Tornando LOTOS executável

Humberto Maia Lima
CPqD TELEBRÁS *

Maurício F. Magalhães
DCA FEE UNICAMP †

28 de outubro de 1988

1 Introdução

Nos últimos anos têm sido apresentadas várias formas para especificação de sistemas de comunicação de dados, cada uma argumentando suas "vantagens" próprias, como justificativas para sua existência: "independente de implementação", "expressividade", "de compreensão fácil", "implementação derivada facilmente". Com isto surgiram desde especificações bastante abstratas [11] até especificações em linguagens de programação convencionais como Pascal e C. É importante observar que tanto interesse é justificado pela necessidade de se ter uma especificação clara e exata do sistema. Face à diversidade de técnicas oferecidas, órgãos internacionais se empenharam na busca de uma técnica de descrição formal (TDF) padrão. A ISO (International Standard Organization) e também o CCITT (Comité Consultatif International Télégraphique e Télephonique) padronizaram a linguagem LOTOS para especificação dos protocolos e serviços para interconexão de sistemas abertos. LOTOS (Language of Temporal Ordering Specification) foi desenvolvida no período de 1981-1986 por experts do ISO/TC97/SC21/WG1/FDT/subgrupo C. A idéia básica em LOTOS é expressar o sistema através da relação temporal entre as interações que constituem o comportamento observável do sistema. Estes conceitos foram inicialmente apresentados por Robin Milner [6] em seu trabalho "A calculus of Communicating Systems" (CCS), de onde LOTOS herdou praticamente todos os conceitos básicos. Entretanto outro trabalho correlato, desenvolvido paralelamente ao trabalho de Milner por C. A. L. Hoare [2], também contribuiu no desenvolvimento de LOTOS.

*Caixa Postal 1579 CEP 13085, Campinas, SP
†Caixa Postal 6101 CEP 13081, Campinas, SP

LOTOS utiliza um segundo componente, para descrição dos dados e valores. Esta parte de LOTOS é baseada em teoria formal de tipos de dados abstratos, sendo a parte de conceitos inspirada na técnica descrita em ACT ONE [3].

LOTOS é assim constituída basicamente de uma linguagem cuja parte de controle é baseada em CCS, e uma linguagem para descrição de dados abstratos representada por ACT ONE.

Ainda que não tenha sido o objetivo principal, LOTOS foi desenvolvida de modo a ser executável. Há vantagens e desvantagens de ter especificações executáveis. Através da execução da especificação é possível se ter um protótipo do sistema numa fase inicial do desenvolvimento, permitindo uma avaliação do mesmo ainda neste estado inicial. Entretanto uma especificação executável pode forçar o especificador a tomar decisões de interesse apenas da implementação. Em [12] [10] [13] [14] são evidenciados outros aspectos a respeito de especificações executáveis.

Neste trabalho são apresentadas as várias etapas necessárias a fim de se executar uma especificação LOTOS. Assume-se que o leitor tem um conhecimento da linguagem; entretanto, é feita uma pequena apresentação de LOTOS para tornar o trabalho completo.

2 A linguagem LOTOS

Em LOTOS, sistemas distribuídos são descritos em termos de processos. Cada processo pode ser refinado e ser representado através de sub-processos que por sua vez também podem ser refinados. Assim, uma especificação em LOTOS é formada por um conjunto de processos organizados hierarquicamente através de refinamentos sucessivos. O comportamento do processo é descrito através de expressões de comportamento $[EC]$. Uma expressão EC_1 pode ser associada a uma expressão EC_2 originando uma nova expressão EC. As regras de formação de novas expressões a partir de expressões já existentes são dadas pelos "operadores", que formam parte essencial da linguagem.

O formato de um processo em LOTOS é o seguinte:

Process <*nome*> <*parâmetros*> :=
EC
Endprocess

Onde <*nome*> é o nome através do qual o processo pode ser referenciado, <*parâmetros*> são os parâmetros formais do processo, e EC é a expressão de comportamento que representa o processo.

