A Process-Graph Based Formulation of the Syndrome-Decoding Problem

Balázs POLGÁR[1], Szilárd NOVÁKI[2], András PATARICZA[1], Ferenc FRIEDLER[2]

[1] Budapest University of Technology and Economics, Faculty of Electrical Engineering and Informatics, Department of Measurement and Information Systems, Pázmány Péter sétány 1/d. Budapest, Hungary, H-1117, Phone: 36-1-463-3579 Fax: 36-1-463-2667 Email: polgar@mit.bme.hu
[2] University of Veszprem, Department of Computer Science, Egyetem u. 10, Veszprem, Hungary, H-8200

I. Introduction

Diagnostics is one of the core problems in assuring the dependability of complex information technology systems. Nowadays diagnostics does not only a simple activity for identifying faulty hardware resources, but more and more it is a complex activity necessitating to cover both the hardware and software aspects. The increasing frequency of transient errors require more and more sophisticated diagnosis technologies, which also support the subsequent phases of fault handling as well, like damage confinement and reconfiguration. In practice, however one of the basic problems is the proper handling of the high complexity of the diagnostic process.

One of the intrinsic assumptions behind all diagnostic algorithms is the notion of a maximum likelihood diagnosis. Frequently this assumption appears simply as “more faults occur with lower frequency than a few ones”. Thus, the diagnostics aims at a minimal set of faulty elements compatible with the syndrome. Other approaches use an explicit notion of fault probabilities and assign different probabilities to different elements.

In the present work, the diagnostics has been formulated as a maximization problem, moreover, it has been solved by state of the art technologies from the field of operations research [5].

The well-known multiprocessor-testing problem has been considered as test bed, which is the simplest one in its structure. However, it has to be pointed out that a similar approach can be used even to a wider class of systems. A universal framework from this study will be provided by the ongoing research on application of operational research methods for integrated diagnosis [4].

System-level self-checking is one of the important methods to guarantee reliability in a multiprocessor systems [1]. Self-checking has two basic steps: in the first one processors test each other, and testers recognize the state of the tested units to be good or faulty. Then in the second step the set of test results – the syndrome – is processed by an algorithm, which marks each unit as good or faulty (or unidentified). The difficulty in the second step is that any tester can be faulty invalidating the test result. Thus, the faulty processors have to be identified rejecting test results of these processors. This is known as the syndrome-decoding problem.

There exist many traditional syndrome-decoding algorithms, and almost all of them originate in the model introduced by Preparata, Metze and Chien in 1967. This model has the nonrealistic assumption that the test results of a good processor are always valid, i.e. it cannot do anything with tests having fault coverage less then 100 percent. Furthermore it cannot handle the ‘probability of the uncertainty’, i.e. the probability of the *good* or *faulty* test result in the case of a faulty tester and either good or faulty tested unit. More accurately it handles the probability, but considers it as fifty percent. If more than one fault distribution can ‘generate’ the same syndrome, i. e. there exist multiple correct diagnoses to a given syndrome, these traditional algorithms need further restrictions or the diagnosis will be incomplete (set of units remain in an unidentified state).

If the syndrome-decoding problem is regarded as a process network synthesis problem, a much more general model can be constructed without the assumptions mentioned above. An advantage of it

is that synthesis algorithms can choose from the possible diagnosis hypotheses the one of the maximum likelihood without any further restriction. Another advantage is that the reliability of each processor can be considered, i.e. it is possible to have processors in the system with different reliability measures.

