) A especificação do portão é dada a seguir:

```
process Gate[...]:=
  ( lower; [0, γdown]down; Gate[...]
    []
    raise; [0, γup]up; Gate[...]
  ) <up, down]
  { up: warning_control_center; stop,
    down: siren; stop
  }
endproc
```

O controlador enviará um sinal para que o portão seja abaixado (lower), que ocorrerá efetivamente dentro do limite de tempo possível ([0, γ_{down}] 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}] up). Se ocorrer alguma falha nesta situação, a central de controle é avisada.

3) A especificação do controlador é dada por:

```
process Controller[...]:=
  Control[...] [> Exception[...]
where
  process Control[...]:=
    approach; [β]i; lower; Control_Entries[...] >> raise; Control[...]
  where
    process Control_Entries[...]:=
      leaving; exit [] approach; leaving; leaving; exit
    endproc
  endproc
  process Exception[...]:=
    siren; stop_the_train; stop
  endproc
endproc
```

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] i é utilizado simplesmente 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, porém, é necessário limitar o número de trens que podem ser encontrados dentro da região R, onde a especificação acima, em particular, limita este número para dois trens. Assim, se o portão sinalizar alguma situação de alarme (siren), o controlador promove a parada dos trens.

4) Finalmente, a especificação do cruzamento é dada pela composição dos processos Train, Gate e Controller (Figura 5.6):

```
specification RailRoad[...]
behaviour
   ( ( Train[...] ||| Train[...]
     ) |[approach, leaving, stop_the_train]|
     Controller[...]
   ) |[lower, raise, siren]|
   Gate[...]
endspec
```

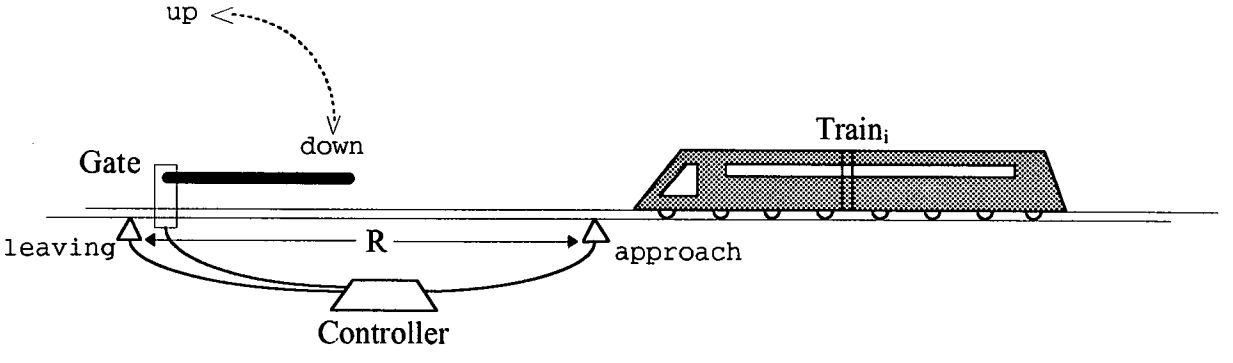


Figura 5.6 - Cruzamento Rodovia-Ferrovia

5.4.3 Verificação de propriedades do sistema

Primeiramente, é estabelecida a fórmula que representa a propriedade relativa a segurança do sistema, ou seja, se algum trem entrar no cruzamento, o portão deverá estar fechado, devido à entrada de um trem anterior, ou o portão deverá ser fechado no limite de tempo de seu funcionamento, ou ainda ocorrer uma falha no fechamento do portão, sendo este fato sinalizado

por um sinal de alarme. Esta propriedade é representada através da fórmula (1).

$$\text{init} \Rightarrow \forall\Diamond(\text{after}(\text{approach}) \Rightarrow \forall\Diamond_{[\beta,\ \gamma_{down}]}(\text{enable}(\text{down}) \text{ or } \text{enable}(\text{siren})) \text{ or } \text{after}(\text{down})) \quad (1)$$

A outra propriedade a ser verificada é a de utilização, estabelecendo que após a saída de um trem do cruzamento, o portão deverá ser aberto no limite de tempo de seu funcionamento, caso não exista outro trem no cruzamento, ou ainda ocorrer uma falha na abertura do portão. Esta propriedade é representada através da fórmula (2).

$$\text{init} \Rightarrow \forall\Diamond(\text{after}(\text{leaving}) \Rightarrow \forall\Diamond_{[0,\ \gamma_{up}]}(\text{enable}(\text{up}) \text{ or } \text{enable}(\text{warning\_control\_center})) \text{ or } \text{enable}(\text{in}) \text{ or } \text{enable}(\text{out})) \quad (2)$$

Resultados

O Grafo Temporizado gerado a partir da especificação anterior possui 334 vértices, 1023 arcos e 11 relógios, lembrando que o número máximo de trens permitido na região R é 2. Os resultados da verificação são apresentados na Tabela 5.4.

Fórmula	1	2
Iterações	40	66
Tempo (seg.)	298,18	203,92

para: $\varepsilon_1 = 30$, $\varepsilon_2 = 45$, $\beta = 1$, $\gamma_{down} = 10$, $\gamma_{up} = 12$

Tabela 5.4 - Resultados da verificação do cruzamento Rodovia-Ferrovia (KRONOS)

A satisfação das fórmulas apresentadas acima é verificada através da ferramenta KRONOS, sendo os seus conjuntos característicos iguais ao conjunto de vértices do Grafo Temporizado que representa o sistema, validando a especificação.

5.5 PROTOCOLO DO BIT ALTERNANTE

Este exemplo segue a notação utilizada em [Garavel89], sendo que a sua especificação em RT-LOTOS Completo é encontrada em [Camargo95]. Neste trabalho, será apresentada uma especificação semelhante a esta, utilizando-se apenas RT-LOTOS Básico, possibilitando a sua verificação pela ferramenta disponível.

5.5.1 Descrição do protocolo

O protocolo do bit alternante consiste em fornecer um serviço que permita o encaminhamento de

mensagens de uma entidade emissora para uma entidade receptora, sendo que a transmissão destas mensagens deve ser efetuada de forma confiável, ou seja, não é permitida a ocorrência de duplicação de mensagens, perdas ou entregas fora da ordem de emissão.

Para cada mensagem enviada pela entidade emissora, a entidade receptora deverá, após receber a mensagem, enviar uma mensagem de confirmação para a entidade emissora. No entanto, o meio de comunicação entre as entidades não é confiável, possibilitando perdas de mensagens.

Assim, no caso de uma perda de mensagem, o meio de comunicação poderá ou não sinalizar este fato à entidade de destino, sendo que utiliza-se um bit de controle adicionado à mensagem para detectar as possíveis perdas não sinalizadas. O valor do bit de controle de uma confirmação será igual ao bit de controle de sua respectiva mensagem, sendo que os bits de controle alternam a cada mensagem emitida.