Em LOTOS a comunicação entre processos é feita através de um mecanismo síncrono chamado "interação". A comunicação se dá através de uma oferta de

interação em uma determinada porta (gate). Assim, uma oferta de interação em uma porta p pode ser como segue:

```
p ? x:int ! true
```

Isso significa que o processo oferece uma interação na porta *p*, e são negociados dois valores: *x* que é do tipo *int* e não possui valor conhecido, e o valor *true*. Se um outro processo oferece:

```
p ! 3 ?y:bool
```

A interação entre os dois processos é possível pois ambos referenciam a mesma porta *p*, e há um casamento entre a informação (dados) trocada. Do resultado da interação teríamos em cada porta p $<3,true>$, onde 3 seria associado ao *x* do primeiro processo e *true* ao *y* do segundo processo. Nem sempre são necessários valores conhecidos para haver uma interação. A tabela 1 descreve os tipos possíveis de interação entre processos.

Proc A	Proc B	condição	tipo de int.	efeito
!E1	!E2	Valor(E1)= Valor(E2)	Casamento de valores	sincronização
!E	?X:t	Valor(E) ∈ domínio(t)	Passagem de valor	sincronizaçao com x=Valor(E)
X:t	Y:u	t=u	Geração de valor	sincronização com X=Y=V, onde V ∈ domínio(t)

Tabela 1: Tipos de interação

Um processo que oferece uma interação em uma porta aguardará até que um outro processo ofereça uma interação que case. Se isto não ocorrer, o processo permanecerá em espera infinita (deadlock).

A oferta de interação é o mecanismo básico de construção de expressões de comportamento em LOTOS. Outras expressões mais complexas podem ser construidas através de composição de expressões básicas com uso de *operadores*. Esses *operadores* permitem exprimir seqüencialidade, alternativa, paralelismo, internalização de portas, habilitação e desabilitação como descritos a seguir:

- *seqüencialidade* : o operador ";" é usado para prefixar uma *EC* de uma oferta de evento, indicando que inicialmente o processo oferece o evento e em seguida continua com a *EC* seguinte.

Por exemplo:

$$\overbrace{\overbrace{p?x : int!true}^{evento}}^{antecede} \; ; \; \overbrace{EC}^{expressão}$$

- *alternativa*: o operador "||" é usado para exprimir várias alternativas possíveis para o processo. Assim, se EC_1, EC_2, ..., EC_n são expressões de comportamento, $(EC_1 \,||\, EC_2 \ldots ||\, EC_n)$ significa que o processo tem como alternativa seguir qualquer uma das expressões de comportamento listadas.

  A alternativa será escolhida segundo um dos critérios:

  - totalmente indeterminístico, isto é, mais de uma alternativa é possível de ser escolhida, e a escolha será totalmente não determinística;
  - determinada por evento ofertado, isto é, se apenas uma dada alternativa contém um evento inicial, e está sendo oferecido um evento complementar (a sincronização é possível) esta expressão será a escolhida;
  - determinada por uma condição de guarda, isto é, se uma expressão é "guardada" (contém uma condição a ser satisfeita) então a expressão será escolhida se a condição for satisfeita

- *paralelismo*: os operadores "|||", "||", "||p||" são três formas disponíveis para expressar paralelismo. Se EC_1 e EC_2 são expressões de comportamento, EC_1 *op* EC_2, onde *op* é qualquer um dos operadores significa uma composição paralela de EC_1 e EC_2, onde EC_1 e EC_2 serão executadas paralelamente. O operador utilizado determina a relação de dependência entre as expressões paralelas no que diz respeito às portas em que elas podem sincronizar. No primeiro caso as duas expressões não se sincronizam em nenhuma porta comum. No segundo caso, em todas as portas comuns e finalmente no terceiro caso apenas nas portas referenciadas entre "||" e "||".

- *habilitação*: o operador ">>" habilita uma expressão EC_2 após a execução com sucesso de uma expressão EC_1. Ou seja, $EC_1 >> EC_2$ significa que seqüencialmente a execução com sucesso de EC_1, EC_2 estará habilitada a ser executada.

