AN INTEGRATED, DEEP-SHALLOW EXPERT SYSTEM FOR
MULTI-LEVEL DIAGNOSIS OF DYNAMIC SYSTEMS

by

Ameen Abu-Hanna and Yaron Gold

Technical Report #504
March 1988
AN INTEGRATED, DEEP-SHALLOW EXPERT SYSTEM
FOR MULTI-LEVEL DIAGNOSIS OF DYNAMIC SYSTEMS.

by

Ameen Abu-Hanna

and

Yaron I. Gold

Computer Science Department
Technion, Israel Institute of Technology
Haifa 32000, Israel

ABSTRACT

An integrated expert system architecture and strategy for diagnosis of dynamic systems with feedback loops and synchronous or asynchronous state transitions is presented. The dynamic system under diagnosis is modeled using structural and behavioral representations in multiple levels of abstraction.

The diagnosis process integrates shallow and deep expertise. It recursively navigates through the structural hierarchy, and at each level tries the shallow expertise first. If it fails it switches to deep, simulation based expertise.

A multilevel simulator assists the diagnosis process in verification and elimination of hypothesized suspects. The simulator shifts on demand through several levels, from coarse qualitative modeling to detailed quantitative modeling. Knowledge of pathological behavior (failure modes) of lower level components is incorporated in the simulator.

Learning is exhibited as deep-to-shallow expertise transfer, as well as up the abstraction levels of the simulator itself, to improve future efficiency. In addition, knowledge about pathological behavior can be used for off-line training by artificially generating new cases for diagnosis.
1. INTRODUCTION

AI Diagnostic systems were traditionally based on so called shallow knowledge [2,9,12]. Such knowledge is meant to capture the diagnostic methodologies performed by experts in their domains. It represents possible shortcuts that an expert may perform, when faced with a complaint of system malfunction, to locate the problem source (e.g. associating symptoms to possible faults in the diagnosed system). Shallow expertise in a specific domain often results in impressive performance when cases are within the narrow scope of the coded expertise. Problems outside that scope may often result in steep degradation of performance. Also, due to the lack of "understanding" knowledge of the system under diagnosis, such systems can provide only shallow and limited explanations about how a solution was arrived at.

Recently, AI diagnostic systems researchers proposed diagnosis from "first principles" [3,6,7,8,11,13,15], utilizing deep knowledge of the diagnosed system itself beside the knowledge about diagnosis methodologies. Deep knowledge may contain models of the systems under diagnosis, with granularities that depend on the systems' properties. The use of deep knowledge, though generally less efficient, can greatly increase robustness as well as support deeper explanation of the system's processes.

A theory of diagnosis from first principles is proposed by Reiter [13], defining and formalizing the diagnostic task. Poole's et al. THEORIST [11] is a programming tool for default reasoning and diagnosis, viewing them as processes of theory formation. Both Reiter and Poole point out the close relation between diagnosis and non-monotonic logic.

In [3], Davis discusses troubleshooting of digital combinational circuits using structural and functional descriptions of the circuit, along with layers of assumptions that are relaxed as needed. The approach in [3] uses a "constraint suspension" technique which requires the ability to compute the function and inverse of each sub-component in order to perform efficiently. Such a requirement is hardly realistic in dynamic systems. The same is true for the suspect computation and test generation technique used in Genesereth's Diagnostic Assistance Reference Tool (DART [7]) that also exploits the design description of the diagnosed system. In [8], Hamscher and Davis discuss the difficulty of diagnosing digital circuits with state. They propose a way in which the "constraint suspension" technique in [3] may be extended to include clocked (synchronous) circuits with state by using temporal abstraction of behavior. No treatment is given in [8] to asynchronous systems.

Yamada and Motoda [15] propose a method for diagnosing dynamic systems (specifically - power plants) using a (single level) description of the diagnosed system, without structural and behavioral abstraction. As demonstrated later, efficient diagnosis of most complex dynamic systems requires introducing one or more levels of such abstraction.

Human experts utilize both deep and shallow expertise. Generally the more efficient shallow expertise is tried first, and if it fails (mainly when presented with more esoteric cases) deeper, more comprehensive understanding of first principles is called upon. The experience gained can be used to augment and refine shallow expertise for future more efficient handling of similar problems.

In [6], Fink and Lusth present what they call the Integrated Diagnostic Model (IDM) which integrates a shallow expert and a deep expert for diagnosis of electrical and mechanical systems. The shallow expert relates symptoms to possible causes of the malfunction, and supplies appropriate testing information. If it fails, the system turns to the deep expert, whose knowledge base contains representations of structure and functionality of the diagnosed system. A component's functionality belongs to a predefined set of generic "functional primitives" (e.g., reservoir, regulator, transformer) that make it possible to qualitatively simulate the device under diagnosis, using "second principles". The deep expert traverses the structural graph through the individual components, providing the user with information for testing the components. A given component is found faulty if all those feeding it are OK, while it's own output is not. Qualitative modeling (see work by De Kleer and Brown in [5]) alone is not well developed for modeling devices of great complexity as discussed by Shragger et al. in [14]. The deep inference used in [6] uses a single level of qualitative, coarse knowledge. Also, feedback loops are not treated in [6].
In this paper an attempt is made to integrate shallow and deep knowledge, each with its own inference engine, in order to diagnose the malfunction of dynamic (synchronous or asynchronous) systems with state and feedback loops. Deep Knowledge consists of structural and behavioral representations of the diagnosed system, in multiple levels of abstraction. A multilevel simulator that shifts on demand from coarse qualitative modeling to detailed quantitative modeling, assists the diagnosis process in verification and elimination of hypothesized suspects. Knowledge of pathological behavior (failure modes) of lower level components is incorporated in the simulator. Learning is exhibited as deep-to-shallow expertise transfer, as well as up the abstraction levels of the simulator itself, to improve its efficiency in subsequent cases.

