Journal of Information Processing Vol.20 No.3 767–773 (July 2012)

[DOI: 10.2197/ipsjjip.20.767]

Regular Paper

Efficient Root Cause Detection in Complex Embedded Systems with Abstract Model-based Diagnosis

Takuro Kutsuna[1,a)] Shuichi Sato[1,b)] Naoya Chujo[2,c)]

Received: October 5, 2011, Accepted: April 2, 2012

Abstract: The increasing complexity of embedded systems in information and communication technology causes a problem with locating faults during system failures. One reason for this problem is that system components that receive abnormal input data from other components may also output abnormal data, even if they are not in abnormal states, and consequently many redundant faults are detected in the system. In this paper, we present a diagnosis method for locating the origin of faults automatically in systems where fault propagation may occur. We use a model-based diagnosis scheme and abstract behavior modeling technique to deal with complex software components. We propose a new approach to diagnose systems that have data flow loops. Finally, we propose a one-stage approach for solving the abstract model-based diagnosis based on its formulation into the partial maximum satisfiability problem.

Keywords: diagnosis, embedded system, maximum satisfiability problem

1. Introduction

Embedded systems are getting increasingly large-scaled and complex in many fields of information and communication technology. The automotive control system is one of such complex systems that is composed of a lot of electronic control units (ECUs). For example, small-class vehicles are equipped with dozens of ECUs, and this number reaches a hundred or more for luxury-class vehicles. These ECUs communicate with one another via automotive networks such as CAN, LIN, and FlexRay. Increasing demands on vehicle safety, driver comfort, and environmental protection will continue to drive this trend.

After product launch, complex embedded systems tend to suffer from locating faults during system failures. Locating software faults is more difficult than locating most hardware faults, because no software faults can be found by looking. Furthermore, software components that receive abnormal input data from other components may also output abnormal data, even if they are not themselves in an abnormal state. Consequently, many redundant faults are detected in the system and the root cause analysis becomes an increasing challenge. We call this problem the fault-propagation problem, which will be more likely in systems in which many software components collaborate closely.

In this paper, we aim to develop a diagnosis method to locate the origin of faults automatically in a system where fault propagations may occur. We use a model-based diagnosis (MBD) scheme developed in the field of artificial intelligence [3], [11]. It is necessary to model the behavior of each component in a system to apply the original MBD scheme, in other words, we need to describe the relationship between the input and the output of each component in a system precisely. Unfortunately, however, it is very difficult to model the behavior of complex software components, such as ECUs, because they have too many branches in the calculation of their output to describe the relationship between the input and the output. Moreover, it will be computationally intractable to diagnose a system that includes complex software components even though we managed to model their behavior. Therefore, we apply the abstract behavior modeling technique proposed in Refs. [9] and [13], which enables us to handle complex software components within the MBD scheme without modeling their concrete behavior. Instead, unlike for the original MBD, we need some criteria to judge whether or not data flows in a system are normal. Hereafter, we refer to such MBD as "abstract MBD" (AMBD). Although the original AMBD can be applied to a system that has no data flow loops, problems occur when it is applied to a system with them. The original AMBD detects no faults when all data flows on a loop are abnormal, because all components on the loop can claim that they output abnormal data due to the abnormal input data. Automotive systems that include ECU components may have data flow loops because most communications among ECUs are mutual and not one-way. In this paper, we propose a modified AMBD to overcome this problem and enable a diagnosis that at least one of the components on a loop is abnormal, even when all data flows on the loop are abnormal.

MBD traditionally employs a two-stage approach based on the notions of conflict and diagnosis. First, conflicts are calculated using a theorem prover [11], [13] or constraint propagation [2]. Second, diagnoses are calculated from conflicts using a hitting set algorithm [11], [13] or a prime implicant algorithm based on boolean decision diagrams [2]. In this paper, we formulate AMBD into a partial maximum satisfiability problem (PMAXSAT). PMAXSAT is an optimization extension of the

[1] Toyota Central R&D Labs. Inc., Nagakute, Aichi 480–1192, Japan
[2] Aichi Institute of Technology, Toyota, Aichi 470–0392, Japan
[a)] kutsuna@mosk.tytlabs.co.jp
[b)] shuichi-sato@mosk.tytlabs.co.jp
[c)] ny-chujo@aitech.ac.jp

boolean satisfiability problem that has been studied for a long time. Although PMAXSAT is an NP-hard problem, many of large practical problems can be solved by state-of-the-art PMAXSAT solvers due to technical advancements in recent years [6]. Compared with traditional approaches, ours can receive the benefit of modern sophisticated algorithms easily, because sound solvers can be used to diagnose the system. Moreover, we extend our method so as to get diagnoses in the order of their occurrence probabilities.

This paper is organized as follows: Section 2 describes the existing AMBD. Section 3 discusses a drawback of the existing AMBD and presents our modified approach. Section 4 describes how we formulate AMBD into the partial maximum satisfiability problem and proposes some extensions. Section 5 shows some experimental results. Section 6 summarizes this work and discusses future work.