II. System-level Diagnosis

A system with system-level self-check consists a set of *intelligent units*, e.g. processors, and some of them are interconnected by *links*. The state of units can be *good* or *faulty* and the collection of the states of all of the units in the system is called *fault-pattern* or *fault distribution*. Via the links units test the state of each other, this determines the *test topology* (e.g. ring, square lattice). Accordingly, the *tester* and the *tested unit* (UUT – unit under test) can be distinguished. Of course, the same unit can have different roles in different tests. The *test result* can be either *'good'* or *'faulty'* and it is correct or *valid* if it is the same as the real state of the tested unit. The collection of all test results is the *syndrome*. *Syndrome-decoding algorithms* try to find the states of units upon a syndrome, the result of the algorithm is called the *diagnosis*. A diagnosis is *complete* if all the units are labeled as 'good' or 'faulty' and a diagnosis is *correct* if all labels correspond to the reality.

State of tester	State of UUT	Test result
good	*good*	*good*
good	*faulty*	*faulty*
faulty	*good*	$c \in \{$ *good, faulty, any* $\}$
faulty	*faulty*	$d \in \{$ *good, faulty, any* $\}$

Table 1 Possible test invalidation models

The traditional approach supposes that the test result of a good unit is always correct, while faulty testers can produce incorrect results. During syndrome decoding the faulty state of a tester invalidates its test results as defined by *test invalidation* relation (***Error! Reference source not found.***). The most general model – called PMC – says that the test result of a faulty tester can be anything independently from the state of the tested unit, i.e. both *c* and *d* is equal to *any*. Another widely used model is the BGM, which suppose that – in the case of complex units – the probability that two units fail in the same way is very small, hence a faulty tester will also detect the failure of the tested unit, i.e. *d=faulty* while *c* remains *any*.

If multiple fault-patterns can generate identical syndromes then the diagnostic algorithm needs more information in order to give a complete diagnosis or there will remain a set of units with label *unidentified*. *Deterministic algorithms* give complete diagnosis with the extra information of the so called *t-bound*, which suppose that at most *t* units can be faulty. *Probabilistic algorithms* try to find the most probable fault-pattern supposing that the probability of the faulty state of a unit is smaller than that of the good one [2].

The test topology and the syndrome can be represented graphically by the *testing graph*. Vertices of testing graph are the units of the system and arcs are representing tests directing from the tester to the tested unit. Test results are labels on arcs. Label 0 represents the *good*, and label 1 represents the *faulty* test result. A five-unit example can be seen on *Figure 1*.

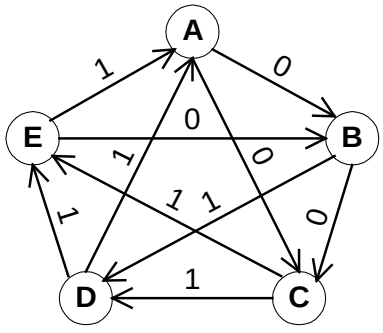

Figure 1. A testing graph with five unit

III. Process Network Synthesis (PNS) problems

PNS problems are given by the set of desired products, available raw materials, and plausible operating units, which units consume and produce their input and output materials, respectively. Moreover, constraints and cost function can be defined for the process network. Our aim is to synthesize the optimal network of operating units producing the desired products while consuming raw materials [6].

PNS problems can be solved by mathematical programming methods, e.g. mixed integer non-linear programming (MINLP). Unfortunately these methods does not exploit the peculiar characteristics of the MINLP model of the synthesis problem, thus they are unnecessary complex, while the size of solvable problems is rather small. Friedler *et al.* [8] developed algorithms for enhancing the effectiveness of mathematical programming methods by exploiting these unique features of process networks.

Conventional graphs are incapable of uniquely representing process structures in synthesis, thus an innovative graphical representation, process-graph or P-graph has been introduced [7]. A P-graph is a directed bipartite graph, i.e. the set of vertexes is the union of two disjoint sets, the set of nodes representing materials and those representing the operating units. The direction of arcs between these two sets determines the direction of the flows in the network. For example, the P-graph representation of a system producing pure components A, B and C from the mixture of A, B and C, i.e. ABC feed stream is in *Figure 2*. Operating unit 1 produces pure component A and a mixture of B and C components, while operating unit 2 produces pure component C and a mixture of A and B components. Operating unit 3 separates the mixture of A and B and operating unit 4 separates the mixture of B and C.[1]