The proposed architecture and diagnosis strategy are outlined in the next section. In Section 3, some aspects of the architecture and knowledge representation are elaborated on. The diagnosis strategy, integrating the two experts, is discussed in some detail and illustrated in Section 4. A conclusion is given in Section 5.

2. SYSTEM OVERVIEW

The architecture and strategy outlined here are for the diagnosis of systems which include, among others, the following properties:

(a) The system under diagnosis may be dynamic, i.e. observable outputs may vary with time.
(b) Components may have internal states (i.e., an output value may depend on the components history in addition to the current input values).
(c) State transitions may be asynchronous (i.e., the system need not be clocked).
(d) Components may have varying propagation delay, depending on their inputs or state (e.g., the delay of piston stroke in an internal combustion engine is affected by heat, mixture ratio, compression ratio, etc.).
(e) Feedback loops are allowed at all structural levels.

The architecture described in this paper consists of two layers: a Shallow Expert (SE) which contains traditional symptom-to-fault associations along with a hypothesis-testing mechanism, and a Deep Expert (DE) in which a hierarchical model of the diagnosed system is represented in multiple levels of structural and behavioral abstraction. While structural abstraction reflects the structural hierarchical design of the diagnosed system, the behavioral abstraction constitutes a hierarchy of properties assigned to components. Special attention is given to teleological classification of properties into responsibilities on one hand and other properties that are meaningful in the diagnostic process, on the other hand.

Integration between the DE and the SE layers is manifested in two ways:

(a) The diagnosis process recursively navigates through the levels of the system’s structural hierarchy, and at each level switches to deep, simulation-based expertise when the current shallow expertise is insufficient for solving the particular level.
(b) As the deep expertise is being utilized, new shallow expertise (i.e., direct symptom-to-fault association) is automatically constructed, augmenting the SE layer for future more efficient diagnosis.

Suspect computation in the SE is guided by links leading from the symptoms to suspects. The DE suggests suspects by following behavioral nets, implied by structure, of the system under diagnosis (which, for abbreviation will be called the d_system). Assumptions of single fault (SFA) and non-intermitency (NIA) are used when tests are applied to check the consistency of suspected components.
The knowledge representation in the DE consists of two hierarchical descriptions:

1. **Structural hierarchy** - components are described by modules and their interconnection. Modules are constructed from other modules and/or primitives.

2. **Behavioral abstraction** - behavior of components at any level of the structural hierarchy is abstracted, from quantitative and exact temporal statements of input and output values and the relationship between them, to qualitative, lumped descriptions of properties and responsibilities of components, defining causal nets which can also be used for causal explanations (see De Kleer in [4]).

A multi-level simulator that can shift, if needed, from coarse qualitative modeling, through more refined qualitative modeling, down to detailed quantitative modeling, assists the diagnosis process in two significant ways:

(a) generation of observable normal behavior for discrepancy analysis and suspect elimination, and

(b) controlled generation of observable faulty behavior for verifying hypotheses or suspects when access to a suspected component is costly and the hypothesis must be tested by checking nearby components that are affected by it. This is made possible by including knowledge about pathological behavior (failure modes) in lower level components.

In addition to the SFA and NIA, the following assumptions are also needed for increasing the expert system efficiency:

(a) The design of the diagnosed system naturally lends itself to hierarchical structural representation.

(b) Components can be identified with specific responsibilities in the d_system's design. This determines the qualitative behavioral levels.

(c) Important attributes are generally observable. This helps probing the d_system in order to discriminate between competing hypotheses. A cost of observability is associated with each attribute.

(d) The d_system is validly designed. A valid design does not allow an input to change as long as the old input has not yet affected the output. This assumption is used when encoding newly acquired qualitative descriptions of behavior.

(e) Pathological behavior patterns (modes of failure) of some "hard to reach" primitives are known.

In addition to deep-to-shallow expertise transfer, higher levels of behavioral abstraction are automatically constructed during operation of the simulator. The ground is set for this type of learning, by having the knowledge engineer predefine attributes that the system should watch for. Finally, off-line training of all layers can be realized by artificially generating new cases, using knowledge about pathological behavior of primitive components.
3. ARCHITECTURE AND KNOWLEDGE REPRESENTATION

As stated in the introduction, two different knowledge types are supported: deep and shallow. They are different in character and require different inference engines. In this section we discuss in some detail a few aspects of the two layers. PROLOG is used to implement the representation language and inference engines of the two layers, exploiting PROLOG's declarative style, pattern matching, and backtracking capabilities.

3.1 Representation Of Deep Knowledge

The knowledge associated with the DE consists of a model of the diagnosed system. It also provides the ground for constructing the shallow expertise and communicating with it. The d_system model includes representation of its structure (in terms of components, modules, sub-modules and primitives, and their interconnection), and behavior (in terms of input and output descriptions and relationships, responsibilities, and other observable properties).

