

On the automatic synthesis of functional dependency graphs from libraries of component models

Luca Console, Roberta Meo, & Daniele Theseider Dupré

Dipartimento di Informatica, Università di Torino

Corso Svizzera 185, 10149 Torino, Italy

EMail: {lconsole,meo,dtd}@di.unito.it

Abstract

The IDEA system for car diagnosis developed by Centro Ricerche Fiat is one of the most successful practical applications of automated diagnosis. However, IDEA is based on functional dependency models that have to be built ad-hoc for each specific device to be diagnosed. Thus, the models are not easily reusable and this increases the cost of building new applications.

In this paper we show how models of the sort used by IDEA can be derived automatically from a library of generic component models, given a description of the structure of the device to be diagnosed. In this way we can reconcile two goals: (i) exploiting libraries of component models that can be reused in many applications (both in diagnostic applications and for other tasks), cutting down the cost of developing a new application and (ii) changing the IDEA system as less as possible. The latter is a major goal since IDEA is a fully engineered system, installed in more than 1500 car repair centers in Italy and integrated with on-board and off-board equipment.

1 Introduction

One of the major advantages of the component-oriented approach to model-based diagnosis [4] is the possibility of re-using models. Given a device, according to such an approach, a diagnostic system requires:

- a description of the structure of the device, i.e., a decomposition of the device into minimal diagnosable (replaceable or repairable) components and a description of the connections between them;

- a description of the behavior of each type of component.

While the former is device-dependent, the latter can be device independent. Thus, libraries of models for component types can be built and used to provide models of different systems that have components in common. For example, a generic description of the behavior of a pipe in the library can be re-used for diagnosing any device having a pipe as a component. Moreover, the description can be independent of the reasoning task (diagnosis) so that it can be used for other tasks (e.g., simulation can be applied to automate FMEA [6]).

In order to achieve this goal the description of the component types in the library must be context-independent, fulfilling the “no-function-in-structure” principle [3]: a component’s behavior must be described independently of the specific function that the component can have in different devices.

Reusing libraries of models can dramatically reduce the cost of developing a new diagnostic application. However, there are many diagnostic systems which are based on models that are not easily reusable. This is the case, in particular, of all those systems whose models rely on the behavior of aggregates of components.

The diagnostic system IDEA [1] is an example of these systems. IDEA has been developed by CENTRO RICERCHE FIAT to diagnose different car subsystems; currently the applications concern most of the electronic and electrical subsystems of all cars of the FIAT group.

IDEA is an advancement with respect to earlier systems developed by CENTRO RICERCHE FIAT and based on fault trees. A specific knowledge representation and acquisition methodology has been devised, which allows for cheaper knowledge acquisition, as discussed in [1]. However, the types of models used by IDEA (which will be described in the next section) are still not directly reusable and indeed there is a group of technicians in FIAT working on the development of IDEA models for new devices. On the other hand, IDEA is a fully engineered and broadly used system. Currently, more than 1500 installations have been made in car repair centers and the system is used frequently. Moreover a significant engineering effort has been made to make the system easily usable: IDEA has a friendly interface, sophisticated explanation capabilities and, which is more important, is able to dialogue with all electronic equipment used in the modern cars and in the repair centers. Thus modifying the implemented system would be very costly.

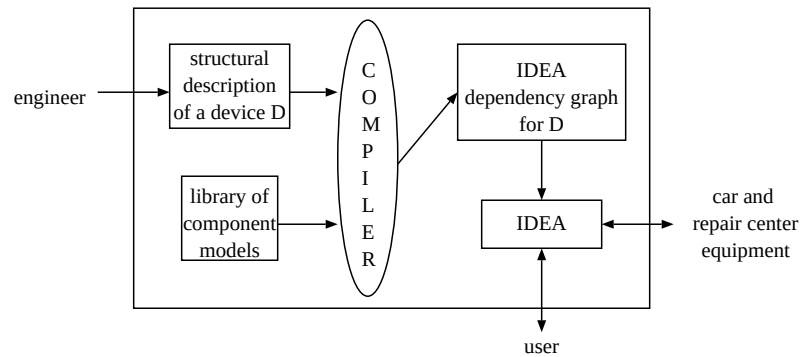


Figure 1: Architecture of the system.