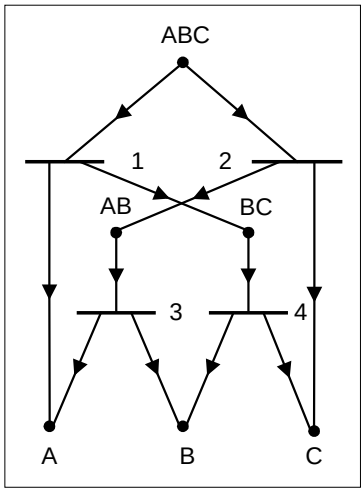

Figure 2 P-graph for separating components from mixture

A P-graph is said to be feasible process structure if it satisfies the five axioms (S1) through (S5) of the combinatorially feasible process structures (see e.g. [7], [8]). Axiom (S1) implies that each product is produced by at least one of the operating units of the system; axiom (S2), a material is not produced by any of operating unit of the system if and only if this material is a raw material; axiom (S3), only the plausible operating units of the problem are taken into account in the synthesis; axiom (S4), any operating unit of the system has a series of connections eventually leading to the operating unit generating at least one of the products; and axiom (S5), each material appearing in the system is an input to or an output from at least one operating unit of the system.

The complete set of combinatorially feasible process structures or solution structures can be generated via algorithm SSG, i.e. solution structure generator. Algorithm SSG maintains a set called ‘materials to produce’ which is initially the set of desired products. In the first step the algorithm selects an arbitrary material from this set, and produces it every feasible way. For example if both operating units 1 and 3 can produce material A, then the structure generation can continue on three feasible branches: A is produced by operating unit 1; A is produced by operating unit 2; or A is produced by both operating unit 1 and 2. In the second step the previously selected – and already produced – material is extracted from ‘materials to produce’ set, and the input materials of operating units which was chosen to generate the selected material are included into set ‘materials to produce’. After second step algorithm SSG recursively calls itself. The algorithm halts on a branch if the set of

[1] The structure and logic properties of P-graphs strongly resemble to Petri-nets. The merging of the theoretical foundations of this two mathematical fields is subject of an ongoing research.

'materials to produce' is empty – which means the structure generated is a combinatorially feasible structure – or the selected material cannot be produced – in this case the branch does not lead to solution structure.

IV. Syndrome decoding based on P-graphs

The idea to use the process network synthesis algorithms for syndrome decoding is that *information can be regarded as material and the transformation of information as the function of an operating unit*.

Information in a syndrome-decoding system is associated with the states of the units ('*unit A is good*', denoted by A_G, or '*unit A is faulty*', denoted by A_F) and the test results, (e.g. '*the tester unit A reports that the tested unit B is faulty*', denoted by AB_F), while the transformations are implication rules, which determine the possible test results depending on the states of the tester and tested unit (e.g. 'IF *unit A is good* AND *unit B is faulty* THEN *A says that B is faulty*') depending on the actual test invalidation model. In the P-graph representation each row in ***Error! Reference source not found.*** determines one or two 'operating units', i.e. one or two transformations. The input of the transformation is the states of the tester and the tested unit, while the consequence of it, the test result is its output. If the test result is *any*, i.e. it can be either good or faulty, then two operating unit will be generated for a single row.

