Centro de Estudios en Microelectrónica y Sistemas Distribuidos
Cemisid - Facultad de Ingeniería - Universidad de Los Andes

Título:

Exploration of the Syntactical Modeling Capabilities in PROMELA / SPIN of Synergestic Distributed Systems based on any type of ω-automata

Autores:

Páez, Charles R. * y Ruiz, J.N.
Cemisid - Facultad de Ingenieria - Universidad de Los Andes

Contacto:

Páez, Charles.R. * Apartado 135 Merida - Venezuela 5101A
e-mail: paezc@ing.ula.ve o cemchp@faces.ula.ve

ABSTRACT

Syntactical modeling of synergestic distributed systems in high level specification languages like PROMELA, its validation and simulation under state of the art tools like SPIN is an open field in computer science. We propose an experimental and theoretical extension suitable for PROMELA/SPIN by using the semantic capabilities of PROMELA to express the acceptance condition F of all types of ω-automata (Büchi, Muller, Rabin, Street, Kurshan and Manna-Pnueli) with special never claims constructs. Some examples are illustrated.

Key Words : Distributed systems, modeling, validation, ω-automata, temporal logic, VLSI.

April - 1996
Mérida - Venezuela

Accepted in the International Conference on Information System Analysis and Synthesis ISAS-96 - Orlando USA July 1996.

Exploration of the Syntactical Modeling Capabilities in PROMELA / SPIN of Synergestic Distributed Systems based on any type of ω-automata

Páez, ChR. and Ruiz, J.N.
University of Los Andes - Cemisid
Mérida, 5101, Venezuela

ABSTRACT

Syntactical modeling of synergestic distributed systems in high level specification languages like PROMELA, its validation and simulation under state of the art tools like SPIN is an open field in computer science. We propose an experimental and theoretical extension suitable for PROMELA/SPIN by using the semantic capabilities of PROMELA to express the acceptance condition F of all types of ω-automata (Büchi, Muller, Rabin, Street, Kurshan and Manna-Pnueli) with special never claims constructs. Some examples are illustrated.

Key Words : Distributed systems, modeling, validation, ω-automata, temporal logic, VLSI.

1. THEORETICAL SCENE IN ω–AUTOMATA

Holzman [1-2] designed a validation modeling language PROMELA and a tool for analyzing and validate the logical consistency of concurrent and distributed systems named SPIN. PROMELA allows the abstract modeling of synergestic distributed system based on global processes and local and global message channels and variables. Processes in PROMELA specify behavior. Channels and variable specify the enviroment in which processes run. Process interaction and process coordination is at the very basis of the language. Each process models an infinite automaton as an ordered set of statements. The execution of every statement is conditional on its executability. So, statement are blocked or executable. Executability is the basic mechanism of sinchronization.

It is obvious, PROMELA/SPIN community can get used to think that the only type of ω- automata we can model are Büchi ones. We proceed to evaluate PROMELA in order to expand its theoretical scope of modeling capabilities to all six types of ω- automata in literature.

2. ω–AUTOMATA

A *non-deterministic ω-automaton* M over an finite alphabet Σ is a tuple (S, s_0, δ,F), where S is a *finite set of states*, s_0 is an *initial state*, δ: S x S → P(S) is a *transition relation* and F is an *acceptance condition*.

A ω-automaton M is *deterministic* if $\forall s \in S, \forall a \in \Sigma : |\delta(s,a)| \leq 1$. The ω-automaton M is *complete* if $\forall s \in S, \forall a \in \Sigma : |\delta(s,a)| \geq 1$.

A *path* p in M is an infinite sequence of states $s_0s_1s_2... \in S$, that starts in the initial state s_0 and has the property that $\forall i \geq 1, \exists a_i \in \Sigma : \delta(s,a) \ni s_{i+1}$. A path $s_0s_1s_2...\in S^\omega$ in M is a *run* r of an infinite word $a_0a_1a_2... \in \Sigma^\omega$ if $\forall i\geq 1 : \delta(s,a) \ni s_{i+1}$.

An *infinite word is accepted by an ω-automaton* M if it satisfies the F acceptance condition. The *infinitary set of sequences* $s_0s_1s_2...\in S^\omega$, $inf(s_0s_1s_2 ...)$ is the set of all the states that appears infinitely often in the sequences.