3.1.1 Structure Representation

The DE is provided with a library of parameterized component types. Components are represented in terms of specific instances of such types (for example one may define a specific 3-input AND_gate using the library's generic N-input AND_gate type) and their interconnections. A type may be a module or a primitive. While a module is constructed from lower level components (other modules and/or primitives), a primitive is the basic structural unit. Structural hierarchy is a key feature of the proposed architecture. This hierarchical abstraction guides the diagnosis process, and thus determines its efficiency.

For the purpose of structure representation, class-concepts such as primitive, module, part, port, conduit and connection are defined. These class-concepts are used as predicate names in the representation of type structures (representation syntax is fairly similar to the one used in [1]). For example, the following represents the structure of a primitive type inverter with a single boolean input port, \( i1 \), and a single boolean output port, \( o1 \):

\[
\text{primitive( inverter, [i1], [o1])}.
\]
\[
\text{port( inverter, i1, input, bool)}.\]
\[
\text{port( inverter, o1, output, bool)}.\]

The following is an example of a module which represents the circuit in Figure 1:

\[
\text{module( box, [in1], [out1])}.
\]
\[
\text{part_of( box, inverter1, inverter_type)}.\]
\[
\text{part_of( box, nand_gate1, nand_type)}.\]
\[
\text{map_in( box, in1, [i1, inverter1])}.\]
\[
\text{map_in( box, in1, [i2, nand_gate1])}.\]
\[
\text{map_out( box, out1, [o1, nand_gate1])}.\]
\[
\text{connection( box, wire, [o1, inverter1], [i1, nand_gate1], unidirectional)}.\]
The predicate module indicates a non-primitive component. The box module consists of two parts, an inverter and a nand gate. The two map in clauses indicate that the input \( in1 \) of box is connected to inputs \( il \) of inverter1 and \( i2 \) of nand gate1. map out indicates that \( o1 \) of nand gate1 is connected to the output \( out1 \) of box. connection indicates connections internal to box, namely that \( o1 \) of inverter1 and \( i2 \) of nand gate1 are wired together, with intended unidirectional information flow from source (\( o1 \) of inverter1) to destination (\( i2 \) of nand gate1).

The box module of Figure 1 can itself be a building block in a more complex module.

### 3.1.2 Behavior Representation

Representation of behavioral knowledge is used by the simulator. This kind of knowledge reflects the behavior of the d_system in multiple levels, differing in detail granularity ranging from coarse qualitative to fine quantitative descriptions.

If there is no explicit knowledge of a given high-level property (which must be predefined) then the simulator can compute it, if needed, using lower level and quantitative knowledge.

Sub-components of a module in the structural hierarchy are generally assigned sub-functions which together perform the overall function of the module. Attributes for describing and abstracting behavior are chosen based on teleology. The multilevel hierarchy of qualitative and quantitative behavioral representation levels is a key factor in system performance.

#### Quantitative behavioral Knowledge

In general, computing behavior constitutes of computing outputs and new states. In our prototype system, the predicates \( outEQN \) and \( stEQN \) are used to compute outputs and states respectively.

For example, the following clause is the quantitative representation of the input to output relation of the inverter type represented structurally earlier:

\[
outEQN(1, \text{inverter}, [o1], [i1], 0.5, \text{if}(i1 == \text{HI}, \text{LO}, \text{HI})).
\]

meaning: output equation number 1 of an inverter computes the output \( o1 \) from the input \( il \) (in general a primitive may have more than one output, hence more equations may be needed). Output is \( \text{LO} \) if \( il \) is \( \text{HI} \), otherwise it is \( \text{HI} \). Input to output propagation delay is 0.5 time units. In this example the '0.5' and \( \text{if}(i1==\text{HI},\text{LO},\text{HI}) \) \(^1\) are the (rather simple) expressions describing the delay time and output

\(^1\) Value "returned" by expression 'if(A,B,C)' is value of expression B if A is true, otherwise it is value of expression C.
respectively. These expressions can be quite complex, using a specially designed description language.

State variables of a given component are defined by the state predicate. For example:

\[
\text{state( latch, st1, bool).}
\]
\[
\text{state( latch, st2, bool).}
\]
defines \(st1\) and \(st2\) as boolean states of a latch and

\[
\text{stEQN( latch, [st1, st2], if( latch_enable == HI, setlist([il, i2]), no_change)).}
\]

indicates that if the input \(latch\_enable\) is HI then the two boolean state variables \(st1\) and \(st2\) get values of the inputs \(i1\) and \(i2\) respectively, otherwise \(st1\) and \(st2\) stay unchanged.

**Qualitative Behavioral Knowledge**

The various levels of qualitative behavior are abstractions of either quantitative or lower level qualitative behavior. One form of such abstraction is quantization: where a continuous spectrum of property values can be observed, quantization into discrete qualitative intervals is performed. Quantization facilitates learning about relations between components’ behavior patterns, due to the manageable number of different value combinations. It also provides a more convenient language for interaction with the user. Another kind of qualitative abstraction is the notion of qualitative component states which, when defined at some behavioral level, abstract several lower level sub-component states.