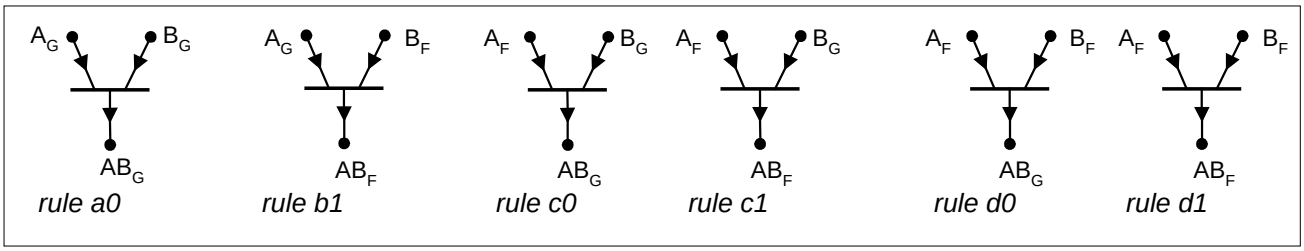

Figure 3 P-graph representation of the consequence rules of PMC model

The P-graph representation of the PMC model can be seen on *Figure 3*. If BGM model is considered (i.e. *d=faulty*), then just *rule d0* should be discarded. Similarly if *d=good*, then *rule d1* is superfluous, and so on. *Figure 4* shows the simplified – but equivalent – P-graph representation of the transformations of the BGM model.

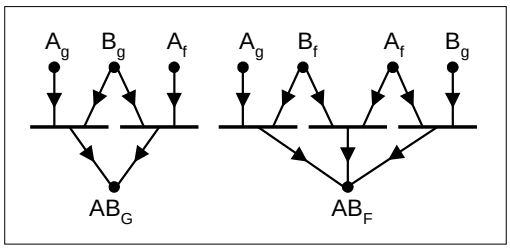

Figure 4 P-graph representation of the BGM test invalidation

If no a priori information is available on the probability of failing, the diagnostic object is the minimum number of faulty units.

Each operating unit has cost, the probability of generating the given test result. The cost of operating units belonging to rows in ***Error! Reference source not found.*** containing a single test result (i.e. not *any*) is 1, while the cost of operating units representing non-deterministic test results (like in case of BGM model *rules c0* and *c1*) is 0.5, because it can be either good or faulty with equal probability.

It seems to be redundant to have for example two operating unit for generating AB_G on *Figure 4*, because it is independent from the state of *A*, but these cannot be merged, because they have different costs.

So thus, the elements of the process network are the following ones:

- *raw materials*: information of the state of each processor (A_G, A_F, B_G, B_F, etc.),
- *products*: information of each test result (AB_G, AC_F, BC_F, BD_F, etc.),
- *operating units*: all transformations representing consequence rules that can 'produce' one of the test results (*rule a0, c0, d0* for AB_G, the adequate *rule b1, c1, d1* for AC_F, etc.).

Constraints should also be formulated in order to guarantee the consistency of the states of units, i.e. not to allow both *good* and *faulty* states of a unit in the result. Mathematically:

$$A_G \in \{0,1\},\ A_F \in \{0,1\} \text{ and } A_G + A_F = 1$$

The aim of the synthesis algorithm is to determine the states of the processors, which means the decoding of the syndrome, that is to find the set of raw materials that can produce the given products. The algorithm first executes SSG. It generates all of the combinatorially feasible process structures. Subsequently, the inconsistent ones – which contains two different states of a single unit – are filtered out from the feasible structures, remaining the set of solution structures. If this set consists multiple elements then the most probable one will be selected, the one with maximum cost. This means that the fault pattern will be chosen, which generates the syndrome with maximum probability.

V. Generalisation of the Test Model

The traditional test invalidation models (see ***Error! Reference source not found.***) can be generalised in order to relax their intrinsic restrictions. In this section we show that the approach sketched above can be used to relax the assumption of 100 percent fault coverage thus to get closer to the real system behaviour.

In the generalised model probabilities are associated to the possible test results. For instance a good tester will detect a fault in the testing unit with a probability of p_{b0}, and a fault will escape with a probability p_{b1} ($p_{b0} + p_{b1} = 1$) (see *Table 2*). Similarly a faulty tester will report a good unit as good or faulty with probabilities p_{c0} and p_{c1}, where $p_{c0} + p_{c1} = 1$, and so on.

