Available online at www.sciencedirect.com

ScienceDirect

Progress in Natural Science 19 (2009) 489–494

Progress in Natural Science

www.elsevier.com/locate/pnsc

Deriving all minimal consistency-based diagnosis sets using SAT solvers

Xiangfu Zhao [a,b], Liming Zhang [a,b], Dantong Ouyang [a,b,*], Yu Jiao [a,b]

[a] Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China
[b] School of Computer Science and Technology, Jilin University, Changchun 130012, China

Received 27 April 2008; received in revised form 7 July 2008; accepted 7 July 2008

Abstract

In this paper, a novel method is proposed for judging whether a component set is a consistency-based diagnostic set, using SAT solvers. Firstly, the model of the system to be diagnosed and all the observations are described with conjunctive normal forms (CNF). Then, all the related clauses in the CNF files to the components other than the considered ones are extracted, to be used for satisfiability checking by SAT solvers. Next, all the minimal consistency-based diagnostic sets are derived by the CSSE-tree or by other similar algorithms. We have implemented four related algorithms, by calling the gold medal SAT solver in SAT07 competition – RSAT. Experimental results show that all the minimal consistency-based diagnostic sets can be quickly computed. Especially our CSSE-tree has the best efficiency for the single- or double-fault diagnosis.

Keywords: Conjunctive normal form; Consistency-based diagnosis; Model-based diagnosis; SAT solver

1. Introduction

Model-based diagnosis (MBD) is one of the active branches in artificial intelligence (AI). It plays an important role of test-bed of some approaches to knowledge representation and reasoning [1]. Its basic principle is to employ the model of a device to judge faults logically, according to the difference between the model's prediction and the actual observation.

Traditionally, conflict recognition, aiming at generating all minimal conflict sets, and candidate generation, aiming at generating all minimal hitting-sets, are of the two important steps towards the final diagnostic results. However, both of them are NP-complete problems. Therefore, the efficiency of each step greatly affects the final diagnosis.

In AI, most path-finding problems, notably AI planning [2] and model-checking [3], have been very successfully reduced to the propositional satisfiability problems (SAT). Recently, Grastien et al. [4,5] have also reduced diagnosis of discrete-event systems to SAT problems. Instead, in this paper, we consider determining a consistency-based diagnostic set and show how it can be reduced to SAT and efficiently solved by SAT solvers.

The structure of the paper is as follows: Some preliminaries of MBD and SAT are given in Section 2. Related algorithms are presented in Section 3. Some experimental results are given in Section 4. Some related work and comparisons are discussed in Section 5. Finally, conclusions and some future directions are put forward in Section 6.

2. Preliminaries

Firstly, let us introduce some definitions involved in model-based diagnosis.

Definition 1 [6]. A system is a triple $(SD, COMPS, OBS)$, where SD (system description) is a set of first order sentences; $COMPS$ (system components) is a finite set of constants; OBS (system observation) is a finite set of first order sentences.

[*] Corresponding author. Tel.: +86 138 4300 6163.
E-mail address: ouyangdantong@163.com (D. Ouyang).

In the following, a unary predicate AB(·) is interpreted to mean "abnormal". AB(c) is true iff c is abnormal, where $c \in COMPS$.

Definition 2 ([6,7]). Given $\Delta \subseteq COMPS$, Δ is called a consistency-based diagnosis for ($SD, COMPS, OBS$) if $SD \cup OBS \cup \{\neg AB(c) | c \in COMPS\text{-}\Delta\}$ is satisfiable.

A consistency-based diagnosis for ($SD, COMPS, OBS$) Δ is a minimal one (MCBD), iff for no proper subset of Δ is a diagnosis for ($SD, COMPS, OBS$).

In the following, let us have a look at SAT solvers. The purpose of a SAT solver is to accept a conjunctive normal formula (CNF) P, in clauses normal form, and then to evaluate whether P is true (satisfiable).

Recently, the field of SAT solving has advanced dramatically: a CNF containing hundreds of thousands or even millions of literals can now be handled by the state-of-the-art solvers, such as SATO [8] or zChaff [9] to name just two of the most popular solvers.