A third kind of qualitative abstraction which may be performed in multiple levels, is designed to capture important properties of the behavior while disregarding others. In [8] we saw a simple form of this kind of abstraction, which reflects a difference in temporal granularities (a coarser behavioral level is obtained by less frequent sampling of the signal). A more sophisticated form is used in our system, where the function or responsibility is emphasized. Function is the intention behind behavior. For example, the intention of driving an electric current through a wound wire may be to emit light (like in an electric light bulb). But it could be intended to emit heat (like in a heater) and then light is just a side-effect. In another case the designer could be interested in the magnetic field generated by the current. So in choosing attributes for describing qualitative behavior one should (1) identify responsibilities assigned to sub-components such that together they perform the overall responsibility of the more complex module, and (2) define how such a responsibility can be computed3.

As an example, consider the circuit in Figure 1. The modeled circuit was referred to as box. In box there are two primitives, namely an inverter and a nand gate, both are modeled structurally and behaviorally (by primitive and outEQN respectively). Note that the steady state of the output \(out1\) is HI, but due to the inverter’s propagation delay, a negative pulse is produced at \(out1\), whenever the input \(in1\) changes from LO to HI. The generation of the pulse is a property of the box module. If this pulse generation is teleological i.e. intended by the designer and without it the system would be considered faulty, we classify this property as a responsibility of box and a (boolean) responsibility named pulse generated can be defined for module box. It’s value is determined by the simulator when it goes (if the need arises) through the deep quantitative knowledge. Each time it “fires” the outEQN rule of the nand gate, and thus the output of the box module is updated with respect to time or value, a

---

2 The concept responsibility was introduced by Milne [10] who discusses analog electrical circuit diagnosis based on a “theory of responsibilities”. We use the words ‘responsibility’ and ‘function’ alternately, meaning the same.

3 In our system the relevant qualitative properties and responsibilities are not derived automatically, certainly a tough task to accomplish, but are predefined by the human expert or knowledge engineer who determines how to calculate them from lower level values.
properties/responsibilities updating routine is invoked. This routine activates a property daemon clause which indicates if and how the new calculated output (value and time) affects each predefined property of the corresponding component. The following is quite a simple example of a property daemon:

```
property_daemon([pulse_generated at out1 of box],
cond_exp = change_in(out1),
update_exp = if(out1 == LO, TRUE, FALSE)).
```

indicating that whenever expression cond_exp evaluates as TRUE, then value of the pulse_generated property, assigned to output out1 of box is updated according to the update_exp expression. In this example, cond_exp is TRUE when a change occurs in the new calculated value of out1. According to update_exp, pulse_generated is TRUE if new value of out1 is LO (Suppose that out1 is initially LO). In this example, many parameters were omitted from the property daemon clause such as "local" variables of the clause, which are usually used in evaluating update_exp and "remembering" intermediate values, their updating expressions, time_stamp of property, all initializing values, and other possible properties assigned to out1 which rely on pulse_generated.

If simulation determines that the responsibility of generating a pulse holds in some case then failing to observe such a pulse in the d_system violates this expectation, suggesting the responsibility holder (in this case the box module) as a suspect of causing the malfunction. Note that properties other than responsibilities (i.e. "side effects") may also be meaningful in diagnosis as indicators of the component functionality, comparing their simulated values with the ones actually observed in the d_system. A given component's responsibility may affect other nearby components' responsibilities. Such relations are represented in causal nets discussed later in this section.

**Levels Of Qualitative Properties**

Figure 2 describes a simple Burglar-Alarm system that produces audio (intermittent tone beeps) and visual (flashing light) alarm signals. This system is used to illustrate concepts such as levels differing in granularity, and in the next section it is used as an example for illustrating the diagnosis process.
Behavior of the Burglar Alarm circuit is described (qualitatively) as follows: As long as the enable_sw is in the right hand position, the output of e_box is forced at HI and one_shot can not be triggered. The alarm system is enabled by moving the enable_sw to its left hand position. Closing the trig_sw (e.g., by a burglar) will cause a short LO (negative) pulse at the input Il of one_shot, causing it to produce a fixed-width HI (positive) pulse which is inverted by inv2, and fed back into the e_box input. The trailing edge of this negative pulse is detected by e_box and causes it to produce a short negative pulse which is fed to the input of one_shot. As a result a pulse train is generated and fed into the modulator and TFF inputs as long as the alarm is enabled. The modulator feeds the amplification transistor T1 with the Audio Frequency (AF) signal, modulated by the pulses produced by one_shot. So the speaker generates periodic beeps at a rate equal to the one_shot pulse rate and at the AF pitch. The output of the T-Flip_Flop (TFF) module causes the the Light Emitting Diode (LED) to flash at half the pulse rate produced by the one_shot module. As designers we expect the one_shot module output to generate pulses, so a boolean property named pulse_train_generated with its obvious meaning that more than one pulse was observed during a given period of time, could be associated with that output. A FALSE value of this property while the circuit is enabled indicates a malfunction, detected at one_shot.

To refine this, we add a property named pulse_rate. To get less abstract information we propose the pulse_width property and even more refined information is obtained by introducing a pulse_shape or wave_form property (assuming for this example that we have deep quantitative description of

Figure 2 - Wiring diagram of the Burglar Alarm circuit.

---

4 When enabled, the e_box module is behaviorally equivalent to the box module in Figure 1, e_box stands for box with an enable input.