- *desabilitação*: O operador "|>" é utilizado para permitir que uma expressão de comportamento EC_1 seja interrompida e uma outra seja iniciada. Por exemplo, se $E = EC_1 \,|> EC_2$, na ocorrência da oferta do evento "*p*"

nome	expressão de comportamento
inaction	stop
termination	exit
action – prefix	$a; B$
	$i; B$
choice	$B1 \| \| B2$
composition	$B1 \| [a_1, \ldots, a_n] \| B2$
	$B1 \| \| B2$
	$B1 \| \| \| B2$
hiding	$HIDE\ a_1, \ldots, a_n\ IN\ B_1$
enabling	$B1 >> B2$
disabling	$B1 \| > B2$
instantiation	$p < a_1, \ldots, a_n >$

Tabela 2: Sintaxe LOTOS

pelo ambiente, onde p é um evento inicial de EC_2 ($EC_2 = p; EC_2'$), o comportamento da expressão E será dada por EC_2.

internalização de portas: uma das características de LOTOS é poder representar um sistema como uma caixa preta onde só são visíveis as portas através das quais o sistema se comunica com o ambiente. É conveniente então possibilitar esconder as comunicações entre processos dentro do sistema, ou dentro de outros processos, restringindo a visibilidade das portas a um determinado escopo. O operador "HIDE" permite delimitar a visibilidade de portas. Assim, na construção HIDE $p_1, p_2, \ldots, p_n$ IN EC, onde $p_1, p_2, \ldots, p_n$ são portas e EC é uma expressão de comportamento, as portas $p_1, p_2, \ldots, p_n$ serão visíveis apenas no escopo de EC, estando escondidas fora dela.

A sintaxe LOTOS pode ser vista na tabela 2.

3 Características de um simulador LOTOS

Um simulador LOTOS pode ser visto como uma máquina capaz de possibilitar a "animação" de uma especificação, permitindo que o comportamento dinâmico do sistema seja observado. Algumas características desta máquina são:

- permitir a apresentação do estado atual: o estado atual é representado por um nó, da árvore de execução[1] , ou um cojunto de nós no caso de expressões

[1] a árvore de execução será descrita em item a seguir

paralelas;

- permitir a apresentação dos próximos eventos possíveis referente ao estado atual: esta lista de eventos representa os comportamentos como possíveis sincronizações internas entre processos, eventos fornecidos sem sincronização interna (possível sincronização com o ambiente) ou alternativas em uma escolha

- evoluir para próximo estado: esta evolução é baseada na lista de ações possíveis no estado atual e eventuais escolhas determinadas pelo usuário do simulador ou através de escolha não-determinística pela própria máquina

- regredir ao estado anterior: a regressão ao estado anterior permite a experimentação de outros comportamentos do sistema através de uma escolha de uma ação que leva a um estado diferente do atual

Além dessas características mais internas da máquina, esta deve possuir uma interface com o usuário de modo a permitir completo domínio das fases de execução como:

- pontos de parada: pontos de parada são estados (ou nós) da árvore de execução onde a simulação deve ser interrompida caso a máquina evolua até este estado

- registro de histórico de execução: o registro de histórico de execução permite a obtenção de uma parte finita da árvore de comportamento dinâmico do sistema

4 Semântica dinâmica de LOTOS

A semântica dinâmica de LOTOS, ou seja, o comportamento esperado durante a fase de execução, é descrita em termos de regras de inferências e axiomas . Estas regras são definidas usando uma relação $-a \rightarrow$ entre expressões de comportamento. Se EC_1 e EC_2 são expressões de comportamento, então $EC - a \rightarrow EC'$ significa que o comportamento de EC após a ocorrência de "a" é dado por EC'. Por exemplo, para o operador ";" temos o seguinte:

$EC = g; EC'$, $EC - g \rightarrow EC'$ é um axioma.