2. Abstract Model-based Diagnosis

The abstract behavior modeling technique in MBD was first introduced in Ref. [9] for the purpose of VHDL debugging. Then it was extended to diagnose robotic control systems that include complex software components in Ref. [13]. We define terms and review AMBD below.

2.1 Terms

First, we define terms required to explain AMBD.

Definition 1 $COMP$ is a set of components in the system, and DF is a set of data flows that includes both communication signals among components and input/output signals of the system. $OUT(c)$ is a function that returns a set of output data flows of component c, and $IN(c, s)$ is a function that returns a set of input data flows of component c related to output s.

Example 1 **Figure 1** is an example of an abstracted system where $COMP = \{C_1, C_2, C_3\}$ and $DF = \{a, b, c, d, e, f\}$. In the figure, $OUT(C_1)$ is $\{c, d\}$, and $IN(C_1, d)$ is $\{a, b\}$ for example.

For an automotive system, we regard ECUs and signals communicated via networks as components and data flows, respectively. Moreover, for a software system with an operating system, we regard tasks and messages among them as components and data flows, respectively.

2.2 System Description and Observations

We use the notions of system description (SD) and observations (OBS) in AMBD which is the same as in the original MBD.

Definition 2 SD is a set of first-order sentences that represents obvious relationships among components and data flows in a system. OBS is a set of first-order sentences that represents observations of a system.

In AMBD, SD is basically derived in the following manner:

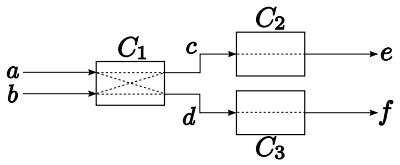

Fig. 1 An example of the abstracted system.

$$SD = \bigwedge_{c \in COMP} \bigwedge_{s \in OUT(c)} \left[ok(c) \wedge \bigwedge_{s' \in IN(c,s)} ok(s') \rightarrow ok(s) \right], \quad (1)$$

where a predicate $ok(x)$ means that x is normal. The idea behind Eq. (1) is, if both a component and its input data flows are normal, the corresponding output data flow must be normal. The main difference between the original MBD and AMBD is that there is no need to describe components' concrete behavior in AMBD while it is essential to describe every component's behavior in the original MBD.

OBS is defined as a function of $ok(s)$ where $s \in DF$ in AMBD. Unlike for the original MBD, we need some criteria to judge whether or not data flows in the system are normal for AMBD. For example, a criterion based on a periodic feature that data flow s_1 must be produced every n milliseconds is used in Ref. [13].

Example 2 SD for Fig. 1 is given by Eq. (1) as:

$$\begin{aligned} SD = {} & \{ok(C_1) \wedge ok(a) \wedge ok(b) \rightarrow ok(c)\} \\ & \wedge \{ok(C_1) \wedge ok(a) \wedge ok(b) \rightarrow ok(d)\} \\ & \wedge \{ok(C_2) \wedge ok(c) \rightarrow ok(e)\} \\ & \wedge \{ok(C_3) \wedge ok(d) \rightarrow ok(f)\}. \end{aligned} \quad (2)$$

And if we observe that data flows $\{a, b, f\}$ are normal, $\{c, e\}$ are abnormal, and the state of $\{d\}$ is unknown in Fig. 1, we get the following OBS:

$$OBS = ok(a) \wedge ok(b) \wedge \neg ok(c) \wedge \neg ok(e) \wedge ok(f). \quad (3)$$

Note that literals related to data flows whose states are unknown do not appear in OBS, such as data flow d in Eq. (3).

2.3 Diagnoses

We first define a function $\mathcal{D}$ in order to define diagnoses.

Definition 3 $\mathcal{D}$ is a function that assigns every component in a system as either normal or abnormal. $\mathcal{D}$ is given as follows:

$$\mathcal{D}(C_f) = \left[\bigwedge_{c \in C_f} \neg ok(c) \right] \wedge \left[\bigwedge_{c \in COMP \setminus C_f} ok(c) \right],$$

where C_f is a set of components assigned to abnormal.

We define a diagnosis as follows:

Definition 4 $C_f \subseteq COMP$ is a diagnosis if the following is satisfiable:

$$SD \wedge OBS \wedge \mathcal{D}(C_f). \quad (4)$$

Given both SD and OBS, we calculate diagnoses from them using Eq. (4). However, there may be an exponential number of diagnoses ($2^{|COMP|}$). We define a minimal diagnosis as follows:

Definition 5 A diagnosis C_f is a minimal diagnosis if no subset of C_f is a diagnosis.

Example 3 Let OBS in Fig. 1 be Eq. (3). Then we obtain $\{C_1\}$ as a minimal diagnosis by using Eqs. (2), (3), and (4). Thus, it is possible to locate the origin of a fault in a system where a fault that occurs in C_1 propagates to c and e through C_2.

