# SPECIFICATION AND VERIFICATION OF PARAMETRIZED PARALLEL PROGRAMS

**Isabelle Vernier**
Laboratoire MASI
Université Pierre et Marie Curie
4, place Jussieu
75252 PARIS CEDEX 05
Email : vernier@masi.ibp.fr
Tel : (33-1) 44-27-71-04
Fax : (33-1) 44-27-62-86

**Abstract**: In this paper, we extend the class of algorithms that can be verified with our method presented in [6]. We present a model that allows to specify parallel and distributed parametrized programs. The behavior of the processes is modeled by an extended Petri net. We can verify required properties using temporal logic formulae.

## 1. INTRODUCTION

Parallelism requires new programming languages that support communication and shared variables management mechanisms as CSP [7], ADA [1]. When a program is written, its author has requirements on the way it behaves. Therefore it is necessary to verify that the reality is as the wishes. As for the programming languages, new verification methods are necessary. As it is not possible to verify the behavior of a program directly from its programming language specification, models have been developed.

The verifications method, based on the state graph, do not take into account that many parallel programs involve a finite but unknown number of processes. Some solutions have already been developed to avoid this problem [3], [13], [14]. But none of them gives full satisfaction.

In [6] we have presented an automatic method. We first build a parametrized state graph where each state is a set of states of an ordinary graph. Once the graph built, we can evaluate CTL formulae by an adequate algorithm that is an adaptation of the classical algorithm [2].

A common feature of the previous methods is the low level of the studied models. All the processes must have exactly the same behavior. Many parallel algorithms can hardly or can not be modeled with them. It is sometimes useful to distinguish a process. This is the case of most of the algorithms for mutual exclusion[11]. To validate such programs, we need a specification model that allows automatic verification methods.

In this paper we present a model that fills in the lacks of the previous one and that can be automatically verified. The model is an extension of Petri nets [12]. Some abbreviations allow to simplify the modelization of the algorithms and the understanding of the model. To complete the specification, we express the required properties with temporal logic formulae. From the model, we can build a parametrized state graph. Therefore we can verify the properties of a program by model checking. The two algorithms that allow this verification are presented in [6].

In section 2 we introduce the model. In section 3 we give an example. In section 4 we show how to express properties with temporal logic formulae. Finally in section 5 we conclude and present further works.

## 2 FROM PETRI NETS TO EXTENDED PARAMETRIZED NETS

The Petri nets are a powerful model and can be automatically validated. Ordinary Petri nets [12] do not offer the possibility to model parametrized parallel programs. Colored Petri nets [8] are too complex to be analyzed. Therefore we have extended the ordinary Petri nets to be able to model parametrized parallel programs. And the extensions are manageable since they allow the building of the parametrized state graph.

### 2.1 Control variables

The behavior of processes is modeled by state machines. If a process is distinguished from the others, its behavior must be modeled by another state machine. With ordinary Petri nets, the number of places and transitions is doubled. Then, if there is more than one distinguished process, the size of the net will be multiplied by the number of distinguished processes. To avoid this multiplication, we have introduced control variables. They are defined in a definition part of the net. A control variable abbreviates a state machine. Its value is the place where the process is. We associate a control variable with each process that is distinguished.
The control variables allow to govern the firing of some transitions. Conditions associated with the transition specify the processes that can (or can not) fire it. If the condition is equal to true, all the processes can fire the transition.
The firing of a transition can modify the value of some control variables. This new value is the place reached by the new distinguished process. These modifications are specified with actions that are executed after the firing of the transition and before the firing of another.

### 2.2. Test places

The progress of a process is usually dependent on the state of the other processes. Therefore a process must be able to test the state of the others. In the case of ordinary Petri nets, this is modeled with inhibitor arcs between transitions and places. The firing conditions are merged with the state machines. In our model, we propose to abbreviate inhibitor arcs with a test place. Then, the test places and the processes places differ from each other.
A test place is a list of processes places. The value tested is the number of processes in the places of the list. A test place can be restricted with control variables. In this case we count only the processes with which the control variables are associated.
To test the value of a test place, we allow to kind of arcs: the inhibitory arcs and the test arcs. They both are allowed only between test places and transitions. An inhibitory arc has the usual meaning. A test arc has the opposite effect on transitions as an inhibitory arc. A transition, linked to a test place by a tesr arc, may be fired only if the value of the test place is more or equal to the valuation of the arc. A test arc is denoted by an arc with arrows at the two ends. Graphically, an inhibitory arc has a circle at one of its ends.
To differentiate the test places from the processes places, they are denoted with a question mark inside.