5 The pulse width is about $1.1 \times R4 \times C1$ seconds, $(\text{In } 3) \times R4 \times C1)$, produced by the 555chip which is wired as a mono_stable device.
transistors, resistors, capacitors etc.), to represent the shape of the waveform describing the pulse. The output signal of T1 (point c) can be plotted against time as in Figure 3.

![Figure 3 - Output waveform at point c.](image)

The following are possible properties defined for the output signal at the output (c) of T1:
- "envelope" properties: envelope_amplitude, envelope_is_periodic, envelope_frequency, and even
- Attack_time, Sustain_time and Decay_time; AF properties: AF_exists, AF_freq, AF_amplitude.

Properties of the speaker output might include beep_heard, beep_volume, beep_pitch, beep_rate, etc. The envelope_amplitude at (c) of T1 affects beep_volume, envelope_frequency affects beep_rate and the LED's light intermittency. Similarly the attack, sustain and decay times contribute to the speaker sound quality.

An implicit causal net (CN) of the d_system is contained in its structural and behavioral representations. Each node of CN represents an output point of some component. A directed edge from node i to node j means that the former directly affects the latter.

A causal sub-net (CSN) at a given structural level is a sub-net of CN, in which all nodes represent outputs of components at that same structural level. A symptom (SYM) is a discrepancy (e.g. indicated by "wrong", "TOO LOW", etc.) between the observed and the expected values of an attribute at any (structural or behavioral) level of abstraction. A causal path (CP) is a directed path in a causal sub-net. A causal sub-net of a given symptom SYM (which is a node in CSN) is the collection of all causal paths to SYM. Attributes and values of various behavioral levels are associated with each node of a causal-net. The appropriate level is chosen by the inference engine according to need. Values are determined during the course of simulation. Not all attributes are relevant at all behavioral levels.

An example of a CSN of the circuit of Figure 2 at the d_system's structural level, showing attributes relevant to the highest behavioral level, is described in Figure 4. Property names are shown inside the circles, where corresponding structural components are shown outside the property circles. The fact that the output of trig_mod is irrelevant at this level is indicated by drawing it with thinner lines, and not filling in an attribute name.

---

4. We say that output i of module A directly affects output j of module B if modules A and B are at the same structural level (i.e., have a common father in the structural tree), and output i is connected to an input of module B that may affect output j.
Figure 4 - Causal net at $d_{\text{system}}$ level, attributes at coarsest behavioral level.

In Figure 5 four partial instances are illustrated with properties for $enab_{\text{mod}}$, $e_{\text{box}}$, $one_{\text{shot}}$ and $inv2$ of the same causal net scheme, at increasing refinement levels. In these examples property values are instantiated to represent given (normal) behavior. Figure 5 is also referred to in the diagnosis case examples of Section 4.

---

7 The part enclosed by dashed lines in Figure 4.
Nets similar to those presented in Figures 4 and 5 define a qualitative behavior schema based on design (structure and intention). Learning about qualitative behavior is facilitated by recording (through deep knowledge simulation) the specific property values (mostly qualitative) at points on boundaries of modules. For example on the boundaries of "super-module" d_system a relation could be expressed as:

```
component_name : [d_system]
Inputs : (AF = TRUE, enable_switch_position = LEFT)
States : (trigger_state = TRIGGER_AFTER_ENABLE)
cause
(beeps @speaker = TRUE, flashing_light @LED = TRUE)
```

Causal sub-nets like those in Figures 4 and 5 are given for each sub-module. For example, zooming into module one_shot by going down in the structural hierarchy results in observing similar behavioral associations between the sub_components of the module.

In addition to providing greater efficiency by allowing "top-down" diagnosis, behavioral abstraction also eliminates false detections of "faults" due to insignificant discrepancies in fine details between outputs observed at the d_system and those computed by deep quantitative simulation. For example in the particular configuration in Figure 2, the pulse (PLS) produced by the e_box module is intended to trigger the one_shot module. As long as the width of PLS varies within some tolerance limits, its actual value does not affect the proper function of the whole system. In diagnosis, pulse generation is checked first, and if not found no further questions to the user about the pulse width are presented at that level\(^8\). Also, at the higher levels a dynamic module may be treated as quasi-static (the notion of time is hidden in properties such as frequency), requiring simpler diagnosis methods.

---

\(^8\) It is possible that the pulse width is too long and is actually causing the complained malfunction. This will be detected at a more refined level but only after all components are dealt with at the more abstract level.
3.2 Representation of Shallow Knowledge