In this paper we present an approach to reconcile the two following goals:

- changing the IDEA system as less as possible;
- converting IDEA to exploiting libraries of reusable components models, thus reducing the cost of knowledge acquisition.

The approach is based on the automatic synthesis of the models used by IDEA starting from:

- a library of generic components models;
- a description of the structure of the device to be diagnosed.

Thus the final architecture of the system is the one reported in figure 1. In order to build a new application (for a new device or a new version of a device), only a description of the structure of the device must be given. The actual model used by IDEA is generated automatically by a compiler. If new components, that are not modeled in the library, are used in a specific device, such — presumably few — component types must be added to the library.

Additional advantages of the approach are the following:

- the models can be reused for other reasoning tasks (e.g. automated FMEA or simulation);
- diagnostic assumptions and strategies different from those of IDEA can be experimented on the same models;

- in particular, some extensions of IDEA can be devised to exploit the same way of synthesizing models, but relaxing some of the restrictive assumptions in the system.

The paper is organised as follows. In the next section we briefly sketch the formalism and diagnostic algorithm used by IDEA. We then introduce the languages for defining generic component models and for describing the structure of a device. In the fourth section we discuss how the models used in IDEA can be automatically synthesized from a model library. We then sketch some extensions to IDEA that can be obtained using the compilation approach from a library of models.

2 The IDEA diagnostic system

In this section we shall briefly describe the core of IDEA [1], namely the formalism it uses for modeling the device to be diagnosed and the diagnostic strategies it exploits.

2.1 The formalism

The representation formalisms distinguishes three types of entities that can occur in a model:

1. *components*, which are the minimal entities that can be replaced or repaired; each component is characterized by a set of modes of behavior (including the correct mode and, possibly, a set of fault modes);
2. *signals* between such components (e.g. inputs and outputs to components);
3. *functional dependencies* between signals and components or between signals and signals.

The signals are the most important entities in the formalism; each signal corresponds to an interface between two (or more) components and can assume two values: a *normal* one and an *abnormal* one. Some signals may be observable; according to IDEA terminology these are called *symptoms*.

Functional dependencies are the core of IDEA representation. Basically, two types of dependencies are considered:

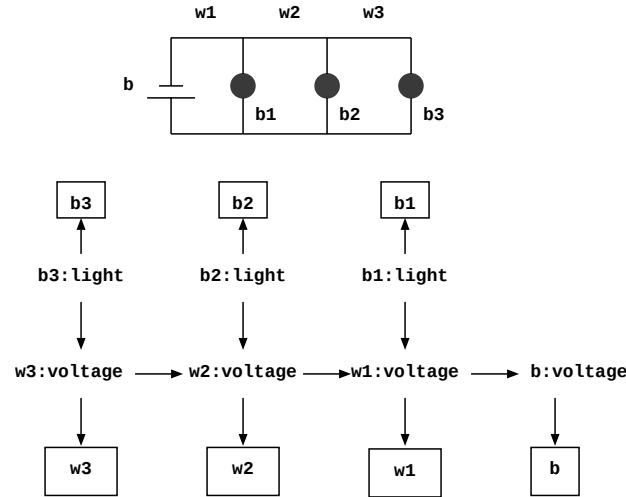


Figure 2: The classical example of the three bulbs and the corresponding dependency graph in the IDEA formalism.

- Dependencies between signals. If a signal s_1 depends on s_2 , then s_2 is regarded as a precondition for s_1 . For example, given a bulb, the signal “light_from_bulb” depends functionally on the signal “voltage_to_the_bulb”.
- Dependencies between signals and components. A signal may depend on the correct or faulty behavior of a component. For example, the signal “light_from_bulb” depends functionally on the fact that the bulb is OK.

Given a device, the functional dependencies between the signals and the components in the device can be represented by means of a graph such that:

- nodes correspond to components (or to modes of behavior of the components) or to signals;
- arcs correspond to functional dependencies.

Figure 2 reports the example electrical circuit in [8] and the corresponding dependency graph: a battery **b** is connected in parallel to three bulbs **b1**, **b2**, **b3**. For the sake of simplicity each wire **wi** corresponds to a pair of wires of the circuit (for example, **w1** corresponds to the wires connecting **b** to **b1**). In the dependency graph,

the nodes corresponding to the components are in square boxes; the other nodes correspond to signals. As an example the signal **b2:light** (corresponding to the fact that the bulb **b2** is lit or not) depends on the component **b2** (i.e., on the fact that **b2** is correct or faulty) and on the signal on wire **w2** (**w2:voltage**), which in turn depends on the component **w2** and on the signal on wire **w1**.

