Referências Bibliográficas

[Acc04] Accellera. Ieee p1850 - standard for psl - property specification language, 2004. 3.2.4, 7

[AHM+98] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. MOCHA: Modularity in model checking. In *Computer Aided Verification*, pages 521–525, 1998. 2.2.2

[BB87] Tommaso Bolognesi and Ed Brinksma. Introduction to the iso specification language lotos. *Comput. Netw. ISDN Syst.*, 14(1):25–59, 1987. 6.2

[BBE+99] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. Technical report, USC, Computer Science Department, 1999. 1, 2.1

[BD87] S. Budkowski and P. Dembinski. An introduction to estelle: a specification language for distributed systems. *Comput. Netw. ISDN Syst.*, 14(1):3–23, 1987. 6.2

[BGL+00] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari. An overview of sal. In C. Michael Holloway, editor, *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, Hampton, VA, 2000. 1, 2.3, 6.1, 7, 7

[BGM02] Marius Bozga, Susanne Graf, and L. Mounier. IF-2.0: A validation environment for component-based real-time systems. In K.G. Larsen Ed Brinksma, editor, *Proceedings of Conference on Computer Aided Verification, CAV'02, Copenhagen*, number 2404 in LNCS, pages 343–348. Springer Verlag, June 2002. 1, 3.2.6, 6.1, 6.2, 7, 7

[BHE04] Carlos Bazilio, Edward Hermann Haeusler, and Markus Endler. Towards a methodology for verifying mobile protocols, 2004. I Simpósio de Tecnologia da Informação da Marinha. 2.3, 5, 5

[BHE05a] Carlos Bazilio, Edward Hermann Haeusler, and Markus Endler. Binding network topologies to specifications via pronouns, 2005. 2nd South-East European Workshop on Formal Methods - SEEFM05 (to appear). 2.3

[BHE05b] Carlos Bazilio, Edward Hermann Haeusler, and Markus Endler. Binding network topologies to specifications via pronouns. *The Annals of Mathematics, Computing & Teleinformatics (AMCT)*, 1(3), 2005. 2.3

[BHE05c] Carlos Bazilio, Edward Hermann Haeusler, and Markus Endler. Language-oriented formal analysis: a case study on protocols and distributed systems, 2005. SBMF-2005 Brazilian Symposium on Formal Methods (to appear). 2.3

[BLL+95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243, 1995. 1, 2.2.2, 6.4

[BMO01] Bernhard Bauer, Jorg P. Muller, and James Odell. Agent uml: a formalism for specifying multiagent software systems. In *First international workshop, AOSE 2000 on Agent-oriented software engineering*, pages 91–103, Secaucus, NJ, USA, 2001. Springer-Verlag New York, Inc. 6.2

[Cas98] G. Del Castillo. The ASM Workbench: an Open and Extensible Tool Environment for Abstract State Machines. In *5th International Workshop on Abstract State Machines*, Magdeburg University, 1998. Otto-von-Guericke-Universität. 6.1, 7

[CBD02] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002. 3.2.2, 7

[CCGR00] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000. 1, 1.1, 2.2.2, 3.3.3, 6.3, C.4

[CGMT04] Ana Cavalli, Cyril Grepet, Stephane Maag, and Vincent Tortajada. A validation model for the dsr protocol. *24th International Conference on Distributed Computing Systems Workshops - W6: WWAN (ICDCSW'04)*, 06:768–773, August 2004. 5, B

[CGP00] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000. 1, 2.2.2, 3.2.2, A.2

[CL00] Daniel Câmara and Antonio A.F. Loureiro. A novel routing algorithm for ad hoc networks. In *33rd Hawaii International Conference on System Sciences*, page 8022, Washington, DC, USA, 2000. IEEE Computer Society. 1, 6.4

[CLN01] S. O. Cruz, C. J. P. Lucena, and J. L. M. Rangel Netto. Identifying objects through pronouns, 2001. Monographs in Computer Science (in portuguese); 39/01, Rio de Janeiro : PUC, Dep. of Informatics, 2001. 3.2.1

[Cor04] James R. Cordy. Txl - a language for programming language tools and applications . *Electr. Notes Theor. Comput. Sci.*, 110:3–31, 2004. 3.1, 6.3

[CW00] Giuseppe Del Castillo and Kirsten Winter. Model checking support for the asm high-level language. In *TACAS '00: Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 331–346, London, UK, 2000. Springer-Verlag. 6.1

[DAC98] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. Technical Report UM-CS-1998-035, Santos Laboratory, Kansas State University, 1998. 2.2.2, 4.1.1, 6.4, 7

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 411–420, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press. 3.2.1, 3.2.4

[DCN+00] Louise A. Dennis, Graham Collins, Michael Norrish, Richard J. Boulton, Konrad Slind, Graham Robinson, Michael J. C. Gordon, and Thomas F. Melham. The PROSPER toolkit. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 78–92, 2000. 7

[Dil96] David L. Dill. The murhi verification system. In *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, volume 1102 of *Lecture Notes in Computer Science*, pages 390–393. Springer, 1996. 1, 2.2.2, C.5

[DKR82] Danny Dolev, Maria M. Klawe, and Michael Rodeh. An o(n log n) unidirectional distributed algorithm for extrema finding in a circle. *J. Algorithms*, 3(3):245–260, 1982. C, C.1, C.3, C.7

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos, 1996. 2.2.2

[DR99] Yifei Dong and C. R. Ramakrishnan. An optimizing compiler for efficient model checking. In *FORTE XII / PSTV XIX '99: Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, pages 241–256, Deventer, The Netherlands, The Netherlands, 1999. Kluwer, B.V. C.5

[ea05] Alessandro Armando el. al. The avispa tool for the automated validation of internet security protocols and applications. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV 2005*, volume 3576, pages 281–285. Springer Verlag, 2005. 6.2

[EMS03] Steven Eker, José Meseguer, and Ambarish Sridharanarayana. The maude LTL model checker and its implementation. In Thomas Ball and Sriram K. Rajamani, editors, *Proceedings of the 10th SPIN Workshop*, volume 2648 of *Lecture Notes in Computer Science*, pages 230–234, Berlin, May 2003. Springer. 1, 2.2.2

[ESO00] Markus Endler, Dilma M. Silva, and Kunio Okuda. RDP: A result delivery protocol for mobile computing. *ICDCS Workshop on Wireless Networks and Mobile Computing*, pages D36–D43, 2000. 1, 4.1.1, 4.1.1, 7

[FvS05] E. M. L. Figueiredo and Arndt von Staa. Avaliação de um modelo de qualidade para implementações orientadas à objetos e orientada à aspectos, 2005. Monografias em ciência da computação ; 14/05, Rio de Janeiro : PUC, Dep. de Informática, 2005. 4.2

