Petri Nets Analysis: Complexity and Finite Coverability Graph in Modular Design

Didier Crestani, Alain Jean-Marie, Christine Coves

Petri nets analysis : Complexity and finite coverability graph in modular design

D. Crestani [1], A. Jean-Marie [1], C. Covès [1]

1 - Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier - LIRMM - 161, rue Ada, 34392 Montpellier cedex 5, France - Tel. (+33) 467 418 513 - Fax : (+33) 467 418 500 - Email: crestani@lirmm.fr

Summary: The Petri net is a very efficient model to describe and analyse the behaviour of Discrete Event Systems. However, faced to the complexity, modular design is needed to deal with large systems. The coverability graph is a useful tool allowing to analyse system's properties. But its capacities are limited to finite coverability graph merging for modular design. This paper studies the temporal complexity of finite coverability graph construction using the minimal coverability graph algorithm. It focuses on modular design using shared transitions and concludes on the advantages and drawbacks of this class of approach.

Key words: Petri nets, Modular design, Finite coverability graph, Algorithm complexity

1. Introduction

The Petri net is an efficient model to describe and analyse Discrete Events Systems. Very important properties like boundedness and quasi-liveness can be studied by using algorithms allowing to generate the reachability graph of a Petri net describing a system behaviour. Faced to the state explosion problem, many techniques have been developed. Some of them try to reduce the complexity of the initial net by using reduction rules [1] [2]. Others use structural analysis [3] [4] or limited unfolding approaches [5] [6] [7]. However, the coverability graph due to its simple algorithms remains a largely used technique [8].

The classical coverability graph algorithm has been proposed by Karp and Miller [9]. It has been optimised latter by Finkel [10] who has proposed an algorithm allowing to construct the minimal coverability graph.

This paper focuses on the asymptotic temporal complexity (the worst case complexity), which allows to estimate the maximal duration of the computation of the minimal coverability graph. We revisit this problem, rarely studied as far as we know, taking into account the size of the studied net (P Places and T Transitions) but also considering the number of nodes N of the final coverability tree. Furthermore we more precisely analyse the temporal complexity generated when a modular design using shared transitions is needed to describe strongly connected systems. In this case, from sub-

systems Petri nets, only finite coverability graph can be merged to analyse the global system. Then we study the advantages and drawbacks of this "divide and conquer" approach in terms of design facilities, memory size complexity and temporal complexity.

We firstly remind the necessity of modular design to deal with systems complexity. Then we present the minimal coverability graph and point out the limitation of coverability graph merging when unbounded systems are associated. The asymptotic temporal complexity of the minimal coverability graph is then studied for finite reachability graph. To deal with modular design, a merging algorithm is presented and its associated temporal complexity estimated. Finally the obtained results and the relevance of a "divide and conquer" approach using modular design and shared transitions are discussed.

2. Modular Design to deal with complexity

Despite its exceptional capacities to deal with Discrete Events Systems by modelling easily and graphically sequentiallity, concurrency, rendez-vous, etc., behaviours, and analysing efficiently the modelled systems, Petri nets remain difficult to use when large and complex systems must be modelled. Then, the problem is not a problem of the model capacities but a problem of Human handling of world complexity. It is difficult to construct, understand and more generally consider a Petri net which cannot be drawn on a A4 format page. To deal with this mental limitation, a classical mean is to use modular design.

A first class of modular design concerns the top-down approaches. In this kind of design, each place (transition) can be associated to a complex operation. A multi-level refinement mechanism allows to describe more and more deeply the operations complexity. The main advantage of this class of modular approach is to preserve easily the system properties by using well formed nets for place refinement [11] [12] [13] or transition refinement [14] [12]. However, it presents also two main drawbacks. On the first hand, the designer is limited in his design because the sub-level nets must be well-formed nets. On the second hand, the complexity power of description is limited because the sub-systems cannot be interconnected due to the refinement mechanism.

A second class of modular design concerns the bottom up approaches. Then each sub-system is modelled by a sub-nets of limited complexity. These different nets are merged together by using shared places or transitions. This approach has been largely used to model flexible manufacturing systems [15] [16] [17] or to facilitate the global invariant computing [18] [19] [20]. The main advantage of the sharing design is that it allows to model strongly connected systems. Unfortunately, the analysis facilities are limited and the local results cannot always allow the full analysis of the global design.