### 2.3 Resources and control process places

In our model, the communication component is a control process. It is modeled by an ordinary Petri net. This control process manages the resources placed at processes' disposal. The control process places are bold. The processes state machine and the control process Petri net are synchronized by the transitions.

## 3. EXAMPLE

We want to model the writing and reading accesses to a file by an undefined number of processes. The same processes can alternately be reader or writer. One process is identified as the owner of the file. The last process that has written becomes the owner. The owner process has writing priority. A process can not write if the owner process is waiting for writing or if some processes are reading or writing. These conditions are modeled by test places and Write transitions that are not the same for the owner process and the other. A process can not read if one process is writing or if the owner process is waiting for writing. The number of allowed simultaneous readings is defined by a ressource of the control process and modeled by a place. In this example, we allow two simultaneous readings

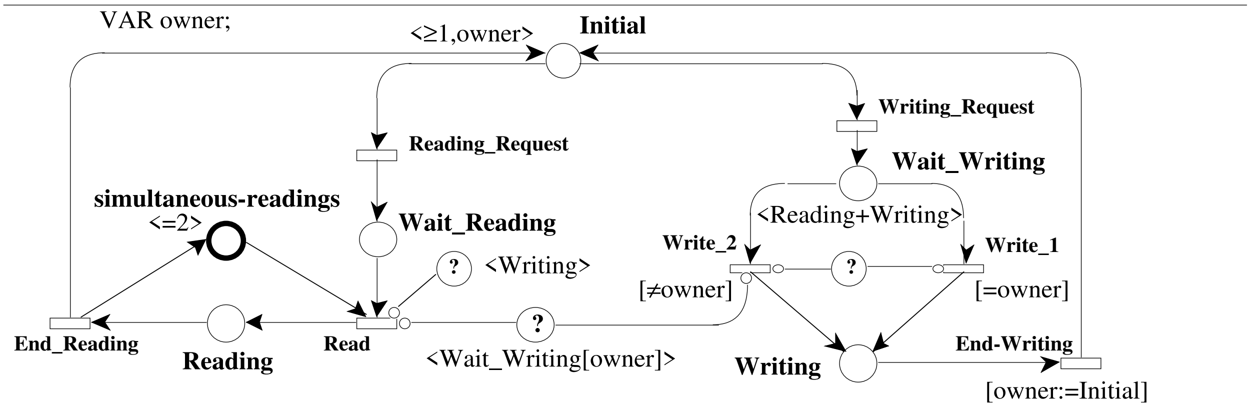
A model is complete if its initial state is specified. As the model is a parametrized one, the initial marking is too. To each place we have to specify the number or the minimal number of processes that are in it and if a distinguished process is too.

Initially two tokens are in place simultaneous_readings. The initial parametrized marking associated with this place is <=2>.

All the processes are in the initial state. We only know that there is at least one token in this place. The owner process is in this place too. The parametrized marking associated with the place Initial is <≥1,owner>.

No process is in the other places. Therefore we associate no parametrized markings with them.

The model is given by the figure 1. The parametrized Petri net model holds 11 places, 16 transitions and 67 arcs as the extended parametrized net model holds 9 places, 7 transitions and 21 arcs. The extended parametrized net offers a better readability. The parametrized state graph can be built directly from this model. No unfolding is necessary.




## 4. PROPERTIES SPECIFICATION AND VERIFICATION

To prove that a program runs "correctly", it is necessary to express what correctly means. Its signification is different for each program and has to be specified by the programmer. Temporal logic formulae seem to be an appropriate specification language.