Any set of propositional forms can be transformed into a CNF. For instance, a set of propositional forms $\{A \rightarrow B, B \rightarrow C, \neg C, A\}$, can be described with a CNF file as follows (variables A, B, and C are denoted by 1, 2, and 3, respectively):

```
p cnf 3 4
−1 2 0
−2 3 0
−3 0
1 0
```

where in the first line, "p cnf" are keywords, "3" is the number of total variables, and "4" is the number of total clauses in this CNF file. Every negative number denotes that the corresponding variable is negative, for example, "−1" denotes "$\neg A$". "0" marks the end of a clause.

According to Definition 2 and SAT, now we can give the basic idea of judging whether a component set *SubCOMP* is a consistency-based diagnosis. Firstly, create a CNF file with all the component description clauses extracted from *SD* (in a CNF file) related to the component set (*COMPS* – *SubCOMP*), and all the corresponding clauses specifying the OK (i.e. ¬AB) mode for every related component. Then, call a SAT solver with this CNF. If "true" is returned, *SubCOMP* is a diagnosis.

3. Description of algorithms

In this section, we first show how a system model and observations can be described with CNF files. Next, an algorithm IsDiag is given to determine whether a subset of COMPS is a diagnosis. Then four variant CS-tree algorithms can derive all minimal diagnoses by calling IsDiag.

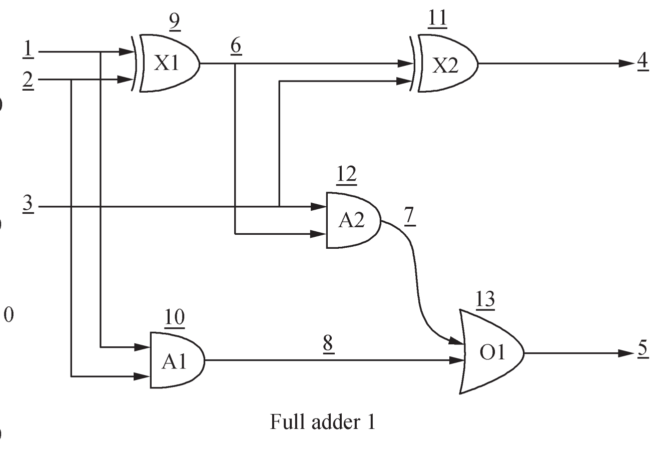
3.1. Modeling a system and observations with CNF

Given a system to be diagnosed, in contrast to the traditional approach [6], we use propositional logic for modeling. Not only the system components but also all the links between components are denoted by variables. For each component c, we use c and $\neg c$ to show that this component is in OK mode or in AB mode, respectively, for simplicity. Every component behavior is described with a propositional statement. All the component behavioral descriptions make up the system description: SD.cnf, a CNF file.

Let us take the fulladder shown in Fig. 1, for example, in which "1", "2" and "3" denote input variables; "4" and "5" denote output variables; "6", "7", and "8" denote all the internal link variables. All the gates, including XOR

SD.cnf

```
p cnf 13 17
−9 −1 −2 −6 0
−9 1 2 −6 0
−9 −1 2 6 0
−9 1 −2 6 0
−10 −1 −2 8 0
−10 1 −8 0
−10 2 −8 0
−11 −3 −6 −4 0
−11 3 6 −4 0
−11 −3 6 4 0
−11 3 −6 4 0
−12 −3 −6 7 0
−12 3 −7 0
−12 6 −7 0
−13 7 8 −5 0
−13 −7 5 0
−13 −8 5 0
```



Description of X1:	Description of A1:	Description of O1:
$\neg OK(X1) \vee \neg 1 \vee \neg 2 \vee \neg 6$	$\neg OK(A1) \vee \neg 1 \vee \neg 2 \vee 8$	$\neg OK(O1) \vee 7 \vee 8 \vee \neg 5$
$\neg OK(X1) \vee 1 \vee 2 \vee \neg 6$	$\neg OK(A1) \vee 1 \vee \neg 8$	$\neg OK(O1) \vee \neg 7 \vee 5$
$\neg OK(X1) \vee \neg 1 \vee 2 \vee 6$	$\neg OK(A1) \vee 2 \vee \neg 8$	$\neg OK(O1) \vee \neg 8 \vee 5$
$\neg OK(X1) \vee 1 \vee \neg 2 \vee 6$		

Fig. 1. Fulladder and its SD.cnf.

gates X1 and X2, AND gates A1, A2, and OR gate O1 are denoted by "9", "11", "10", "12", and "13", respectively. Therefore, we can get a CNF file of the system description SD.cnf shown in Fig. 1.

