LOTOS : Linguagem para Especificação de Ordenação Temporal

Ricardo Neimo Belem Lima

1 Introdução

LOTOS (Language of Temporal Ordering Specification) é uma das duas Técnicas de Descrição Formal (FDT) desenvolvidas dentro da ISO para especificação formal de sistemas abertos distribuidos e em particular para arquitetura OSI.

A idéia básica que LOTOS desenvolveu foi que sistemas podem ser especificados pela definição da relação temporal entre as interações que constituem o comportamento observável externamente de um sistema. Ao contrário do que o nome possa sugerir, LOTOS não está relacionado com lógica temporal, mas é baseado em métodos algébricos introduzidos por Milner em CCS.

LOTOS também inclui a descrição de estrutura de dados e expressão de valores. Essa parte é baseada na teoria formal de tipos abstratos de dados. A maioria desses conceitos foi inspirado em ACT ONE, embora existam algumas diferenças.

Os objetivos principais desta FDT são:

- Descrição de especificações precisas, completas e não ambíguas;
- Documentos legíveis para usuários, implementadores e testadores;
- Uma base bem definida formalmente para verificação e validação de padrões e para teste de conformância de suas implementações.

É claro que esses objetivos são particularmente importantes numa arquitetura padrão distribuída como a OSI. Máquinas que se comunicam e cooperam umas com as outras usando especificações informais e ambíguas podem levar à implementações incompatíveis.

2 Processos

2.1 Processos

Em LOTOS, um sistema distribuído é visto como um processo, possivelmente composto de alguns sub-processos. Um sub-processo é um processo dentro de si mesmo, e em geral uma especificação LOTOS descreve um sistema via hierarquia de definições de processos. Um processo é uma entidade capaz de realizar ações internas (não observáveia) e interagir com outros processos, da mesma forma com seu ambiente. Interações complexas entre processos são unidades elementares de sincronização chamadas eventos, ou interaç oes (atômicas) ou simplesmente aç oes.

Eventos implicam sincronização, porque os procesos que interagem num evento (dois ou mais) participam de sua execuç ao no mesmo instante de tempo. Tais sincronizaç oes podem envolver troca de dados. Eventos são atômicos quando ocorrem instantaneamente, sem consumo de tempo. Um evento é visto como uma ocorrência num ponto de interação, ou porta, e no caso de sincronização sem troca de dados, o nome do evento e o nome da porta coincidem.

2.2 Definições de Processos e Expressão Comportamental

A definição de processo será a especificação de seu comportamento através da definição das seqüências de ações observáveis que podem ocorrer (serem observadas) nas portas de um processo. Cada comportamento será representado como uma árvore de ações.

Uma componente essencial de uma definição de processo é sua expressão comportamental.

Uma expressão comportamental representa um estado de um processo. Um conjunto pré-definido de operadores é usado combinando ações com expressões comportamentais formando assim outras expressões comportamentais. Existem duas expressões comportamentais pré-definidas, **stop**, que denota *deadlock* (ou inatividade) e **exit**, que denota "término bem sucedido".

As definições de processos são similares às procedures em linguagem de programação e a instanciação de processo é similar à chamada de procedure.

FIGURA

A expressão comportamental de um processo determina quais ações (ou eventos) são possíveis como próximas ações do processo. Existem ações que um processo pode executar independentemente, e são representadas por ações internas **i**. Existem ações que necessitam sincronização com o ambiente para serem executadas.

3 LOTOS básico

LOTOS básico é uma versão simplificada da linguagem empregando um alfabeto finito de ações observáveis. Isso se dá porque ações observáveis em LOTOS básico são identificadas somente pelo nome das portas aonde elas são oferecidas e os processos LOTOS somente podem ter um número finito de portas.

A estrutura de ações será enriquecida em LOTOS completo através de associação de dados às portas e somado possivelmente ao uso de um alfabeto infinito de ações observáveis.

LOTOS básico somente descreve processo de sincronização, enquanto LOTOS completo também descreve comunicação de valores entre processos.

3.1 Operador *prefixo de ação* e Operador *escolha*

O operador *prefixo de ação* (;) expressa uma composição seqüencial de ações. Esse operador é usado para ordenar seqüencialmente as ações. Por exemplo *a;B* denota um comportamento aonde a ação a será executada antes da expressão comportamental B.

O operador *escolha* ([]) denota a escolha entre dois ou mais comportamentos alternativos.

Para melhor entendimento será utilizado o clássico exemplo do Produtor - Consumidor na introdução do uso desses operadores.

