" Um Simulador Prolog para Expressões LOTOS

Gualberto Rabay Filho* Maurício Ferreira Magalhães**

Walter da Cunha Borelli*

Faculdade de Engenharia Elétrica
Universidade Estadual de Campinas
Cx Postal 6101 CEP13081

Resumo: A linguagem LOTOS se caracteriza por possuir um forte formalismo algébrico, que permite especificações com bastante clareza e sem ambiguidades. Apesar de ter sido projetada para ser executável, esta característica ainda não foi bastante explorada. Existe uma relação entre a semântica operacional de LOTOS e a linguagem Prolog que é mostrada neste trabalho. Propomos um simulador, escrito em Prolog, que torna LOTOS executável, possibilitando explorar suas características. Após uma breve introdução à programação lógica apresentamos um resumo da sintaxe e semântica de LOTOS. Finalmente detalhamos as relações entre LOTOS e Prolog através da implementação de um validador de expressões em LOTOS.

1. Introdução

Os sistemas de comunicação de dados vêm aumentando em complexidade e em diversidade de equipamentos, exigindo um esforço maior no desenvolvimento de software de comunicação. As técnicas de descrição formal vêm sendo utilizadas de maneira crescente no sentido de permitir a construção de especificações de sistemas com bastante clareza e sem ambiguidades. Na ISO, através do grupo de trabalho ISO/TC97/SC21/WG1 foi desenvolvida a linguagem LOTOS [1] como uma das linguagens adequadas à especificação de sistemas através de técnicas de descrição formal. A linguagem LOTOS possui uma estrutura dinâmica baseada no CCS de Milner [2], com idéias do CSP [3], e uma estrutura estática de representação de dados que utiliza a linguagem de tipos abstratos ACT ONE [4].

---

* Depto. de Telemática
** Depto. de Automação
Este trabalho teve o apoio do CPqD da Telebrás

O princípio da linguagem LOTOS é o da execução ordenada de ações atômicas observáveis pelo ambiente. A linguagem possui um grande poder de abstração e um formalismo algébrico rigoroso que possibilita verificações de programas com bastante precisão [1].
Este formalismo por outro lado dificulta a execução automatizada das especificações. Várias universidades e centros de pesquisa vêm desenvolvendo ferramentas automatizadas que possibilitam uma maior divulgação e consequente melhor utilização da linguagem na especificação dos sistemas distribuídos e protocolos de comunicação[5]. Entre as ferramentas construídas, os simuladores para linguagens comportamentais [6], e entre elas LOTOS têm sido discutidos em vários trabalhos [7],[8].

Neste trabalho propomos o desenvolvimento de um simulador para linguagem LOTOS utilizando a linguagem Prolog. Sua concepção baseia-se na derivação de expressões de comportamento. Uma expressão P' pode ser derivada de uma expressão P através da oferta de um evento "a", notação P--a->P', após avaliação dos operadores contidos na expressão P. Semanticamente os operadores da linguagem são representados por um conjunto de regras de inferência. A programação lógica possui uma sintaxe que permite uma representação das regras de inferência de uma forma direta. A linguagem Prolog [9] vem sendo utilizada como uma boa alternativa na construção de ferramentas de simulação e representações executáveis de protocolos, como pode ser visto em [6],[11],[12] e [13].

2. Programação Lógica

A programação lógica permite descrever problemas de uma forma próxima a do raciocínio humano, de maneira formal através de cláusulas. Neste aspecto a programação lógica difere da programação convencional pois se baseia na construção de uma base de informações e a solução de problemas é feita por consultas e deduções sobre este conhecimento. O elemento básico da programação lógica é a cláusula, através da qual é feita a elaboração das soluções. Uma cláusula tem uma forma genérica:

Q1,Q2, ... Qm <---- P1,P2, ...Pn ($n \geq 0, m \geq 0$) onde Q1 ...Qm constitui a cabeça da cláusula e representa as conclusões, e P1...Pn formam o corpo que representa as premissas.

A linguagem Prolog permite uma execução procedural de cláusulas lógicas Horn incorporando os conceitos de programação lógica [10]. As cláusulas Horn se caracterizam por possuir no máximo uma conclusão :

concl <--- prem1,prem2, ... premN

Em Prolog os seguintes tipos de cláusulas são utilizados:

1) concl:-prem1,... premN, significa que a conclusão é verdadeira se cada uma das premissas, ou objetivos, forem verdadeiras.