An example of a dependency graph for a real application can be found in [1].

As we noticed, the signals considered in IDEA are binary, where one of the two values correspond to the normal one and the other to the abnormal one. The development of IDEA knowledge bases is easy if the following inferences are correct: given a signal s and the component c having s as an output, then:

1. if s is abnormal, then either c is faulty or at least one of the inputs to c that are relevant for s is in turn abnormal;
2. if s is normal, then both c and all of its inputs that are relevant for s must be normal.

This is clearly a restrictive assumption since it imposes that there is a strict duality between normal and faulty behavior. The duality is reflected by the simple form of the dependency graphs. In fact, representing dependencies between signals (and signals and components) is sufficient, without considering values of signals. The node labeled with a signal s corresponds to both the normal and abnormal value of s . Thus, the dependencies starting from s can be read in two different ways: if we assume that s is abnormal, then each dependency points to a signal (component) whose abnormality is a possible cause of s ; if we assume that s is normal then the dependencies point to all the signals (and components) that must be normal.

The assumption above does not hold, for example, in case signals are not binary, and when the signal s produced by c under faulty behavior depends also on the inputs to c . In these cases the construction of IDEA models requires coding the actual signals into binary signals and a hand made transformation of actual dependencies into relations from symptoms to causes. This transformation can be automatised if generic models of component types are available.

In the section entitled “Extended dependency graphs” we shall introduce an extended definition of functional dependency graphs that are not based on the assumptions above but are still similar

to those used by IDEA and can be used adopting similar diagnostic strategies.

2.2 Diagnostic strategy

The diagnostic strategy adopted by IDEA can be regarded as a special form of abduction with corroboration and relies on the assumptions discussed in the previous section. Abduction [5, 2] is used to search for explanations of abnormal signals: if a signal s is abnormal, then at least one of the components or signals on which it depends must be abnormal. Corroboration (see the “alibi” principle in [7]) is used to exonerate components: if a signal s is normal, then all the signals and components on which it depends are assumed to be normal (notice that in this way IDEA cannot deal properly with masking faults, which, anyway, are not very common in the domain of application of the system). Corroboration can also be interpreted abductively in the sense that it gives the minimal abnormality explanation for the normality of s .

The form of reasoning sketched above is obtained by means of constraint propagation on the functional dependency graph. Two forms of propagation are performed by IDEA:

- *failure propagation*, which starts from an abnormal signal (symptom) and labels as candidate explanations all the signals and components modes on which it depends (directly or indirectly through chains of arcs).
- *functionality propagation*, which starts from a signal which is observed to be normal and labels as exonerated (normal) all the signals and components on which it (directly or indirectly) depends.

Let us consider a simple example, referring again to figure 2. Let us suppose that **b2:light** is observed to be abnormal (i.e. **b2** is not lit) and **b1:light** is observed to be normal (i.e. **b1** is lit). Two candidate explanations for the abnormality of **b2:light** are: (i) **b2** is abnormal (broken) and (ii) signal **w2:voltage** is abnormal. The latter can be explained either by a fault of **w2** or by the fact that signal **w1:voltage** is abnormal. In summary, failure propagation leads to four candidate single faults: **b2**, **w2**, **w1** and **b**. These are, in fact, the components on which the signal **b2:light** (directly or indirectly) depends and

thus at least one of them must be faulty when **b2:light** is abnormal. Functionality propagation starting from the observation that **b1:light** is normal leads to exonerating **b1** and **w1:voltage** which in turn exonerates **w1** and (via **b:voltage**) the battery **b**. Thus two candidates remain: **b2** and **w2**. These candidates could be discriminated by making further measurements (in the example **w2:voltage**).

IDEA uses some focusing techniques in order to speed-up search in the functional dependency graph; for example fault probabilities can be used to investigate the most probable faults first.