FIGURA

Esse exemplo ilustra o seguinte fato: o produtor "produz" exatamente dois elementos e então termina. Similarmente o consumidor "consome" os dois elementos e então termina. E o canal trata os dois tipos de mensagem de uma das duas formas a seguir: podendo entregar os dois elementos vindos do produtor diretamente ao consumidor, ou entregar um elemento e esperar que o produtor deposite o outro elemento. A especificação dos processos isoladamente ficaria:

```
process Produtor [pc1, pc2]: exit:=
        pc1; pc2; exit
endproc
```

```
process Consumidor [cc1, cc2]: exit:=
        cc1; cc2; exit
endproc

process Canal [pc1, pc2, cc1, cc2]: exit:=
        pc1; (pc2; cc1; cc2; exit [] cc1; pc2; cc2; exit)
endproc
```

3.2 Operador *habilita*

O operador *habilita* (>>) é usado para expressar a composição seqüencial de duas expressões comportamentais. Por exemplo, se *P1* e *P2* são dois processos, *P1* >> *P2* significa *P1* habilita *P2*. O processo *P2* será habilitado somente se *P1* tiver um "término bem sucedido". Essa habilitação se dá na execução de um **exit** em *P1*.

Por exemplo:

```
a; b;  exit >> c; stop
```

Isto é equivalente a:

```
a; b; c; stop
```

Se acontecesse:

```
a; b; stop >> c; stop
```

Seria equivalente a:

```
a; b; stop
```

A expressão do lado direito do operador não será executada, pois a expressão do lado esquerdo não tem um "término bem sucedido".

3.3 Ação interna

Uma ação interna **i** não é controlada pelo ambiente e portanto, em conjunção com o operador *escolha* representa uma escolha não determinística.

Só a título de ilustração reescreveremos o Canal do exemplo Produtor - Consumidor de modo que se possa simular uma possível perda de um elemento produzido pelo Produtor. Agora a especificação pode entregar o elemento ao Consumidor ou perder esse elemento.

```
process Canal [pc1, pc2, cc1, cc2]: exit:=
        pc1; (pc2; cc1; exit [] cc1; pc2; exit [] i; pc2; exit)
        >> (cc2; exit [] i; exit)
endproc
```

Alterando o processo Consumidor de modo que satisfaça às novas características determinadas acima:

```
process Consumidor [cc1, cc2]: exit:=
        cc1; (cc2; exit [] exit)
        [] cc2; exit
        [] exit
endproc
```

Agora o consumidor pode sincronizar nas seguintes seqüencias de ações: (cc1, cc2, e **exit**); (cc1 e **exit**); (cc2 e **exit**) ou simplesmente **exit** quando ambos elementos são perdidos.

3.4 Paralelismo

Existem três tipos de operadores de composição paralela: o operador de *composição seletiva*, o operador *interleaving* e o operador de *sincronização completa.*

O operador *interleaving* (|||) é usado para expressar o conceito de paralelismo entre comportamentos aonde nenhuma sincronização é necessária.

Como exemplo:

```
out1 ; out2; exit) ||| (in1; in2; exit)
```

Isto é equivalente a:

```
out1; (out2; in1; in2; exit [] in1; (out2; in2; exit [] in2; out2; exit)) []
in1; (in2; out1; out2; exit [] out1; (in2; out2; exit [] out2; in2; exit))
```

O operador de *composição seletiva* (|[**L**]|) é usado quando processos sincronizam com ações comuns e aonde **L** é uma lista de ações de sincronizações que podem ocorrer.

Como exemplo:

```
a; b; c; exit |[a]| d; a; c; exit
```

Isto é equivalente a:

```
d; a; (b; c;  exit ||| c; exit)
```

ou

```
d; a; (b; c; c; exit [] c; b; c; exit)
```

Claramente, quando **L** é uma lista vazia o operador de composição seletiva se torna idêntico ao operador *interleaving*.

Temos também que quando **L** é uma lista com todas portas de sincronização, o operador de composição seletiva se torna idêntico ao operador de *sincronização completa* (||).

3.5 Operador *desabilita*

O operador *desabilita* ([>) modela uma interrupção de um processo por outro processo. Se *P1* e *P2* são dois processos, então *P1* [> *P2* significa que, em algum ponto durante a execução de *P1*, existe uma escolha entre executar uma das próximas ações de *P1* ou uma das primeiras ações de *P2*. Se *P1* não termina com sucesso, as primeiras ações de *P2* são oferecidas, enquanto se *P1* termina bem sucedido, *P2* não começa sua execução.