3. CLASSIFICATION OF ω–AUTOMATA

There are six types of ω-automata depending of the acceptance condition F : B-automaton (Büchi) [3], M-automaton (Muller) [4], R-automaton (Rabin) [5], S-automaton (Street) [6], L-automaton (Kurshan) [7] and ∀-automaton (Manna y Pnueli) [8].

If M is a *B-automaton* the $F \subseteq S$ is a set of states. Therefore, a run r is *accepted by a B-automaton* M if

$$inf(r) \cap F \neq \phi \qquad (1)$$

Thus, some states $s_f \in S$ are specified as *accepting states*. In order for a word to be accepted, these accepting states must occur infinitely often during a run r on the B-automaton M.

$$\exists s_f \in F : s_f \in inf(r) \qquad (2)$$

The acceptance condition of a *M-automaton* M is a set $F \subseteq P(S)$ of sets of states. Therefore, a run r is *accepted by a M-automaton* M if

$$inf(r) \in F \qquad (3)$$

Thus, Muller´s acceptance condition consists of a set in which element is a set of states. In order for a word to be accepted by a M-automaton, the set of states that occurr infinitely often during a run r of M on the word must be one of the elements of the acceptance set.

If M is a *R-automaton*, then the acceptance condition has the form $F = \{ (U_1,V_1),...,(U_n,V_n)\}$, where $U_i,V_i \subseteq S$.

A run r is *accepted by a R-automaton* M if there exists $i \in \{1,...,n\}$ such that

$$inf(r) \subseteq U_i \text{ and} \qquad (4a)$$

$$inf(r) \cap V_i \neq \phi \qquad (4b)$$

For a *S-automaton* its acceptance condition has the form $F = \{ (U_1,V_1),...,(U_n,V_n)\}$. A run r is accepted by a S-automaton if for every $i \in \{1,...,n\}$

$inf(r) \subseteq U_i$ or (5a)

$inf(r) \cap V_i \neq \phi$ (5b)

The acceptance condition for a *L-automaton* M is a pair F = (Z,V), where Z $\subseteq$ P(S) and V $\subseteq$ S. A run is *accepted by a L-automaton* if either :

$inf(r) \subseteq U$ for some $U \in Z$, or (6a)

$inf(r) \cap V \neq \phi$ (6b)

In the case of a $\forall$*-automaton* M its acceptance condition is F = (U,V) $\subset$ SxS. A run r is *accepted by the* $\forall$*-automaton* ,if either :

$inf(r) \subseteq U$, or (7a)

$inf(r) \cap V \neq \phi$ (7b)

4. ω–REGULAR LANGUAGES

The set of words accepted by an ω-automaton M is called the *ω-regular language of M* and denoted by L^ω (M). The *cardinality of* L^ω *(M)* is the number of R words accepted by M.

$L^\omega (M) = \{ a_1a_2... \in \Sigma^\omega \mid a_1a_2... \text{ is accepted by M} \}$ (8)

5. PROMELA CAPACITY FOR MODELING ALL KINDS OF ω–AUTOMATON

First, PROMELA was designed for modeling Büchi automata in mind. Our main target is to explore its intrinsic capabilities for modelling the rest of other types of ω-automata. Due the fact all of them differ in the definition of their acceptace condicion F. We summary the never claim construct for every kind of F acceptance condition of each ω-automata.

PROMELA validation models are finite. So the number of execution sequences that constitute its ω-regular language are bounded and enumerable. Therefore, there are terminating sequences and cyclic sequences. The specification of correctness requirement are defined by propositions on system states. There are three ways in which a correctness criteria on a model can be expressed in PROMELA : assert statement, validation labels (accept, progress, end) and temporal claims.

The acceptance conditions F of every type of ω-automata must be expressed as a condition that cannot happen infinitely often. In order to express such a condition we can use PROMELA segments based on acceptance state labels (proposed in PROMELA to formalize that something cannot happen infinitely often) within never-claims constructs (proposed in PROMELA to formalize linear time temporal logic formulae or behavior that is claimed to be impossible). Therefore, a correctness violation occurs if and only if a temporal claim is matched by a system behavior and it occurs infinitely often. In the following paragraphs we propose the modeling of the corresponding F acceptance condition of every type of ω-automata.

6. ACCEPTANCE CONDITION F OF ω–AUTOMATON IN PROMELA:

For a *Büchi automaton*, F is a set of acceptance states that must be reached infinitely often. So, we define those states $s_f \in S$ and $s_f \in F$ as *progress states* and proceed to validate searching for non-progress states. If we find a non-progress cycle; then there exist at least a run do not accepted for the B-automaton.