In the following we deal with bottom-up modular design of complex system. Furthermore, we focus on shared transitions techniques due to its synchronising capabilities.

3. The Minimal Coverability Graph and Modular Design

A. The minimal coverability graph algorithm

Karp and Miller have proposed in 1969 [9] an algorithm allowing to construct a Petri Net coverability graph. This graph is obtained from the coverability tree by merging the identical nodes. This algorithm has been improved later by Finkel [10] who has developed an algorithm allowing to generate the unique minimal coverability graph. This new algorithm uses properties depending on the monotonic behaviour of nets, to reduce and compact the Karp and Miller graph during its generation.

With these algorithms the following problems can be decided [10]:

- finiteness of the reachability tree;
- the boundedness of the Petri net places;
- the quasi-liveness of the Petri net transitions.

The ***Figure 1*** presents the minimal coverability tree algorithm structure. In this procedure **mi** denotes the marking of a node **ni**, and **n** represents the node currently treated with its marking **m**.

Four cases are distinguished. Cases 1 and 2 stop a useless development of the coverability tree. They refine the equivalent configuration considered in the Karp and Miller algorithm. Case 3 detects a non bounded node by considering the entire tree and removing useless sub-trees if possible. Finally, case 4 generates new tree nodes.

B. Modular design and coverability graph limitation

We consider now a system divided into two sub-systems. The coverability graph of each sub-net can be obtained independently. Then, if we suppose that a bottom-up modular design using shared transitions is used, then it seems at first sight that the global coverability graph can be obtained by composition of the two sub-nets coverability graphs.

Unfortunately, this reasoning cannot be employed when non boundedness sub-nets are considered. For example it is easy to note that in ***Figure 2*** the simple global Petri net PN0 can be considered as the result of the merging of the two sub-nets PN1 et PN2.

It is clear that the global net PN0 is bounded. However, if we consider the Finkel algorithm the merging of the non bounded coverability graphs CG1 and CG2 of PN1 and PN2 leads to a non bounded coverability graph CG0 for PN0 (***Figure 3***).

The same kind of result is also obtained if we consider the classical coverability graph construction algorithm proposed by Karp and Miller (***Figure 4***).

These false results are due to the loss of information induced in the coverability graph when the symbol ω is introduced to deal with non bounded nets. Due to this important limitation we consider in the following, the generation of *finite* coverability graphs and their merging within a bottom-up modular design using shared transitions.

4. The Minimal Coverability Graph temporal complexity

A. Study assumptions

We suppose in the following that the number of places P and transitions T of the studied Petri net is much smaller than the number of nodes N of the generated finite coverability tree.

The temporal complexity evaluates the duration T(n) of an algorithm execution by counting the number of executed instructions for some data of size n. A constant α allows to integrate the computer influence. The notation O(.) is used to express the algorithmic complexity.

In the following we consider that basic operations like reading, writing, or marking comparison have an algorithmic complexity in O(1).

Finally, we suppose that during its execution, the considered algorithm suppresses K nodes.

B. The finite minimal coverability tree temporal complexity

In the following we consider independently the temporal complexity of each case of the algorithm

before evaluating the minimal coverability tree complexity. Let Ti(n) be the total amount of time spent in Case i during an execution of the algorithm, given that the underlying coverability tree is of size N.

1. **Case 1 : m = m1 - Detection of an identical known marking**

The associated complexity is limited because it depends only on the input test. In the worst case less than N+K nodes can be considered to detect if the marking of the current node is equal to another one. The complexity can be expressed by Eq.(1).

$$T1_{Finkel}(N) = a \sum_{n \in X} a(n) = a \,.\, \sum_{n=1}^{N+K} N+K-1 = a\,.(N+K).(N+K-1) \qquad \text{Eq. (1)}$$

In Eq.(1) n represents the currently treated node, X the set of all the tree nodes (of size less than N+K), a(n) the set of the previously generated nodes (of size less than N+K-1), and α the computer dependent coefficient.

From the previous expression it is easy to conclude that the complexity of case 1 is $O((N+K)^2)$.

$$T1_{Finkel}(N) = O((N+K)^2) \qquad \text{Eq.(2)}$$

2. **Case 2 : m < m1 - Detection of a higher known marking**

This case is not treated in the Karp and Miller algorithm. It presents to develop unnecessarily the coverability tree and permits to remove the current node. For a bounded net, which is the studied configuration, the K removals are realised in this part of the algorithm. For the case 2 the complexity depends on the input test complexity and on the complexity of the removal operation.