Ou seja, após a ocorrência do evento "g" o comportamento de toda a expressão será EC'. Da mesma forma o comportamento de expressões compostas podem ser inferidos a partir do comportamento de seus componentes. A tabela 3 mostra um resumo das regras de inferência para os demais operadores. Uma descrição mais precisa pode ser encontrada em [1].

nome	expressão B	axiomas e regras de inferências
inaction	$STOP$	nenhuma regra
termination	$EXIT$	$B - \delta \rightarrow STOP$
action-prefix	$a; B'$	$B - a \rightarrow B'$
	$i; B'$	$B - i \rightarrow B'$
choice	$B_1 \Box B_2$	$\frac{B_1 - a \rightarrow B_1'}{B - a \rightarrow B_1'}$, $\frac{B_2 - a \rightarrow B_2'}{B - a \rightarrow B_2'}$
composition	$B_1 \|[g1,...,gn]\| B_2$	$\frac{B_1 - a \rightarrow B_1', a \notin \{g1,...,gn\}}{B - a \rightarrow B_1' \|[g1,...,gn]\| B_2}$
		$\frac{B_2 - a \rightarrow B_2', a \notin \{g1,...,gn\}}{B - a \rightarrow B_1 \|[g1,...,gn]\| B_2'}$
		$\frac{B_1 - a \rightarrow B_1', B_2 - a \rightarrow B_2', a \in \{g1,...,gn\}}{B - a \rightarrow B_1' \|[g1,...,gn]\| B_2'}$
	$B_1 \|\|\| B_2$	$\frac{B_1 \|[]\| B_2 - a \rightarrow B'}{B - a \rightarrow B'}$
	$B_1 \| B_2$	$\frac{B_1 \|[g1,...,gn]\| B_2 - a \rightarrow B'}{B - a \rightarrow B'}$
hiding	$HIDE\ g1,...,gn\ IN\ B'$	$\frac{B' - a \rightarrow B'', a \notin \{g1,...,gn\}}{B - a \rightarrow HIDE\ g1,...,gn\ IN\ B''}$
		$\frac{B' - a \rightarrow B'', a \in \{g1,...,gn\}}{B' - i \rightarrow HIDE\ g1,...,gn\ IN\ B''}$
enabling	$B_1 >> B_2$	$\frac{B_1 - a \rightarrow B_1', a \neq \delta}{B - a \rightarrow B' >> B_2}$, $\frac{B_1 - \delta \rightarrow B_1'}{B - i \rightarrow B_2}$
disabling	$B_1 \|> B_2$	$\frac{B_1 - a \rightarrow B_1', a \neq \delta}{B - a \rightarrow B_1' \|> B_2}$, $\frac{B_1 - \delta \rightarrow B_1'}{B - \delta \rightarrow B_1'}$
		$\frac{B_2 - a \rightarrow B_2'}{B_2 - a \rightarrow B_2'}$

Tabela 3: Semântica Dinâmica de LOTOS

5 Arquitetura do simulador

A arquitetura do simulador aqui descrito é mostrada na figura 1. Ele é bastante semelhante a interpretadores para linguagens convencionais, no sentido em que a partir de uma forma textual do programa é gerada um forma intermediária, computável, sobre a qual é feita a interpretação.

Conforme a figura1, pode-se verificar que os módulos funcionais do simulador são:

- Analisador sintático: executa a análise sintática e gera a partir da forma textual da especificação uma árvore sintática abstrata.
- Analisador semântico: utilizando a árvore gerada pelo analisador, efetua todas as verificações relativas à semântica estática das especificações. Gera uma representação interna, chamada de "árvore de execução".
- Máquina de execução: a máquina de execução é o núcleo do interpretador. A partir das informações existentes na árvore de execução a máquina permite que o comportamento dinâmico do sistema seja observado.
- Interface com o usuário: o objetivo deste módulo é fornecer, como sugere o nome, uma interface através da qual o usuário do simulador interfira na execução através de opções. As opções desejadas pelo usuário são examinadas pela máquina de execução durante o processo de simulação.