Portanto, se a entidade emissora receber uma confirmação com o bit de controle errado, ou receber uma indicação de perda da confirmação, ou ainda, não receber nenhuma comunicação até um limite de tempo máximo de espera de uma confirmação, então ela retransmite a última mensagem enviada. Da mesma forma, se a entidade receptora recebe uma indicação de perda de mensagem ou uma mensagem com o bit de controle errado, então ela retransmite a última confirmação enviada.

5.5.2 Especificação formal do protocolo em RT-LOTOS

A Figura 5.7 ilustra a estrutura geral do protocolo do bit alternante:

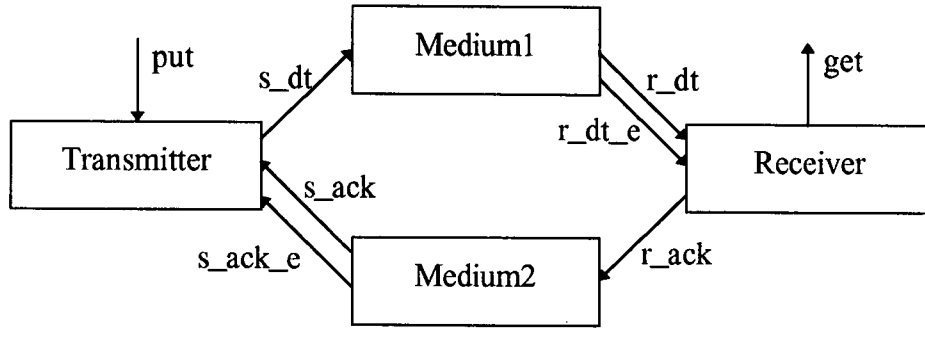


Figura 5.7 - Estrutura Geral do Protocolo do bit alternante

1) A entidade emissora é representada pela seguinte especificação:

```
process Transmitter[...]:=
  put; Transmit[s_dt_m_off, s_ack_off, s_ack_on, s_ack_e, s_ack]
  >> put; Transmit[s_dt_m_on, s_ack_on, s_ack_off, s_ack_e, s_ack]
  >> Transmitter[...]
where
  process Transmit[s_dt_m_bit, s_ack_ok, s_ack_not_ok, s_ack_e, s_ack]:=
    s_dt_m_bit;
    ( ( [0, τlim]s_ack;
        ( s_ack_ok; exit [] s_ack_not_ok; Transmit[...]
        ) <s_ack>{s_ack: Transmit[...]}
      )
      []
      s_ack_e; Transmit[...]
    )
  endproc
endproc
```

A entidade emissora (`Transmitter`) inicia com a emissão de uma mensagem (`put`), onde o valor bit de controle associado a esta mensagem é igual a zero (`off`). Ocorre então o envio da mensagem (`s_dt_m_off`) para o meio de comunicação que transporta a mensagem enviada para a entidade receptora. Logo após, aguarda-se o envio da confirmação da mensagem (`s_ack`), onde o valor do bit pode ser correto (`s_ack_off`) ou errado (`s_ack_on`). Assim, se a confirmação for enviada dentro do limite de tempo permitido ($[0, \tau_{lim}]$), e se o valor do bit estiver correto, então inicia-se uma nova transmissão de mensagem, onde o valor do bit de controle associado a esta será agora igual a 1 (`on`). Caso o valor do bit for errado, ou ocorrer a sinalização de perda da confirmação, ou ainda, expirar o limite de espera da confirmação, então a mensagem será retransmitida, através de um novo lançamento do processo `Transmit`.

2) A entidade receptora é representada pela seguinte especificação:

```
process Receiver[...]:=
  Rec[get, r_dt_m_off, r_dt_m_on, r_ack_off, r_ack_on, r_dt_e]
  >> Rec[get, r_dt_m_on, r_dt_m_off, r_ack_on, r_ack_off, r_dt_e]
  >> Receiver[...]
where
```

```
  process Rec[...]:=
    r_dt_m_bit; get; r_ack_ok; exit
    []
    r_dt_m_not_bit; r_ack_not_ok; Rec[...]
    []
    r_dt_e; r_ack_not_ok; Rec[...]
  endproc
endproc
```

A entidade receptora (`Receiver`) inicia a recepção das mensagens esperando receber uma mensagem com o bit de controle igual a zero (off). Caso a mensagem recebida possua o valor do bit igual a zero (`r_dt_m_off`), então esta é apresentada ao usuário que solicitou o serviço (`get`), enviando também a confirmação para a entidade emissora (`r_ack_off`). Mas, caso o valor do bit não for correto (`r_dt_m_on`), ou se for comunicada uma perda de mensagem, então a entidade receptora envia uma confirmação cujo valor do bit de controle é incorreto (`r_ack_on`). A recepção correta de uma mensagem faz com que a entidade receptora espere uma nova mensagem, agora com o valor do bit igual a 1 (on).

3) O meio de comunicação em cada sentido (`Medium1` e `Medium2`) é especificado da seguinte forma:

```
process Medium1[...]:=
  s_dt_m_off; [0, π1]i;
  ( r_dt_m_off; Medium1[...] [] r_dt_e; Medium1[...] [] i; Medium1[...]
  ) []
  s_dt_m_on; [0, π1]i;
  ( r_dt_m_on; Medium1[...] [] r_dt_e; Medium1[...] [] i; Medium1[...])
endproc
process Medium2[...]:=
  r_ack_off; [0, π2]i;
  (s_ack; s_ack_off; Medium2[...] [] s_ack_e; Medium2[...] [] i; Medium2[...]
  ) []
  r_ack_on; [0, π2]i;
  ( s_ack; s_ack_on; Medium2[...] [] s_ack_e; Medium2[...] [] i; Medium2[...]
  )
endproc
```

A estrutura das especificações dos meios de comunicação (`Medium1` e `Medium2`) é semelhante,

diferenciadas apenas pelas ações que representam o comportamento, e pelo limite de tempo de atraso de envio (π_1 e π_2). Assim, quando um meio recebe uma mensagem ou confirmação (`s_dt_m` e `r_ack`), ele pode transmití-a com sucesso (`r_dt_m` e `s_ack`), transmitir um sinal de perda de mensagem ou confirmação (`r_dt_e` e `s_ack_e`), ou não transmitir nada (`i`).