2.4 AMBD with Time Consideration

It is possible to incorporate the notion of time in AMBD. Let

$TIME$ be a set of discrete time points to be considered, e.g., $TIME = \{0, \ldots, T\}$. Then, we can derive SD that includes delays between input and output as follows:

$$SD = \bigwedge_{t \in TIME} \bigwedge_{c \in COMP} \bigwedge_{s \in OUT(c)} \left[ok(c,t) \wedge \bigwedge_{s' \in IN(c,s)} ok(s',t) \rightarrow ok(s, t+\delta(s,c)) \right], \quad (5)$$

where a predicate $ok(x,t)$ means that x is normal at time t and $\delta(s,c)$ means a delay of output s in component c. OBS is extended to include time and defined as a function of $ok(s,t)$ where $s \in DF$ and $t \in TIME$. We also need to update $\mathcal{D}$ as:

$$\mathcal{D}(C_f^t) = \left[\bigwedge_{\{c,t\} \in C_f^t} \neg ok(c,t) \right] \wedge \left[\bigwedge_{\{c,t\} \in CT \setminus C_f^t} ok(c,t) \right],$$

where $CT = \{\{c,t\} | c \in COMP, t \in TIME\}$ and $C_f^t \subseteq CT$. Then, we can calculate diagnoses based on Eq. (4) where C_f is replaced with C_f^t.

Considering time in AMBD has the advantage that we can deal with intermittent faults because component faults are detected with time-indices. However, we need information about delays between input and output and time-indexed observations to incorporate time in AMBD as mentioned above. For an automotive system, it is quite difficult to estimate input-output delays in each ECU because it changes depending on a lot of factors. Moreover, if we use a periodic feature to judge whether or not data flows in the system are normal, it is not easy to assign the result to particular time point observations. For that reason, we do not consider time in AMBD in the following sections.

3. Diagnosing Systems with Loops

AMBD without time consideration works well when a system has no data flow loops, as in Fig. 1. However, a problem can arise when using AMBD to diagnose systems that have data flow loops. In this section, we explain the problem and propose a modified approach.

3.1 A Problem of Existing Methods

Figure 2 is an example of a system that has data flow loops. By using Eq. (1), we calculate SD for Fig. 2 as follows:

$$\begin{aligned} SD = & \{ok(C_1) \wedge ok(a) \wedge ok(e) \wedge ok(f) \rightarrow ok(b)\} \\ & \wedge \{ok(C_2) \wedge ok(b) \rightarrow ok(c)\} \\ & \wedge \{ok(C_2) \wedge ok(b) \rightarrow ok(e)\} \\ & \wedge \{ok(C_3) \wedge ok(b) \rightarrow ok(d)\} \\ & \wedge \{ok(C_3) \wedge ok(b) \rightarrow ok(f)\}. \end{aligned} \quad (6)$$

Now, assume that we have the following OBS:

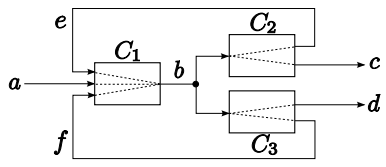

Fig. 2 An example of a system with data flow loops.

$$OBS = ok(a) \wedge \neg ok(b) \wedge ok(c) \wedge ok(d) \wedge \neg ok(e) \wedge ok(f). \quad (7)$$

Then, we get an empty set as a minimal diagnosis from Eqs. (4), (6), and (7). However, it is inappropriate to conclude that no components are abnormal even though abnormal data flows are observed in the system. As this example shows, AMBD described in Section 2 does not work well when the system has a data flow loop and all data flows on the loop are abnormal.

3.2 A Modified Approach

We first define a nonrecurrent loop, and then propose our modified approach. A path p in the system is denoted as:

$$p = (s_1, c_1, s_2, c_2, \ldots, s_k), \quad (8)$$

where $s_i \in DF$ and $c_i \in COMP$. A loop is a path such that $s_1 = s_k$ in Eq. (8). For example, (b, C_2, e, C_1, b) is a loop in Fig. 2. We define a nonrecurrent loop as follows:

Definition 6 A nonrecurrent loop is a loop that does not contain the same loop more than once.

Example 4 In Fig. 2, the path $(b, C_2, e, C_1, b, C_3, f, C_1, b)$ is a nonrecurrent loop, whereas the path $(b, C_2, e, C_1, b, C_2, e, C_1, b)$ is not one because it contains the loop (b, C_2, e, C_1, b) twice.