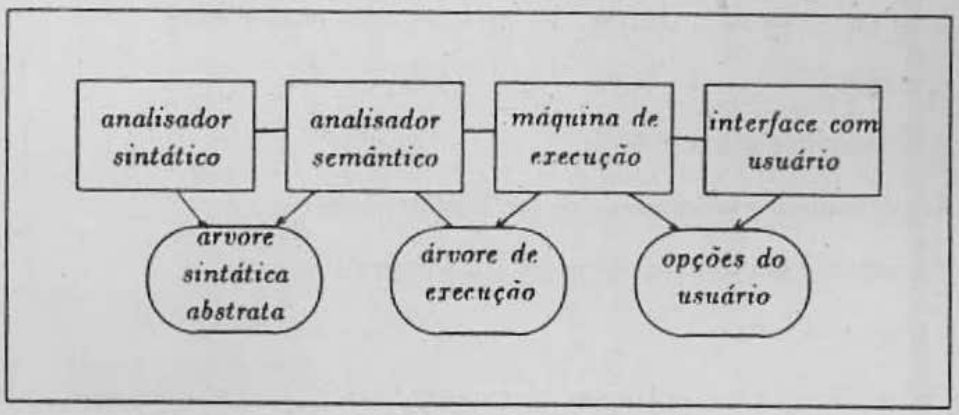


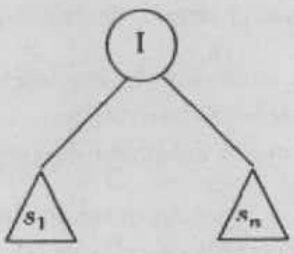
Figura 1: Estrutura do simulador

Nos ítens seguintes serão discutidas em maiores detalhes a árvore e a máquina de execução, visto que estes representam o núcleo da execução de LOTOS.

6 A árvore de execução

A execução da especificação LOTOS é obtida através de uma "máquina" que percorre uma árvore. As características desta árvore devem refletir a semântica estática da especificação de onde foi derivada. O comportamento dinâmico é obtido pela interpretação desta árvore.

A árvore de execução LOTOS é uma representação abstrata da especificação, onde os elementos sintáticos (isto é, os elementos apenas de forma) foram retirados, mas todo conteúdo semântico foi mantido. O formato das informações na árvore é fortemente influenciado pelas características do método de simulação utilizado. O objetivo da árvore é facilitar a implementação do simulador, e seu formato geral é o seguinte:



Onde I é um nó contendo uma instrução e $S_1, \ldots, S_n$ são sub-árvores com o mesmo formato. Um nó pode não ter nenhuma sub-árvore associada. À instrução existente no nó estão as seguintes informações e objetos associados:

- P, conjunto de portas visíveis
- S, conjunto de sub-árvores
- A, conjunto das ações possíveis (ou a ação possível)
- O, conjunto dos objetos das ações (sub-árvores)
- i, código da instrução

As instruções são derivadas dos operadores LOTOS. A cada instrução está associado um objeto sob o qual a instrução opera. São as seguintes as instruções:

- SEQ: equivalente ao operador ";" e está associado ao um objeto que representa um "gate" e eventualmente uma oferta de evento. Esta instrução é básica no processo de simulação e é portanto chamada de *primitiva*.

- CHOICE: equivalente ao operador "||", tem como objeto associado um conjunto de sub-árvores que representam as diversas possibilidades de evolução.
- HIDE: equivalente ao operador "HIDE", tem como objeto uma lista de portas. Esta instrução provoca uma alteração de contexto na evolução da sub-árvore seguinte.
- INST: equivale a instanciação de processos, tem como objeto uma lista de portas e uma lista de valores, relativos aos parâmetros reais da instanciação, e a sub-árvore que descreve o processo instanciado.
- DISR, ENAB: equivalem aos operadores "|>" e ">>" respectivamente e têm como objeto uma sub-árvore; são relativos às operações de habilitação ou "disrupt".
- PAR: equivale aos operadores "|||", "||" e "|||", tem como objeto uma lista de portas e uma lista de sub-árvores. Esta instrução provoca uma multiplicação da máquina de execução e é portanto chamada de "expansão".
- EXIT: equivale ao operador "EXIT", tem como objeto uma lista de expressão de valores.
- STOP: equivale ao operador "STOP". Não possui nenhum objeto associado e indica que o processo que a executa está inativo.
- ENTER: esta instrução não tem um operador LOTOS equivalente . Ela identifica o ponto de entrada na instanciação de processos.