3 Extending IDEA to exploit model libraries

The models used by IDEA are tailored for a diagnostic task and strategy and do not include a notion of component type to be reused for all components of the same type. In the rest of the paper we show how they can be derived from a library of task-independent models of component types. Such generic models can then be reused:

- to generate input to IDEA for different instances of the same component type in the same device and in different devices;
- for diagnostic strategies different from those of IDEA;
- for reasoning tasks different from diagnosis, e.g. qualitative simulation for FMEA.

In this section we describe the language we assumed for the library of models; the compilation algorithm will be discussed in the next section. The languages we shall use are fairly general and indeed several other languages could have been used to the purpose of the paper.

The library must contain a description of each type of component of the systems to be diagnosed. In particular, we assume that such a description contains at least the following set of items:

- A set of *modes* of behavior of the component. This must include the correct (“ok”) mode and, possibly, a set of distinguished fault modes.
- A list of the interface variables for the component. In particular this includes the list of input and output ports of the component. The domain of each variable is also specified; this is the set of the qualitative values that the variable can assume.

Component: bulb		
Modes: ok, broken		
Interface variables:		
voltage: { on, off }		
light: { on, off }		
Behavior:		
mode	voltage	light
ok	on	on
*	off	off
broken	*	off

Figure 3: The model of a bulb in our model library; “*” means any value.

- A description of the behavior of the component, of the form:

$$MODE \times IV_1 \times \dots \times IV_n \quad (1)$$

where *MODE* denotes the set of modes of behavior of a component and each *IV_j* denotes the domain of an interface variable. Such a relation provides constraints on the correct and faulty behavior of the component and can be defined extensionally (e.g., by means of tables) or by means of equations, logical formulae and similar.

Thus, the description of each type of component is very general and is independent of the role that the component can assume in different devices. For example, figure 3 reports the model of a generic bulb whose behavior is defined extensionally by means of a table.

The program for compiling dependency graphs uses the model library and takes as input a description of the structure of the specific device to be diagnosed. Also in this case we make very general assumptions on the language we use. The description must include at least:

- A declaration of the components of the device. For each component, its type must be defined.

Components:
 b:battery, b1, b2, b3:bulb,
 w1, w2, w3:wire
Connections:
 < b:voltage , w1:voltage >,
 < w1:voltage , w2:voltage >,
 < w1:voltage , b1:voltage >,
 ...

Figure 4: Description of the structure of the circuit in figure 2.

- A description of the connections between the components. Two components can be connected using their interface variables; thus the description of the connections is a set of pairs of interface variables. If two components c_1 and c_2 are connected, the set will contain a pair $\langle v_1, v_2 \rangle$, where v_1 is a variable of c_1 and v_2 is a variable of c_2 . In this description one can specify, for each specific component, which of the interface variables play the role of inputs to the component and which play the role of outputs. One simple way to specify inputs and outputs is by way of ordered pairs where the first element corresponds to an output port and the second one to an input port.

Figure 4 reports the structural description of the circuit in figure 2. The first pair in the “Connections” item specifies that the output **b:voltage** is connected to **w1:voltage**.

4 Automatic synthesis of functional dependencies

In this section we describe the module for the automatic synthesis of dependency graphs (the “compiler” according to figure 1). The inputs to the module are the model library and the description of the structure of the device. The output is the graph of functional dependencies to be used by IDEA for performing diagnosis, according to the strategy discussed previously.

4.1 The duality case

Let us start by describing how the models can be derived under the restrictive assumptions that all signals are binary (i.e., their qualitative values are simply “normal” and “abnormal”) and that the duality between normal and abnormal behavior holds, i.e. that for each component in the system, the model for its component type in the library is of the following form (where we assume that O is the only output of the component and $I_1, \dots, I_n$ are the inputs on which O depends; this can be generalized to multiple outputs, defining a table for each output):

$mode$	I_1	$\dots$	I_n	O
ok	normal	$\dots$	normal	normal
ab	*	* $\dots$ *	*	abnormal
*	abnormal	* $\dots$ *	*	abnormal
$\vdots$	$\vdots$	$\vdots$	$\vdots$	abnormal
*	*	* $\dots$ *	abnormal	abnormal

This means that the output is abnormal if and only if the mode is abnormal (independently of the inputs) or one input is abnormal (independently of the mode and the other inputs).

Under these hypotheses, the inferences mentioned in the section introducing IDEA are correct.