3.6 Prioridade de Operadores

Como em algumas linguagens, existe uma prioridade em LOTOS de modo a se reduzir o número de parênteses necessário. A ordem de prioridade dos operadores é:

```
;     []  |[L]|  [>  >>  hide  choice
```

4 Tipos de dados em LOTOS completo

As representações de valores, expressões de valores e estrutura de dados em LOTOS completo são derivadas da linguagem de especificação para tipos abstratos de dados (ADT) ACT ONE. A escolha de tipos de dados abstratos para LOTOS se deve à necessidade de uma abstração dos detalhes de implementação de um processo.

Tipos concretos de dados implicam uma descrição de como os valores de dados são representados na memória, e como procedures associadas os operam. Por exemplo, uma fila Pascal é definida com uma lista de records e um par de procedures que a manipula realiza operações de adição e remoção.

Tipos abstratos de dados podem ser vistos como a especificação formal de uma classe de tipos concreto de dados. Não há indicação de como valores de dados são atualmente representados e manipulados na memória, mas somente define as propriedades essenciais do dado e operações que alguma implementação requer.

Ultimamente, uma definição ADT identifica um objeto matemático (álgebra) formado por conjuntos de valores de dados (*data carriers*) e um conjunto de operações associadas.

A forma mais básica de especificação de tipos de dados em LOTOS consiste de uma assinatura (*signature*) e possivelmente uma lista de equações (*equations*).

4.1 Assinatura (*signature*)

O primeiro passo na especificação de um tipo de dado consiste na definição dos data carriers e suas operações. Os nomes dos data carriers são atribuidos aos *sorts*. A declaração de cada operação será incluída nesse domínio, que consiste numa lista de nenhum ou mais sorts. Os sorts e as operações de um tipo de dado são referidos como a assinatura daquele tipo de dado.

Abaixo está uma definição dos números naturais, que consiste somente de uma assinatura. A definição é chamada “Números-naturais”, e pode ser referido por outras definições e também combinado com elas.

```
type Numeros-naturais is
    sorts nat
    opns  0 :  nat
          succ :  nat  nat
endtype
```

4.2 Equações (*equations*)

Suponha que nós queiramos definir a operação soma que combina dois termos naturais num outro termo natural.

Precisa-se de uma nova construção que expresse propriedades dessa equação.

Uma definição das propriedades do operador soma é:

```
eqns
    forall x, y : nat
    ofsort nat
         x + 0 = x;
         x + succ(y) = succ (x + y);
```

Estendendo a definição de números naturais adicionando o operador soma seria:

```
type Numeros-naturais-estendidos is
    sorts nat
    opns  0 :  nat
          succ :  nat  nat
          x + y : nat, nat  nat
    eqns
       forall x, y : nat
       ofsort nat
           x + 0 = x;
           x + succ(y) = succ (x + y);
endtype
```

5 LOTOS completo

5.1 LOTOS completo

Em LOTOS completo é possível descrever processos de sincronização envolvendo troca de valores de dados. Estruturas de dados e expressões de valores são definidos usando-se, como dito antes, tipos abstratos de dados de ACT ONE.

Em comparação com LOTOS básico, LOTOS completo é um conjunto mais completo e enriquecido. Os valores expressos podem ser:

- oferecidos nas portas, e trocado entre processos;
- usados para expressar condições para satisfazer um dado comportamento;
- usado para generalizar o operador escolha;
- passados por habilitação de processos.

5.2 Declarações de valor

Uma declaração de valor é da forma “!E”, aonde E é uma expressão de valor. Exemplos de declarações de valores são:

```
!(3+5)  !(x+1)  !TRUE  !‘example’  !not(x)  !min(x,y)
```

Se combinarmos uma declaração de valor com uma porta *tsap*, temos agora a descrição de uma ação. Exemplo:

```
tsap !(3+5)
```

Descreve a ação *tsap*<8>. Outro exemplo, se x=3 e y=5 em:

```
tsap !min(x,y)
```

Temos a descrição do evento *tsap*<3>.

5.3 Declarações de variáveis

Uma declaração de variáveis tem a forma "$?x : t$", aonde x é o nome de uma variável e t é um sort. Como já explicado um sort indica o domínio de valores que x pode assumir. Exemplos de declarações de variáveis são:

```
?x:integer  ?text:string  ?x:nat  ?active:boolean
```

Se combinarmos uma declaração de variável com uma porta *gate*, temos agora a descrição de um conjunto de ações. Exemplo:

```
a ?x:nat ; b ?y:nat ; c !min(x,y) ; stop
```

Este exemplo significa que na porta a foi recebido um valor x, depois na porta b recebeu-se um valor y, por fim oferece-se através da porta c o menor número entre x e y e termina (**stop**).