[Gol87] Robert Goldblatt. *Logics of time and computation*. Center for the Study of Language and Information, Stanford, CA, USA, 1987. 5.2

[GV03] Alex Groce and Willem Visser. What went wrong: Explaining counterexamples. In *10th International SPIN Workshop on Model Checking of Software*, pages 121–135, Portland, Oregon, May 9–10, 2003. 6.4

[Hol97] Gerard J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997. 1, 1.1, 2.2.2, 3.2, 3.3.2, 6.3, 6.4, 7, C.3, C.4

[HP00] Klaus Havelund and Thomas Pressburger. Model checking JAVA programs using JAVA pathfinder. *International Journal on Software Tools for Technology Transfer*, 2(4):366–381, 2000. 2.3

[JHA+96] J. -C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. CADP: a protocol validation and verification toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 437–440, New Brunswick, NJ, USA, / 1996. Springer Verlag. 3.2.6, 6.1, 7

[JMB01] D. Johnson, D. Maltz, and J. Broch. DSR the dynamic source routing protocol for multihop wireless ad hoc networks, 2001. In Ad Hoc Networking, edited by Charles E. Perkins, chapter 5, pages 139–172. Addison-Wesley, 2001. 1, 4.1.2, 6.4

[JMH04] David B. Johnson, David A. Maltz, and Yih-Chun Hu. The dynamic source routing protocol for mobile ad hoc networks (dsr), July 2004. INTERNET-DRAFT - IETF MANET Working Group. 4.1.2, 4.1.2

[Kan03] S. H. Kan. *Metrics and models in software quality engineering, 2nd edition*. Pearson Education, 2003. 4.2

[KG02] Shmuel Katz and Orna Grumberg. A framework for translating models and specifications. In *IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods*, pages 145–164, London, UK, 2002. Springer-Verlag. 1, 3.2.6, 6.1, 7, C.6

[KM94] M. Kaufmann and J. Moore. Design goals of acl. Technical Report 101, Computational Logic, Inc., August 1994. 2.2.1

[KNP01] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. *Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pages 7–12, 2001. 2.2.2

[Lau02] Tanara Lauschner. Verificação formal e análise de protocolos de roteamento de redes móveis ad hoc, March 2002. Dissertação de Mestrado, UFMG. 5, 6.4

[Lin01] Jurgen Lind. Specifying agent interaction protocols with standard uml. In *AOSE '01: Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II*, pages 136–147, London, UK, 2001. Springer-Verlag. 1, 6.2

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996. 1, 1.1, C

[McG04] Tommy M. McGuire. Correct implementation of network protocols, 2004. Ph.D. thesis, The University of Texas at Austin. 6.1

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. 6.3, C.4

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. 3.2.1

[M.J88] M.J.C. Gordon. Mechanizing programming logics in higher-order logic. In G.M. Birtwistle and P.A. Subrahmanyam, editors, *Current Trends in Hardware Verification and Automatic Theorem Proving (Proceedings of the Workshop on Hardware Verification)*, pages 387–439, Banff, Canada, 1988. Springer-Verlag, Berlin. 2.2.1

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts i and ii. *Information and Computation*, 100(1):1–77, 1992. 3.2.1

[MS] Faron Moller and Perdita Stevens. Edinburgh Concurrency Workbench user manual (version 7.1). Available from http://homepages.inf.ed.ac.uk/perdita/cwb/. 6.3

[MS04] Nicolas Markey and Ph. Schnoebelen. Tsmv: A symbolic model checker for quantitative analysis of systems. In *QEST*, pages 330–331, 2004. 7

[OPB00] J. Odell, H. Parunak, and B. Bauer. Extending uml for agents, 2000. 6.2

[ORS92] S. Owre, J. M. Rushby, , and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag. 2.2.1, 2.3

[PFO98] Kalyan S. Perumalla, Richard Fujimoto, and Andrew Ogielski. TED - a language for modeling telecommunication networks. *SIGMETRICS Performance Evaluation Review*, 25(4):4–11, 1998. 1

[PJ96] Charles E. Perkins and David B. Johnson. Mobility support in IPv6. In *Mobile Computing and Networking*, pages 27–37, 1996. 1

[Plo81] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981. 3.2.6

[Pro] Veritech Project. Examples for veritech. Available: http://www.cs.technion.ac.il/Labs/ssdl/research/veritech/examples/index.html [Accessed 05.02.2006]. C.6

[RE01] R.C.A. Da Rocha and M. Endler. MobiCS: An environment for prototyping and simulating distributed protocols for mobile networks. *3rd. IEEE International Conference in Mobile and Wireless Communication Networks (MWCN´2001)*, pages 44–51, August 2001. 1, 2.1, 6.4

[RE02] M.F. Ribeiro and M. Endler. Design and simulation of atomic multicast protocols for mobile networks. In *Proc. Of the IV Workshop de Comunicação Sem Fio e Computação Móvel (WCSF2002)*, pages 3–14, October 2002. 1

[Rus97] John Rushby. Specification, proof checking, and model checking for protocols and distributed systems with PVS. Tutorial presented at FORTE X/PSTV XVII '97: Formal Description Techniques and Protocol Specification, Testing and Verification, 1997. 2.2.1

[SEM97] S. Campos, E. Clarke, and M.Minea. The verus tool: A quantitative approach to the formal verification of real-time systems. In O. Grumberg, editor, *Proc. 9th International Conference on Computer Aided Verification (CAV'97)*, volume 1254, pages 452–455. Springer Verlag, 1997. 6.4

[SSK95] Ken Slonneger, Kenneth Slonneger, and Barry Kurtz. *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 1, A.1

[(st06] Stefano Tonetta (stonetta@dit.unitn.it). Lcr algorithm for synchronized processes, 2006. Specification provided through a discussion at NuSMV mailing list. C.4

[TC96] Stavros Tripakis and Costas Courcoubetis. Extending promela and spin for real time. In *TACAs '96: Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*, pages 329–348, London, UK, 1996. Springer-Verlag. 7

[vO05] D. von Oheimb. The high-level protocol specification language hlpsl developed in the eu project avispa. In *APPSEM 2005 Workshop*, 2005. 6.2

[Wik05a] Wikipedia. Ad hoc protocol list. From Wikipedia, the free encyclopedia [Online], 2005. Available: http://en.wikipedia.org/wiki/Ad_hoc_protocol_list [Accessed 01.08.2005]. 1

[Wik05b] Wikipedia. List of network protocols. From Wikipedia, the free encyclopedia [Online], 2005. Available: http://en.wikipedia.org/wiki/List_of_network_protocols [Accessed 01.08.2005]. 1

[Win01] Kirsten Winter. Model checking abstract state machines, 2001. Ph.D. thesis, Technical University of Berlin, Germany. 6.1