For the input test complexity, it is easy to remark that it is equivalent to the case 1 complexity (Eq.(3)).

$$T2_{Finkel}(\text{input test}) = O((N+K)^2) \qquad \text{Eq.(3)}$$

For the remove operation complexity, if β represents the cost of a removal (supposed to be independant of N) and α the computer depending factor, then the global cost of the K removing can be expressed by Eq.(4).

$$T2_{Finkel}(\text{removing operations}) = \alpha.K.\,\beta \qquad \text{Eq.(4)}$$

It is evident that the removal complexity can be neglected, compared with the input test one. So, finally, case 2 complexity is Eq.(5).

$$T2_{Finkel}(N) = O((N+K)^2) \qquad \text{Eq.(5)}$$

3. **Case 3 : non boundedness detection**

Since we consider in this paper a bounded net, during the algorithm execution there is only evaluation

of the input test. This test detects if the marking of node under treatment is lower than the marking of one the tree nodes.

Like for cases 1 or 2 this complexity can be expressed by :

$$T3_{Finkel}(N) = O((N+K)^2) \qquad \text{Eq.(6)}$$

4. Case 4 : New nodes generation

For this last case, the fireable transitions must be detected for each tree marking. The corresponding complexity can be evaluated by the following equation:

$$T4_{Finkel}(N) = a \sum_{n \in X} (t + C(n)) \qquad \text{Eq. (7)}$$

In this equation C(n) corresponds to the nodes generation complexity from node n. The sum Σ C(n) is the total number of nodes generated and is therefore less than N+K, and t the complexity associated to the consideration of all the possible fireable transitions (less than T).

Finally, this complexity can be estimated by Eq.(8):

$$T4_{Finkel}(N) = O((N+K)+(N+K).T) = O((N+K).T) \qquad \text{Eq.(8)}$$

5. Finite coverability tree temporal complexity

Table 1 summarises the previously obtained results:

The complexities of cases 1, 2 and 3 are equivalent and higher than the complexity of case 4 which is linear. So, it is easy to conclude that the temporal complexity of the finite minimal coverability tree construction is quadratic with the number of tree nodes (Eq.(9)).

$$T(Finkel\_tree) = O((N+K)^2) \qquad \text{Eq. (9)}$$

C. The finite minimal coverability graph temporal complexity

From the finite minimal coverability tree, the coverability graph is constructed by identifying and merging identical nodes. It can be proved, that this kind of transformation can be realised with a linear temporal complexity estimated by Eq.(10) [21].

$$T(tree/graph) = O(N.P) \qquad \text{Eq. (10)}$$

Moreover, the Finkel graph algorithm removes the useless arcs. However this operation does not change the temporal complexity transforming the tree in a graph.

Finally, since the complexity of graph transformation can be neglected compared with the complexity of the tree construction we can conclude that:

$$T(\text{Finkel finite coverability graph}) = O((N+K)^2) \quad \text{Eq. (11)}$$

5. Modular Analysis using Minimal Coverability Graph Merging

A. The merging algorithm

The merging algorithm proposed by Finkel [22] is directly inspired by the previous developed version for the global coverability graph construction [10]. However, the author has not taken into account the construction limitation shown previously. ***Figure 5*** presents a restricted version allowing to merge two finite sub-coverability graphs FCG1 et FCG2, to construct a global finite coverability graph FCG0. These graphs correspond to sub-Petri nets R1 and R2 having shared transitions to design a global Petri net R0.

This algorithm uses the following notation rules:

- n represents the node under treatment and m its marking;
- M01 (M02) is the initial marking of R1 (R2);
- T1 (T2) is the set of transitions of R1 (R2) not shared with R2 (R1) and Ts the set of shared transitions;
- **merge**(m1,m2) permits to merge a marking m1 of R1 with a marking m2 of R2 in a unique marking m = m1|m2 corresponding to the marking of R0.

B. The temporal complexity

To estimate the temporal complexity of this algorithm we suppose, as previously, that the basic algorithm functions (merging, node creation and removing) can be executed efficiently with an O(1) complexity.