The starting point of the compilation algorithm is the set of observable signals which correspond to the outputs of the device to be diagnosed. The algorithm builds the dependency graphs by moving backward from such signals; in this way it can determine the components and other signals on which they depend.

Let S be a set of signals; S is initialised with the set of observable signals. In particular we consider only the abnormality values for such signals. The algorithm proceeds iteratively until S is empty (i.e., until dependencies for each signal have been computed); at each step:

- it selects a signal s from S and it determines the component c having s as output;
- given the model of the component c , it determines all cases in which c can produce s as output. Two different situations may arise:

1. for every fault mode m_i of c that can generate s , a dependency arc is created from s to m_i .
2. if s can be generated by c under normal behavior (i.e., in the *ok* mode), the algorithm determines the set $\{i_1, \dots, i_k\}$ of input signals on which the output s depends. From the structural description of the device, the algorithm determines the signals $\{o_1, \dots, o_k\}$ (outputs of some other component) which are linked to $\{i_1, \dots, i_k\}$. A dependency between s and each one of the o_h is created and the signals $\{o_1, \dots, o_k\}$ are added to S .

4.2 Extended dependency graphs

As we noticed in the previous sections, the assumptions that all signals are binary and that there is a strict duality between correct and faulty behavior can be restrictive in some domains.

Let us consider a hydraulic example in which we have a pump and a tank whose generic models are reported in figure 5. The pump can have two faults: it may either pump less than the normal value or more than the normal value. The tank may have a hole or it may leak¹. In the model we distinguish different qualitative values for the interface variables. In particular, the values **low** and **high** for the output of the pump (and the input of the tank) should be intended as deviations from the normal value. Therefore, the value **low** means any value lower than normal, including zero. This is the reason why, in the behavior of the tank, when the input is **low**, **waterLevel** may be either **low** or **zero**.

The specific device we consider is formed by a pump **p1** and a tank **t1**, connected in such a way that the output of **p1** is the input to **t1** (see figure 6). The only observable signal is the level of water in **t1** (**waterLevel**).

In this example the signals are not binary and the observable effects of correct and faulty behavior of the tank depend on the input to the tank and can coincide for some particular inputs to the tank. Moreover, we may have fault masking: if the pump is **abHigh** (i.e., it provides more water than it should) and the tank is **leaking**, then the

¹Actually, this is a dynamic system and the model should relate the behavioral modes and the input to the tank to the derivative of the water level, rather than to the water level itself. Here we simplified the model abstracting these dynamics and assuming that the system is only observed in a steady state.

Component: pump

Modes: ok, abLow, abHigh

Interface variables:

out:{normal, low, high}

Behavior:

mode	out
ok	normal
abLow	low
abHigh	high

Component: tank

Modes: ok, leaking, holed

Interface variables:

in:{normal, low, high}

waterLevel:{zero, low, normal, high}

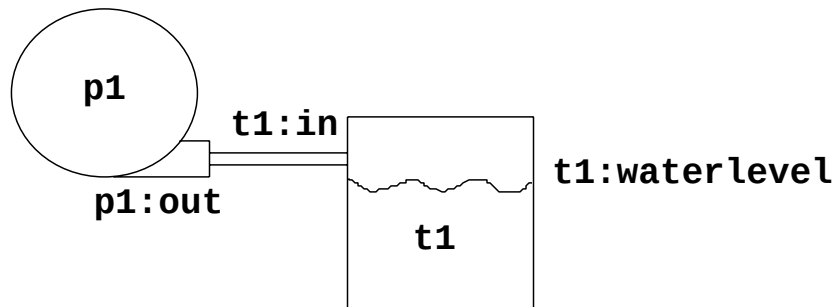
Behavior:

mode	in	waterLevel
ok	normal	normal
ok	high	high
ok	low	low or zero
leaking	low	zero
leaking	normal	low
leaking	high	normal
holed	*	zero

Figure 5: The generic models of a pump and a tank.

level of water in the tank is normal. Thus, building the dependency graphs according to the assumptions made by IDEA would not be possible in such a case.

In this section we introduce an extension of the dependency graph formalism of IDEA. The extended formalism can easily cope with the cases discussed above. Basically, the extension consists in the fact that separate dependencies have to be built for each value that a



Structural description:

Components: p1:pump; t1:tank;

Connections: { p1:out, t1:in }

Figure 6: Structural description of an hydraulic device.