A figura 2 é um exemplo de um processo produtor/consumidor especificado em LOTOS [17]. A respectiva árvore de execução está na figura 3.

7 A máquina de execução

Neste item é descrito o núcleo do simulador, chamado aqui de "máquina de execução". A execução de uma especificação é uma tarefa de relativa complexidade, pois a linguagem LOTOS oferece operadores bastante poderosos e a combinação desses operadores pode gerar expressões de comportamento bastante complexas. Além disso, LOTOS permite construções paralelas, isto é, expressões de comportamento que se intercalam no tempo. A linguagem também oferece a possibilidade de se parametrizar a especificação, sendo os parâmetros reais conhecidos apenas em tempo de execução. Os núcleos de simuladores de LOTOS (ou de outras linguagens de comportamento) citados em literatura foram implementados em sua maioria em PROLOG [5] [4] [7] e PARLOG [8] [9],

```
process produce [a,b,c,d] : noexit :=
    item available [ a,b,c ]
     || item acceptable[a,b,d]
where
    process item_available[a,b,c] : noexit :=
       a ; (   b ; item available[a,b, c]
           [] c  ; item_available[a,b,c]
           )
    endproc

    process item_acceptable[a,b,d] : noexit :=
       a ; (   b ; item_acceptable[a,b,d]
           [] c  ; item_acceptable[a,b,d]
           )
    endproc
endproc
```

Figura 2: exemplo LOTOS de um produtor/consumidor

havendo implementações também em LISP [15], utilizando-se da facilidade oferecida por esssas linguagens que é ter a "parte de controle" embutida dentro da própria linguagem. A abordagem utilizada no simulador aqui descrito difere das anteriores por utilizar uma linguagem de programação imperativa (no caso, a linguagem "C", [16]), para implementação de todo o simulador, inclusive o "núcleo de execução". O uso de "C" permite maior portabilidade para o simulador além do que possibilita uma melhor performance especialmente em computadores menores onde implementações de PROLOG tendem a ser bastante ineficientes. Outra vantagem do "C" é a flexibilidade oferecida pela linguagem para implementação de algoritmos específicos.

O processo de funcionamento da máquina de execução é baseado na idéia da combinação de sub-máquinas que são idênticas entre si e que executam paralelamente. Assim, a máquina de execução é composta de:

- um conjunto de sub-máquinas SM e
- um estado global

A cada sub-máquina sm $\in$ SM estão associados:

- um estado local, que é um nó na árvore de execução que representa a expressão de comportamento que está sendo executada por esta sub-máquina

- um contexto local

O contexto local de cada sub-máquina é representado pelo conjunto de informações dinâmicas acumuladas durante a execução. Este contexto é formado pelos nomes reais das portas (incluindo as restrições de visibilidade e sincronização), expressão de valor associada à variáveis e a lista de expressões de comportamento (se houver) geradas pelas instruções DISR e ENAB.

O algorítmo de uma sub-máquina (simplificado) é dado a seguir, com base nas instruções:

- SEQ: altera o contexto global, oferecendo sincronização na porta associada
- HIDE: altera o contexto local, modificando a visibilidade das portas na lista associada
- DISR, ENAB: alteram o contexto local, inserindo a sub-árvore associada na lista correspondente
- INST: guarda a posição na sub-árvore atual e aplica o mesmo algorítmo na sub-árvore associada. Altera o contexto local com os parâmetros reais da instanciação do processo
- EXIT: recupera a posição na sub-árvore onde foi executada a instanciação. Altera o contexto local com os valores fornecidos na lista de valores associada (funcionalidade do processo). Aplica o mesmo algorítmo na sub-árvore recuperada
- CHOICE: para cada sub-árvore associada, aplica algorítmo de execução recursiva até que a instrução seja SEQ. Altera o contexto global fornecendo o conjunto de sincronizações possíveis (mas mutuamente excludentes)
- PAR: o contexto local é alterado e para cada sub-máquina associada à instrução é criada uma sub-máquina que passa a executá-la. Cada sub-máquina herda o contexto local da sub-máquina que a criou. Essa instrução portanto expande o número de sub-máquinas existentes, formando uma árvore onde somente as sub-máquinas nas folhas são "ativas". As sub-máquinas são nós da árvore.