4) O protocolo do bit alternante é representado pela seguinte especificação global:

```
specification Alternating_Bit_Protocol[...]
behaviour
   ( Transmitter[...] ||| Receiver[...]
   ) |[s_dt_m_off, s_dt_m_on, r_dt_m_off, r_dt_m_on, r_dt_e,
       r_ack_off, r_ack_on, s_ack_off, s_ack_on, s_ack_e, s_ack]|
   ( Medium1[...] ||| Medium2[...])
endspec
```

5.5.3 Verificação do protocolo

Uma propriedade a ser satisfeita pelo Grafo Temporizado estabelece que, após o envio da mensagem pela entidade emissora, e dentro do limite de tempo estabelecido , τ_{lim}, existem quatro possibilidades para a entidade emissora:

- receber uma confirmação, com o bit de controle indicando que a transmissão foi efetuada com sucesso;
- receber uma confirmação, mas com o valor do bit de controle errado;
- receber uma sinalização de perda da confirmação; ou
- não receber nada do meio de transmissão.

Portanto, a fórmula abaixo expressa a propriedade descrita acima.

```
init => ∀ (after(put) =>
            ∀◊((enable(s_dt_m_off) => ∀◊{<=τlim}(enable(s_ack_off) or
                                        ((enable(s_ack_on) or
                                          enable(s_ack_e) or
                                          enable(i)) => enable(s_dt_m_off))))
               or (enable(s_dt_m_on) => ∀◊{<=τlim}(enable(s_ack_on) or
                                        ((enable(s_ack_off) or
                                          enable(s_ack_e) or
                                          enable(i)) => enable(s_dt_m_on))))))
```

Resultados

O Grafo Temporizado que representa o protocolo do bit alternante possui 502 vértices, 1366 arcos e 7 relógios. Foram necessárias 40 iterações e 59,12 segundos para que a ferramenta KRONOS retornasse que a fórmula é satisfeita em todos os vértices, sendo que os valores adotados para as constantes são: $\tau_{lim} = 300$, $\pi_1 = 180$ e $\pi_2 = 120$. É fácil de observar que existe uma restrição sobre as constantes, pois se $\tau_{lim} < \pi_1 + \pi_2$, a fórmula acima não é mais satisfeita.

5.6 CONSIDERAÇÕES FINAIS A RESPEITO DA AVALIAÇÃO DA METODOLOGIA E DA FERRAMENTA

Neste capítulo, foram apresentados diversos exemplos com uma complexidade viável para a representação em RT-LOTOS. Neste trabalho, iniciou-se ainda várias especificações: Sincronização de Lábios [SHH92], e outras atualmente em curso de realização.

O tradutor mostrou-se eficiente quanto ao mapeamento de RT-LOTOS para Grafos Temporizados. Verificou-se que o tradutor desenvolvido neste trabalho possui algumas vantagens sobre o tradutor desenvolvido em [Yovine93], que gera um Grafo Temporizado a partir de especificações escritas em ATP [Nicollin92], em particular, na redução do número de vértices do Grafo Temporizado, através da representação de ações temporizadas.

Em alguns exemplos, o tamanho destes grafos pode parecer complexo, comparado a sua especificação. Porém, é bom lembrar que o operador de composição paralela geralmente provoca um considerável aumento no número de vértices. Verificou-se também que o algoritmo que atribui os relógios do Grafo Temporizado deve ser aprimorado, de forma a reduzir o número total de relógios, através de uma análise da representação dos operadores.

O tradutor limita-se às especificações escritas em RT-LOTOS Básico. A representação da variável especial, @t, da linguagem RT-LOTOS Completo, em particular, para realizar a verificação através da ferramenta KRONOS, não é atualmente possível, pois a versão utilizada não possui nenhum artifício semelhante.

CAPÍTULO 6

CONCLUSÕES

O uso de métodos formais para especificação de sistemas onde o tempo intervém, e de metodologias de verificação constitue o contexto deste trabalho. Foi apresentada uma metodologia baseada na álgebra de processos RT-LOTOS [Camargo95], que permite especificar tais sistemas e verificar sobre esta especificação a satisfação de propriedades estabelecidas durante a concepção.

A abordagem apresentada neste trabalho consiste na geração de um Grafo Temporizado, que representa a especificação do sistema, escrita na linguagem RT-LOTOS, e a partir da qual é possível verificar propriedades, utilizando um método de verificação simbólica de modelos (*Symbolic Model Checking*).

Foi desenvolvido neste trabalho um tradutor automático de especificações, escritas em RT-LOTOS, para o formalismo de Grafos Temporizados, sendo que uma etapa preliminar é utilizada para gerar uma representação intermediária na forma de uma Rede de Petri, que permitiu simplificar todo o mecanismo de mapeamento. Num segundo tempo, o Grafo Temporizado é gerado a partir desta rede, através da simulação da Rede de Petri obtida.

A verificação das propriedades sobre o Grafo Temporizado foi realizada através da ferramenta KRONOS [OY93], que implementa um algoritmo de verificação simbólica de modelos [HNSY92]. As propriedades a serem verificadas são expressas através de fórmulas escritas utilizando a lógica temporal tempo-real TCTL [ACD90].

Os estudos de casos apresentados demonstraram que a linguagem RT-LOTOS Básico permite representar situações diversas, onde as restrições temporais dos sistemas puderam ser expressas de uma forma clara e eficiente. No entanto, constatou-se que existe uma limitação, quanto a representação de comportamentos onde é necessário um tratamento mais geral, usando variáveis, por exemplo. Neste caso, torna-se necessária a simplificação da especificação para que seja possível realizar a verificação. Detectou-se ainda casos onde não foi possível realizar a verificação, por exemplo, quando é necessário armazenar o tempo de ocorrência de uma ação, pois, apesar da linguagem RT-LOTOS Completa possuir uma variável que desempenha este papel (@t), isto pode levar em muitos casos a não-decidibilidade do processo de verificação [DOY94].

A verificação de sistemas a partir de Grafos Temporizados tem sido apresentada em vários trabalhos existentes na literatura [CF95, DOY94, Yovine93]. Em [Yovine93], foi desenvolvido um tradutor de especificações escritas através da álgebra de processos ATP [Nicollin92] para Grafos Temporizados, porém, a linguagem ATP possui limitações quanto a representação da ocorrência de ação em um intervalo de tempo e do tratamento de exceções temporais. Em [DOY94] é apresentada uma proposta de mapeamento para uma extensão temporal de LOTOS, diferente da deste trabalho e chamada ET-LOTOS; entretanto, nenhum mecanismo de tradução automático foi desenvolvido nem implementado, deixando incompleto uma validação da abordagem para as extensões temporais de LOTOS. Neste trabalho, constatou-se que a metodologia de geração de Grafos Temporizados, a partir de especificações escritas em RT-LOTOS, possui vantagens sobre a metodologia apresentada em [Yovine93], pois o número de vértices do Grafo Temporizado pôde ser reduzido, através da representação de ações temporizadas.