[WPP04] Oskar Wibling, Joachim Parrow, and Arnold Neville Pears. Automatized verification of ad hoc routing protocols. In *FORTE*, pages 343–358, 2004. 5, 6.4

[YRS03] Ping Yang, C. R. Ramakrishnan, and Scott A. Smolka. A logical encoding of the pi-calculus: Model checking mobile processes using tabled resolution. In *VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 116–131, London, UK, 2003. Springer-Verlag. 1, 3.2, 6.3, C.7

[Z.102] Z.100. Specification and description language, 2002. Z.100 ITU-T Recommendation - Status: in force (08/02). 1, 3.2, 6.2, C.2

[ZBG98] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998. 1, 2.1

A
Definições

A.1
Gramática de Atributos

Uma gramática de atributos (SSK95) é uma gramática livre de contexto G=(N, Σ, P, S), onde N o alfabeto de não-terminais, Σ é o alfabeto de terminais, S é o símbolo inicial e P é o conjunto das regras de produção na forma $X_0 ::= X_1\ X_2 ... X_{np}$, onde $n_p \geq 1, X_0 \in N\ and\ X_k \in N \cup \Sigma\ for\ 1 \leq k \leq n_p$.

Para cada categoria sintática $X \in N$, existem 2 conjuntos disjuntos I(X) e S(X) de atributos *herdados* e *sintetizados*, respectivamente. Para X=S, o símbolo inicial, I(X)=ϕ.

Seja A(X)=I(X)$\cup$S(X) o conjunto de atributos de X. Cada atributo Atb$\in$A(X) tem seu valor associado à algum domínio semântico. Assim, considerando as regras de produção, cada atributo sintetizado Atb$\in$S(X_0) tem seu valor definido em termos dos atributos do conjunto $A(X_1) \cup ... \cup A(X_{np}) \cup I(X_0)$. Cada atributo herdado Atb$\in$ I(X_k), onde $(1 \leq k \leq n_p)$, tem seu valor definido em termos dos atributos do conjunto $A(X_0) \cup S(X_1) \cup ... \cup S(X_{np})$.

A.2
Gramática de Grafos

Uma gramática de grafos (*graph grammar*) (CGP00) é uma quíntupla G=$(\Sigma_n, \Sigma_t, \bigwedge, S, R)$, onde o alfabeto de não-terminais (Σ_n), o alfabeto de terminais (Σ_t) e o alfabeto de arestas ($\bigwedge$) são conjuntos mutuamente disjuntos e não-vazios. $S \in \Sigma_n$ é o símbolo inicial e R é o conjunto finito e não-vazio de regras de produção. Cada elemento em R é uma quádrupla r=(A, D, I, O), onde $A \in \Sigma_n$, D=(N, ϕ, ψ) é um grafo conectado sobre $\Sigma = \Sigma_n \cup \Sigma_t$ e $\bigwedge$, $I \in N$ é o conjunto de *nós de entrada* e $O \in N$ é o conjunto de *nós de saída*.

Dado um grafo cujos nós são rotulados, um novo grafo é derivado usando uma das regras de produção da gramática. Iniciamos com o grafo nó inicial S. Durante a derivação, um nó com rótulo A é substituído por um grafo D, dada uma regra da forma (A, D, I, O). Toda aresta que termina em A, também

terminará em cada nó de entrada de D (I). Toda aresta que se inicia em A, iniciar-se-á em cada nó de saída de D (O).

A.3
Gramática de Grafos com Atributos

Uma gramática de grafos com atributos é uma quíntupla G=$(\Sigma_n, \Sigma_t, \bigwedge, S, R)$, onde o alfabeto de nós não-terminais (Σ_n), o alfabeto de nós terminais (Σ_t) e o alfabeto de arestas ($\bigwedge$) são conjuntos mutuamente disjuntos e não-vazios, $S \in \Sigma_n$ é o nó inicial e R é um conjunto finito não-vazio de regras de produção. Cada regra em R é uma quádrupla r=(A, D, I, O), onde $A \in \Sigma_n$, D=(N, ϕ, ψ) é um grafo conexo sobre $\Sigma = \Sigma_n \cup \Sigma_t$ e $\bigwedge$, $I \in N$ é o conjunto de nós de entrada e $O \in N$ é o conjunto de nós de saída. Para cada X$\in \Sigma_n \cup \Sigma_t$, existem 2 conjuntos finitos disjuntos I(X) e S(X) de atributos herdados e sintetizados, respectivamente.

Como o próprio nome indica, uma gramática de grafos de atributos é uma gramática de grafos decorada com atributos. Com isto, podemos definir elementos cuja semântica dependa da topologia da rede gerada pela gramática de grafos. Essencialmente essa será nossa forma de definir algumas construções abstratas existentes na linguagem de especificação introduzida na seção 3.2.

B
Gramática de LEP em BNF

Neste apêndice apresentamos a gramática em BNF de LEP, comentando as construções mais importantes e/ou menos intuitivas.

```
LIST(X) ::= X ',' LIST(X) | X
```

LIST é apenas uma construção na metalinguagem para a definição de lista de elementos.

```
Espec ::=
      <Top>
      { <Mod> }+
      { <Prop> }
```

Uma especificação em LEP é composta pela expressão que define a topologia, um conjunto de módulos e um conjunto de propriedades a serem validadas.

Especificação das topologias

```
Top ::= [ <Agg> ] <Top-Expr>
```

A especificação da topologia contém uma possível definição de AGG (<Agg>), para introdução de uma nova topologia, e sua instanciação (<Top-Expr>).

```
Agg ::= { <IdM> '⇒' <Gr> { <Atribs> }* ';' }+
Gr ::= <Gr> <Oper-Seta> ['(' <Id> ')'] <Gr> | <Atom>
Atom ::= <Id> | <IdM> | 'in"('<Atom>')' | 'out"('<Atom>')'
Oper-Seta ::= '→' | '←' | '↔'
```

Especificação da gramática de grafos com atributos (AGG).

```
Top-Expr ::= 'topology is' ( <Top-Conexoes> | <Top-Macro> )
Top-Conexoes ::= '{' LIST(<Indice> - {LIST(<Descritor-Nohs>)}) '}'
Top-Macro ::= <Id> '(' LIST(<Descritor-Nohs>) ')' <Params-Top>
Descritor-Nohs ::= [<Id> ':'] <Limite> ['..' <Limite>] | <Indice>
Limite ::= <Num> | '*'
Indice ::= <Id> ['[' <Num> ']']
Params-Top ::=
    'unreliable' | 'reliable' | 'undirected' | 'directed' |
  'unsecure' | 'secure' | 'disconnected' | 'connected' |
  'dynamic' | 'static'
```

Especificação da expressão que instância uma topologia definindo a dimensão das redes, a topologia inicial adotada e os parâmetros de configuração que determinam o comportamento das redes.