The role of the shallow expert (SE) is to suggest suspects and test their consistency, using direct links from observed symptoms. A suspected component of an observed symptom SYM is a component which, when faulty, may cause SYM. A most specific suspected property (MSP) of SYM is a property assigned to a suspected component of SYM at the lowest relevant structural level, and which has at least once in the past been encountered as a possible cause for SYM. The SE uses causal nets, similar to those described with relation to the DE, with the addition of direct links to MSPs. An MSP can be reached via a direct link from each node that represents an output of an embedding component manifesting a symptom and is on the causal path from MSP to SYM. Each link is associated with an occurrence number (OCN), acting as a certainty measure reflecting the number of times MSP was confirmed as the cause of SYM, and with a context indicating the inputs and states of the module at the time MSP was confirmed. The property values used by the SE are faulty values, as recorded when the specific SYM to MSP link was created (as a result of learning). At high behavioral levels, values reflect the discrepancy (e.g. TOO LOW, WRONG, etc.). The ovals in Figure 6, which are on the path from MSP to SYM, represent faulty values of properties, that are expected, due to the faulty MSP, at component outputs (where 'components' are the direct sub-components of the module manifesting SYM). For example, the diagram in Figure 6 indicates that "TOO HIGH capacitance" of C1 is an MSP of the symptom "TOO LOW flashing_rate" of LED at the d_system level, and also of the symptom "TOO LOW pulse_rate" at the one_shot's level. The leftmost oval represents the "input" property value of one_shot which is the output property of e_box as was observed when capacitance of C1 was confirmed as TOO HIGH (note that due to feedback loop in this case it is also affected by the MSP). Each property is associated with test information, including some measure of the test cost. This is represented in Figure 6 by the arrows pointing to the component outputs to be tested, (the dotted rectangles represent components taken from the structural representation of the DE with dashed arrows representing flow of information.

![Causal net of symptom TOO LOW flashing_rate at LED related to capacitance of C1.](image)

Shallow knowledge is constructed as a result of learning. Such learning takes place under two circumstances:

(a) After completion of a successful diagnosis, i.e. locating the faulty component, by constructing a new net or updating the OCNs.

---

9 They may in fact be implemented as a subset of the DB's causal nets.
10 That is, antecedent components in the structure tree.
(b) After simulation with a faulty component's behavior, (e.g. "capacitance of C5 is LOWER than normal") and observing the violated properties (e.g. HIGHER flashing rate of LED). Here the created link is attached with OCN of zero (that could be changed to 1 if the faulty component is eventually found to be the cause of anomaly).

4. THE DIAGNOSIS PROCESS

4.1 Outline

The goal of the diagnosis process is to locate the the minimum number of most specific consistent suspect components. Recall that a suspect component is a component that the DE or SE hypothesize as a possible cause of the complaint symptom(s). A suspect component is said to be consistent if it is not ruled out by current knowledge.

The diagnosis process is driven recursively, guided by the structural hierarchy, starting with the "super module" d_system, and zooming in through module boundaries until the lowest possible level faulty component is identified. Both shallow and deep diagnosis use the concepts of causal sub-nets and paths described in the previous section.

4.2 Shallow Inference

In SE the causal paths from a hypothesized MSP to the observed symptom SYM (in the complaint behavioral level), along with "input" properties, are used to identify the points at which outputs must be tested for expected faulty values, in order to verify (or rule out) the hypothesis. The union of nodes along all these paths plus "input" nodes that affect MSP is called the test-set of MSP as cause of SYM. For example, in Figure 6 the test-set of MSP "TOO HIGH capacitance at C1" as cause of SYM "TOO LOW flashing rate at LED" consists of pulse_rate@TFF, pulse_rate@one_shot and the "input" property pulse_rate@e_box.

If more than one symptom is observed, SFA implies that only the intersection of the corresponding MSPs need to be checked for consistency. The first test-set considered is the one corresponding to the MSP with the highest OCN value associated with an incoming link. The order in which nodes in a given test-set are selected for test is determined so as to minimize the cost of MSP verification (or elimination). If any test fails (i.e. the expected faulty output was not observed) then the MSP is ruled out. Note that each test-set, like CSN, is associated with a given structural level. If a test-set at the current level succeeds, i.e. a consistent component embedding the one with the MSP is identified (for example one_shot in Figure 6), then the process is repeated recursively until the identified consistent component is the one with the MSP11 (e.g. capacitor C1 in Figure 6). Test results at all levels are made available on a blackboard for later use by the system. Note that cycles in the net (resulting from structural feedback) imply that any test obtained from a point in the cycle gives information about all members of that cycle. As a consequence, MSPs in a cycle, with the same context, where one member is found consistent, are all consistent. If the SE fails to find a faulty component (i.e. all possible hypotheses are eliminated), then the DE is called upon.

11 Note too that in this case it is not necessary to perform all tests implied by the suspect list, as if any test (other than "input" property test) succeeds, all affected properties on the causal path from the tested property to SYM need not be verified as they are bound to manifest faulty valued outputs.
4.3 Deep Inference

In DE the intersection of all causal sub-nets of the symptoms at a given structural level constitutes the suspect-list containing the outputs that must be checked. Expected normal values of these outputs (derived from simulation at the appropriate behavioral level), are interactively checked against values observed at the d-system.

Direct testing of points that lie deep in the structural hierarchy may be quite costly. For testing such points, knowledge of "failure modes" of primitive and lower level components is used. The d-system may be simulated under various hypothesized faults of such components, in order to provide the resulting faulty behavior at more "testable" (i.e. less costly) points. The simulator always tries to use rules that apply directly to the behavior at the level of the simulated module. However, if no direct qualitative rules apply for a given module, simulation may have to start from low level quantitative behavioral rules of its sub-components.

While the simulator is running, a history of property values of each component is recorded with respect to time. This history is used for deriving higher level properties and in resolving feedback loops as well as for constructing higher level behavioral rules for future use.