O estado global da máquina de execução é dado pela tupla $< O, COR, PROX >$ onde:

- O é o conjunto das ofertas de sincronização
- COR é o conjunto dos estados correntes locais das sub-máquinas

- *PROX* é o conjunto dos próximos estados alcançáveis a partir de um estado corrente tal que $< cur, o, prox >$ onde cur $\in$ CUR, o $\in$ O e prox $\in$ PROX.

A tupla $< cur, o, prox >$ é chamada de *transição*. O algorítmo global da máquina de execução é portanto evoluir cada sub-máquina até o ponto em que seja ofertada uma sincronização e em seguida evoluir para um novo estado global da máquina através das transições possíveis.

8 Conclusão

O presente trabalho teve como objetivo traçar os pontos básicos para implementação de um simulador para a linguagem de especificação LOTOS. A viabilidade desta atividade foi comprovada a partir do estudo da própria linguagem e da tendência mundial nesta área. Do simulador aqui descrito já se encontra implementada toda a parte relativa à verificação da sintaxe e semântica estática de LOTOS. O trabalho está prosseguindo em duas áreas: a da análise/simulação dinâmica e a da implementação da parte relativa a dados. Sendo assim mudanças em premissas aqui estabelecidas podem vir a acontecer. Dada a complexidade da linguagem ACT ONE, a parte de LOTOS refente a dados será implementada com com simplificações.

Referências

[1] ISO, Information Processing Systems, Open System Interconnection, *LOTOS - A formal description technique based on the temporal ordering of observational behaviour*, ISO IS 8807, 1988

[2] C.A.R. Hoare, *Communicating sequencial process*,Prentice-Hall Intl, 1985.

[3] H. Ehrig, B. Mahr, *Fundamentals of algebraic specification 1*, Springer-Verlag,Berlin,1985

[4] J.P. Briand, M.C. Fehri, L. Logrippo, A. Obaid, *Executing LOTOS specifications*, Protocol Specification, Testing, and Verification V, North-Holland, Amsterdan, 1986.

[5] B. Berthomieu,*Le Langage CCS et son interprete: Une implementation experimentale de CCS*, Rapport Technique CNRS-LAS, Toulouse, 1985.

[6] R. Milner,*A calculus of communicating systems*,LCNS 92, Springer-verlag, Berlin, 1980.

[7] A. Obaid,*Aplication of the inference rules of CCS and LOTOS*, University of Ottawa Technical Report TR-85-14, Ottawa,1985.

[8] D. R. Gilbert,*Implementing LOTOS in PARLOG*, MSc thesis, Department of Computing, Imperial College, London , 1986.

[9] D. R. Gilbert,*Executable LOTOS: using Parlog to implement a FDT*, Protocol Specification, Testing, and Verification V, North-Holland, Amsterdan, 1986.

[10] D. P. Sidhu, *Protocol Verification Via Executable Logic Specifications*, Protocol Specification, Testing, and Verification III, North-Holland, Amsterdan, 1983.

[11] R. L. Schwartz, P.M. Melliar, *From state machines to temporal logic : Specification methods for protocol standards*, IEEE Transactions on Comunnications, December 1982

[12] G.D. Schultz, D.B. Rose, C.H. West, J.P. Gray, *Executable description end validation of SNA*, IEEE Transactions on Comunnications, April 1980

[13] B. Meandzija, *Towards the automatic generation of executable specification for communications sistems*, IEEE Workshop Languages for Automation, November 1984

[14] H. Ural, R. L. Probert, *Stepwise validation of communication protocols and services*, Computer Networks and ISDN systems, November 1986.

[15] P. Van Eijk,*A comparison of behavioural language simulators*, Protocol Specification, Testing and Verification V,North-Holland, Amsterdan, 1986.