Especificação dos módulos

```
Mod ::=
    'module' <Id>
        [<Locals>]
        { <Trans> }+
    'endmodule'
```

Um módulo em LEP é composto de um identificador, um conjunto de variáveis locais que definem o estado de uma instância do módulo e um conjunto de transições que definem o comportamento deste.

```
Locals ::= { <Tipo> LIST(<Id>) }*
Tipo ::=
    'int' |
    'bool' |
    <Id> |
    <Tipo> '#' <Tipo> |
    'seq:' <Tipo> |
    'set:' <Tipo> |
    '(' <Tipo> ')'
```

Na declaração de tipos, além dos tipos básicos, podemos ter o identificador de um módulo como tipo, sequências, conjuntos e combiná-los através do construtor de tipos '#'. Por exemplo, *(seq:int)#int* define uma estrutura com

dois (2) componentes: uma sequência inteiros e um inteiro.

```
Trans ::= <Pre-Cond> '->' { <Acao> }*
```

Numa transição temos a pré-condição <Pre-Cond> que é um guarda para a execução das ações correspondentes.

```
Pre-Cond ::=
    <Recebimento> |
    <Bool-Expr> |
    'init' |
    'true' |
    'else'
```

Pre-Cond contém as possíveis pré-condições de uma transição em LEP.

```
Recebimento ::= <Ident> '?' <Id> '(' { LIST(<Id>) } ')'
Ident ::=
    <Id> ['#' <Num>] ['.' (<ComandosSet> | <ComandosSeq>)] |
    <Pronome> ['(' (<Num> | <Id>) ')']
```

Um identificador pode ser, dentre outras opções, o seletor de uma estrutura. Por exemplo, para o tipo *(seq:int)#int*, se declaramos uma variável x deste tipo, *x#2* é uma referência à segunda componente da estrutura, ou seja, o tipo *int*.

```
ComandosSet ::=
    'add' '(' <Ident> ')' |
    'remove' '(' <Ident> ')' |
    'contains' '(' <Ident> ')' |
    'clean' |
    'empty' |
    'size'
ComandosSeq ::=
    'next' |
    'previous' |
    'first' |
    'last' |
    'range' '(' <Ident> ',' <Ident> ')' |
    <ComandosSet>
```

Estas são as operações que podemos realizar em elementos do tipo *set* ou *seq*.

```
Pronome ::=
      'this' |
      'any' |
      'anyother' |
      'everyone' |
      'neighbours' |
      'sender' |
      'none'
Acao ::=
      <Atrib> |
      <Envio> |
      <If> |
      <While> |
      <Executa> |
      <Termina> |
      <Trans> |
      '{' LIST(<Acao>) }
```

Dentre as ações listadas, podemos definir uma sequência não-deterministicamente colocando as ações entre chaves ({}). Além definirmos transições com a mesma pré-condição, esta é uma outra forma de inserirmos não-determinismo na especificação.

```
Atrib ::= <Ident> '=' <Expr> | <Ident> '=' <Executa>
Envio ::= <Pronome> '!' <Id> ['(' LIST(<Ident>) ')']
If ::= 'if' <Bool-Expr> 'then' { <Acao> }* 'endif'
While ::= 'while' <Bool-Expr> 'do' { <Acao> }* 'endwhile'
Executa ::= 'start' <Ident>
Termina ::= 'stop' [ <Pronome> ]
Expr ::=
    <Bool-Expr> |
    <Int-Expr> |
    { LIST(<Expr>) } |
    <Ident>
Bool-Expr ::=
    <Ident> <Oper-Bool> <Bool-Expr> |
    'not' <Bool-Expr> |
    <Int-Expr> <Oper-Comp> <Int-Expr> |
    <Ident> | <Ident> '.' <Ident> Oper-Bool ::= 'and' | 'or'
Int-Expr ::= ( <Id> | <Num> ) <Oper-Int> <Int-Expr>
| ( <Id> | <Num> )
Oper-Int ::= '+' | '-' | '*' | '/'
Oper-Comp ::= '==' | '<>' | '>' | '<'
Id ::= { <Idm>, <IdM> }+
Idm ::= {'a',..,'z'}
IdM ::= {'A',..,'Z'}
Num ::= {'0',..,'9'}+
```

Especificação das propriedades

```
Prop ::=
    <Quantif> <Prop> |
    <Sub-Formula> [<Oper-Prop> <Prop>] |
    '(' <Prop> ')'
Sub-Formula ::=
    <Bool-Expr> |
    <Recebimento> |
    <Envio> |
    <Id> '$' <Ident> '$'
Quantif ::= { '<>', '[]' }*
Oper-Prop ::= '->' | 'U' | 'W'
```

Como quantificadores temporais de LEP temos <> (eventualmente algo necessariamente ocorre), '[]' (sempre), 'U' (*strong until*) e 'W' (*weak until*). Os operadores <>, '[]' e 'U' funcionam conforme a figura B.1.

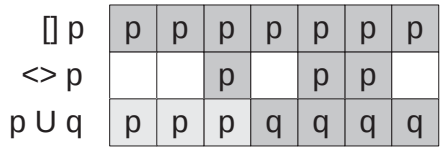

Figura B.1: Sequências no tempo onde as fórmulas são válidas

O operador 'W' pode ser definido em função de '[]' e 'U' como: $\boxed{[]p \textit{ or } (p\ U\ q)}$.

Como vimos, uma sub-fórmula pode ser uma expressão booleana, um comando que atesta o envio ou o recebimento de uma mensagem ou uma fórmula que verifica se um determinado trecho de comando (entre '$') foi executado. Um comando que atesta o envio ou o recebimento de uma mensagem tem o seguinte formato:

envio: *remetente!msg(destinatário)*
recebimento: *destinatário?msg(remetente)*

, onde *remetente* e *destinatário* são pronomes ou referências explícitas a elementos no protocolo. Ou seja, no expressão que atesta o envio de uma mensagem, a sintaxe é singelamente diferente do comando de envio numa especificação, a qual deixa implícito o remetente (módulo onde o comando ocorre).

Quanto à formula que verifica se um determinado trecho de comando foi executado, o objetivo é permitir a verificação da especificação baseada em seus próprios requisitos, como encontrado em (CGMT04). Por exemplo, pensando no uso correto das construções de LEP, poderíamos definir que sempre que o pronome *sender* é executado por um agente, necessariamente esse agente recebeu alguma mensagem: $[]\ p\ \$sender!msg_1\$ \rightarrow p?msg_2$.