2) concl. , representa uma cláusula unitária,sem corpo e significa uma asserção ou fato

3) :-prem1,... premN, conhecida como cláusula objetivo, representa um questionamento.

Um programa portanto deve ser composto de uma ou mais cláusulas do tipo 1 e 2, e a execução ativada via um questionamento através de uma cláusula do tipo 3.

Em Prolog os objetos de dados são chamados de termos. Um termo pode.representar uma constante, uma variável ou um termo composto.Um termo composto é caracterizado por um nome e uma sequência de um ou mais termos chamados argumentos. Uma cláusula do tipo termo composto estabelece uma relação entre os argumentos. Por exemplo

pai(josé,joão).

representa uma cláusula constituída por termo composto onde pai representa o nome, e joão e josé os argumentos. A cláusula estabelece uma relação de paternidade entre os argumentos.

A execução de um programa consiste na unificação de cláusulas. O interpretador tenta encontrar uma cláusula que unifique com o problema questionado a partir do topo da base de dados. Encontrada esta cláusula, tenta atender os objetivos do corpo da cláusula, partindo da esquerda para a direita. Para atender estas premissas podem ser chamadas outras cláusulas, obedecendo a ordem de cima para baixo e da esquerda para a direita. Se uma cláusula falha o interpretador tenta, através de um retrocesso, escolher uma outra alternativa para a cláusula que foi executada antes da falha.

O uso de Prolog para implementar regras de inferência que representam a semântica de LOTOS facilita um mapeamento direto destas para cláusulas Prolog. Os detalhes desta implementação serão vistos nos próximos itens.

3. LOTOS

Neste item faremos uma breve descrição da linguagem LOTOS, uma leitura mais completa pode ser encontrada nas referências [1],[9],[14].

Uma especificação em LOTOS representa dinamicamente o comportamento de um sistema de comunicação através de eventos atômicos observáveis que ocorrem de forma ordenada no tempo. A especificação apresenta uma estrutura de dados abstrata e uma parte dinâmica que representa os processos. Esta dualidade permite maior flexibilidade para se utilizar linguagens de tipos de dados diferentes e fazer especificações onde a descrição dos dados possua diferentes níveis de abstração. Uma especificação em LOTOS observa a seguinte sintaxe:

specification [specification identifier]
type definition(*ACT ONE*)

```
        endtype

        process [process identifier]
        [gate list][formal parameters]
            := behaviour expression
        endproc
    endspec
```

A definição de tipos de dados, construída a partir de ACT ONE [4], é constituída de uma parte sintática e uma parte semântica. A parte sintática é caracterizada por uma assinatura que consiste de um conjunto de classes (nomes de portadores de dados) e de um conjunto de operadores. Dada uma assinatura é possível gerar todos os símbolos de uma classe através da execução recursiva dos operadores. Cada símbolo gerado chama-se termo. Como exemplo, a representação dos números naturais é possível a partir da definição de uma constante zero, e aplicação recursiva da operação sucessor.

```
        0:--->nat,
        suc:nat --> nat
(0,suc(0),suc(suc(0)), ...
```

A semântica está relacionada com o conjunto de equações de uma especificação. Uma equação é um par de termos de um mesma classe, relacionada portanto com uma dada assinatura.

Em LOTOS existe disponível uma biblioteca de tipos de dados que auxilia na implementação das especificações.

A parte dinâmica visa modelar algebricamente a comunicação entre processos. [Process identifier] representa o nome pelo qual o processo deve ser chamado durante a especificação. Os gates da [gate list] descrevem a relação de pontos de interação do processo com o ambiente. Os parâmetros são valores trocados nas interfaces entre processos. O corpo do processo é representado pelas expressões de comportamento. Uma expressão de comportamento representa as possíveis ações que um processo pode executar ao se comunicar com o ambiente.

As interações entre processos representam o elemento básico da comunicação em LOTOS. As interações através das portas(gates) que devem ser nomeadas como entrada ou saída, e os tipos de dados que serão passados nas interações devem ser definidos. Na tabela 1 são mostradas as possíveis interações existentes em LOTOS e seus efeitos.