Definition 7 Let L_n be a set of all nonrecurrent loops in the system. For $l \in L_n$, $\pi(l)$ is a function that returns a set of unique components in l, $\phi(l)$ is a function that returns a set of unique data flows in l, and $in(l)$ is a function defined as follows:

$$in(l) = \bigcap_{c \in \pi(l), s \in \phi(l)} IN(c,s) \setminus \phi(l).$$

Example 5 In Fig. 2, there are three nonrecurrent loops: $l_1 = (b, C_2, e, C_1, b)$, $l_2 = (b, C_3, f, C_1, b)$, and $l_3 = (b, C_2, e, C_1, b, C_3, f, C_1, b)$. For example, $\pi(l_1) = \{C_1, C_2\}$, $\phi(l_1) = \{b, e\}$, and $in(l_1) = \{a, f\}$.

In order to overcome the problem mentioned above, we derive SD' as:

$$SD' = \Phi_c \wedge \Phi_l, \quad (9)$$

where Φ_c is given by the right side of Eq. (1) and Φ_l is given as follows:

$$\Phi_l = \bigwedge_{l \in L_n} \left[\bigwedge_{c \in \pi(l)} ok(c) \wedge \bigwedge_{s \in in(l)} ok(s) \rightarrow \bigwedge_{s' \in \phi(l)} ok(s') \right]. \quad (10)$$

The idea behind Eq. (10) is, if all components on a loop and input data flows into the loop are both normal, all the data flows on the loop must be normal. While Φ_c represents obvious relationships that hold in each component, Φ_l represents obvious relationships that hold in each nonrecurrent loop.

Example 6 SD' for Fig. 2 is given as follows by using Eq. (9):

$$\begin{aligned} SD' = & \text{ (the right side of Eq. (6))} \\ & \wedge \{ok(C_1) \wedge ok(C_2) \wedge ok(a) \wedge ok(f) \rightarrow ok(b) \wedge ok(e)\} \\ & \wedge \{ok(C_1) \wedge ok(C_3) \wedge ok(a) \wedge ok(e) \rightarrow ok(b) \wedge ok(f)\} \\ & \wedge \{ok(C_1) \wedge ok(C_2) \wedge ok(C_3) \wedge ok(a) \\ & \quad \rightarrow ok(b) \wedge ok(e) \wedge ok(f)\}. \end{aligned} \quad (11)$$

Let OBS be Eq. (7). Then we obtain $\{\{C_1\},\{C_2\}\}$ as a set of minimal diagnoses from Eqs. (4), (7), and (11). Thus, when all data flows on a loop are abnormal, at least one of the components on the loop is abnormal.

It is essential to take nonrecurrent loops into consideration, otherwise problems occur when a system has overlapping elementary loops and all data flows on those loops are abnormal. We show this through the following example.

Example 7 Let SD for Fig. 2 be Eq. (12), where clauses related to nonrecurrent loops are removed from Eq. (11):

$$\begin{aligned} SD = \ & \text{(the right side of Eq. (6))} \\ & \wedge\{ok(C_1) \wedge ok(C_2) \wedge ok(a) \wedge ok(f) \rightarrow ok(b) \wedge ok(e)\} \\ & \wedge\{ok(C_1) \wedge ok(C_3) \wedge ok(a) \wedge ok(e) \rightarrow ok(b) \wedge ok(f)\}. \end{aligned} \tag{12}$$

Assume that we have the following OBS:

$$OBS = ok(a) \wedge \neg ok(b) \wedge ok(c) \wedge ok(d) \wedge \neg ok(e) \wedge \neg ok(f). \tag{13}$$

Then, we get an empty set as a minimum diagnosis from Eqs. (4), (12), and (13). This happens because all elementary loops can claim that they output abnormal data due to the abnormal input data from other elementary loops.

3.3 Finding All Nonrecurrent Loops

In our modified approach, it is necessary to find all nonrecurrent loops in a system. In this section, we propose an algorithm to find all nonrecurrent loops in a system based on the notion of an elementary loop.

Definition 8 An elementary loop is a loop that does not contain the same data flow more than once except for the beginning and the end of the loop.

Example 8 In example 5, l_1 and l_2 are elementary loops, but l_3 is not an elementary loop because it contains data flow b more than once.

We propose an algorithm for finding all nonrecurrent loops in **Fig. 3**. We first find all elementary loops in the system, which is done effectively using the BACKTRACK algorithm [14]. Then we express the overlapping of elementary loops as a graph in Step 2 and calculate the connected subgraphs of that graph in Step 3. Note that each connected subgraph corresponds to a nonrecurrent loop in the system. Finally we calculate a component set and a data flow set of the corresponding nonrecurrent loop in Step 4.

Step 1. Let L_e be a set of all elementary loops in the system.
Step 2. Let G_e be a graph where each node represents an elementary loop in L_e. For every node pairs $\{l_i, l_j\}$ $(i \neq j)$ in G_e, add an edge between them if $\phi(l_i) \cap \phi(l_j) \neq \emptyset$.
Step 3. Calculate all connected subgraphs in G_e, and denote the set of those subgraphs as S_n.
Step 4. For each subgraph $G_n \in S_n$, calculate π and ϕ of the corresponding nonrecurrent loop l_n as:

$$\pi(l_n) = \bigcup_{l_e \in N(G_n)} \pi(l_e),\ \phi(l_n) = \bigcup_{l_e \in N(G_n)} \phi(l_e),$$

where a function $N(G_n)$ returns a node set of G_n.