A integração a outras ferramentas de verificação baseadas em Grafos Temporizadas (tais como HyTech [HH95]) faz parte das perspectivas de continuação deste trabalho, bem como a junção destas ferramentas num ambiente completo, contendo também um simulador para a obtenção de cenários. A utilização da abordagem e da ferramenta associada, apresentada neste trabalho, está prevista no projeto PROTEM-CC (fase 3), aprovado e denominado como DAMD (Design de Aplicações Multimídias Distribuídas.

Toda a abordagem apresentada neste trabalho será de grande interesse a um futuro padrão para a extensão temporal de LOTOS, pois será possível promover uma adaptação ao modelo eleito.

REFERÊNCIAS BIBLIOGRÁFICAS

[ACD90] Alur, R.; Courcoubetis, C.; Dill, D.; *Model Checking for Real-Time Systems*. In Proceedings of the 5th IEEE Symposium on Logic in Computer Science, 1990.

[AD91] Alur, R.; Dill, D.; *The Theory of Timed Automata*. Proccedings of the REX Workshop, Lecute Notes in Computer Science No. 600, Mook, The Netherlands, June, 1991.

[AH92] Alur, R.; Henzinger, T.A.; *Logics and Models for Real Time: A Survey*. Proccedings of the REX Workshop, Mook, The Netherlands, June, 1991, Lecture Notes in Computer Science No. 600, Springer-Verlag, 1992.

[ASU86] Aho, A.V.; Sethi, R.; Ullman, J.D.; *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[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.

[BB87] Bolognesi, T.; Brinksma, E.; *Introduction to the ISO specification language LOTOS*. Computer Networks and ISDN Systems, North-Holland, num. 14, 1987.

[BD88] Boullier, P.; Deschamp, P.; *Le Système SYNTAX: Manuel d'Utilisation et de Mise en Oeuvre sous UNIX*. Project Langages et Traducteurs, INRIA, Septembre, 1988.

[BD89] Boullier, P.; Deschamp, P.; *SYNTAX: User Commands and C Library Functions*. Project Langages et Traducteurs, INRIA, Juin, 1989.

[BLT93] Bolognesi, T.; Lucidi, F.; Trigila, S.; *Converging Towards a Timed-LOTOS Standard*. Reserarch Report, CNUCE/C.R.N., Pisa, Itália, 1993.

[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.

[CES86] Clarke, E.M.; Emerson, E.A.; Sistla, A.P.; *Automatic verification of finite-state concurrent systems using temporal logic specifications*. ACM Transactions on Programming Languages and Systems, 8(2):244-263, 1986.

[CF95] Camargo, M.S. de; Farines, J.-M.; *Uma abordagem para especificação e verificação de sistemas dependentes do tempo*. IX SBES, 1995.

[CO94] Courtiat, J.-P.; Oliveira, R.C.; *About time nondeterminism and exception handling in a temporal extension of LOTOS*. In Protocol Specification, Testing and Verification XIV, pages 37-52, Canadá, 1994.

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

[Fischer85] Fischer, M.; Re: Where are you? E-mail message to Leslie Lamport. Arpanet message number 8506252257.AA07636@YALE-BULLDOG.YALE.ARPA (47 lines), June, 1985.

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

[HL94] Heitmeyer, C.; Lynch, N.; *The Generalized Railroad Crossing: A case study informal verification of real-time Systems*. Laboratory for Computes Science and Massachusetts Institute of technology, November, 1994.

[HH95] Henzinger, T. A.; Ho, Pei-Hsin; *HyTech: The Cornell HYbrid TECHnology Tool*. In Hybrid Systems II, Lecture Notes in Computer Science 999, Springer-Verlag, 1995, pp. 265-294.

[HNSY92] Henzinger, T.A.; Nicollin, X.; Sifakis, J.; Yovine, S.; *Symbolic model-checking for real-time systems*. In Proceedings of 7th LICS, IEEE Computer Society Press, 1992.

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

[ISO88] LOTOS, *A Formal Description Technique Based on the Ordering of Observational Behaviour*. ISO IS 8807, November, 1988.

[Koymans90] Koymans, R.; *Specifying real-time propreties with metric temporal logic*. Real-Time Systems, 2(4), pp. 255-299, November, 1990.

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

[LL94] 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 OSI95 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, Berne, Swtzerland, October, 1994.

[MCF96] Martins, R.F.; Camargo, M.S. de; Farines, J.-M.; *Sobre a Verificação de Especificações em RT-LOTOS*. Submetido ao X Simpósio Brasileiro de Engenharia de Software (SBES96), a ser realizado em São Carlos-SP, nos dias 22-25 de Outubro de 1996.

[Merlin74] Merlin, P.; *A study of the recoverability of Computer Systems*. PhD Thesis, University of California, Irvine, 1974.

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

[Milner80] Milner, R.; *A Calculus of Communicating Systems*. Lecture and Notes in Comp. Science, No. 92, Spriger Verlag, 1980.

[Nicollin92] Nicollin, X.; *ATP: une algèbre pour la spécification et l'analyse des systèmes temp reel*. Thèse, Institut National Polytechnique de Grenoble, France, Mai, 1992.

[NSY92] Nicollin, X.; Sifakis, J.; Yovine, S.; *Compiling real-time specifications into extended automata.* IEEE TSE Special Issue on Real-Time Systems, September, 1992.

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

[OY93] Olivero, A.; Yovine, S.; *KRONOS: A Tool for Verifying Real-Time Systems*, User's Guide and Reference Manual. Montbonnot Soint Martin, França, August, 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), pp. 127-143, 1993.

[SHH92] Stefani, J.-B.; Hazard, L.; Horn, F.; *Computational model for distributed multimedia applications based on a synchronous programming language.* Computer Communications, 15(2):114-128, March, 1992.

[Sun88] SUN Microsystems; *Programming Utilities and Libraries Manual.* Chapters 9-10, Revision A of 27 March, 1990.

[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, Mai, 1993.

[YPD94] Yi, W.; Pettersson, P.; Daniels, M.; *Automatic Verification of Real-Time Communicating Systems by Constraint-Solving.* In Proc. FORTE'94, Berne, Swtzerland, October, 1994.

ANEXO A

UMA INTRODUÇÃO À LINGUAGEM LOTOS

A álgebra de processos LOTOS (*Language of Temporal Ordering Specifications*) é uma linguagem de especificação formal padronizada pela ISO [ISO88], sendo constituída de duas componentes: uma componente de controle, também conhecida como LOTOS-básico, baseada nas álgebras de processos CCS [Milner80] e CSP [Hoare85], onde é utilizado um alfabeto finito de ações observáveis, além da ação interna *i* (ação invisível), para representar o comportamento de um sistema; e a componente de tipos de dados, baseada na teoria formal de tipos abstratos de dados algébricos, que trata da descrição de estruturas de dados e expressões de valores.