Similarly, we can give an observation file in CNF. For instance, suppose that we have the observation on input and output variables, "1", "2", "3", "4", and "5", then the observation on the CNF file includes the following clauses:

−1 0
2 0
−3 0
4 0
−5 0

3.2. Algorithm IsDiag

Given a system to be diagnosed, we suppose that the system description SD.cnf and observations OBS.cnf have been created off-line. Then in this section we will give an algorithm IsDiag to judge whether a component set is a diagnosis, by calling SAT solvers as follows:

Function IsDiag(*SubCOMP*[ ])

Input: *SubCOMP*, a subset of *COMPS* to be judged currently

Output: It returns "**true**" if *SubCOMP* is a diagnosis, else it returns "**false**".

Step 1: Extract all the related component descriptions in SD.cnf to the complementary set (*COMPS* – *SubCOMP*) of *SubCOMP*, and put them in a new CNF file SD_C_OBS.cnf.

Step 2: Add the OK mode clause of each component in (*COMPS* - *SubCOMP*) to SD_C_OBS.cnf.

Step 3: Add the content of OBS.cnf to SD_C_OBS.cnf.

Step 4: Call a SAT solver with SD_C_OBS.cnf, **if** it is satisfiable, return "**true**" (i.e. *SubCOMP* is a diagnosis), **else** return "**false**" (i.e. *SubCOMP* is not a diagnosis).

According to the definition of consistency-based diagnosis, in order to determine whether *SubCOMP* is a diagnosis, we need to check whether all the normal behaviors related to the complementary set (*COMPS* – *SubCOMP*) are consistent with all observations. Therefore, only all the related normal behavioral descriptions in SD.cnf are considered (Step 1). In addition, all the OK modes of these are collected (Step 2). Then, all the obtained observations are also constrained (Step 3). Afterwards, a new CNF file related to (*COMPS* – *SubCOMP*) is created, being called by SAT solvers (Step 4).

It should be noted that given the system description SD.cnf, we can build a map from each component to all the related clauses off-line. Therefore, from this map, we can find related clauses to (*COMPS* – *SubCOMP*) directly in Step 1, and the time complexity is linear to the length of (*COMPS* – *SubCOMP*).

3.3. Deriving all MCBDs with SAT solvers

Next, we can derive all MCBDs with the CSSE-tree that was proposed by us in Ref. [10] or with the inverse CS-tree with Mark Set in Ref. [11], using the function IsDiag for checking a diagnosis. The basic idea is to enumerate all necessary subsets of *COMPS* according to some order (such as width-first or depth-first, from shorter subsets to longer subsets or from longer subsets to shorter subsets), and check each of them with IsDiag. As some pruning rules or marking rules are introduced, the efficiency is improved. More details of the four algorithms can be found in related papers, omitted here for simplicity.

The complexity of each algorithm mentioned above is $O(2^k)$ in the worst cases, where k is the length of the system component set *COMPS*. However, many nodes have been pruned by pruning rules, and hence the complexity is much less than $O(2^k)$ in general. Especially, when we consider single- or double-fault diagnosis, the complexity of CSSE-tree will be reduced to $O(k^2)$ in the worst situation (as the CSSE-tree is extended by width-first and from shorter subsets to longer subsets, the generated nodes are all the single components and the combination of all the double components, i.e. the maximum number of nodes generated is $C(k,1)+C(k,2)=1/2(k^2+k)$, where function $C(\cdot)$ denotes the combination), therefore the efficiency is improved greatly. We will also compare them in detail by the experiments in the following section:

4. Experimental results and analysis

We have implemented the four algorithms and tested a lot of examples to evaluate their performance. The environment for implementation and tests was Dell Dimension C521, MD Athlon(tm) 64×2 Dual Core Processor

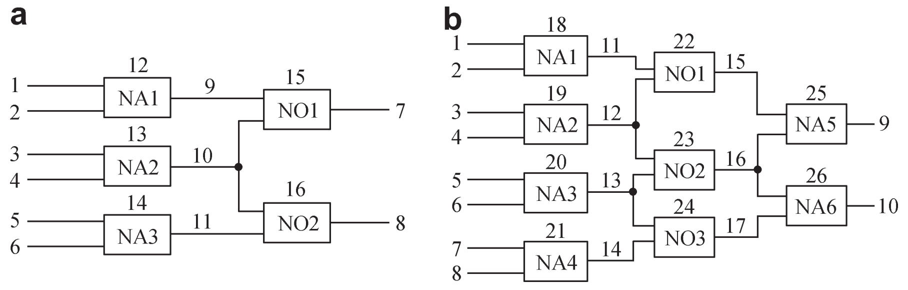


Fig. 2. Polybox_5 (a) and Polybox_9 (b).