1. Case 1 : m = m1 - Detection of an identical known marking

The structure of this part of the merging algorithm is strictly equivalent to global form. Then its complexity can be expressed by the same expression Eq. (12).

$$T1_{merging\_Finkel}(N) = O((N+K)^2) \quad \text{Eq. (12)}$$

2. Case 2 : m < m1 - Detection of a higher known marking

The same remark can be done for this part of the merging algorithm. The corresponding complexity

is then Eq. (13):

$$T2_{merging\_Finkel}(N) = O((N+K)^2) \qquad \text{Eq. (13)}$$

3. Case 3 : New nodes creation

This case needs to detect the fireable transitions from at least one sub-coverability graph FCG1 and/or FCG2. The temporal complexity associated to each of the relevant configurations (fireable transition belonging to FCG1, FCG2 or shared) is identical to the one estimated for the same case of node creation in the global Finkel algorithm (Eq. (6)). So, we can write Eq. (14):

$$T3_{merging\_Finkel}(N) = a \sum_{n \in W} (t + C(n)) \qquad \text{Eq. (14)}$$

In Eq.(14) W represents the set of nodes of FCG0, C(n) the complexity of nodes creation from the treated node. Since finally, there was N+K created nodes, ΣC(n) is necessarily lower than N+K. Finally, t corresponds to the temporal complexity of transition taking into account from one node. Globally in the algorithm this complexity is lower than (N+K)T.

Then the complexity of nodes creation for the merging algorithm can be estimated by:

$$T3_{merging\_Finkel}(N) = O((N + K) + (N + K).T) = O((N + K).T) \qquad \text{Eq. (15)}$$

4. Finite coverability graph temporal complexity

Table 2 summarises the previous obtained results.

Therefore the temporal complexity of the coverability construction with merging or global approach have the same quadratic expression. Since the temporal cost of graph construction from a tree structure is linear with the number of nodes, and can be neglected, we can finally conclude that:

$$T_{FCG0}(N) = O((N+K)^2) \quad \text{Eq. (16)}$$

C. The modular approach : global temporal complexity

The previously obtained expression corresponds only to the temporal complexity of the construction of the final graph FCG0 from the two sub-graphs. In fact, to estimate the real temporal complexity of the modular approach, the calculus must include the temporal cost of the construction of FCG1 and FCG2.

The global expression of the modular approach is then Eq. (17):

$$T_{MODULAR\_FINKEL} = T_{FCG0}(N) + T_{FCG1}(N1) + T_{FCG2}(N2) \quad \text{Eq. (17)}$$

This expression supposes that the finite coverability trees of the nets R1 and R2 have N1 and N2

nodes respectively.

By explaining all the terms of the previous equation we obtain Eq. (18):

$$T_{MODULAR\_FINKEL} = ((N+K)^2)) + ((N1+K1)^2)+ ((N2+K2)^2) \quad \text{Eq. (18)}$$

Where K1 and K2 represent the number of removing nodes during FCG1 and FCG2 construction respectively.

Finally, if we suppose that generally N is greater than N1 and N2 the final expression of the temporal complexity of finite coverability graph construction using modular approach with shared transitions is Eq. (19):

$$T_{MODULAR\_FINKEL} = ((N+K)^2)) \quad \text{Eq. (19)}$$

Then, unfortunately, the use of a modular bottom-up approach leads to the same temporal complexity than the direct approach.

6. Conclusion

In this paper we have studied the temporal complexity of finite minimal coverability graph using the Finkel's algorithm. We have demonstrated that this complexity is quadratic with respect to the final number of nodes in the finite minimal coverability tree. Moreover, we have discussed the advantages and drawbacks of a modular design. The top down approach facilitates the construction of well-formed nets. However the possibilities of design are limited for the user. We have focused on modular design using a bottom up approach with shared transition. In this case, the user can construct very complex structures highly interconnected. But the validation capacities are weak. Then we have investigated the impact of modular design on modular finite coverability construction. Our hope was that the divide and conquer approach like modular design would have a positive effect in space or time with regards to the global approach. Unfortunately, it is obvious that the space complexity of the modular approach is higher than the global one, since the final (global) coverability graph and each of the sub-coverability graphs must be present in the computer memory. We have also demonstrated that the temporal complexity of the merging algorithm is equivalent to the complexity of the direct approach generating the global coverability graph. This demonstration implies that, in fact, a modular approach needs more time than a global one since moreover the time lost to construct each sub-coverability graph must be added. Then, we can conclude that the main but very important advantage of a modular design with shared transitions is that it permits to design step by step, while using the natural system partitions. But these design capacities cannot be exploited to generate a finite coverability graph more efficiently than a direct and global approach.