[16] B. W. Kernigham, D. M. Ritchie,*The C programming Language*, PrenticeHall, New Jersey, 1978

[17] E. Brinksma, *A tutorial on LOTOS*, in: ISO IS 8807 annex C, 1988

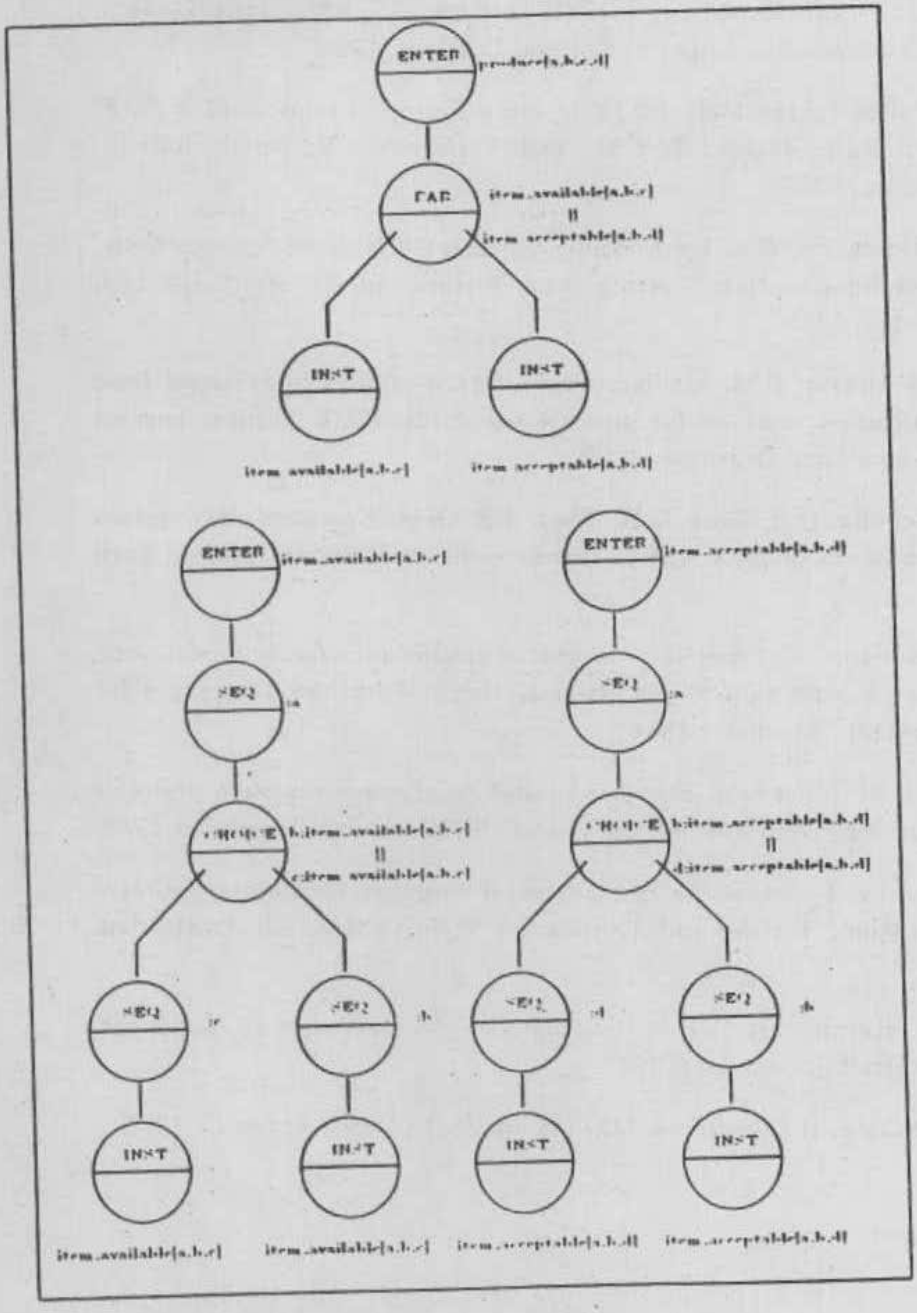


Figura 3: Árvore de execução LOTOS