3600+, 1.9 GHz, 1022 MB RAM, Windows XP, and VC++ 6.0. The SAT solver we used in IsDiag is RSAT—the gold medal SAT solver in SAT07 competition [12].

We ran all the four algorithms on the circuit c17 of ISCAS-85, two full-adders (one-bit fulladder and two-bit fulladder), and two Polybox circuits [6] (with NAND gates and NOR gates instead of multipliers and adders, respectively). Two Polybox circuits are Polybox_5 (three NAND gates and two NOR gates) and Polybox_9 (six NAND gates and three NOR gates), shown in Fig. 2, where NAi ($1 \leqslant i \leqslant 6$) denotes a NAND gate and NOj ($1 \leqslant j \leqslant 3$) denotes a NOR gate. For each circuit, firstly we give the system description SD.cnf. Then all possible observations can be produced by combining all various values of all input and output variables, and every OBS.cnf is tested using the four algorithms. At the same time, we record the time cost and nodes generated in all for processing each OBS.cnf. Finally, the average time and nodes are totaled in Table 1. (Note in this table, A, B, C and D denote the CSSE-tree, inverse CS-tree, CS-tree with Mark Set, and CSISE-tree, respectively.)

From Table 1, we can see clearly that the CSSE-tree proposed by us generates the fewest nodes and has the best efficiency in general (only for Fulladder_2, more or less the same as the CS-tree with Mark Set). The other three methods sorted by the number of nodes generated are the inverse CS-tree, CS-tree with Mark Set, and CSISE-tree. This is because the shorter component sets can be checked first in the CSSE-tree (width-first search) and inverse CS-tree (depth-first search), and single or double faults are more popular in average. Additionally, some pruning rules have been introduced by the CSSE-tree. In contrast, the longer component sets are first checked by the CS-tree with Mark Set, and CSISE-tree.

However, we can also see from Table 1 that though more nodes have been generated by the CS-tree with Mark Set than the inverse CS-tree, the corresponding efficiency of the former is better than that of the latter in some examples, such as Polybox_9 and Fulladder_2. That is mainly because the Mark Set is introduced in the former, which can be seen as a heuristic strategy.

Generally, for each algorithm, the time cost is increasing with the increment of generated nodes in total. Here, a Polybox_9 example is depicted in Fig. 3.

Next, let us have a look at the relationship between the efficiency and the number of faults. Two better algorithms, CSSE-tree and CS-tree with Mark Set, are depicted in Table 2 and Table 3, where average time costs (seconds) are listed.

We can further see clearly that the CSSE-tree is generally better than the CS-tree with Mark Set when the maximum number of fault components is smaller; especially when the number is 1 or 2, i.e. in single- or double-fault cases. This also mainly results from the different orders of extension of the corresponding trees.

Table 1
Average time cost and nodes generated.

	ISCAS-85_c17		Polybox_5		Polybox_9		Fulladder_1		Fulladder_2	
	Average time (s)	Average nodes	Average time (s)	Average nodes	Average time (s)	Average nodes	Average time (s)	Average nodes	Average time (s)	Average nodes
A	1.8870	9.6797	1.6194	7.9297	17.1777	70.9902	1.2890	6.6563	29.3283	112.4883
B	3.0988	21.7031	2.3188	16.6094	29.6580	207.9443	1.7228	12.1250	46.1013	323.7031
C	3.6797	41.4766	2.5005	24.1602	27.3227	273.9072	2.0249	22.1563	29.2773	379.1992
D	7.9090	55.5781	3.3839	24.4844	57.9165	399.3506	3.7394	26.9063	119.9656	826.6953