In those cases where it is not appropiate to validate the model using non-progress state approach, we can use temporal claims in order to probe the no existance of a ciclic sequence of states $s_j \in S$ and $s_j \notin F$. If we label correctly the states $s_f \in S$ and $s_f \in F$, the body of the never-claim must be of the form shown in Table 1. So, if in any run it is executed infinitely often a a ciclic sequence of states $s_j \in S$ and $s_j \notin F$ will exist an error. This condition looks like a typical non-progress condition.

Table 1 . PROMELA Never-claim for B-automaton

F Acceptance Condition	PROMELA never-claim

$F \subseteq S$
 	
$inf(r) \cap F \neq \emptyset$
	never {
	accept:
	do
	::!(process[pid]@F_1 ||
	...|| process[pid]@F_n)
	od
	}

We propose to use an acceptance-state within a never claim construct in order to satisfy the *ω-automata Muller's* F acceptance condition is a set F $\subseteq$ P(S) of sets of states contained in a state transition relation, as shown in Table 2.

Table 2. PROMELA Never-claim based on state transition in a process for M-automaton

F Acceptance Condition	PROMELA never-claim

$F \subseteq S$
 	
$inf(r) \in F$
	never {
	do
	:: skip
	:: process[pid]@STATE1
	-> break
	od;
	accept1:
	do
	:: !(process[pid]@STATE2)
	:: process[pid]@STATE2
	-> break
	od;
	accept2:
	do
	:: !(process[pid]@STATE3)
	od
	}

The acceptance condition F = { $(U_1,V_1),...,(U_n,V_n)$}, in a *Rabin automaton*, is a set of pair of states $U_i,V_i \subseteq S$. So, in order to accept a run r, each R-automaton must reach any state U_i and infinitely often must visit those states V_i. This condition can ve validated in two steps, as shown in Table 3.

First we verify the run r reached a Ui state by using a monitor process that initialize a global variable to

TRUE. and making the variable FALSE whitin the process when it reach a state U_i .
Then, the second part of the validation consist in to verify that the run reach the Vi state infinitely often. This can be made in a B-automaton style. We propose an alternative way. Labeling the states Vi as STATE_V and verifying an imposible temporal claim
The acceptance condition F of an *Street automaton* consists of pairs (U,V) of states such that in order the run r to be accepted by the automaton, it must go through any state $s_i \in U$ *or* through any state $s_j \in V$ infinitely often. In order to model this acceptance condition F is enough to add an additional possibility in the never claim that we used for the Muller automaton : The option of that the run r have circulated through any state $s_i \in U$. We use a binary variable FLAG with value1 by default that change its value to 0 whenever the run goes through any state $s_i \in U$. All states $s_j \in V$ are labeled as STATE_V. Thus we propose the following never claim construct.

Table 3. PROMELA Never-claim for R-automaton

F Acceptance Condition	PROMELA never-claim

$F = \{ (U_1,V_1),\ldots,(U_n,V_n)\}$, where $U_i,V_i \subseteq S$
 	
$inf(r) \subseteq U_i$ and $inf(r) \cap V_i \neq \phi$
	First of all:
	byte flag = 1;/* global var.*/
	...
	inside process where is STATE_U
	STATE_U:
	statement;
	statement;
	flag = 0;
	...
	proctype monitor()
	{ accept: flag == 1 }
	
	and as a second part :
	
	never {
	do
	:: skip
	:: !(process[pid]@STATE_V)
	-> break
	od;
	accept:
	do
	:: !(process[pid]@STATE_V)
	od;
	}

For a *Kurshan automaton*, its acceptance condition F is a pair (Z,V), where Z is a set of state transitions and V a set of states. In order to be accept an infinite run r in a L-automaton, it must goes infinitely often through states $s_i \in V$ or through states transitions in Z. In order to model this condition as a impossible assertion we may set a label STATE_V in any state $s_i \in V$, and assuming that in any state transition in Z, a binary variable VZ goes from 1 to 0.

Then we propose the following never claim construct. Thus, if the variable VZ goes from 1 to 0 or if the run goes through any state labeled as STATE_V; then the stablished condition in the never claim it is impossible and we can assert that the model of the L-automaton is correct.