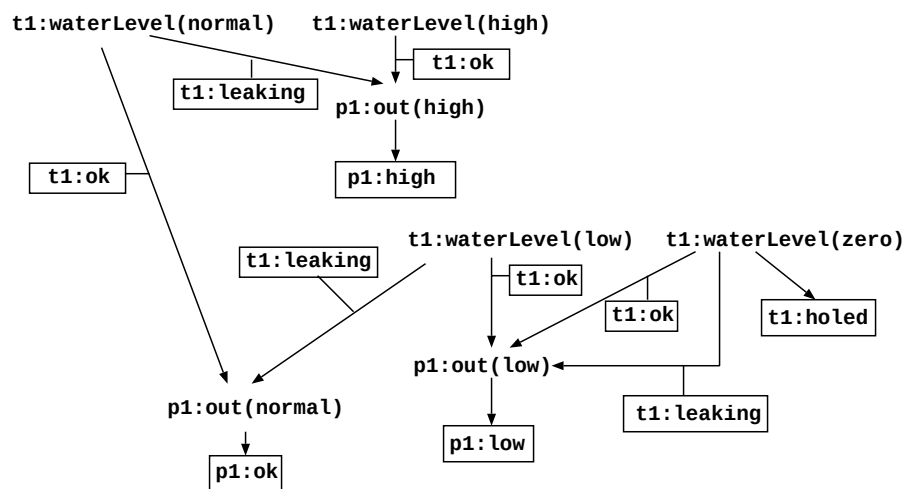


Figure 7: Extended dependency graph for the hydraulic example.

signal can assume. The dependency graph will still contain dependencies between signals and between signals and nodes corresponding to modes of behavior of components. However, arcs corresponding to dependencies between signals are labeled with the fact that a compo-

nent is in a specified behavioral mode. In particular, given a signal $s(x)$ which is output of a component c , we can have the following types of dependencies:

- a dependency from $s(x)$ to a mode m_i of c , if c can produce the output $s(x)$, regardless of the input when in mode m_i ;
- a dependency from $s(x)$ to another signal $s'(y)$, with a label m_j if c can produce the output $s(x)$, when in mode m_j and with the input $i(y)$ corresponding to the output $s'(y)$ of some other component.

For example, the extended dependency graph for the device in figure 6 is reported in figure 7.

This slightly extended form of dependency graphs can describe more general forms of behavior than the simpler ones discussed in the previous section.

A modified reasoning strategy wrt the one in IDEA should be used on extended dependency graphs. The diagnostic process starts from the nodes corresponding to the observed symptoms and moves along the dependency arcs looking for explanations of the symptoms. The dependency arcs starting from a signal s correspond to candidate explanation of s . If an arc points to a node corresponding to a mode m of a component c , then this mode is a candidate explanation of s . If an arc points to another signal s' and is labeled with the mode m' of a component c , then the union of m' and of a candidate explanation of s' is a candidate explanation of s .

For example, starting from the observation **t1:waterLevel(zero)** in figure 7, we have that **t1:holed** is a candidate explanation; other explanations can be obtained by assuming **t1:leaking** and looking for explanations of **p1:out(low)** (in this case we get only one explanation { **t1:leaking, p1:low** }); similarly, following the other arc with the label **t1:ok**, we exonerate the tank **t1** and we obtain the candidate { **p1:low** }.

The extended dependency graphs discussed in this section can be easily compiled by the following variation of the algorithm introduced in the previous section:

- all possible values for observable signals have to be considered; this means that the set S of signals is formed by all possible instances of observable signals;

- in case a signal s can be produced by a component c in mode m_i , regardless of the inputs, then we create a dependency from s to m_i ;
- in case a signal s can be produced by a component c in mode m_i with input signals $\{i_1, \dots, i_k\}$, which correspond to output signals $\{o_1, \dots, o_k\}$ of some other components, then we create a dependency from s to $\{o_1, \dots, o_k\}$ with the label $m_i(c)$. Then $\{o_1, \dots, o_k\}$ have to be added to S .

4.3 Simplifications under the single-fault assumption

The resulting dependency graph can be further postprocessed to be able to deal with non-binary observations, but restricting the diagnostic strategy to reason on single faults. In this way we achieve an intermediate result between dealing only with simple systems that satisfy the duality condition discussed above, and dealing with systems with arbitrarily complex interactions between faults.