Por exemplo, as interações abaixo

```
...p ?x:int !x+4 ?z:bool !4 ?w:int ...
...p !3 ?y:int !true !4 ?k:int ...
```

sincronizam e após a sincronização os seguintes valores são gerados:

x = 3; y = 7 ; z = true; !4 e !4 apenas sincronizam os dois processos, e k e w assumem um valor inteiro qualquer. Além

destas existem as interações não observáveis pelo ambiente, que são os eventos internos, não determinísticos, representados pelo símbolo " i ".

processo A	processo B	condição sincron.	interação	efeito
g! E1	g! E2	valor(E1) = valor(E2)	casamento de valor	sincronização
g! E	g?x:t	valor(E) ∈ Dom(t)	passagem de valor	após sincronização x=valor(E)
g? x:t	g? y:u	t=u Dom(t) = Dom(u)	geração de valor	após sincronização x = y = v, v é qualquer valor no Dom(t)

TABELA 1 Tipos de Interações em LOTOS

Os operadores da linguagem são mostrados a seguir:

-Ação: o operador ação representado por ";", quando utilizado conjuntamente com um nome "a" constitui um prefixo de uma expressão de comportamento. A expressão a;B , indica que apenas após oferta do evento "a "a expressão B pode ser executada.

-Escolha(Choice): possibilita a escolha não determinística entre ocorrências de eventos. B1 [] B2 especifica um comportamento em que as expressões B1 ou B2 podem ser escolhidas de acordo com os eventos oferecidos pelo ambiente, por eventos internos ou por atendimento de uma guarda.

-Composição: este operador possibilita a execução concorrente entre processos. A expressão P1|[a1, ... an]|P2 significa uma composição entre P1 e P2 sincronizados através da lista de portas a1 ..an.

- Restrição: este operador torna inacessível ao ambiente as portas de um processo, por exemplo P1, que são restritas através da sintaxe "hide a1...an in P1".

-Composição Sequencial: P1>>P2 indica que o processo P2 inicia apenas após o término bem sucedido de P1. Se P1 é uma composição de processos sua terminação com sucesso implica que todos os processos de P1 tenham sido completados com sucesso.

-Desabilitação: este operador permite que um processo P2 possa interromper de forma não determinística um processo P1 se P2 desabilita P1, notação P1 [> P2. Se qualquer evento de P2 ocorrer, P1 não volta a executar.

Além destes operadores LOTOS permite fazer declarações de

valores locais através do comando let x = ... in P1 . As chamadas de processos dentro de uma especificação e a instanciação de parâmetros é semelhante às declarações e chamadas de procedimentos em linguagens procedurais.

O princípio básico da semântica dinâmica de LOTOS é o da derivação de expressões de comportamento. Dizemos que P'é a derivada de P se P --a-->P'.

A representação da evolução das expressões de comportamento pode ser feita através de regras de inferência para o operador prefixo de ação e pela eliminação do rótulo que prefixava a expressão. Se a expressão a;B recebe um evento a, a expressão resultante é B, ou seja a;B -- a --> B. Qualquer representação de operadores em LOTOS pode ser redutível a representações de operadores ação e operadores escolha. A semântica operacional de LOTOS é traduzida através de um conjunto de regras de inferência, que como veremos na seção seguinte permite fazer mapeamento para a linguagem Prolog.

4. Implementação do Simulador

Vimos que uma cláusula Prolog pode ser escrita como: conclusão:- prem1, ... premN. A semântica operacional de LOTOS é escrita através de um conjunto de regras de inferência R, dada por

$$R:\ \frac{P1,\ \ldots\ Pn}{Q}$$

, onde P1 ... Pn são as premissas e Q a conclusão. Esta regra pode ser diretamente reescrita em Prolog da forma:

```
Q:-P1,..Pn.
```

Desta maneira é possível construir um programa Prolog que faça inferências sobre expressões de comportamento escritas em LOTOS. As vantagens de usar Prolog neste caso seriam :

-permitir um mapeamento direto entre as regras de inferência que representam a semântica de LOTOS e o programa.

-como o programa é constituído de uma parte de controle e outra de dados, independentes, facilitar as alterações no controle ou expansão no conhecimento.

O modelo proposto permite através da execução simbólica da especificação verificar se os comportamentos observados correspondem ao projetado. As duas formas de operação do simulador são:

-Validação de Sequências de Eventos: analisa se uma sequência de eventos fornecidos pelo usuário é válida como possível sequência de eventos obtida da execução da especificação.

-Geração de Sequências de Eventos: gera automaticamente conjuntos

de sequências resultantes da execução da especificação. Nesta forma de operação é possível gerar sequências de comprimento finito definido pelo usuário, ou obter as sequências possíveis que fazem a especificação derivar um estado final, definido pelo usuário, a partir de um estado inicial. Este tipo de operação pode gerar sequências de teste para a especificação.