5.4 Construções condicionais

Este ítem mostra dois exemplos de construções condicionais:

- Seleção de predicados:

  ```
  sap1 ?x:nat [x<3] ; sap2 !x ; stop
  ```

- Expressão guardada:

  ```
  [x>0]  processo 0 [] [x<=0]  processo1
  ```

O primeiro ítem significa que a porta *sap1* recebe valor x, e põe como saída na porta *sap2* este valor, se ele for menor que 3 e depois termina (**stop**).

O segundo ítem significa que se x for maior que 0 haverá a execução do processo 0 e se x for menor ou igual a 0 haverá a execução do processo 1.

5.5 Expressões comportamentais

Para ilustrar essa parte de LOTOS completo, recorreremos ao uso de exemplo do Produtor - Consumidor, com uma modificação. Agora o consumidor aceita uma única mensagem, com número de seqüência par e então sai.

A especificação do exemplo com esta modificação ficaria:

```
process Consumidor [In_Ele]: exit:=
        In_Ele ?Msg:nat ?Msg_seq:nat [(Msg_seq mod 2) = 0] ; exit
endproc
```

5.5.1 "Término bem sucedido" com passagem de parâmetros

A possibilidade de passagem de parâmetros como resultado do processo é ilustrada abaixo, reescrevendo o exemplo anterior:

```
process Consumidor [In_Ele]: exit(nat):=
        In_Ele ?Msg:nat ?Msg_seq:nat [(Msg_seq mod 2) = 0] ; exit(Msg_seq)
endproc
```

5.5.2 Instanciação de processo com passagem de parâmetros

Para ilustrar a instanciação de processo com passagem de parâmetros, reescreveremos o exemplo do Produtor - Consumidor, só que agora o consumidor ao receber as mensagens com o número de seqüência, contará o número de mensagens com número de seqüência ímpar e o número de mensagens com número de seqüência par. Ao final (**exit**) mostrará o número total de mensagens recebidas.

```
process Consumidor [In_Ele](Num_Msg_impar, Num_Msg_par : nat): exit(nat, nat):=
        In_Ele ?Msg:nat ?Msg_seq:nat [(Msg_seq mod 2) = 0] ;
        Consumidor [In_Ele](Num_Msg_impar, Num_Msg_par + 1)
        []
        In_Ele ?Msg:nat ?Msg_seq:nat [(Msg_seq mod 2) <> 0] ;
        Consumidor [In_Ele](Num_Msg_impar + 1, Num_Msg_par)
        []
        exit(any nat, Num_Msg_impar + Num_Msg_par)
endproc
```

5.5.3 Composição seqüêncial com passagem de valores (*enable* com *accept*)

A composição seqüencial com passagem de valores é da seguinte forma:

```
B1 >> accept X1:S1,....,Xn:Sn in B2
```

A expressão comportamental *B1* será executada até terminar. Se ele sai (**exit**), então na execução da próxima expressão comportamental (*B2*), as variáveis X1 a Xn de *B2* serão substituidas pelos resultados de *B1* (parâmetros do **exit**).

O número e o sort dos valores que são passados por *B1* têm que ser os mesmos daqueles declarados no campo *accept*.

6 Conclusões

LOTOS (Language of Temporal Ordering Specification) é uma das duas Técnicas de Descrição Formal (FDT) desenvolvidas dentro da ISO para especificação formal de sistemas abertos distribuidos e em particular para arquitetura OSI.

Embora os protocolos tenham sido e serão a área principal de utilização de LOTOS, um grande número de artigos têm sido publicados recentemente mostrando que LOTOS pode ser usado tão bem em outras áreas.

Entre outras áreas de utilização de LOTOS estão aplicações da linguagem em especificação formal de processamento de chamada telefônica, aplicações em especificação de mecanismos de segurança e aplicações em especificação de algoritmos distribuídos.

Este trabalho abordou a linguagem LOTOS de uma maneira introdutória e fazendo uso de exemplos de modo a esclarecer o uso de sua sintaxe. Mostrou um subconjunto de LOTOS (LOTOS básico), que é uma versão simplificada da linguagem, mas que por si só já consiste numa poderosa ferramenta para especificação de sistemas.

Por fim, este trabalho mostrou LOTOS completo, que é uma expansão de LOTOS básico, com uso de troca de valores de dados. LOTOS completo utiliza tipos abstratos de dados de ACT ONE e é um conjunto poderoso para especificação de sistemas.

References

[1] P. H. J. van Eijk, C. A. Vissers e M. Diaz, eds., "The Formal Description Technique LOTOS", (North-Holland, Amsterdan, 1989).

[2] L. Logrippo, M. Faci e M. Haj-Hussein, "An Intodución to LOTOS: learning by examples", Computer Networks and ISDN Systems 23 (1992) pp. 325-342.