7. References

1. BOUSSIN, J., **Synthesis and Analysis of Logic Automation Systems**, 7th Triennial World Congress, Helsinki, 1978.

2. BERTHELOT, G., **Transformations and decompositions of nets**, Advances in Petri Nets, Lecture Notes in Computer Science, Vol. 254, pp. 359-376, Springer-Verlag, 1987.

3. ESPARZA, J., SILVA, M., **Circuits, Handles, Bridges and Nets**, Advances in Petri Nets, Lecture Notes in Computer Science , Vol. 483, pp. 210-242, Springer-Verlag, 1990.

4. DESEL, J. and ESPARZA, J., **Free Choice Petri Nets**, Cambridge Tracts in Theoretical Computer Science No. 40, Cambridge University Press, 1995.

5. KONDRATYEV, A., KISHINEVSKY, M., TAUBIN, A., TEN, S., **A Structural Approach for the Analysis of Petri Nets by Reduced Unfoldings**, ICATPN'96, Advances in Petri Nets, Lecture Notes in Computer Science, Vol. 1091, pp. 346-365, Springer-Verlag, 1996.

6. GRAVES, B., **Computing reachability properties hidden in a finite net unfolding**, 17th Foundations of Software Technology and Theoretical Computer Science Conference, Lecture Notes in Computer Science, Vol. 1346, pp. 327-341, Springer-Verlag, 1997.

7. COUVREUR, J.M. and POITRENAUD, D., **Detection of illegal behaviours based on unfoldings**, ICATPN'99, Lecture Notes in Computer Science, Vol. 1630, pp. 364-383, Springer-Verlag, 1999.

8. FALKO, B., **Analysis of Petri Nets with a Dynamic Priority Method**, Application & Theory of Petri Nets 1997, ICATPN'97, Lecture Notes in Computer Science, Vol. 1248, pp. 215-234, Springer-Verlag, 1997.

9. KARP, R.M. and MILLER, R.E., **Parallel Program Schemata**, Journal of Computer and System Sciences 3, pp. 147-195, 1969.

10. FINKEL, A., **The Minimal Coverability Graph for Petri Nets**, Advances in Petri Nets 1993, Lecture Notes in Computer Science, Vol. 674, pp. 210-243, Springer Verlag, 1993.

11. SILVA, M., **Sur le concept de Macro-place et de son utilisation pour l'analyse des réseaux**, RAIRO Automatique, vol. 15, n° 4, pp. 335-345, 1981.

12. SUZUKI, I. and MURATA, T., **A Method for Stepwise Refinement and Abstraction of Petri Nets**, Journal of Computer and System Sciences, Vol. 27, n° 1, pp. 51-76, 1983.

13. ZHOU, M. C. and DI CESARE, F., **Parallel and Sequential Mutual exclusions for Petri Net Modelling of Manufacturing Systems with Shared Resources**, IEEE Transaction on Robotics and Automation, vol. 7, n° 4, pp. 515-527, 1991.

14. VALETTE, R., **Analysis of Petri Nets by Stepwise refinement**, Journal of Computer and System Sciences, Vol. 18, pp. 35-46, 1979.

15. AUSFELDER, C., CASTELIN, E., GENTINA, J. C., **A Method for Hierarchical Modeling of the Command of Flexible Manufacturing Systems**, IEEE Transaction on Systems, Man and Cybernetics, Vol. 24, n° 4, pp. 564-573, 1994.

16. WANG, L., **Gestion Hiérarchisée de Systèmes de Production Discrets : Une Approche basée sur les Réseaux de Petri**, Ph. D. Thesis, Université de Metz, Décembre 1995.

17. JENG, M., **A Petri Net Synthesis Theory for Modeling Flexible Manufacturing Systems**, IEEE Transaction on Systems, Man and Cybernetics, Vol. 27, n° 2, Avril 1997.

18. AGERWALA, T. and CHOED-AMPHAI, Y. C., **A Synthesis rule for concurrent systems**, Proc. Of Design Automation Conference, pp. 305-311, 1978.

19. NARAHARI, Y. and VISWANADHAM, N., **On the Invariants of Coloured Petri Nets**, Advanced in Petri Nets 85, Lecture Notes in Computer Science, Vol. 222, pp. 330-345, Springer-Verlag, 1986.