LOTOS é normalmente aplicada a sistemas distribuídos, sistemas de processamento de informação e sistemas concorrentes, dentre outros. Além de possuir um poder de expressão que lhe permite representar as diversas relações de causalidades, LOTOS apresenta facilidades para representar comportamentos complexos a partir da composição de especificações, onde um sistema é visto como sendo um processo, com a possibilidade deste ser constituído de vários sub-processos.

A.1 SINTAXE DA LINGUAGEM

Como o interesse neste trabalho está na componente de controle de LOTOS, pois o tradutor não trata da parte de tipos de dados, apresenta-se de forma resumida a sintaxe e o significado informal de cada operador da linguagem [BB87]. Os símbolos B, B_1 e B_2, que aparecem a seguir representam expressões de comportamento.

Inação: O operador de inação, representado pela palavra "**stop**", denota um processo completamente inativo, onde nenhuma ação (observável ou interna) pode ser oferecida ao ambiente.

Terminação com sucesso: Representado pela palavra "**exit**", denota um processo onde a ação δ representa o fim bem sucedido do mesmo, sendo que após esta ação ele transforma-se em um processo "**stop**".

Prefixação de ações: Este operador pode ser utilizado de duas formas: ***i; B***, representando o oferecimento da ação interna ***i***, seguido do comportamento ***B***; e ***g; B***, representando o

oferecimento de uma ação observável ***g*** (*gate*), seguido do comportamento ***B***.

Escolha: O operador de escolha é representado por B_1 [] B_2, denotando que um processo se comportará como B_1 ou como B_2.

Ocultação: O operador de ocultação (*Hiding*) é representado por **hide** $g_1,...,g_n$ **in** ***B***, onde as ações observáveis, $g_1,...,g_n$, do processo ***B***, são transformadas em ações não observáveis.

Composição Paralela: Este operador pode ser representado através de três formas:

- B_1 |[$g_1,...,g_n$]| B_2 : representando a sincronização dos processos, B_1 e B_2, que oferecem ações em $\{g_1,...,g_n\}$;
- B_1 ||| B_2 : representando o entrelaçamento (*interveaving*) das ações de B_1 e B_2; e
- B_1 || B_2 : representando a sincronização completa, isto é, o conjunto de ações a serem sincronizadas é o conjunto de todas as ações possíveis.

Composição Seqüencial: A composição seqüencial de dois processos (*enable*), B_1 seguido de B_2, é representada por B_1 >> B_2, sendo que se B_1 finalizar com uma terminação com sucesso, o processo B_2 é habilitado.

Desabilitação: Este operador, também conhecido como operador de preempção (*disable*), é representado por B_1 >> B_2, expressando a possibilidade do processo B_1 ser desabilitado pelo processo B_2.

Instanciação de Processos: Representado por $P[g_1,...,g_n]$, denotando uma instanciação do processo ***P***.

ANEXO B

A GRAMÁTICA E A SEMÂNTICA OPERACIONAL DA LINGUAGEM RT-LOTOS

Neste anexo, são apresentadas a gramática e a semântica operacional atribuídas à linguagem RT-LOTOS, utilizadas no desenvolvimento do tradutor. Assim, a gramática é dada por:

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*] { *recproc* }
term1 → *hide_op term1* | *term2*
procs → *proc* | *proc procs*
action → *name* | *name* , *action*
recproc → *name* : *body* | *name* : *body* , *recproc*
hide_op → **hide** *list* **in**
term2 → *term2 seq_op term3* | *term3*
proc → **process** *func proc_def*
seq_op → >>
term3 → *term3 preemp_op term4* | *term4*
proc_def → := *body processes* **endproc**
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*

Onde a representação utilizada acima diferencia os elementos terminais da gramática (as palavras-chave, os operadores e os sinais utilizados para delimitação, ex.: **specification**) dos elementos não-terminais (identificadores e regras gramaticais, ex.: *processes*).

O não-terminal *string* representa qualquer cadeia de caracteres formadas pelas letras do alfabeto (maiúsculas e minúsculas), além do caracter "_". E o não-terminal *integer* representa um número pertencente ao conjunto dos números naturais.

A semântica operacional da linguagem RT-LOTOS é apresentada na Tabela B.1.

Tabela B.1 - Regras da semântica operacional de RT-LOTOS

(1) $\dfrac{}{stop \xrightarrow{t} stop}$ $(t \in D^{\omega})$

(2) $\dfrac{}{exit \xrightarrow{\delta} stop}$ $\dfrac{}{exit \xrightarrow{t} exit}$ $(t \in D^{\omega})$

(3) $\dfrac{}{[0,t]a; E \xrightarrow{a} E}$ $(t \in D^{\omega},\ a \in Act^{i})$ $\dfrac{}{[0,0]a; E \xrightarrow{a^*} stop}$ $(a \in Act)$

$\dfrac{}{[0,t+s]a; E \xrightarrow{s} [0,t]a; E}$ $(t, s \in D^{\omega},\ s > 0,\ a \in Act^{i})$

$\dfrac{}{[t_1+s, t_2+s]a; E \xrightarrow{s} [t_1,t_2]a; E}$ $(s, t_1, t_2 \in D^{\omega},\ s > 0,\ a \in Act^{i})$

(4) $\dfrac{E \xrightarrow{a} E'}{E[\,]F \xrightarrow{a} E' \quad F[\,]E \xrightarrow{a} E'}$ $(a \in Act^{i,\delta})$ $\dfrac{E \xrightarrow{a^*} E'}{E[\,]F \xrightarrow{a^*} E'[\,]F \quad F[\,]E \xrightarrow{a^*} F[\,]E'}$ $(a \in Act)$

$\dfrac{E \xrightarrow{t} E',\ F \xrightarrow{t} F'}{E[\,]F \xrightarrow{t} E'[\,]F'}$ $(t \in D^{\omega})$

(5) $\dfrac{E \xrightarrow{a} E',\ F \xrightarrow{a} F'}{E|[L]|F \xrightarrow{a} E'|[L]|F'}$ $(a \in L \cup \{\delta\})$ $\dfrac{E \xrightarrow{a^*} E',\ F \xrightarrow{a^*} F'}{E|[L]|F \xrightarrow{a^*} E'|[L]|F'}$ $(a \in L)$

$\dfrac{(E \xrightarrow{a^*} E') \wedge (F \not\xrightarrow{a} \vee F \not\xrightarrow{a^*})}{E|[L]|F \xrightarrow{a^*} E'|[L]|F \quad F|[L]|E \xrightarrow{a^*} F|[L]|E'}$ $(a \in L)$ $\dfrac{E \xrightarrow{a} E'}{E|[L]|F \xrightarrow{a} E'|[L]|F \quad F|[L]|E \xrightarrow{a} F|[L]|E'}$ $(a \notin L \cup \{\delta\})$