No simulador proposto é utilizada uma sintaxe intermediária para representar a descrição dos processos, que é diretamente executável em Prolog. Os operadores LOTOS têm uma sintaxe própria no simulador e as regras de inferência são traduzidas diretamente em cláusulas Prolog. A sintaxe intermediária considera os processos como listas ordenadas de eventos. Ao executar uma validação de uma de sequência de eventos, o simulador faz uma comparação entre os eventos da sequência definida pelo usuário com os elementos das listas que descrevem o processo. Caso a lista fornecida pelo usuário não corresponda à sequência de execução do processo são apontados os eventos inválidos. Se a comparação for bem sucedida o programa devolve a expressão de comportamento final após execução da lista. Como exemplo mostramos uma tabela resumida com apenas três operadores, mostrando a notação interna, e as regras de derivação associadas.

operador	notação interna	regra de derivação
ação	" ;"	der([a,B],[a|_],B).
composição independente	par	der([B1,par,B2],A,[B1'par B2]:- der(B1,A,B1'). der([B1,par,B2],A,[B1,par,B2']:- der(B2,A,B2'). der([B1,par,B2],A,[B1',par,B2']:- A=[i|_],der(B1,[a1|_],B1'), der(B2,[a2|_],B2'),compl(a1,a2).
escolha	esc	der([esc(B1,B2)],A,B1'):- der(B1,A,B1). der([esc(B1,B2)],A,B2'):- der(B2,A,B2').

A primeira derivação constitui um fato em Prolog. Se a expressão a;B é comparada com uma lista de eventos onde 'a' é a cabeça da lista, a expressão resultante é B. A derivação para a expressão de composição independente B1\|\|B2, é representada internamente por [B1,par,B2]. Ao ser submetida a uma lista de eventos A, três possíveis derivações podem ocorrer:

a)[B1,parB2]--A-->[B1',par,B2], se a lista A possui como cabeça um elemento que derive a expressão B1 pela regra do operador ação;

b)[B1,par,B2]--A-->[B1,par,B2'], se a lista A possui como cabeça um elemento que derive a expressão B2 pela regra do operador ação;

c)[B1,par,B2]--A-->[B1',par,B2'], se a lista A possui como cabeça um evento interno e as listas que representam os processos B1 e B2 possuem elementos complementares(ex:a? e a!);

A derivação para o operador escolha gera B1 ou B2 dependendo se a cabeça da lista de eventos A unifica com o próximo evento de B1 ou de B2. Como a execução em Prolog segue a unificação de cláusulas de acordo com a sua posição na base de dados, foi necessário implementar mecanismos que simulem um não determinismo para a execução de alguns operadores como escolha e composição.

Os operadores têm notação infixa, com exceção do operador esc, conforme mostrado na tabela de operadores.

Como exemplo seja o processo abaixo:

```
process teste(a,b,c,d):=
                        a;b;stop [] c;d;stop
endproc
```

A evolução do processo teste pode ser vista através das possíveis sequências de eventos geradas a partir de sua execução. A execução de eventos de uma lista leva a expressão de comportamento inicial a expressões intermediárias derivadas da expressão inicial.Se for submetida a lista [a,c] ao processo teste teremos:

```
                a
teste(a,b,c,d) ---> teste'(a,b,c,d):=b;stop[]c;d;stop
                c
teste(a,b,c,d) ---> teste''(a,b,c,d):=b;stop[]d;stop
```

No simulador, internamente, a forma de executar a lista acima é a seguinte:

```
:-? der([teste(a,b,c,d)],[a,c],F).
```

a expressão resultante é F=[b,stop,par,d,stop], que corresponde à expressão teste''(a,b,c,d).

Exemplo2

```
process Sender[ConRq,ConCf,DtRq,DisRq):=
        Connection_Phase[ConRq,ConCf];
        Data_Phase[DtRq,DisRq]
where
  process Connection_Phase[ConRq,ConCf]:=
          ConRq;ConCf;exit
  endproc
  process Data_Phase[DtRq,DisRq]:=
          (DtRq;Data_Phase[DtRq,DisRq]
          [] DisRq;stop
           )
```

```
endproc
endproc
```