Interestingly enough, the simplified graphs that result from the postprocessing resemble closely the structure of dependency graphs in IDEA.

In order to define the postprocessing, notice that a path in the original graph identifies a mode assignment, i.e. an assignment of correct or faulty modes to components, such that hypothesizing such mode assignment implies the values for signals along the path. The goal is then to reconstruct a graph with a similar property, but without labels on the edges, therefore with only paths corresponding to single faults. The postprocessing can then be defined as follows:

- initial nodes (i.e. nodes with no entering edge) with normal observations are ignored.
- A label L with a mode for a component is moved to the node to which the edge points; if a label has already been put, a copy of the node is created. From the node, the label L is moved along the edges, except in case of an edge E labeled with a fault mode if L also contains a faulty mode²; in this case the edge entering the node where E comes out is removed. The label moving is continued until a final node is found with a mode of a component. If such a mode is an ok mode and L contains a

²This would correspond to a multiple fault.

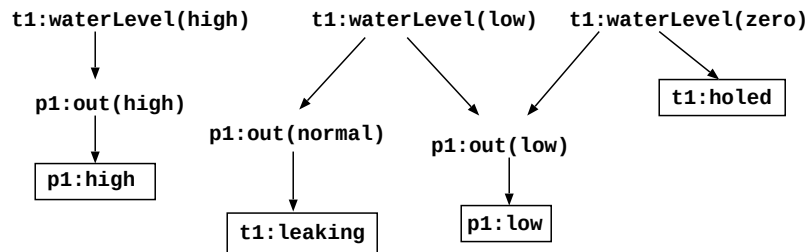


Figure 8: Simplified dependency graph for the hydraulic example.

faulty mode, the mode in L is put in place of the ok mode. If both modes are faulty modes, the last arc is removed³.

- Finally, labels are removed from nodes.

The result of this postprocessing on the dependency graph in figure 7 is shown in figure 8. Such graphs resemble more closely a decision-tree-like structure and can therefore be used more efficiently than general dependency graphs.

5 Conclusions

In this paper we presented an approach to the automatic synthesis of functional dependency graphs from a library of reusable component models and a description of the structure of the device to be diagnosed. The graphs we generate can be given as input to the IDEA diagnostic system. In this way we showed that the process of generating the models used by IDEA can be significantly simplified.

In the paper we pointed out the assumptions that simplify the development of diagnostic applications in IDEA and we introduced an extended notion of functional dependency graph which can overcome such limitations. Moreover, we showed that also these extended graphs can be generated automatically starting from a library of reusable component models, and that they can be simplified under the single-fault assumption.

The compiler has been implemented in C++; we plan to experiment the approach to derive functional dependencies for electrical/electronic and hydraulic devices.

³This again corresponds to a multiple fault.

6 Acknowledgements

The authors are grateful to Fulvio Cascio and Marialuisa Sanseverino for their contribution to the work presented in this paper.

References

- [1] F. Cascio and M. Sanseverino. IDEA (Integrated Diagnostic Expert Assistant) model-based diagnosis in the the car repair centers. *IEEE Expert*, 12(6), 1997. A preliminary version appeared in *Proc. DX 96, 7th Int. Workshop on Principles of Diagnosis*, 1996.
- [2] L. Console, D. Theseider Dupré, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.
- [3] J. de Kleer and J.S. Brown. A qualitative pyhsics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.
- [4] W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.
- [5] D. Poole. Normality and faults in logic-based diagnosis. In *Proc. 11th IJCAI*, pages 1304–1310, Detroit, 1989.
- [6] C. Price, D. Pugh, M. Wilson, and N. Snooke. The Flame system: automating electrical Failure Modes and Effects Analysis (FMEA). In *Proc. Ann. Reliability and Maintainability Symposium*, pages 90–95, 1995.
- [7] O. Raiman. Diagnosis as a trial: The alibi principle. In *Proc. Int. Workshop on Model-based Diagnosis*, Paris, 1988.
- [8] P. Struss and O. Dressler. Physical negation - integrating fault models into the general diagnostic engine. In *Proc. 11th IJCAI*, pages 1318–1323, Detroit, 1989.