Fig. 3 An algorithm for finding all nonrecurrent loops.

4. Formulation into the Partial Maximum Satisfiability Problem

Two-stage approaches based on the notions of conflict and diagnosis are used traditionally to diagnose systems in MBD [2], [11] and in AMBD [9], [13]. In this section, we propose a new one-stage approach for AMBD. We formulate the problem of diagnosing systems into the partial maximum satisfiability problem that can be solved effectively using state-of-the-art solvers.

4.1 The Maximum Satisfiability Problem and Its Extensions

The maximum satisfiability problem (MAXSAT) is an optimization problem of assigning variables so as to satisfy as many clauses in a given boolean formula as possible. The input boolean formula is given in conjunctive normal form (CNF). The partial MAXSAT (PMAXSAT) is an extension of MAXSAT where a certain subset of clauses in a given formula is treated as a hard constraint that must be satisfied. In PMAXSAT, unsatisfiable (UNSAT) is returned as a solution if it is impossible to satisfy all hard constraints simultaneously. The weighted MAXSAT (WMAXSAT) is another extension of MAXSAT where each clause has a weight and variables are assigned so as to maximize the sum of weights of satisfied clauses. The weighted partial MAXSAT (WPMAXSAT) is a problem where both hard constraints and weights are considered.

Example 9 Let us consider the following formula with four clauses:

$$(a \vee \neg b) \wedge (\neg a \vee c) \wedge (b \vee c) \wedge \neg c.$$

Assume that the first and second clauses are hard constraints, and the weights of the third and fourth clauses are 3 and 4 respectively. Then, the assignment $(a, b, c) = (0, 0, 0)$ is a solution of WPMAXSAT where the first, second, and fourth clauses are satisfied.

MAXSAT is an NP-hard problem, so it would be intractable as the size of the input formula gets larger. However, due to technical advancements based on the Davis-Putnam-Logemann-Loveland algorithm [1] and its recent extensions, state-of-the-art MAXSAT solvers can solve large practical problems strictly. We used YICES [6] as a MAXSAT solver in our experiments in Section 5. Note that YICES is a solver for the satisfiability modulo theories problem, which is a generalization of the boolean satisfiability problem, but can also deal in MAXSAT (including PMAXSAT, WMAXSAT, and WPMAXSAT).

4.2 Formulating AMBD into PMAXSAT

We first need to convert Eq. (4) into CNF because the input formula of MAXSAT must be in CNF. Suppose that OBS is given in CNF as Eq. (3). Then we need only to convert SD' into CNF because $\mathcal{D}(C_f)$ is already CNF. We easily convert SD' into CNF using the following equation:

$$\bigwedge_{x \in X} x \rightarrow \bigwedge_{y \in Y} y = \bigwedge_{y \in Y} \left[\bigvee_{x \in X} \neg x \vee y \right], \tag{14}$$

Step 1. Let $C_a = \emptyset$ and $\Delta' = \Delta$.
Step 2. Solve Δ' as PMAXSAT and denote the solution as x^*.
Step 3. If $x^* \neq$ UNSAT, then go to Step 4, else go to Step 5.
Step 4. Let C^* be a set of components assigned to $\neg ok$ in x^*. Update C_a and Δ' as:

$$C_a = C_a \cup C^*,$$
$$\Delta' = \Delta' \wedge \left[\bigvee_{c \in C^*} ok(c) \right],$$

where the added clause in the second equation is treated as a hard constraint. Then go back to Step 2.
Step 5. Output C_a as a set of minimal diagnoses.

Fig. 4 An algorithm for calculating all minimal diagnoses with PMAXSAT solvers.

where X and Y are arbitrary sets of variables. We denote the CNF conversion of SD' as SD'_c.

Now, define Δ as:

$$\Delta = SD'_c \wedge OBS \wedge \mathcal{D}(\emptyset), \tag{15}$$

where clauses in both SD'_c and OBS are treated as hard constraints. Let x^* be a solution of PMAXSAT of Δ. Then a set of components that are assigned to $\neg ok$ in x^* is a minimal diagnosis of the corresponding AMBD because x^* is a solution that satisfies Eq. (4) and in which as many components are assigned to ok as possible.