20. FINOTTO, P., CRESTANI, D., PRUNET, F., **A bottom-up approach for efficient qualitative analysis**, First International Workshop on Manufacturing and Petri Nets, ICATPN'96, pp. 129-142, Osaka, June 1996.

21. LANLIGNEL, J.M., RAYNAUD, O., THIERRY, E., **Pruning Graph with Digital search Trees. Application to Distance Hereditary Graphs**, STACS'00, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, Vol. 1770, pp. 529-541, Lille, 2000.

22. FINKEL, A. and PETRUCCI, L., **Composition / Décomposition de Réseaux de Petri et de leurs Graphes de Couverture**, Informatique Théorique et Applications, Vol. 28, n°2, pp. 73-124, 1994.

Tables and Figures

```
Procedure Tree_F(PN : Petri net ; var FT : tree)
begin
        unprocessednodes := {create_node(r,M0)} ;
        processednodes := ∅ ;
        while unprocessednodes ≠ ∅ do
                Select some node n ∈ unprocessednodes;
                unprocessednodes = unprocessednodes – {n};
                Case n : [1,4] of
                Case 1 : there is a node n1 ∈ processednodes such that m = m1
                        processednodes = processednodes + {n};
                        exit;
                Case 2 : there is a node n1 ∈ processednodes such that m < m1
                        remove_node(n;FT);
                        exit;
                Case 3 : there is a node n1 ∈ processednodes such that m1 < m
                        m2 := m ; ancestor := false;
                        for all ancestors n1 of n such that  m1 < m do
                                for all places p such that m1(p) < m(p) do m2(p) := ω ;
                                end for
                        end for
                        if n1 is the first ancestor such that m1 < m2 then
                        ancestor := true ;
                        m1 := m2 ;
                        remove_tree(n1;FT); *removes the subtree(n1) whose root is n1 in FT*
                        remove in unprocessednodes + unprocessednodes all nodes of subtree(n1);
                        unprocessednodes := unprocessednodes + {n1};
                        endif
                        for every n1 ∈ processednodes such that m1 < m2 do
                        remove in unprocessednodes + unprocessednodes all nodes of subtree(n1);
                        remove_tree(n1,FT) ;
                        end for
                        if ancestor = false Then
                        unprocessednodes := unprocessednodes + {n};
                        endif
                        exit;
```

```
                    Case 4 : otherwise
                        for every transition t such that m(t>m' do
                        create_new_node(n,t,n');
                        unprocessednodes := unprocessednodes + {n'};
                        end for
                        processednodes = processednodes + {n};
                        exit;
            end while
end
```