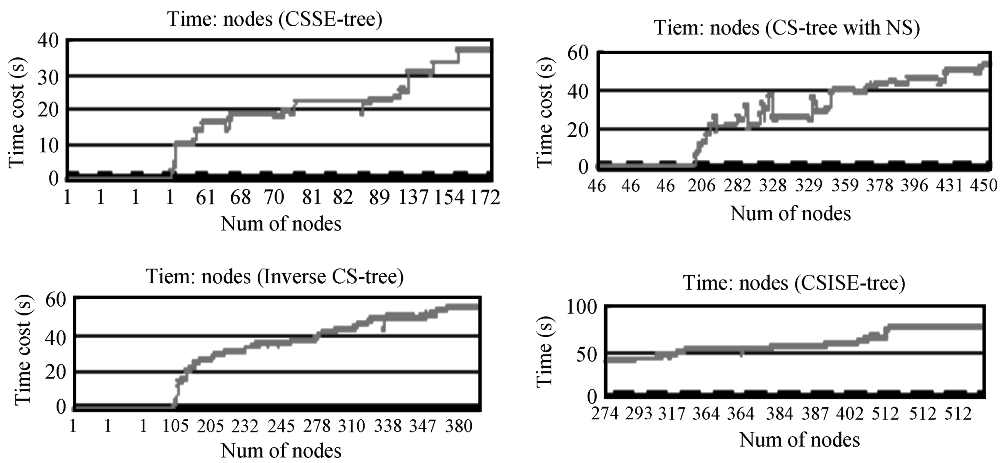


Fig. 3. Time (s): nodes in Polybox_9.

Table 2
Time (s): maximum number of fault components in c17, Polybox_5, Fulladder_1.

Total faults	ISCAS-85_c17		Polybox_5		Fulladder_1	
	CSSE-tree	CS-tree with MS	CSSE-tree	CS-tree with MS	CSSE-tree	CS-tree with MS
0	0.146392	1.010266	0.143459	0.824069	0.151481	0.863559
⩽1	0.821498	2.050138	0.660813	1.524773	0.784292	1.576730
1	1.760776	3.496916	1.350618	2.459045	1.290540	2.147267
⩽2	1.763235	3.485248	1.358411	2.280260	1.244020	1.988916
2	2.572540	4.718546	2.147613	3.134951	1.880567	2.559634
⩽3	1.886956	3.679675	1.619399	2.500500	1.289037	2.024858
3	3.522822	6.250428	2.843141	3.533181	2.684551	3.139069

Table 3
Time (s): maximum number of fault components in Polybox_9, Fulladder_2.

Total faults	Polybox_9		Fulladder_2	
	CSSE-tree	CS-tree with MS	CSSE-tree	CS-tree with MS
⩽1	1.453956	3.846129	15.748950	12.359005
1	8.521352	16.941680	29.620212	21.944777
⩽2	8.299941	15.081002	19.022576	22.309946
2	17.674624	30.465692	22.292435	30.781896
⩽3	13.662437	23.199518	27.624513	27.797646
3	23.355471	37.874223	40.605235	36.032242
⩽4	17.177651	27.322747	29.328317	29.277322
4	31.051707	43.596556	54.875494	51.354207

5. Discussion

As we know, in traditional methods for model-based diagnosis, conflict recognition and candidate generation are the two important steps. There are many algorithms to solve the two problems, such as ATMS [13], DART [14], the approach with structural information [15], and variant CS-trees [10,11] for deriving all minimal conflict sets; and HS-tree [6], HST-tree [16], BHS-tree [17], the approach based on Boolean algebra [17], the approach with genetic algorithm [18], and our HSSE-tree [19], etc. for deriving all minimal hitting sets. However, both of the two steps are NP-complete problems. Therefore, the efficiency of the final diagnosis is heavily affected by that of the two steps.

Instead, in this paper, we can directly compute all minimal consistency-based diagnosis sets without needing the two steps additionally. And the efficiency is very high in general. Especially our CSSE-tree is better than other algorithms for single- or double-diagnosis in general.

6. Conclusion and future work

In this paper, a universal algorithm of checking whether a component set is a diagnosis is proposed using SAT solvers. Then all minimal diagnoses can be derived by the CSSE-tree or by the other three similar algorithms. Experimental results show that our CSSE-tree generates fewest nodes, and has the best efficiency in general.

The hierarchical way has been applied in the MBD field, such as [20,21] and our hierarchical approach to model-based diagnosis of discrete-event systems [22]. Every independent part of the system can be solved in parallel, thus the efficiency will be improved, and then all the independent solutions are merged to obtain the final results. This can also be a new research direction to derive all minimal diagnoses.