C
Especificações do Algoritmo de Eleição de um Líder

Neste apêndice listaremos exemplos de especificações do Algoritmo de Eleição de um Líder em diferentes linguagens. Estas especificações estão baseadas na solução originalmente proposta (Lyn96), a qual tem ordem $O(n^2)$ com respeito ao número de mensagens trocadas, e numa versão otimizada de ordem *nlogn* (DKR82). Estas especificações se baseiam numa topologia em anel, unidirecional, parametrizadas com a quantidade de candidatos à eleição. Com exceção da especificação em SDL, todas as outras foram obtidas dos próprios autores das linguagens, ou de terceiros, de forma a tornar mais legítima a comparação com LEP.

C.1
LEP

Em LEP, como descrito na seção 3.2.5, temos o Algoritmo de Eleição de um Líder descrito em 2 partes (figura C.1), como comumente encontrado nas especificações em LEP: o comportamento de cada candidato e a topologia sobre a qual estes se interconectam. Como vimos, acreditamos que esta separação facilita o entendimento e permite uma certa independência entre estas partes, possibilitando o reuso.

Para a versão otimizada proposta em (DKR82), temos a seguinte especificação (figura C.2):

Para estas especificações, podemos verificar a seguinte propriedade *liveness*:

[] (<> someone!win)

, a qual verificará se sempre, em algum momento, necessariamente alguém é eleito. Podemos verificar também a propriedade *safety*:

not(<> none!win)

, a qual define que nunca, necessariamente em algum momento ninguém vence. Ou seja, necessariamente, sempre alguém vence.

Para as linguagens à seguir nos basearemos numa das duas (2) implementações e nas propriedades de *liveness* e *safety* dadas.

```
topology is Ring(5), directed, reliable;
module candidate
        vars my, p, count : int;
        init -> count = 0; my = this; neighbours!msg(my, count);
        this?win -> stop;
        this?msg(p, count) ->
                if ((p > my) or
                    ((p == my) and (count < topology.size))) then
                        my = p;  count = count + 1;
                        neighbours!msg(my, count);
                else
                        if ((p == this) and (count > topology.size)) then
                                everyone!win; stop;
                        endif
                endif
endmodule
```

Figura C.1: Especificação da versão original do Algoritmo de Eleição de um Líder em LEP


```
topology is Ring(5), directed, reliable;
module candidate
        vars my, p, count, Active, n : int;
        init -> Active = 1; count = 0; my = this; neighbours!one(my,
count);
        this?win -> stop;
        this?one(p, count) ->
                if Active then
                        if (p <> my) then
                                count = count + 1;
                                neighbours!two(p, count);
                                neighRight = p;
                        else
                                if (count > topology.size) then
                                        everyone!win;
                                endif
                        endif
                else
                        neighbours!one(p, count);
                endif
        this?two(p, count) ->
                if Active then
                        if ((neighRight > p) and (neighRight > my)) then
                                my = neighRight;
                                neighbours!one(neighRight);
                        else
                                Active = 0;
                        endif
                else
                        neighbours!two(p, count);
                endif
endmodule
```

Figura C.2: Especificação da versão otimizada do Algoritmo de Eleição de um Líder em LEP

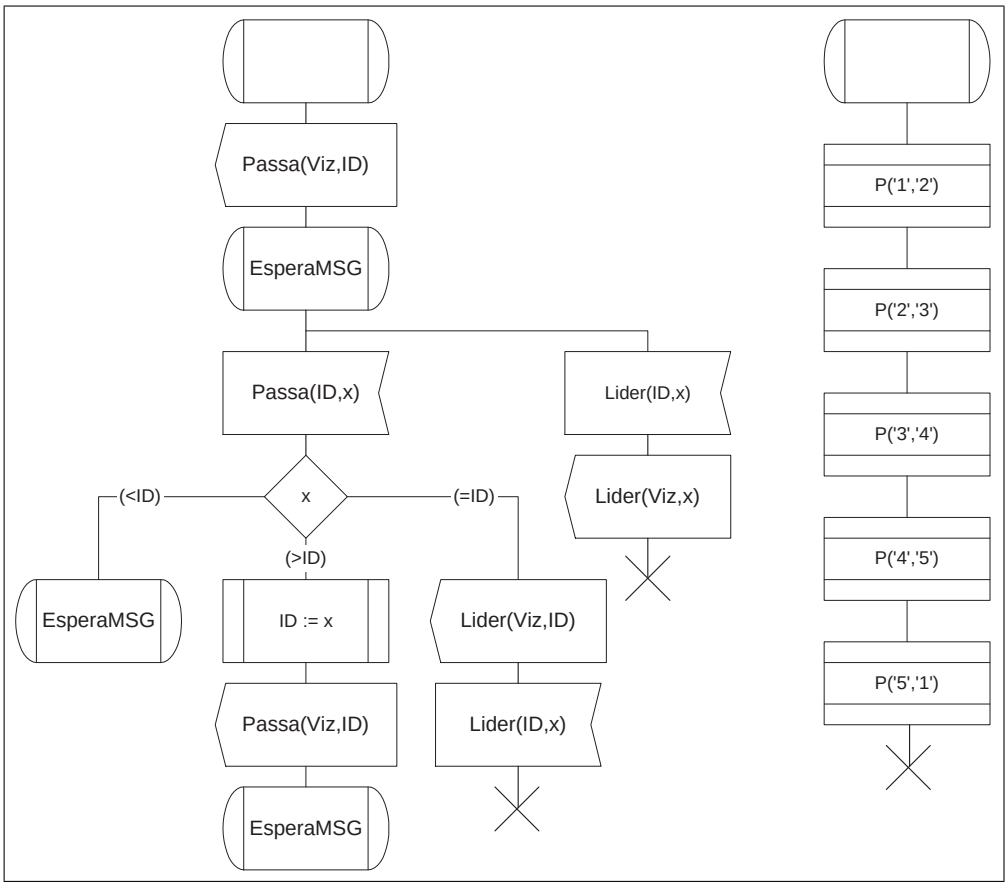

Figura C.3: Especificação de Eleição de um Líder em SDL

C.2 SDL

Nesta seção apresentamos como poderíamos ter uma especificação deste algoritmo em SDL (Z.102). Devido ao fato de não encontrarmos nenhum exemplo disponível na literatura, fizemos um com a mesma idéia dos que serão listados.

Na figura C.3, o lado esquerdo é processo que representa o comportamento de cada candidato, enquanto que o lado direito é o processo que inicia os cinco (5) participantes.

Como SDL foi projetada para ser uma linguagem de especificação que, dentre outros usos, pode ser integrada a ambientes de verificação, sua versão original não contém mecanismos para a especificação de propriedades a serem verificadas sobre o modelo especificado.

C.3 Promela

Nesta seção apresentamos uma especificação em Promela, a qual é um exemplo distribuído juntamente com o verificador de modelos Spin (Hol97).