Figure 1 : The minimal coverability algorithm structure

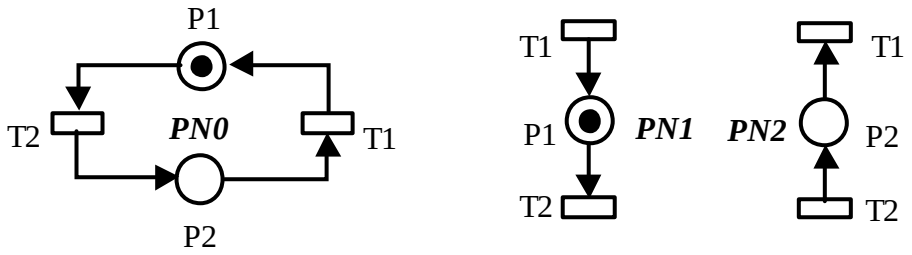

Figure 2 : The global Petri net PN0 and its sub-nets PN1 and PN2

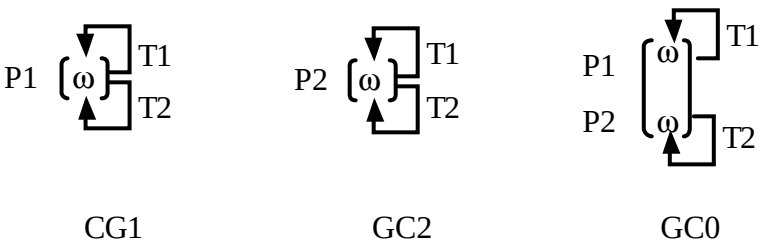

Figure 3 : Finkel Algorithm : coverability graphs CG1, CG2 and CG0

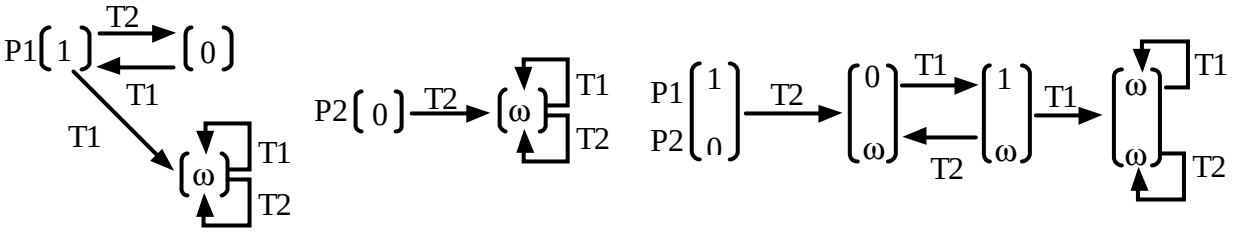

Figure 4 : Karp and Miller Algorithm : coverability graphs CG1, CG2 and CG0

Procedure Merging_FCG *(FCG1, FCG2 : Finite Coverability Graphs ; T1, T2 : set of no shared*

transitions ;Ts : set of shared transitions, var FCG0 : global Finite Coverability Graph)

begin

unprocessednodes := {create_node(r, merge(M01,M02))} ; * *(M0 = M01|M02)* *

processednodes := ∅ ;

While unprocessednodes ≠ ∅ **do**

Select some node n with a marking m = m1|m2 ∈ unprocessednodes;

unprocessednodes = unprocessednodes – {n} ;

Case n : [1..3] of

Case 1 : there is a node n' ∈ processednodes such as m = m'

processednodes = processednodes + {n};

exit;

Case 2 : there is a node n' ∈ processednodes such as m < m'

remove_node(n,FCG0);

exit;

Case 3 : otherwise :

nc1 := nodes of FCG1 having a marking mc1 equivalent to m1 ;

nc2 := nodes of FCG2 having a marking mc2 equivalent to m2 ;

for every transition t ∈ T1 such as there is an arc mc1 |t> mc1' in FCG1 and m1 |t> m1' **do**

create_node(n', merge(m1',m2)) ;

unprocessednodes = unprocessednodes + {n'};

create in FCG0 an arc connecting n and n' with the label t;

and for

for every transition t ∈ T2 such as there is an arc mc2 |t> mc2' in FCG2 and m2 |t> m2' **do**

create_node(n', merge(m1,m2'));

unprocessednodes = unprocessednodes + {n'};

create in FCG0 an arc connecting n and n' with the label t;

end for

for every transition t ∈ Ts such as there is an arc mc1 |t> mc1' in FCG1 and an arc mc2 |t> mc2' in FCG2 and m1 |t> m1' and m2 |t> m2' **do**

create_node(n', merge(m1',m2')) ;

unprocessednodes = unprocessednodes + {n'};

create in FCG0 an arc connecting n and n' with the label t;

end for

processednodes = processednodes + {n};

exit;

end while

Identify the nodes having the same label;

end

Figure 5 : The merging algorithm

	Finite minimal coverability tree temporal complexity
Case 1	$O((N+K)^2)$
Case 2	$O((N+K)^2)$
Case 3	$O((N+K)^2)$
Case 4	O((N+K).T)

Table 1 : Coverability tree cases temporal complexity

	Finite minimal coverability tree merging temporal complexity
Case 1	$O((N+K)^2)$
Case 2	$O((N+K)^2)$
Case 3	O((N+K).T)

Table 2: Merging coverability tree cases temporal complexity

8. Biographical notes

Didier Crestani is Assistant Professor at the university of Montpellier, France. He develops his research activities at the LIRMM, French Laboratory of Informatics, Robotics, and Microelectronics. His areas of interest are enterprise modelling, Discrete Event Systems modelling and analysis using Petri nets.

Alain Jean Marie is Research Director from INRIA (Institut National de Recherche en Informatique et en Automatique). He develops his research activities at the LIRMM. His areas of interest are system performance evaluation using applied probabilities, queuing theory, control theory, games theory.

Christine Covès is Doctor of the university of Montpellier and presently engineer in a software society. She has developed his research activities at the LIRMM. Her areas of interest are enterprise modelling, Discrete Event Systems modelling and analysis using Petri nets.