However, we can get only one of the whole minimal diagnoses by solving Eq. (15) as PMAXSAT. Hence we propose an algorithm for calculating all minimal diagnoses with PMAXSAT solvers in **Fig. 4**. In Fig. 4, a new clause is added to Δ as a hard constraint in Step 4 so as not to obtain the same diagnosis as a solution of PMAXSAT again. If we do not need to get all minimal diagnoses but only a subset of them, we can add a condition to Step 3 such as "go to Step 5 if the size of C_a exceeds the given threshold." Furthermore, if we would like to get not only the minimal diagnoses but also all possible diagnoses, it is enough to modify the equation for updating Δ' in Step 4 as follows:

$$\Delta' = \Delta' \wedge \left[\bigvee_{c \in COMP \setminus C^*} \neg ok(c) \right].$$

4.3 Consideration of Fault Probabilities

Suppose that a system has n components C_i $(i = 1, \ldots, n)$ and the fault probability of C_i is p_i. Define x_i as:

$$x_i = \begin{cases} 1 & \text{if } ok(C_i), \\ 0 & \text{otherwise.} \end{cases} \qquad (i = 1, \ldots, n) \tag{16}$$

If each component is assumed to break down independently, then the occurrence probability of $x = (x_1, \ldots, x_n)$ is given as follows:

$$P(x) = \prod_{i=1}^{n} (1 - p_i)^{x_i} \, p_i^{1 - x_i}. \tag{17}$$

By taking logarithms of both side of Eq. (17) and transforming its right side, we obtain the following equation:

$$\log P(x) = \sum_{i=1}^{n} \left\{ \log \frac{1 - p_i}{p_i} \right\} x_i + \text{Const}, \tag{18}$$

where $\text{Const} = \sum_{i=1}^{n} \log p_i$. From Eqs. (16), (18), and the fact that log is a monotone increasing function, it follows that we can get the diagnosis that has the highest occurrence probability by setting the weights of clauses $ok(C_i)$ $(i = 1, \ldots, n)$, which appears in $\mathcal{D}(\emptyset)$ of Eq. (15), as

$$\log \frac{1 - p_i}{p_i} \quad (i = 1, \ldots, n)$$

respectively and solving Eq. (15) as WPMAXSAT. Moreover, we can get diagnoses in the order of their occurrence probabilities by applying the algorithm in Fig. 4 where PMAXSAT is replaced with WPMAXSAT.

5. Experiments

In this section, we present experimental results to show how the proposed method works. The experiment is done on a Microsoft Windows XP machine with an Intel Core i7 CPU (2.80 GHz, 3.5 GB RAM).

5.1 Diagnosis of An Automotive Control System

5.1.1 System Overview

We assume the automotive control system in **Fig. 5**, which is quoted from Ref. [12] and modified herein for the sake of simplicity. There are 8 ECUs in the system and they communicate with one another via the controller area network (CAN), which is one of the commonly used automotive networks. The system has 20 data flows, labeled 9 to 28, which are shown in the figure as the numbers on the arrows. The dashed lines in each ECU in the figure indicate dependencies between the input data and the output data of the ECU; however, we can only presume them because we could not extract information about them from Ref. [12].

5.1.2 Observations and Fault Probabilities

Assume that we observe the abnormalities of the data flows in the system as shown in Fig. 5 where solid, dashed, and dotted arrows mean that the data flow is normal, abnormal, and unknown, respectively. Then, OBS is derived in a form such as Eq. (3). And we assume that the fault probability of ECU i $(i = 1, \ldots, 8)$ equals $0.001 \times i$ $(i = 1, \ldots, 8)$ respectively.

5.1.3 Deriving the System Description

We implement the algorithm to derive SD'_c automatically from $(COMP, DF, IN, OUT)$ in the system according to Eqs. (9), (14), and the algorithm in Fig. 3. We find 10 elementary loops and 77 nonrecurrent loops for the system in Fig. 5, resulting in an SD'_c with 946 clauses. SD'_c becomes large for the size of the system because the system has many overlapping elementary loops. If the elementary loops were less overlapped, SD'_c would be much smaller.

5.1.4 Diagnosing the System

We implement the diagnosing algorithm described in Section 4 and calculate the minimal diagnoses of the system in Fig. 5. The result is shown in **Table 1** where minimal diagnoses are sorted in the order of their occurrence probabilities. We can get minimal diagnoses in the same order as in Table 1 one after another using the proposed algorithm. All minimal diagnoses in the table contain ECU1 because the abnormalities of the data flows $\{10, 16\}$ can be explained by fault propagation from the abnor-

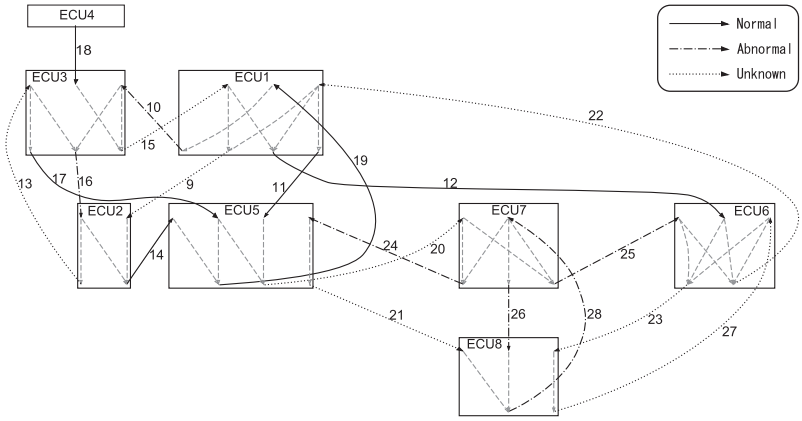

Fig. 5 Automotive control system that has 8 ECUs and 20 data flows (labeled 9 to 28). The solid arrow means that the data flow is normal, the dashed arrow means that it is abnormal, and the dotted arrow means that its state is unknown.