Esta especificação está baseada na modificação da versão original do algoritmo de eleição de um líder proposta em (DKR82). Nesta nova versão, temos 2 tipos de mensagens (*one* e *two* na especificação em Promela) e 2 estados para cada candidato (ativo - *Active=1* - e inativo - *Active=0* - na especificação). A idéia é diminuir o número de candidatos ativos pela metade, a cada ciclo de mensagens, até que o líder seja encontrado. Esta observação é dada a título de didática, já que esta modificação não interfere no tipo de considerações que estão sendo feitas acerca das especificações.

Na especificação em Promela temos os módulos que define o comportamento do candidato (módulo *node*) e que inicia os candidatos para a eleição (módulo *init*).

Comparando com a versão deste algoritmo em LEP, novamente temos o protocolo sendo definido juntamente com a topologia sobre a qual este basear-se-á.

Para a especificação das propriedades em Promela/Spin, definimos uma variável global (nr_leaders, neste caso) e predicados em função desta variável.

```
#define elected (nr_leaders > 0)
#define noLeader (nr_leaders == 0)
#define oneLeader (nr_leaders == 1)
---
![] noLeader
<> elected
[] (noLeader U oneLeader)
```

Esta forma de especificarmos as propriedades baseadas em variáveis globais, as quais são atualizadas ao longo da especificação, é interessante mas permite equívocos como o correto ponto de atualização na especificação. Naturalmente, esta dificuldade aumenta a medida em que estas especificações vão ficando maiores.

C.4 NuSMV

Como vimos na seção 6.3, NuSMV (CCGR00) é o resultado de uma reengenharia, reimplementação e extensão do SMV (McM93). Esta especificação do algoritmo de Eleição de um Líder num anel síncrono foi gentilmente cedida por Stefano Tonetta ((st06) na lista de discussões do NuSMV, a qual foi automaticamente gerada e é baseada na primeira versão deste algoritmo.

Na figura C.5 temos o comportamento de cada candidato. Nesta temos dois (2) estados possíveis para um candidato (*unknown* e *leader*). Os parâmetros do módulo *node* (*receive* e *send*) indicam qual é o valor enviado

```
#define N   5           /* nr of processes (use 5 for demos) */
#define I   3           /* node given the smallest number    */
#define L   10          /* size of buffer  (>= 2*N) */

mtype = { one, two, winner };
chan q[N] = [L] of { mtype, byte};
byte nr_leaders = 0;

proctype node (chan in, out; byte mynumber)
{       bit Active = 1, know_winner = 0;
        byte nr, maximum = mynumber, neighbourR;

        printf("MSC: %d\n", mynumber);
        out!one(mynumber);
end:    do
        :: in?one(nr) ->
                if
                :: Active ->  if
                              :: nr != maximum ->
                                      out!two(nr);
                                      neighbourR = nr
                              :: else ->    /* Raynal p.39:  max is greatest number */
                                      assert(nr == N);
                                      know_winner = 1;
                                      out!winner,nr;
                              fi
                :: else ->    out!one(nr)
                fi
        :: in?two(nr) ->
                if
                :: Active ->  if
                              :: neighbourR > nr && neighbourR > maximum ->
                                      maximum = neighbourR;
                                      out!one(neighbourR)
                              :: else ->    Active = 0
                              fi
                :: else ->    out!two(nr)
                fi
        :: in?winner,nr ->
                if
                :: nr != mynumber -> printf("MSC: LOST\n");
                :: else ->    printf("MSC: LEADER\n");
                              nr_leaders++;
                              assert(nr_leaders == 1)
                fi;
                if
                :: know_winner
                :: else -> out!winner,nr
                fi;
                break
        od
}
init {
        byte proc;
        atomic {
                proc = 1;
                do
                :: proc <= N ->
                              run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
                              proc++
                :: proc > N -> break
                od
        }
}
```

Figura C.4: Especificação de Eleição de um Líder em Promela

```
MODULE node(receive,send)

VAR
id: 1..3;
status : {unknown,leader};

ASSIGN
next(id) := id;

init(status) := unknown;
next(status) :=
  case
    receive = id : leader;
    1 : status;
  esac;

init(send) := id;
next(send) :=
  case
    receive > id : receive;
    1 : 0;
  esac;
```

Figura C.5: Especificação de Eleição de um Líder em SMV

```
MODULE main

VAR
q : array 0..2 of 0..3;
node1 : node(q[0],q[1]);
node2 : node(q[1],q[2]);
node3 : node(q[2],q[0]);

DEFINE
nr_leaders := (node1.status=leader)+
                (node2.status=leader)+
                (node3.status=leader);
elected     :=          (nr_leaders > 0);
noLeader    :=          (nr_leaders = 0);
oneLeader   :=          (nr_leaders = 1);
uniqueid :=  node1.id!=node2.id &
             node1.id!=node3.id &
             node2.id!=node3.id;
```

Figura C.6: Especificação de Eleição de um Líder em SMV

pelo candidato anterior, no anel, e que valor será enviado para o candidato posterior, respectivamente.

Na figura C.6 temos o módulo principal, o qual define os processos a serem criados na especificação e como estes se comunicarão. Além disso, temos a definição de propriedades, as quais serão usadas como predicados nas fórmulas para a verificação.

Na figura C.7 temos as especificações das fórmulas em LTL a serem verificadas, as quais foram extraídas da distribuição do Spin (Hol97) e estão baseadas nas propriedades definidas na figura anterior (figura C.6).

Podemos observar claramente que nesta especificação existe uma amarração com a topologia escolhida (anel) e a quantidade de candidatos (3). Ou seja, caso alguma dessas quantidades sejam alteradas, inevitavelmente te-

```
-- LTLSPEC
--   uniqueid -> ! G noLeader

-- LTLSPEC
--   uniqueid -> F elected

-- LTLSPEC
--   uniqueid -> F G oneLeader

-- LTLSPEC
--   uniqueid -> G (noLeader U oneLeader)
```

Figura C.7: Especificação de Eleição de um Líder em SMV

remos que alterar partes da especificação. No caso específico de SMV, que tem algumas limitações como a indexação de vetores apenas por constantes, estas modificações necessariamente implicarão em aumento da especificação.

C.5 Murphi

Murφ (Dil96) é uma linguagem para especificação de modelos concorrentes. Esta foi projetada para possibilitar não-determinismo, escalabilidade e composição de suas descrições.

Nas figuras C.8, C.9, C.10 e C.11 apresentamos a versão otimizada do Algoritmo de Eleição de um Líder em Murφ. Esta especificação foi utilizada como elemento de comparação para o trabalho descrito em (DR99).

Na figura C.8 temos a definição de constantes, tipos e variáveis utilizadas ao longo da especificação.

Na figura C.9 listamos as funções utilizadas para manipulação dos canais de mensagens. Por exemplo, *input* é a função de recebimento de mensagens pelo canal, enquanto que *output* é a função de envio de mensagens.