Simple temporal formulae express the usual properties of parallel programs [4]. Properties of finite state programs are checked by efficient inspection of the accessibility graph [10] [2] [5]. Therefore, we have chosen a temporal logic language to complete the specification of the programs. Though CTL has a less expressive power as other languages (e.g. CTL*), we have chosen it for the following reasons:

- the model checking of a formula on the state graph is linear in the size of both the graph and the formula,

• CTL is sufficient to express usual properties as liveness and safety.

Example:
Concerning the readers-writers example, some properties that we want to verify are :
- there are no simultaneous writings ($P_1$),
- the number of simultaneous readings is equal to two ($P_2$).

In the two cases, we want that each reachable parametrized state verifies the property. The associated formula is ¬∃(true U ¬($P_1$ and $P_2$)). A parametrized marking, pm, verifies this formula if from pm there is no path leading to a state that does not verify ($P_1$ and $P_2$).

The property $P_1$ is ¬(Writing > 1) and the property P2 is ¬(Readings > 2). Therefore the whole formula is ¬∃(true U ¬(¬(Writing > 1) and ¬(Readings > 2))). More simply, the formula is ¬∃(true U ((Writing > 1) or (Readings > 2))).

## 5 CONCLUSION

In this paper we have presented a way to specify a parametrized parallel program. An automatic verification method can be applied to the model.

Further works will deal with the ability to consider order on the processes. It allows to model algorithms that assume that processes are on a ring and can communicate with their neighbors.

Another improvement of our work is the consideration of more complex formulae languages such as CTL*, FCTL [10].

## REFERENCES


**[1]** Booch G., *Software Engineering using Ada*, Benjamin Cummings, 1987.

**[2]** Clarke E.M., Emerson E.A., Sistla J., *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification*, ACM Transactions on Programming languages and systems, vol 8, n°2, April 1986, pages 244-263.

**[3]** Clarke E.M., Grümberg O., *Avoiding The State Explosion Problem in Temporal Logic Model Checking Algorithms*, Proc. 6th ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, August 1987, pages 294-303.

**[4]** Emerson E.A., Srinivasan J., *Branching Time Temporal Logic* , Branching Time and Partial Order in Logics and Models for Concurrency ,Workshop on Linear Time, LNCS 354, 1988, pages 123-172.

**[5]** Emerson E.A.,Sadler T., Srinivasan J., *Efficient Temporal Reasoning*, Proc. of the 16th ACM Symposium of Principles of Programming Languages, Austin, Texas, 1989, pages 166-178.

**[6]** Haddad S., Vernier I., *Evaluation Paramétrée de Formules de Logique Temporelle*, Proc. 5èmes Rencontres sur le Parallélisme, Brest, France, Mai 1993, pages 39-42.

**[7]** Hoare C.A.R., *Communicating Sequential Proceses*, Prentice Hall Int., 1985.

**[8]** Jensen K., Rozenberg G. (Eds), *High-Level Petri Nets*, Springer-Verlag, 1991.

**[9]** Karp R.M., Miller R.E, *Parallel Program Schemata*, Journal of Computer System science, vol. 3, n°2, May, 1969.

**[10]** Lichtenstein O., Pnueli A., *Checking That Finite State Concurrent Programs Satisfy Their Linear Specification*, Proc. 12th ACM Symposium of Principles of Programming Languages, New-Orleans, Louisiana, January 1985, pages 97-107.

**[11]** Raynal M., *Algorithms for Mutual Exclusion*, North Oxford Academic Publishers Ltd, 1986.

**[12]** Reisig W., *EATCS Monograps on Theoretical Computer Science*, vol. 4, Springer Publ. Company, 1985.

**[13]** Sistla A.P., German S.M., *Reasoning with Many Processes*, Proc. Symposium on Logic in Computer Science, Ithaca, New-York, June 1987, pages 138-152.

**[14]** Wolper P., Lovinfosse V.,*Verifying Properties of Large Sets of Processes with Network Invariants*, Automatic, Proc. International Workshop on automatic Verification Methods for Finite State Systems, Grenoble, France, 1989, LNCS 407, Springer Verlag, pages 68-80.