Table 4 . PROMELA Never-claim for S-automaton

F Acceptance Condition	PROMELA never-claim

$F = \{(U_1,V_1),\ldots,(U_n,V_n)\}$, where $(U_i,V_i) \subseteq S$
 	
$inf(r) \subseteq U_i$
or
$inf(r) \cap V_i \neq \phi$
	never {
	do
	:: skip
	::(Zf == 1) &&
	!(process[pid]@STATE_V)
	->break;
	od;
	accept:
	do
	:: !(Zf == 0) &&
	!(process[pid]@STATE_V)
	od;
	}

Table 5 . PROMELA Never-claim for L-automaton

F Acceptance Condition	PROMELA never-claim

$F = (Z,V)$, where $Z \subseteq P(S)$ and $V \subseteq S$.
 	
$inf(r) \subseteq U$ for some $U \in Z$,
or
$inf(r) \cap V \neq \phi$
	never {
	do
	:: skip
	::(VZ==1) &&
	!(process[pid]@STATE_V)
	->break;
	od;
	accept:
	do
	:: !(VZ==0) &&
	!(process[pid]@STATE_V)
	od;
	}

An *automaton of Manna and Pnueli* has an acceptance condition F similar to the one exhibited by a Street automaton. So, the following never claim construct will be used, as shown in Table 6.

Table 6 . PROMELA Never-claim for ∀-automaton

F Acceptance Condition	PROMELA never-claim
$F = (U,V) \subset SxS$. $inf(r) \subset U$, or $inf(r) \cap V \neq \phi$	never { do :: skip ::(Zf==1) && !(process[pid]@STATE_V) -> break; od; accept:

	do :: !(Zf== 0) && !(process [pid]@STATE_V) od; }

7. EXPERIMENTAL EXTENTIONS OF SPIN

As the correctness criteria expressed by temporal claims with acceptance statement in its interior do not specify independent system behavior we can exercise our PROMELA models, independently of the type of ω-automata we used in its specification, with the SPIN tool. We report the sucessfull modeling and validation [10] of three examples proposed in the literature : a producer/consumer system[11], an HDLC communication protocol [12] and a VLSI arbiter [13] based on different ω-automata based on the proposed PROMELA constructs.

9. ACKNOWLEDGEMENT

Thanks to G. Holzman at ATT, creator of PROMELA language and the SPIN verification tool, for his kind early review of the paper and for his opportune comments.

10. REFERENCES

[1] Holzman GJ., *Design and Validation of Protocols*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[2] Holzman, GL., Design and validation of protocols : a tutorial, Computer Networks and ISDN Systems (25):981-1017, 1993.

[3] Büchi, JR., On a decision method in restricted second-order arithmetic. In Proceedings International Congress on Logic Method and Philosophy of Science, 1960, pp.1-12, Stanford University Press, 1962.

[4] Muller, DE., Infinite sequences and finite machines. In Switching Circuit Theory and Logical Design, Proceeding Fourth Annual Symposium, pp.3-16, 1963.

[5] Rabin, MO., Decidability of seco nd-order theories and automata on infinite trees, Trans. American Mathematical Society, 141:1-35, 1969.

[6] Street, RS., Propositional dynamic logic of looping and converse in elementary decidable, Information and Control, 54:121-141,1982.

[7] Kurshan, RP., *Testing containment of ω-Regular Languages*, Technical Report 1121-861010-33-TM, Bell Laboratories, 1986.

[8] Manna, Z. and Pnueli, A., Specification and verification of concurrent programs by ∀-automata, In Proceeding, Fourteenth Annual ACM Symposium on Principles of Programming Languages, pp.1-12, 1987.

[9] Clarke, EM., Draghicescu, LA. and Kurshan, RP., *A unified approach for showing language containment and equivalence between various types of ω-automata*, Technical Report CMU-CS-89-192, Carnegie Mellon U. 1989.

[10] Ruiz, JN., *Modelaje de Soluciones Digitales de Hardware y de Protocolos de Comunicacion*, RT-Cemisid-53-1995, Tesis Ing. Sistemas, Universidad de Los Andes, 1995.

[11] Schlichting, R. Schneider, F., Using message passing for distributed programming: Proof rules ans disciplines, ACM Trans on Programming Languages and Systems , 6-3:402-431, July, 1984

[12] ISO 6225 Data communication - HDLC balanced class of procedures

[13] Bochmann, G., Hardware specification with temporal logic: An example, IEEE Trans on Computers, C-31, 3:223-231, March, 1982.