Na figura C.10 temos as regras a serem aplicadas não-deterministicamente para cada nó. No início do conjunto de regras temos a declaração de uma seção *ALIAS*, a qual cria sinônimos para simplificar a especificação (como pronomes em LEP). Por exemplo, *right:channels[(n+1)%NumNodes]* significa que o termo *right* fará referência ao canal do candidato seguinte a este no anel, ou seja, *(n+1)%NumNodes*.

Finalmente, na figura C.11 temos a definição da regra que inicializa o sistema, definindo os valores locais de cada candidato.

Na especificação das propriedades, podemos novamente definir a propriedade *liveness* para este algoritmo como:

ALWAYS EVENTUALLY leader;

```
CONST
        NumNodes    : 7;
        BufferSize  : 2 * NumNodes;

TYPE
        NodeID      : 0..NumNodes-1;
        BufferLength: 0..BufferSize;
        BufferIndex : 0..BufferSize-1;
        MsgStage    : ENUM { first, second };
        Message     : RECORD
                          stage  : MsgStage;
                          ID     : NodeID;
                      END;
        Channel     : RECORD
                          length : BufferLength;
                          buffer : ARRAY [BufferIndex] OF Message;
                      END;
        NodeState   : ENUM {
                          start,
                          active,
                          inactive,
                          inactiveget,
                          getfirst,
                          getsecond
                      };
        Node        : RECORD
                          state  : NodeState;
                          maxID  : NodeID;
                          nbrID  : NodeID;
                          NID    : NodeID;
                          msgstage : MsgStage;
                      END;

VAR
        channels    : ARRAY [NodeID] OF Channel;
        nodes       : ARRAY [NodeID] OF Node;
        leader      : boolean;
```

Figura C.8: Constantes, Tipos e Variáveis utilizadas na especificação da Eleição de um Líder em Murφ

C.6 CDL

CDL (*Core Description Language*) é a linguagem de especificação do projeto Veritech (KG02), descrito no capítulo 6. Nesta seção listamos o algoritmo de Eleição de um Líder, o qual é um exemplo padrão apresentado na página do projeto (Pro). Nesta especificação temos a definição de 3 módulos: INIT_ELECTION, que inicia o processo de eleição; PROCESS, que especifica o comportamento de cada candidato, e; SYSTEM, que instancia os 5 candidatos e inicia a eleição.

Pela especificação listada, percebe-se que boa parte da codificação é implicação do número de processos (5) criados. A topologia em anel que conecta os processos está descrita na especificação juntamente com o algoritmo (por exemplo, o cálculo utilizando a função módulo (%) para retornarmos ao início do anel).

Sobre a descrição de propriedades, como Veritech visa a integração de diferentes ferramentas de verificação através de tradução de código, CDL não

```
FUNCTION isfull(Chan: Channel) : boolean;
BEGIN
   RETURN Chan.length = BufferSize;
END;

FUNCTION isempty(Chan: Channel) : boolean;
BEGIN
   RETURN Chan.length = 0;
END;

PROCEDURE input(VAR Chan: Channel; VAR stage: MsgStage; VAR ID: NodeID);
VAR
   i : BufferLength;
BEGIN
   assert(!isempty(Chan));
   stage := Chan.buffer[0].stage;
   ID    := Chan.buffer[0].ID;
   Chan.length := Chan.length-1;
   -- shift buffer
   FOR i:= 0 TO Chan.length DO
        Chan.buffer[i] := Chan.buffer[i+1]
   END;
   CLEAR Chan.buffer[Chan.length];
END;

PROCEDURE output(VAR Chan: Channel; stage: MsgStage; ID: NodeID);
BEGIN
   assert(!isfull(Chan));
   Chan.buffer[Chan.length].stage := stage;
   Chan.buffer[Chan.length].ID    := ID;
   Chan.length := Chan.length+1;
END;
```

Figura C.9: Funções para manipulação dos canais de mensagens da especificação da Eleição de um Líder em Murφ

contém por si só mecanismos para a especificação de propriedades.

Como CDL foi projetada para a integração de diferentes ambientes de verificação através de suas respectivas linguagens, está não oferece por si só nenhum mecanismo para a descrição de propriedades a serem validadas numa dada especificação.

C.7 XL

Nesta seção apresentamos uma especificação em XL, que é utilizada como linguagem de entrada do verificador MMC (YRS03) que, como descrevemos anteriormente, é um verificador de modelos para uma linguagem baseada em π-calculus, implementado sobre um compilador Prolog. A especificação apresentada está baseada na mesma modificação do algoritmo de eleição de um líder utilizada na especificação em Promela (DKR82) e pode ser encontrada como exemplo distribuído juntamente com a ferramenta.

Na especificação temos a inicialização dos processos dos candidatos feita de maneira recursiva, comum nas linguagens de especificação baseadas nos paradigmas lógico e funcional. Nesta encontramos diversos predicados, como os que definem os estados de cada candidato (ativo/inativo). Além destes, também temos um predicado para definir um canal (*medium*), o qual contém um buffer para armazenar a fila de mensagens recebidas por um candidato.

```
RULESET n: NodeID DO
ALIAS
        node: nodes[n]; left: channels[n];
        right: channels[(n+1)%NumNodes]
DO
RULE "send first"
        node.state = start & !isfull(right) ==>
BEGIN
        output(right, first, node.maxID); node.state := active;
END;

RULE "active"
        node.state = active & !isempty(left) ==>
VAR     stage   : MsgStage;
BEGIN
        input(left, stage, node.NID);
        IF stage = first THEN
                node.state := getfirst; node.nbrID := node.NID
        ELSIF stage = second THEN
                node.state := getsecond;
        ELSE  ERROR "wrong message type";
        END;
END;

RULE "getfirst"
        node.state = getfirst & !isfull(right) ==>
BEGIN
        IF node.nbrID != node.maxID THEN
                output(right, second, node.nbrID); node.state := active;
        ELSIF node.maxID = NumNodes-1 THEN
                ASSERT(!leader); leader := true; node.state := active;
        ELSE  ERROR "protocol fails"
        END
END;

RULE "getsecond"
        node.state = getsecond & !isfull(right) ==>
BEGIN
        IF node.nbrID > node.NID & node.nbrID > node.maxID THEN
                output(right, first, node.nbrID); node.state := active; node.maxID := node.nbrID;
        ELSE  CLEAR node; node.state := inactive;
        END;
END;

RULE "inactive"
        node.state = inactive & !isempty(left) ==>
BEGIN
        input(left, node.msgstage, node.NID); node.state := inactiveget;
END;

RULE "inactive get"
        node.state = inactiveget & !isfull(right) ==>
BEGIN
        output(right, node.msgstage, node.NID);      CLEAR node; node.state := inactive;
END;
ENDALIAS;
ENDRULESET;
```