Table 1 Diagnostic results of the system in Fig. 5.

Minimal Diagnosis	Probability
{ECU1, ECU8}	7.8×10^{-6}
{ECU1, ECU7}	6.8×10^{-6}
{ECU1, ECU5}	4.9×10^{-6}

Table 2 An overview of nine synthetic systems and the average of computational time and peak memory usage for diagnosing these systems.

ID	$\|COMP\|$	$\|DF\|$	$\|L_n\|$	$\|SD'_c\|$	$E[time]$		$E[peakmem]$	
A	50	250	16	571	0.21	(0.01)	564.2	(1.5)
B	50	250	64	2,331	0.21	(0.01)	586.8	(38.2)
C	50	250	104	2,915	0.21	(0.01)	582.5	(33.0)
D	100	500	8	674	0.21	(0.01)	602.4	(5.3)
E	100	500	75	3,640	0.21	(0.01)	623.6	(51.0)
F	100	500	504	23,473	0.43	(0.04)	734.0	(333.2)
G	200	1,000	19	1,403	0.22	(0.01)	679.3	(10.6)
H	200	1,000	254	17,437	0.53	(0.09)	802.2	(360.6)
I	200	1,000	352	22,815	0.73	(0.07)	741.4	(349.6)

mality of ECU1. Conversely, the abnormalities of the data flows {24, 25, 26, 28} can be explained by fault propagation from either ECU7 or ECU8. Moreover, ECU5 can be the origin of fault propagation through the data flows {24, 25, 26, 28} if we assume that data flow 21 is abnormal. So {ECU1, ECU5} is also regarded as a minimal diagnosis.

The computational time required to get each minimal diagnosis by solving WPMAXSAT is about 0.2 second. Consequently, it takes about 0.6 second in total to get all minimal diagnoses in Table 1.

5.2 Scalability Test with Synthetic Systems

Another experiment is done to check the scalability of the proposed method. We have generated nine synthetic systems randomly. **Table 2** shows an overview of these systems. In Table 2, $|COMP|$ and $|DF|$ means the number of components and data flows in the system respectively. Nonrecurrent loops are calculated by the algorithm in Fig. 3, then SD'_c is calculated according to Eqs. (9) and (14) for each system. In Table 2, $|L_n|$ means the number of nonrecurrent loops in the system and $|SD'_c|$ means the number of clauses in SD'_c. We generate an OBS randomly and calculate a minimal diagnosis by solving WPMAXSAT of Δ in Eq. (15). This procedure is repeated a hundred times for each system. The average time of obtaining a diagnosis is shown in Table 2 as $E[time]$ on the second time scale, and its standard deviation is inside the parentheses. Also, the average of peak memory usage is shown in Table 2 as $E[peakmem]$ on the kilobyte scale. From Table 2, we can see that $|SD'_c|$ largely depends on $|L_n|$, and the average time of diagnosis increases with respect to $|SD'_c|$. The average of peak memory usage increases slightly depending on $|SD'_c|$, on the other hand, its standard deviation increases rapidly as $|SD'_c|$ gets large. We have analyzed the results of the systems {F, H, I} in detail and found that the peak memory usage rarely gets large in these systems depending on the OBS. It seems that the proposed method is applicable to a system whose size is comparable to that of a system in Table 2, because it will take less than a second to diagnose the system with less than a megabyte of memory consumption on average.

6. Summary and Future Work

In this paper, we proposed a method for diagnosing complex embedded systems that include many software components based on the abstract model-based diagnosis. We modified the existing method for deriving the system description in order to diagnose systems that have data flow loops. We also propose a one-stage approach for solving the abstract model-based diagnosis based on its formulation into the partial maximum satisfiability problem.

In using the abstract model-based diagnosis, we need criteria to judge whether or not data flows in a system are normal. Therefore it is necessary to develop methods to derive these criteria effectively. One possible approach is to log the data communicated among components and derive the criteria from them using data mining, machine learning, or statistical methods. Another possible approach is to model software components using the framework of formal methods and certify the criteria using software verification methods. Consequently, the abstract model-based diagnosis can be a bridge between AI-based diagnosis methods and other methodologies. Of course, it is possible to use the result of system reliability analysis, such as failure mode and effect analysis (FMEA) or fault tree analysis (FTA), to derive the criteria practically. Papadopoulos et al. [10] in particular proposed the Interface Focused-FMEA (IF-FMEA) which is a modification of classical FMEA that analyzes how a hardware or software component reacts to failures generated by other components. The

result of IF-FMEA can be applied to compose the diagnostic criteria more effectively.