For illustration of test cost criteria and feedback loop handling, consider Figure 8 where a specific causal net at some behavioral level is shown. SYM is the symptom, S1 to S5 (assigned to M1 to M5 respectively), are the so far suspect properties constituting the suspect list of SYM, NS1 and NS2 are non suspected properties (assigned to MOD1 and MOD2 respectively) which do not appear in suspect computation, but are considered in test cost computation. M1 to M5, MOD1 and MOD2 are direct sub-components of the module manifesting SYM. The DE reasons as follows: it recognizes the cycle in the net (i.e. S2, S3 and S4) and introduces an imaginary module encompassing the cycle (shown as a dashed square labeled IM). This results in a loopless path, defined by S1, IM and S5 leading to SYM, on which 3 test points, labeled a, b and c are considered, which imply test-a, test-b and test-c respectively. We say that test-i succeeds if the expected values of properties, which eventually may affect SYM, at structural point i match the observed values at that point, otherwise the test is said to fail. The DE's role is to single out a component (out of M1, IM and M5) by applying tests. For example, if test-b succeeds then M5 is singled out (this is implied by the SFA), otherwise (e.g. M3 manifesting faulty S3) S3 ceases to be consistent, and test-a is needed to discriminate between M1 and M3.

\[^{12}\text{The order of property testing at a structural point P submit to the behavioral hierarchy where the coarsest level is dealt with first and only if succeeds, lower relevant levels at P are considered.}\]
IM. Note that an observed faulty value of NS2 results in concluding that S3 is faulty\textsuperscript{13} (same applies to NS1 and S1). Hence, the presence of non-suspect properties affected by suspect properties suggests them as applicable test points which can minimize the test cost of test-\textit{i}. Suppose test-\textit{b} fails (e.g. NS2 was found faulty) and test-\textit{a} succeeds, then IM is singled out and diagnosis is applied to IM with S3 as the new symptom. The resolution of the present cycle may need capturing the fault before it propagates through the whole cycle to stay (and then, all participants of the cycle manifest faulty outputs). When a behavioral level does not offer enough discriminatory information, a more refined one is used\textsuperscript{14}, until a direct sub-component can be accused. In some cases e.g. when all cycle-members manifest faulty outputs as a result of some faulty component CCOMP, "opening" of the cycle is required for later deep diagnosis of CCOMP.

In any case, if the singled out component (COMP) resulting from deep diagnosis is not a primitive, the whole process, starting from SE, is recursively repeated on COMP (going one level down in the structural hierarchy). If the DE fails to locate a single COPM at a given structural level then recursion stops and the system returns the module at that level, along with its list of consistent suspects.

The flow of this process is shown in Figure 9.

\textsuperscript{13} But otherwise no conclusion should be deduced about S3.

\textsuperscript{14} This is performed regardless of absence or presence of cycles.
Figure 9 - Flowchart of diagnosis process.
4.4 Examples Of Diagnosis

Case diagnosis examples are given below. Assume that at the outset the SE has no empirical knowledge at all and that the cases are presented in chronological order.

Case 1

The complained symptom:
When operating the triggering switch first and later the enable switch, a single beep was heard. (The user was probably expecting a series of beeps).

Since no shallow knowledge exists at this point, SE fails and the DE is activated. It simulates the case and rejects the "symptom" as it finds the reported behavior normal. During simulation, learning takes place at the levels used by the simulator.

Case 2

Complained symptom:
when the enable switch is operated first, followed by operating the triggering switch, LOW beep rate was heard.

Again applying SE results in returning control to DE which, by simulation confirms that the complaint is indeed a malfunction as it finds that the beep rate should be MEDIUM. It considers a CSN similar to that presented in Figure 4, except that properties are beep_rate (of speaker), modulated_pulses_rate (of TI and modulator), enable (of enable_mod), and pulse_rate (of e_box, one_shot and inv2). Thus the component suspect list constitutes of (speaker, TI, one_shot, e_box, inv2, enable_mod, trig_mod).

The DE introduces an imaginary module IM containing the pulse_rate relationships of one_shot, inv2 and the e_box. The DE applies the following test:

"DE : Is flashing_rate at LED NORMAL (MEDIUM) ?"

-> No.

This test result implies that pulse_rate of the one_shot is wrong, hence, the DE eliminates the modulator, TI, and speaker from the component suspect list. Suppose test of enable at enable_mod succeeds (for example, voltage was found to be more than 3.5 V which indicates a TRUE value of enable since this voltage is considered HI). Now the IM is found to be the only remaining suspect. It consists of a cycle. The cycle's members are checked at this behavioral level.

"DE : Is pulse_rate at e_box NORMAL (MEDIUM) ?"

-> No.

test : "Is pulse_rate at inv2 NORMAL (MEDIUM) ?".

-> No.

\[15\] The same result could have been reached by (property) suspect set intersections if the user indicated the additional symptom at LED.
At this point, the DE realizes that this behavioral level does not provide more discriminatory information so a shift to a more refined level, illustrated in Figure 5 (b) is performed. At this level the pulse polarity of trig_mod appears. Based on the history of properties generated by simulation, the DE checks, through tests, that the trig_mod generates a NEG pulse followed first by a POS pulse by one_shot, and then by a NEG pulse by inv2, and followed by a NEG pulse by e_box. Suppose the above tests succeeded without adding any additional discriminatory information, then a more refined level illustrated in Figure 5 (c) is considered.

"DE : Is pulse_width at one_shot NORMAL (0.8-1.2 sec) ?"