Figura C.10: Regras que descrevem o comportamento de um nó na especificação da Eleição de um Líder em Murφ

```
STARTSTATE
        CLEAR channels;
        CLEAR nodes;
        FOR i: NodeID DO
                nodes[i].maxID := NumNodes-1 - i;
        END;
        leader := false;
END;
```

Figura C.11: Estado inicial da especificação da Eleição de um Líder em Murφ

```
HOLD_PREVIOUS

TYPE
message_state: ENUM {election , elected , no_message};

CONST
NumProcess: 5;

VAR
processes: ARRAY[NumProcess] OF integer; -- i is the i'th process
messages: ARRAY[NumProcess] OF integer; -- i is the i'th process channel
message_exist: ARRAY[NumProcess] OF message_state INITVAL no_message;


MODULE INIT_ELECTION(){

VAR
init: boolean INITVAL true;

        TRANS INIT: -- process 0 sends message to process 1
                enable: init = true;
                assign: message_exist[NumProcess-1]' := election;
                        init' := false;
                relation: (messages[NumProcess-1]' = processes[0]);
}


MODULE PROCESS(i: integer){

VAR
leader: integer INITVAL -1; -- -1 means that nobody is leader

        TRANS RECEIVED_ELECTION_AND_I_AM_GREATER:
                enable: ((message_exist[i]=election)/\(messages[i]>processes[i]));
                assign: message_exist[i]' := no_message;
                        message_exist[(i-1)%NumProcess]' := election;
                relation: (messages[(i-1)%NumProcess]' = messages[i]);

        TRANS RECEIVED_ELECTION_AND_I_AM_LESS:
                enable: ((message_exist[i]=election)/\(processes[i]>messages[i]));
                assign: message_exist[i]' := no_message;
                        message_exist[(i-1)%NumProcess]' := election;
                relation: (messages[(i-1)%NumProcess]' = processes[i]);

        TRANS I_AM_THE_LEADER:
                enable: ((message_exist[i]=election)/\(processes[i]=messages[i]));
                assign: message_exist[i]' := no_message;
                        message_exist[(i-1)%NumProcess]' := elected;
                relation: ((leader' = processes[i]) /\
                        (messages[(i-1)%NumProcess]' = processes[i]));

        TRANS RECEIVED_ELECTED_AND_I_AM_THE_LEADER:
                enable: ((message_exist[i]=elected)/\(processes[i]=messages[i]));
                assign: message_exist[i]' := no_message;
                        message_exist[(i-1)%NumProcess]' := elected;
                        messages[(i-1)%NumProcess]' := leader;

        TRANS RECEIVED_ELECTED_I_AM_NOT_THE_LEADER:
                enable: ((message_exist[i]=elected)/\(processes[i]!=messages[i]));
                assign: message_exist[i]' := no_message;

                        message_exist[(i-1)%NumProcesses]' := elected;
                relation: ((messages[(i-1)%NumProcess]' = messages[i]) /\
                          (leader' = messages[i]));
}

MODULE SYSTEM(){

VAR
proc1: integer INITVAL 0; proc2: integer INITVAL 1; proc3: integer INITVAL 2;
proc4: integer INITVAL 3; proc5: integer INITVAL 4;

(((((INIT_ELECTION()|||PROCESS(proc1))|||PROCESS(proc2))|||PROCESS(proc3))|||PROCESS(proc4))|||PROCESS(proc5))
}
```

Figura C.12: Especificação de Eleição de um Líder em CDL

```
% XSB/LMC Code for Leader Election Protocol:
%    Dolev, Klawe, Rodeh,
%    Adapted from SPIN test suite by Y.S.R., 19 Dec 96.
%    Parameterized wrt size of ring and buffer size by C.R., Aug 98.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% { Leader election Protocol }
% M: number of processes

systemLeader(M) ::= one_node(Left, Right, 0, M) | chain(Right, Left, 1, M).

one_node(Right, Left, I, M) ::= node(Temp, Left, I, M) | medium(Temp, Right, []).

chain(Right, Left, I, M) ::=
        if (I is M-1)          then one_node(Right, Left, I, M)
                else {  one_node(Temp, Left, I, M) | { I1 is I+1; chain(Right, Temp, I1, M)} } .

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% {  the node process involved in electing a leader }

node(Right, Left, Id, N) ::= Right ! first(Id); nodeActive(Right, Left, N, Id, 0).

nodeActive(Right, Left, N, Maxi, Nbr) ::=
        Left ? first(NId);
        if (NId \== Maxi)
           then {Right ! second(NId); nodeActive(Right, Left, N, Maxi, NId) }
           else if (Maxi is N-1)
                   then { action(leader); nodeActive(Right, Left, N, Maxi, NId) }
                   else { action(fail) }
  #
        Left ? second(NId);
        if (Nbr > NId /\ Nbr > Maxi)
           then { Right ! first(Nbr); nodeActive(Right, Left, N, Nbr, Nbr) }
           else nodeInactive(Right, Left).

nodeInactive(Right, Left) ::= Left ? Msg; Right ! Msg; nodeInactive(Right, Left).

medium(Read, Write, Buf) ::=
        Read ? Msg; medium(Read, Write, [Msg|Buf])
   #
        Buf \== []; strip_from_end(Buf, Msg, RNBuf); Write ! Msg; medium(Read, Write, RNBuf).

{*
:- datatype message.

message(first(Id)) :- typeof(Id, integer).
message(second(Id)) :- typeof(Id, integer).
*}

{*
strip_from_end([X], X, []).
strip_from_end([X,Y|Ys], Z, [X|Zs]) :- strip_from_end([Y|Ys], Z, Zs).
*}

I2 ::=      one_node(Right, Left, 0, 2) | one_node(Left, Right, 1, 2).

I21 ::=     node(T1, Left, 0, 2) | medium(T1, Right, []) | node(T2, Right, 1, 2) | medium(T2, Left, []).

I22 ::=     node(T1, Left, 0, 2) | node(T2, Right, 1, 2) | medium(T1, Right, []) | medium(T2, Left, []).
```

Figura C.13: Especificação de Eleição de um Líder em XL

A especificação de propriedades para serem validadas pelo verificador MMC é dada através de μ-calculus. Por exemplo, podemos ter:

$$\text{ae\_leader} \mathrel{+}= < - > \text{tt } /\backslash \text{ [-leader] ae\_leader.}$$

, a qual faz referência a uma ação *leader* utilizada na especificação. Nesta propriedade, a ação não-*leader* ocorre recursivamente até que, em algum momento, a propriedade é satisfeita.

Deste pequeno exemplo podemos perceber que μ-calculus, apesar de bastante expressível, requer uma certa familiaridade do usuário com conceitos mais teóricos como ponto-fixo.