$\dfrac{E \xrightarrow{a^*} E'}{E|[L]|F \xrightarrow{a^*} E'|[L]|F \quad F|[L]|E \xrightarrow{a^*} F|[L]|E'}$ $(a \notin L)$ $\dfrac{(E \xrightarrow{t} E' \wedge F \xrightarrow{t} F'),\ (\forall a \in L\ E \not\xrightarrow{a} \wedge F \not\xrightarrow{a})}{E|[L]|F \xrightarrow{t} E'|[L]|F'}$ $(t \in D^{\omega})$

(6) $\dfrac{E \xrightarrow{a} E'}{\text{hide } L \text{ in } E \xrightarrow{a} \text{hide } L \text{ in } E'}$ $(a \notin L)$ $\dfrac{E \xrightarrow{a} E'}{\text{hide } L \text{ in } E \xrightarrow{i} \text{hide } L \text{ in } E'}$ $(a \in L)$

$\dfrac{E \xrightarrow{a^*} E'}{\text{hide } L \text{ in } E \xrightarrow{a^*} \text{hide } L \text{ in } E'}$ $(a \notin L)$ $\dfrac{E \xrightarrow{t} E',\ (\forall a \in L\ E \not\xrightarrow{a})}{\text{hide } L \text{ in } E \xrightarrow{t} \text{hide } L \text{ in } E'}$ $(t \in D^{\omega})$

(7) $\dfrac{E \xrightarrow{a} E'}{E \gg F \xrightarrow{a} E' \gg F}$ $(a \in Act^{i})$ $\dfrac{E \xrightarrow{\delta} E'}{E \gg F \xrightarrow{i} F}$ $\dfrac{E \xrightarrow{a^*} E'}{E \gg F \xrightarrow{a^*} E' \gg F}$ $(a \in Act)$

$\dfrac{E \not\xrightarrow{\delta},\ E \xrightarrow{t} E'}{E \gg F \xrightarrow{t} E' \gg F}$ $(t \in D^{\omega})$

(8) $\dfrac{E \xrightarrow{a} E'}{E [> F \xrightarrow{a} E' [> F}$ $(a \in Act^{i})$ $\dfrac{F \xrightarrow{a} F'}{E [> F \xrightarrow{a} F'}$ $(a \in Act^{i,\delta})$ $\dfrac{E \xrightarrow{\delta} E'}{E [> F \xrightarrow{\delta} E'}$

$\dfrac{F \xrightarrow{a^*} F'}{E [> F \xrightarrow{a^*} E [> F'}$ $(a \in Act)$ $\dfrac{E \xrightarrow{a^*} E'}{E [> F \xrightarrow{a^*} E' [> F}$ $(a \in Act)$ $\dfrac{E \xrightarrow{t} E',\ F \xrightarrow{t} F'}{E [> F \xrightarrow{t} E' [> F'}$ $(t \in D^{\delta})$

(9) $\dfrac{E \xrightarrow{a} E'}{E < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots, a_n{:}Q_n\} \xrightarrow{a} E' < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots, a_n{:}Q_n\}}$ $(a \in Act^{i})$

$\dfrac{E \xrightarrow{\delta} E'}{E < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots, a_n{:}Q_n\} \xrightarrow{\delta} E'}$ $\dfrac{E \xrightarrow{a_j^*} E'}{E < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots, a_n{:}Q_n\} \xrightarrow{i} Q_j}$ $(a_j \in L)$

$\dfrac{E \xrightarrow{b^*} E'}{E < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots, a_n{:}Q_n\} \xrightarrow{b^*} E' < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots, a_n{:}Q_n\}}$ $(b \in Act - L)$

$\dfrac{E \xrightarrow{t} E'}{E < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots, a_n{:}Q_n\} \xrightarrow{t} E' < L] \{a_1{:}Q_1, a_2{:}Q_2, \cdots a_n{:}Q_n\}}$ $(t \in D^{\omega})$

(10) $\dfrac{E[a_1/a_1', \cdots, a_n/a_n'] \xrightarrow{a} E',\ P[a_1', \cdots, a_n'] := E}{P[a_1, \cdots, a_n] \xrightarrow{a} E'}$ $(a \in Act^{i,\delta})$

$\dfrac{E[a_1/a_1', \cdots, a_n/a_n'] \xrightarrow{a^*} E',\ P[a_1', \cdots, a_n'] := E}{P[a_1, \cdots, a_n] \xrightarrow{a^*} E'}$ $(a \in Act)$

$\dfrac{E[a_1/a_1', \cdots, a_n/a_n'] \xrightarrow{t} E',\ P[a_1', \cdots, a_n'] := E}{P[a_1, \cdots, a_n] \xrightarrow{t} E'}$ $(t \in D^{\omega})$

ANEXO C

A FERRAMENTA KRONOS

KRONOS é uma ferramenta desenvolvida pelo laboratório VERIMAG (França), que promove a verificação simbólica de modelos para Sistemas Tempo-Real. Ela lê um Grafo Temporizado, que descreve o comportamento do sistema, e uma fórmula TCTL, especificando um requisito, e verifica se o Grafo Temporizado satisfaz a fórmula.

KRONOS adota a definição de Grafos Temporizados apresentada em [NSY92], e a definição da lógica TCTL apresentada em [ACD90].

C.1 O GRAFO TEMPORIZADO

O formato do arquivo de entrada que contém o Grafo Temporizado consiste de um cabeçalho, que declara o número total de estados, transições e relógios do Grafo Temporizado, como é mostrado abaixo:

`#states` *s*
Onde *s* é um inteiro positivo que corresponde ao número de estados do Grafo Temporizado.

`#trans` *t*
Onde *t* é um inteiro positivo que corresponde ao número de transições do Grafo Temporizado.

`#clocks` *c* $id_1 \ldots id_c$
Onde *c* é um inteiro positivo que corresponde ao número de relógios do Grafo Temporizado, e $id_1 \ldots id_c$ são os identificadores associados a cada um deles.

Após o cabeçalho, é declarado o corpo do Grafo Temporizado, que consiste de uma lista de estados seguidos pelas listas de transições.

`state:` *n*
Inicia a definição do estado *n*.

`invar:` *invariant*
Associa uma condição invariante ao estado.

`trans:` $trans_1 \ldots trans_k$

Associa uma lista de transições que partem do estado que está sendo definido.