We need information about the system architecture to derive the system description using the abstract model-based diagnosis. There are several languages to describe the system architecture, such as EAST-ADL [5] or architecture analysis and design language (AADL) [8]. Therefore it would be useful to develop tools to extract the system description automatically from source files written in such architecture description languages. Moreover, some of these languages have error modeling functions [7], making it possible to construct the diagnosis system automatically from source files that include information about both system architecture and system errors.

The partial maximum satisfiability problem cannot be applied to the original model-based diagnosis [2] because it can only deal in boolean formulas. However, it is possible to formulate the original model-based diagnosis into the satisfiability modulo theories problem, which is a generalization of the boolean satisfiability problem, and to solve it with the state-of-the-art solvers, such as YICES [6] and Z3 [4], in a similar manner to the proposed method.

References

[1] Davis, M. and Putnam, H.: A computing procedure for quantification theory, *J. ACM*, Vol.7, No.3, pp.201–215 (1960).

[2] de Kleer, J. and Kurien, J.: Fundamentals of model-based diagnosis, *Proc. 5th IFAC Symposium on Fault Detection, Supervision, and Safety of Technical Processes* (*Safeprocess*), Washington, D.C., USA, pp.25–36, Elsevier (2003).

[3] de Kleer, J. and Williams, B.: Diagnosing Multiple Faults, *Artificial Intelligence*, Vol.32, No.1, pp.97–130 (1987).

[4] de Moura, L. and Bjørner, N.: Z3: An efficient SMT solver, *Tools and Algorithms for the Construction and Analysis of Systems*, pp.337–340, Springer (2008).

[5] Debruyne, V., Simonot-Lion, F. and Trinquet, Y.: EAST-ADL – An Architecture Description Language, *18th IFIP World Computer Congress, ADL Workshop*, Toulouse, France, pp.53–62 (2004).

[6] Dutertre, B. and de Moura, L.: The Yices SMT solver, Technical Report, SRI International (2006).

[7] Feiler, P. and Rugina, A.: Dependability Modeling with the Architecture Analysis & Design Language (AADL), Technical Report, Carnegie Mellon University (2007).

[8] Feiler, P., Gluch, D. and Hudak, J.: The Architecture Analysis & Design Language (AADL): An Introduction, Technical Report, Carnegie Mellon University (2006).

[9] Friedrich, G., Stumptner, M. and Wotawa, F.: Model-based diagnosis of hardware designs, *Artificial Intelligence*, Vol.111, No.1–2, pp.3–39 (1999).

[10] Papadopoulos, Y., McDermid, J., Sasse, R. and Heiner, G.: Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure, *Reliability Engineering & System Safety*, Vol.71, pp.229–247 (2001).

[11] Reiter, R.: A Theory of Diagnosis from First Principles, *Artificial Intelligence*, Vol.32, No.1, pp.57–95 (1987).

[12] Sano, Y., Fukatsu, H., Yahata, Y., Sakai, K., Hino, R. and Arashida, T.: Development of New In-Vehicle Communications System, Technical Report, Mitsubishi Motors Technical Review (in Japanese) (2004).

[13] Steinbauer, G. and Wotawa, F.: Detecting and locating faults in the control software of autonomous mobile robots, *16th International Workshop on Principles of Diagnosis*, pp.13–18 (2005).

[14] Tarjan, R.: Enumeration of the Elementary Circuits of a Directed Graph, *SIAM Journal on Computing*, Vol.2, pp.211–216 (1973).

Takuro Kutsuna was born in 1979. He is a researcher at Toyota Central R&D Labs. Inc. He received his M.S. from Kyoto University in 2004. His research interests include artificial intelligence, machine learning and embedded software systems.

Shuichi Sato was born in 1967. He received his Ph.D. in Engineering from Osaka University in 2005. He has been working in Toyota Central R&D Labs. Inc. He has been engaged in the fields of robotics, artificial intelligence, manufacturing system design and marketing science. He is a member of the Japan Society of Mechanical Engineers. He is also a member of the Japan Institute of System, Control and Information Engineers.

Naoya Chujo was born in 1958. He received his B.S., M.S. and Ph.D. from Nagoya University in 1980, 1982 and 2004 respectively. He has been engaged in research on automotive electronics with Toyota Central R&D Labs. Inc. since 1982. He became a professor at Aichi Institute of Technology in 2010. His current research interests are the embedded systems. He received the Best ASIC Prize of DATE Conference in 1999. He is a member of IPSJ, IEICE, IEEJ and IEEE-CS.