Acknowledgments

This work was supported in part by NSFC Major Research Program under Grant Nos. 60496320 and 60496321, Basic Theory and Core Techniques of Non Canonical Knowledge; NSFC under Grant Nos. 60773097 and 60873148; Program for New Century Excellent Talents in University; Jilin Province Science and Technology Development Plan under Grant Nos. 20060532 and 20080107; European Commission under Grant No. TH/Asia Link/010 (111084); the graduate innovation lab of Jilin University under Grant No. 20080115, and “985” project of Jilin University.

References

[1] Console L, Dressler O. Model-based diagnosis in the real world: lessons learned and challenges remaining. In: Proceedings of the 16th international joint conference on artificial intelligence (IJCAI-99), Stockholm, Sweden; 1999. p. 1393–400.
[2] Kautz H, Selman B. Pushing the envelope: planning, propositional logic, and stochastic search. In: Proceedings of the 13th national conference on artificial intelligence (AAAI-96), Portland, Oregon; 1996. p. 1194–201.
[3] Biere A, Cimatti A, Clarke EM, et al. Symbolic model checking without BDDs. In: Proceedings of the 5th international conference on tools and algorithms for the construction and analysis of systems (TACAS-99); 1999. p. 193–207.

[4] Grastien A, Rintanen J, Kelareva E, et al. Diagnosis of discrete-event systems using satisfiability algorithms. In: Proceedings of the 22nd conference on artificial intelligence (AAAI-07), Vancouver, Canada; 2007. p. 305–10.
[5] Rintanen J, Grastien A. Diagnosability testing with satisfiability algorithms. In: Proceedings of the 20th international joint conference on artificial intelligence (IJCAI-07), Hyderabad, India; 2007. p. 532–7.
[6] Reiter R. A theory of diagnosis from first principles. Artif Intel 1987;32(1):57–96.
[7] de Kleer J, Mackworth AK, Reiter R. Characterizing diagnoses and systems. Artif Intell 1992;56(2-3):197–222.
[8] Zhang H. SATO: an efficient propositional prover. In: Proceedings of the international conference on automated deduction (CADE-97). LNAI 1997;1249:272–5.
[9] Moskewicz MW, Madigan CF, Zhao Y, et al. Chaff: engineering an efficient SAT solver. In: Proceedings of the 38th design automation conference (DAC-01); 2001. p. 530–5.
[10] Zhao X, Ouyang D. Improved algorithms for deriving all minimal conflict sets in model-based diagnosis. In: Proceedings of the 3rd international conference on intelligent computing (ICIC-07). LNCS 2007;4681:157–66.
[11] Han B, Lee SJ. Deriving minimal conflict sets by CS-trees with mark set in diagnosis from first principles. IEEE Trans Syst Man Cybernet B 1999;29(2):281–6.
[12] Rsat. UCLA Automated Reasoning Group. Available from: http://reasoning.cs.ucla.edu/rsat/ [2008-4-27].
[13] de Kleer J. An assumption-based tms. Artif Intell 1986;28(2):127–62.
[14] Genesereth MR. The use of design descriptions in automated diagnosis. Artif Intell 1984(24):411–36.
[15] Luan S, Dai G. An approach of diagnosing a system with structure information. Chin J Comput 2005;28(5):801–8, [in Chinese].
[16] Wotawa F. A variant of Reiter's hitting-set algorithm. Inform Process Lett 2001;79(1):45–51.
[17] Lin L, Jiang Y. The computation of hitting sets: review and new algorithms. Inform Process Lett 2003;86(4):177–84.
[18] Lin L, Jiang Y. Computing minimal hitting sets with genetic algorithm. In: Proceedings of the 13th international workshop on principles of diagnosis (DX-02), Austria; 2002. p. 77–80.
[19] Zhao X, Ouyang D. A method of combining SE-tree to compute all minimal hitting sets. Prog Nat Sci 2006;16(2):169–74.
[20] Chittaro L, Ranon R. Hierarchical model-based diagnosis based on structural abstraction. Artif Intell 2004;155(1-2):147–82.
[21] Siddiqi S, Huang J. Hierarchical diagnosis of multiple faults. In: Proceedings of the 20th international joint conference on artificial intelligence (IJCAI-07), Hyderabad, India; 2007. p. 581–6.
[22] Zhao X, Ouyang D. On-line diagnosis of discrete-event systems: a hierarchical approach. In: Proceedings of the IEEE international conference on robotics, automation and mechatronics (RAM-08), Chengdu, China; 2008. p. 785–90.