As transições são como comandos guardas, e possuem o seguinte formato:

guard `=>` *label*`;` `reset { ` $clock_1 \ldots clock_i$ `};` `goto` *k*

A condição *guard* é uma fórmula booleana sobre os relógios. *label* representa a etiqueta que identifica a transição. O conjunto de relógios a serem inicializados pela transição é especificado pela palavra-chave `reset`. A palavra-chave `goto` define o estado de destino desta transição.

Os estados devem ser listados na ordem natural, começando por zero e obedecendo o conteúdo do cabeçalho. KRONOS compila o Grafo Temporizado e gera um código intermediário mais conveniente, armazenado no arquivo com sufixo `.kro`.

C.2 AS FÓRMULAS

KRONOS utiliza a lógica TCTL para representar os requisitos a serem verificados no sistema. As fórmulas são construídas a partir de proposições e restrições sobre os relógios, por meio de operadores lógicos e temporais.

Os principais operadores são:

`init`

Este é o predicado que representa o estado inicial, com todos os relógios inicializados.

`ed` *fórmula*

Significa que durante algum caminho computacional, *fórmula* eventualmente torna-se verdadeira. Este operador corresponde a notação $\exists\Diamond$ de TCTL.

`ed{~`*c*`}` *fórmula*

Significa que durante algum caminho computacional, *fórmula* eventualmente torna-se verdadeira em um tempo *t* que satisfaça *t*~*c*, onde ~{<, ≤, >, ≥, =}.

`ab` *fórmula*

Significa que *fórmula* sempre é verdadeira. Este operador corresponde a notação $\forall$ de TCTL.

`ad` *fórmula*

Significa que durante todos os caminhos computacionais, *fórmula* eventualmente torna-se verdadeira. Este operador corresponde a notação $\forall\Diamond$ de TCTL.

`ad{~c} fórmula`

Significa que durante todos os caminhos computacionais, *fórmula* eventualmente torna-se verdadeira em um tempo t que satisfaça $t \sim c$, onde $\sim \{<, \leq, >, \geq, =\}$.

A sintaxe da lógica usada por KRONOS é:

```
<fórmula> ::= true
            |   init
            |   state(<integer>)
            |   <clockconstraint>
            |   not <fórmula>
            |   <fórmula> and <fórmula>
            |   <fórmula> or <fórmula>
            |   <fórmula> impl <fórmula>
            |   <fórmula> eu <bound> <fórmula>
            |   ed <bound> <fórmula>
            |   eb <bound> <fórmula>
            |   <fórmula> au <bound> <fórmula>
            |   ad <bound> <fórmula>
            |   ab <bound> <fórmula>
            |   (<fórmula>)

<bound> ::= <> | {<rel><integer>} | <interval>

<rel> ::= < | <= | > | >= | =

<clockconstraint> ::= <clock><rel><integer>
                    |   <clock><rel><clock>
                    |   <clock><rel><clock> + <integer>
                    |   <clock><rel><integer> + <clock>
                    |   <integer><rel><clock>
                    |   <integer> + <clock><rel><clock>
                    |   <clock> + <integer><rel><clock>
                    |   <clock> - <clock><rel><integer>
                    |   <integer><rel><clock> - <clock>

<interval> ::= ]<integer>,<integer>[
             |   ]<integer>,<integer>]
             |   [<integer>,<integer>[
             |   [<integer>,<integer>]
```

C.3 CONSIDERAÇÕES GERAIS

A versão atual da ferramenta é executada no sistema UNIX. KRONOS lê dois arquivos contendo um Grafo Temporizado e uma fórmula, e relata o progresso de sua evolução através de mensagens apresentadas na saída padrão. Os argumentos aceitos pelo KRONOS, bem como as formas gerais da linha de comando são apresentados a seguir:

```
kronos [-s step] [-vh] arquivo1 [.tg] arquivo2 [.tctl]
kronos [-s step] -f[-vh] arquivo1 [.tg] string
kronos arquivo1 [.tg]
```

onde:

`-s step`
Estabelece o passo no tempo usado para avaliar as fórmulas `au`. Como padrão, KRONOS adota o passo unitário.

`-h`
Esta opção faz com que as avaliações de todas as sub-fórmulas sejam armazenadas em um arquivo com sufixo `.log`. Como padrão, KRONOS cria somente o arquivo de saída com sufixo `.eval`, com o resultado final da avaliação. Este resultado pode ser *true*, indicando que a fórmula analisada foi completamente satisfeita; *false*, indicando que não é possível satisfazer a fórmula analisada; ou uma lista de condições para que a fórmula seja satisfeita.

`-v`
Esta opção permite as informações sobre o progresso da avaliação seja relatado através da saída padrão. Como padrão, KRONOS não executa esta opção.

`-f`
Esta opção faz com que o arquivo `aux.tctl` seja criado, contendo a fórmula dada em *string*. Neste caso, o arquivo de saída será `aux.eval`.

O Grafo Temporizado é lido através de *arquivo1*, e a fórmula através de *arquivo2* ou pela entrada padrão. Os sufixos `.tg` e `.tctl` podem ser omitidos, e o arquivo de saída é `arquivo2.eval`.

Comentários podem ser adicionados aos arquivos de entrada, bastando delimitar o texto pelos marcadores /* e */, como é feito na linguagem de programação *C*. Números são positivos inteiros, sem o ponto decimal.

ANEXO D

O SISTEMA SYNTAX

Neste anexo, serão apresentados os conceitos básicos do Sistema SYNTAX [BD88, BD89], ferramenta que permite a produção de tradutores, implantada sobre UNIX.

D.1 INTRODUÇÃO

O Sistema SYNTAX agrupa um conjunto de ferramentas que facilitam a concepção e a realização de tradutores, principalmente no domínio da compilação. Os objetivos são os mesmos apresentados na definição dos utilitários LEX e YACC, do UNIX [Sun88], porém, este sistema possui uma arquitetura mais estruturada, em particular, no tratamento de erros.

SYNTAX compreende principalmente dos 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.

D.2 O INTRODUTOR SINTÁTICO - BNF

BNF é o processador de base, que transforma as definições sintáticas para uma forma interna, utilizada pelo sistema. Ele trata de uma gramática independente do 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 proxíma a "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, ele deve conduzir à uma cadeia terminal

(eventualmente vazia), e um não-terminal não deve derivar dele mesmo ($N \overset{+}{\Rightarrow} N$), mas a recursividade é permitida.