State of tester	State of UUT	Test result	
		0	1
good	*good*	p_{a0}	p_{a1}
good	*faulty*	p_{b0}	p_{b1}
faulty	*good*	p_{c0}	p_{c1}
faulty	*faulty*	p_{d0}	p_{d1}

Table 2 Generalised test model

Assumed that both $p_{a1} = p_{b0} = 1$ and $p_{c0} = p_{c1} = p_{d0} = p_{d1} = 0.5$ then this is just the case of the PMC model. With the same probabilities by changing p_{d1} to 1 (and accordingly $p_{d0} = 0$) we get the BGM model. By assigning these probabilities to 0 and 1 all of the meaningful test invalidation models can be reconstructed. In this sense the generalised test invalidation model covers the traditional one.

However, the use of explicit maximisation algorithms allows using probabilities in between 0 and 1. In this case the probability p_{b1} can represent fault coverage, while p_{c0}, p_{c1}, p_{d0} and p_{d1} the distortion of the test results by a faulty tester.

The mathematical model allows to set p_{a1} to nonzero thus covering the case of fault alarms (a good tester finds a good unit to be faulty). This case is not very realistic, but in some test models it can be used to model a non-deterministic test process with a potential fault alarm as outcome.

In case of the generalised model the P-graph described above is extended by two more implication rules in order to cover all the cases (*Figure 5*). *Rules a1* and *b0* indicate imperfect but good testers. As previously, probabilities p_{xy} ($x \in \{a,b,c,d\}$, $y \in \{0,1\}$) are assigned as cost to corresponding operating unit, which generates the given test result.

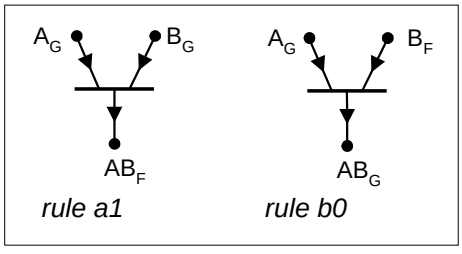

Figure 5 Extra rules for the P-graph of the extended model

The failing probability p_f of processors can also be taken into account. It will modify the cost of operating units with the

probabilities of the states of the tester and UUT (*Table 3*). This way the differences between more and less probable states of units will be much more emphasized (if $p_f = 0.1$ then $(1- p_f)^2 = 0.81$, $(1- p_f) \cdot p_f = 0.09$ and $p_f^2 = 0.01$).

Consider now the situation when we have processors with failing probability 0.1, operating according to the PMC model and have the test result that unit A finds unit B to be good. The possible states of A-B can be 0-0, 1-0 and 1-1. The differences between costs of operating units – generating the given result from possible states of A-B – depending on the consideration of p_f can be seen on *Table 4*.

Indices of op. Units	**0**	**1**
A	$p_{a0} \cdot (1- p_f)^2$	$p_{a1} \cdot (1- p_f)^2$
B	$p_{b0} \cdot (1- p_f) \cdot p_f$	$p_{b1} \cdot (1- p_f) \cdot p_f$
c	$p_{c0} \cdot p_f \cdot (1- p_f)$	$p_{c1} \cdot p_f \cdot (1- p_f)$
d	$p_{d0} \cdot p_f^2$	$p_{d1} \cdot p_f^2$

Table 3 Costs of operating units including failing probability p_f

op. unit	**without p_f**	**with p_f**
a0	*1*	*0.81*
c0	*0.5*	*0.045*
d0	*0.5*	*0.005*

Table 4 Costs of operating units generating 0 test result in PMC model ($p_f = 0.1$)

VI. Example

One advantage of the adaptation of algorithm SSG is that more generalised syndrome-decoding problems can be solved, i.e. problems having more than one correct diagnosis upon a given syndrome. For example, the testing graph in *Figure 1* yields two correct diagnoses in the BGM test invalidation model. One diagnosis can be A, B and C units are good units; D and E units are faulty (see *Figure 6*.a); other correct diagnosis can be that only B and C units are good and units A, D and E are faulty (see *Figure 6.b*).