A representação interna é:

```
proc(sender(ConRq,ConCf,DtRq,DisRq),[connection_phase(ConRq,ConCf),
                    data_phase(DtRq,DisRq)].
proc(connection_phase(ConRq,ConCf),[ConRq,ConCf]).
proc(data_phase(DtRq,DisRq),[esc([DtRq,data_phase(DtRq,DisRq)],
                                 [DisRq,stop])]).
```

Uma lista possível de eventos seria :

```
[ConRq,ConCf,DtRq,DtRq]
```

A execução do processo é dada pela chamada da cláusula **validar**, e em seguida o usuário deve fornecer a identificação do processo e a sequência de ações que pretende validar:

```
?:-validar.
   PROCESSO: [sender(ConRq,ConCf,DtRq,DisRq)]
   EVENTO:  [ConRq,ConCf,DtRq,DtRq]
   SEQUENCIA VALIDA
   ESTADO FINAL ---> data_phase(DtRq,DisRq)
```

Após a execução verificamos que a sequência fornecida é válida e o estado final após consumir os eventos fornecidos é mostrado.

5. Conclusão

Neste trabalho apresentamos uma proposta de implementação de um simulador para a linguagem de descrição formal LOTOS , baseado nos modelos apresentados por [6],[7]. Verificamos que uma forma executável de LOTOS pode ser implementada com vantagens em Prolog pelas seguintes razões:

- tradução direta da semântica operacional de LOTOS para cláusulas Prolog;

- facilidade de modificação do conjunto de cláusulas que representam o controle caso haja alterações na semântica;

- facilidade de modificação na base de dados sem interferência no controle.

O desenvolvimento de ferramentas automáticas de análise para especificações formais incentivará o uso mais generalizado destas técnicas além do ambiente acadêmico. Em particular, a proposta deste simulador é permitir a verificação automática de especificações feitas em LOTOS.

REFERENCIAS BIBLIOGRAFICAS

[1] _,Information Processing Systems-Open Systems Interconnection LOTOS-A Formal Description Technique "Based on the Temporal Ordering of Observational Behaviour", ISO/TC97/SC21 DP8807 revised, julho 1986.

[2] R.Milner, "A Calculus of Communicating Systems", LNCS 92, Springer-Verlag, Berlin, 1980.

[3] C.A.R.Hoare, " Communicating Sequential Processes ", Comm. ACM, vol.21, no.8, 1978.

[4] H.Ehrig, B.Mahr " Fundamentals of Algebraic Specification 1", Spring Verlag, Berlin, 1985.

[5] C.A.Vissers, "Trends and Proliferations of Formal Descriptions Techniques for OSI Standards", III Simpósio Brasileiro de Redes de Computadores, São Paulo, 1987.

[6] G.Pappalardo, "Experiences with a Verification and Simulation Tool for Behavioural Languages", Proceedings of VII Protocols Spec.Test. & Verif. , 1987.

[7] J.P.Briand, M.C.Fehri, L.Logrippo, A.Obaid,"Executing LOTOS Specifications", Proceedings of VI Protocols Spec. Test. & Verif., ed. B.Sarikaya & G.V.Bochmann, North-Holland, 1986.

[8] D.Gilbert, "Executable LOTOS: Using PARLOG to implement an FDT", Proceedings of VII Protocols Spec. Test.& Verif., North-Holland, 1987.

[9] E.Brinksma , " A Tutorial on LOTOS" ,Proceedings of V Protocols Spec. Test. & Verif., ed. M.Diaz, North-Holland, 1986.

[10] W.F.Clocksin, C.S.Mellish, "Programming in Prolog", Springer Verlag, 1981.

[11] L.Logrippo, D.Simon, H.Ural, "Executable Description of the OSI Transport Service in Prolog", Proceedings of IV Protocols Spec.Test. & Verif., ed. Y.Yemini, R.Strom, S.Yemini, North-Holland, Amsterdam, 1985.

[12] G.Bochmann, R.Dssouli, B.Sarikaya, W.L.Souza, H.Ural, "Use of Prolog for Building Protocol Design Tools", Proceedings of V Protocols Spec. Test. & Verif., ed. M.Diaz , North-Holland, Amsterdam, 1986

[13] D.P.Sidhu, C.Crall, " Executable Logic Specifications for Protocols Service Interfaces" , IEEE Trans. on Sotware Engineering, vol 14, no. 1, Jan. 1988.

[14] W.L.de Souza, "LOTOS; uma Técnica para Descrição Formal de Serviços e Protocolos de Comunicação", Anais do IV Simpósio Brasileiro de Redes de Computadores, São Paulo, 1987, pp 121-144.