BNF aceita gramáticas ambíguas, com a condição de que estas ambiguidades 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 *term6* da gramática da linguagem RT-LOTOS, apresentada no anexo B, deve ser representada da seguinte forma:

```
<TERM6>      = <ACT> <PREFIX_OP> <TERM6>
             | <FACTOR>
             ;
<ACT>        = %IDENT
             | "[" %INTEGER "," %INTEGER "]" %IDENT
             | "[" %INTEGER "]" %IDENT
             ;
<PREFIX_OP>  = ";" ;
<FACTOR>     = <NULLARY_OP>
             | "(" <BODY> ")"
             | <ACT>
             | <PROC_INST>
             ;
```

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, sendo que a sua identificação deve ser feita através das aspas, ou iniciados pelo caracter "#", também com a possibilidade de serem definidos ao nível sintático; e os terminais genéricos, que são os

identificadores que representam um conjunto teoricamente infinito de possibilidades de ocorrência, iniciados pelo caracter "%", sendo que estes devem ser definidos obrigatoriamente ao nível léxico.

Comentários podem ser incluídos à definição da gramática, bastando iniciar cada linha de comentário com o caracter "*". Também é possível associar a semântica a cada produção da gramática, mas BNF tratará somente da interpretação das regras sintáticas, sendo o tratamento da semântica feito através de módulos específicos (SEMACT, SEMAT, 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.

D.3 CONSTRUÇÃO DO ANALISADOR LÉXICO - LECL

LECL é o construtor léxico do sistema SYNTAX, 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 seria 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 SYNTAX, como por exemplo a classe *LETTER*, que representa todas as letras do alfabeto, maiúsculas e minúsculas. A seguir, é apresentado um exemplo de definições de algumas classes:

```
Classes
      all_less_EOL      = ANY - EOL;
      keyword_one       = ONE;
      keyword_two       = "t" "w" "o";
      DIGIT             = "0".."9";
```

A classe `all_less_EOL` define que será igual a classe pré-definida `ANY`, que representa o conjunto dos caracteres ASCII, menos a classe pré-definida `EOL`, que representa o código de final de linha. A classe `keyword_one` define a palavra-chave `"ONE"`, sendo que as letras podem ser tanto maiúsculas como minúsculas. A classe `keyword_two` define a palavra-chave `"two"`, mas apenas com letras minúsculas. E a classe `DIGIT` define o conjunto dos dígitos, que vão de 0 à 9.

Outra forma de simplificar a estrutura do texto é utilizando as abreviações, que são iniciadas pela palavra-chave *Abbreviations*. Como exemplo, a definição de identificadores pode ser feita da seguinte forma:

```
Abbreviations
      IDENTIFIER = LETTER { ["_"] (LETTER | DIGIT)};
```

Assim, um identificador é definido como sendo uma cadeia de caracteres que começa com uma letra do alfabeto, podendo ser seguida de ou mais caracteres "_", letras e dígitos, terminando sempre com uma letra ou dígito.

A declaração das expressões regulares é iniciada pela palavra-chave *Tokens*. É bom lembrar que todos os terminais genéricos devem ser definidos. Assim, para os terminais genéricos dados na seção anterios, as seguinte definições são válidas:

```
Tokens
      %INTEGER     = {DIGIT}+;
      %IDENT       = IDENTIFIER @Lower_Case;
```

Estabelecendo que um número inteiro é constituído de um ou mais dígitos, e que um identificador possui o formato dado pela abreviação `IDENTIFIER`, sendo que as letras maiúsculas e minúsculas não são diferenciadas, e a ação pré-definida `@Lower_Case` estabelece a transformação de todas as letras para minúsculo, após o reconhecimento.

Em alguns casos, pode ocorrer um conflito entre duas ou mais expressões regulares, como no exemplo apresentado a seguir:

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

Neste caso, a palavra-chave `"behaviour"` pode ser reconhecida tanto pela primeira expressão como pela segunda, ou seja, o conflito á 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".

Portanto, para resolver os conflitos acima, são acrescentadas as seguintes definições:

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

O segundo conflito apresentado também pode ser resolvido utilizando restrições de contexto, que estabelecem o que pode suceder uma expressão.

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

O texto de definições pode conter comentários, identificados através dos caracteres "--", que marcam o início do mesmo. O final de um comentário será sempre o final da linha.

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*.lc.l, que contém uma listagem completa da análise feita sobre as definições, relatando os possíveis erros.

D.4 CONSTRUÇÃO 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 com pilhas.

A classe de gramática aceita pelo sistema é a LALR(1). Os possíveis conflitos entre as regras da gramática podem ser tratados através do módulo PRIO, porém, é aconselhável procurar contornar tais conflitos para tornar o analisador mais simples. Em [ASU86], é apresentada uma maneira de modificar uma gramática de tal forma a eliminar os conflitos existentes.

Os conflitos possíveis são:

- conflitos Reduce / Reduce: estes conflitos indicam que existem várias possibilidades de reconhecimento simultâneo dos lados da direita das regras envolvidas (reduce); e
- conflitos Shift / Reduce: estes conflitos indicam 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.

Como exemplo, seja a seguinte gramática de expressões:

```
<EXPR> = <EXPR> + <EXPR>;
<EXPR> = <EXPR> - <EXPR>;
<EXPR> = <EXPR> * <EXPR>;
<EXPR> = <EXPR> / <EXPR>;
<EXPR> = - <EXPR>;
<EXPR> = %IDENT;
```

Para resolver os conflitos existentes, as seguintes alterações são necessárias:

```
%left + -
%left * /
<EXPR> = - <EXPR>; %prec *
```

onde é definida a prioridade dos quatro operadores básicos da aritmética, ou seja, todos os operadores são associativos à esquerda, sendo que a adição e a subtração são menos prioritárias que a multiplicação e a divisão. A expressão `%prec *` estabelece que a regra possui a mesma associatividade e prioridade do operador de multiplicação.

Neste trabalho, o módulo PRIO não foi utilizado, pois todos os conflitos foram contornados, utilizando a abordagem apresentada em [ASU86].

PRIO gera como resultado o arquivo *name*.dt, contendo tabelas internas de tratamento de

ambiguidades, utilizadas pelo módulo CSYNT.

CSYNT gera como resultado: 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 os possíveis erros.

D.5 CONSTRUÇÃO 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 herdadas 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, será dada uma descrição mais detalhada deste módulo.

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.

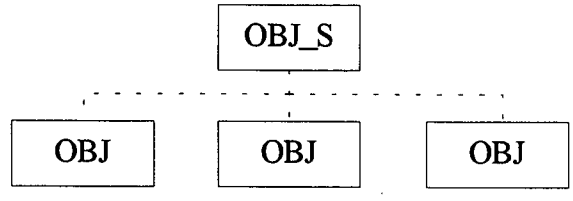


Figura D.1 - Exemplo de uma árvore abstrata

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

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, o pais, os irmãos e os filhos deste nó. É, portanto, possível 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 os 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 o arquivo *name*.att, que contém 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.

D.6 TRATAMENTO DE ERROS - RECOR

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

- uma 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, sendo que 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.

D.7 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 na 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.

D.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).