In order to solve syndrome-decoding problem via the adapted SSG algorithm we define the set of products, which is the set of test result, i.e. P={AB_G, AC_G, BC_G, BD_F, CD_F, CE_F, DA_F, DE_F, EA_F, EB_G}. Moreover, the set of raw materials is defined to be the set of feasible unit states R={A_G, A_F, B_G, B_F, C_G, C_F, D_G, D_F, E_G, E_F}. The set of operating units contains all feasible transformation between the unit states and the test results (see *Figure 7*). For example, the test result AB_G can be generated in two feasible ways, i.e. from unit states A_G and B_G, and from A_F and B_G in BGM test invalidation model. Consequently, test result CE_F can be generated in three feasible ways, i.e. from unit states C_G and E_F; from unit states C_F and E_F ; and from unit states C_F and E_G.

We define constraints and cost as described above; constraints like A_G, $A_F \in \{0, 1\}$, $A_G+A_F=1$ for each unit to establish a consequent structure, i.e. no unit has state good and faulty at the same time. Costs of operating units in BGM test invalidation model are described in previous chapters. The cost of network is the product of the cost of operating units.

Adapted algorithm SSG then generates two consequent networks of good or faulty processors. Each network produces desired products, i.e. the

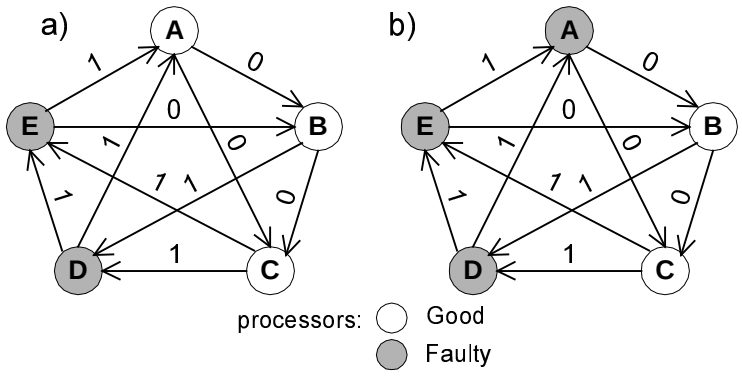

Figure 6 Correct diagnoses for system represented by testing graph in *Figure 1*

syndrome. The cost of network in *Figure 6.a* is 0.25, while the cost of network in *Figure 6.b* is 0.125. The adapted SSG algorithm chooses the structure with the highest cost, i.e. structure with the most probable fault pattern, which is represented by *Figure 6.a*.

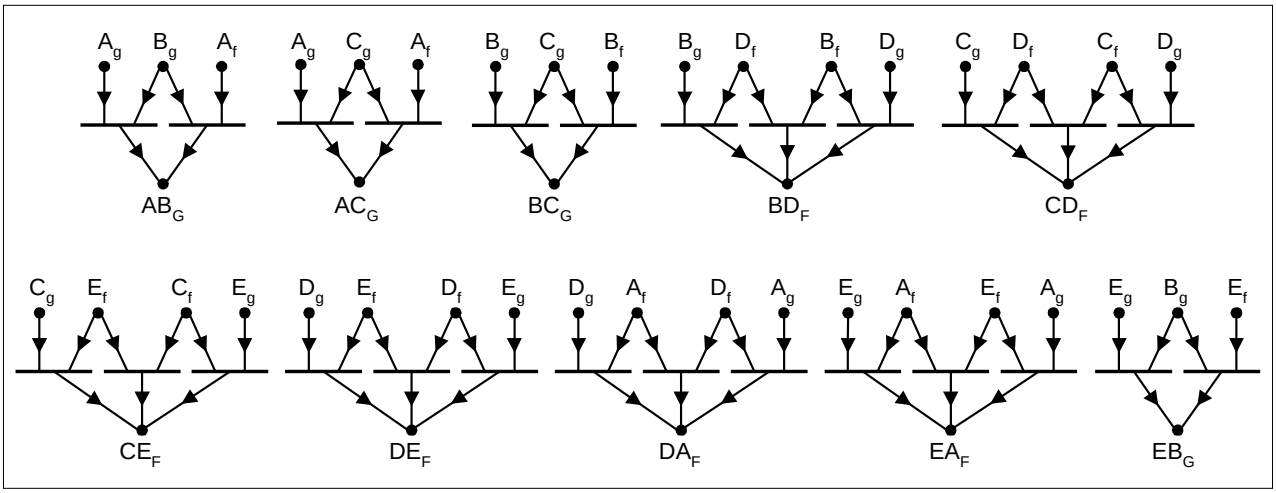

Figure 7 Operating units in adopted SSG algorithm for testing graph in Figure 1

VII. Conclusion

With process-graph based representation of syndrome decoding problem, it is possible to solve much more generalised problems that can be solved with previous algorithms. The assumption of one hundred percent fault coverage can be eliminated; moreover, the solution can be generated in any test invalidation model. The proposed algorithm does not need additional information to determine between fault patterns when more than one correct diagnosis can be established.

These highly desired properties of the algorithm achieved by the recognition, that any information in a system can be regarded as a material; and any transformation of the information can be represented as an operating unit in a process network. The combinatorial algorithms for solving PNS problems are based on rigorous mathematical foundation; therefore, the adapted SSG algorithm provides an effective solution of the generalised syndrome-decoding problem. The preceding recognition may also be established in the modelling of integrated diagnostics, which is the subject of our future work.

As mentioned above the syndrome-decoding problem in multiprocessor systems has a special structure, namely the direct manifestation of internal fault states in the syndromes. In more complex problems obviously the states of the system control logic have to be taken into account in the model to be analysed [3]. These straightforward extensions can be well incorporated into the process-graph-based models due to the strong analogy to Petri-nets. Current work aims at generalisation of the results into this direction.

Acknowledgement

The research has been supported in part by the Hungarian National Research Foundation Grants OTKA T030804 and T029309.

References

[1] Selényi Endre, *Generalisation of system level self-checking*, PhD Thesis *MTA,* 1985.

[2] S. N. Maheshwari, S. L. Hakimi, *On Models for Diagnosable Systems and Probabilistic Fault diagnosis*. IEEE Trans. on Comput. Vol. C-25, pp 228-236, 1976.

[3] A. Pataricza, P. Urbán, *A combination of Petri-nets and linear programming in design for dependability*. Technical report, TUB, 1998.

[4] G. Csertán, A. Pataricza, and E. Selényi, *Dependability analysis in HW-SW co-design*. In IEEE Computer Performance and Dependability Symposium, IPDS'95, pages 306-315, 1995.

[5] A. Pataricza, *Algebraic modelling of diagnostic problems in hw-sw codesign*. In D. Avresky and B. Horst, editors, Digest of Abstracts of the IEEE International Workshop on Embedded Fault-Tolerant Systems, Dallas, Texas, Sept. 1996.

[6] Friedler, F., L. T. Fan, and B. Imreh, *Process Network Synthesis: Problem Definition*, Networks, 28(2), 119-124 (1998).

[7] Friedler, F., K. Tarjan, Y. W. Huang, and L. T. Fan, *Graph-Theoretic Approach to Process Synthesis: Axioms and Theorems*, Chem. Engng Sci., 47(8), 1973-1988 (1992).

[8] Friedler, F., K. Tarjan, Y. W. Huang, and L. T. Fan, *Combinatorial Algorithms for Process Synthesis*, Computers Chem. Engng, 16, S313-320 (1992).