-> No.

Now the one_shot is the only consistent component suspect. The SE is consulted about this suspect, but since it is still "empty" the DE is given control again, and the 555chip, the resistor R4 and the capacitor C1 constitute the new lower level component suspect list. Verification prices of suspect properties are high (for example charging time of capacitor, etc), so the DE considers failure mode simulation.

"DE : What is pulse_width at one_shot ? (TOO SHORT, TOO LONG).

-> TOO LONG.

Possible failure modes of R4 and C1 are actual resistance and capacitance that are lower or higher than nominal values\(^\text{16}\). A likelihood measure of each abnormal behavior determines the order of selecting each one. Say that lower resistance is selected first. Simulation of e_box yields a pulse width which does not explain the symptom of a TOO LONG pulse_width. Nevertheless, this is learnt by the SE (i.e. decrease of R4's resistance results in increase of pulse_width at e_box with OCN = 0). The second choice, increase of resistance, is found to explain the symptom (this is also learnt by the SE with OCN = 0), so R4 is suggested as the cause. Now suppose the user finds R4's actual resistance to be OK, so OCN is not incremented and a third choice of faulty behavior, higher capacitance of C1, is tried which explains the symptom. If the user finds this, as will be supposed, to be the case, (which can be done by actually replacing the capacitor), it implies constructing the appropriate net in the SE's knowledge concerning the d_system, associating increase of capacitance of C1 with decrease in pulse_rate of e_box and inv2 as well as with decrease in beeps_rate at speaker and flashing_rate at LED.

Case 3

Complained symptom:
when the enable switch is operated first, followed by operating the triggering switch, HIGH beep rate was heard.

The SE can't "remember" such a case, the DE accuses the one_shot (after probably asking about the beep_rate at speaker). Now the SE suggests R4 as manifesting lower than normal resistance since this was learnt as a "side effect" of reasoning in case 2, and found that to be the case. The OCN is incremented and new direct links from the symptoms to the LED's flashing_rate and speaker's beep_rate are created with OCN = 1.

\(^{16}\) One way to facilitate quantitative simulation of such behavior is providing a typical value of resistance (e.g. half the correct resistance). Note that here is a hidden assumption implying that failure of a component leads to the same symptom without considering the magnitude of violation, a somewhat strong assumption that can be relaxed if the concept of degree of failure is introduced.
Case 4

Complained symptom: when the enable switch is operated first and then the triggering switch then LOW flashing rate of LED and LOW beep rate of speaker are observed.

The SE suggests $R_4$ first and then $C_1$ which are not confirmed by the user. The DE finds $1M$ as the consistent component suspect and tries to resolve the cycle. It finds that even the refined behavioral level reached in the diagnosis of case 2 is not fine enough, and it goes further down where it considers pulse width of $e_{box}$ (this level is not illustrated in Figure 5) which is found to be TOO LONG. The SE can't suggest suspects, so the DE simulates $e_{box}$'s behavior using knowledge on faulty behavior of inv1 (longer than normal propagation delay). This is indeed found to be the cause, and appropriate learning actions take place. (This case is quite hypothetical, if we view inv2 as the "common" inverter with relatively very short propagation time delay).

5. CONCLUSIONS

In this paper an attempt was made to integrate shallow and deep knowledge, each with its own inference engine, thus defining two distinct experts that alternate in role in order to diagnose the malfunction of dynamic (synchronous or asynchronous) systems with state and feedback loops. Deep knowledge consists of structural and behavioral representations of the diagnosed system, in multiple levels of abstraction. A multilevel simulator that shifts on demand from coarse qualitative modeling to detailed quantitative modeling, assists the diagnosis process in verification and elimination of hypothesized suspects. Knowledge of pathological behavior (failure modes) of lower level components is incorporated in the simulator. Learning is exhibited as deep-to-shallow expertise transfer, as well as up the abstraction levels of the simulator itself, to improve its efficiency in subsequent cases.

Explanation facilities can be easily incorporated in the present system's architecture, as the causal nets can provide causal explanations on multiple levels of detail.

A prototype of the system is being implemented using PROLOG, exploiting it's declarative style and backtracking facilities. PROLOG's efficiency in simulation (especially quantitative) is relatively low, but quite adequate for our purposes as a test-bed for the ideas developed in this paper.

Some of the conclusions that can be drawn from our work are listed below:

1) Integration of experts adds efficiency by utilizing the shallow expertise whenever possible, with the deep expertise in stand by for the more difficult cases and for deeper explanations.

2) Multiple levels of behavioral abstraction greatly contribute to diagnosis efficiency and to explanation clarity, by matching the behavioral representation of the system under diagnosis to the appropriate difficulty level of the problem or the level of explanation required. In addition to providing greater efficiency by allowing "top-down" diagnosis, behavioral abstraction also eliminates false detections of "faults" due to insignificant discrepancies in fine details between outputs observed at the d_system and those computed by deep quantitative simulation.

3) Qualitative behavior modeling alone is not adequate for complex dynamic systems which may need explicit time representation in low behavioral levels. Hence, quantitative knowledge is still needed, both in normal behavior simulation and pathological simulation in the more detailed levels, where quantitative knowledge is more faithful to the no-function-in-structure concept. The latter is deliberately ignored in normal expected behavior at higher levels.

4) Compiling qualitative behavior by learning mitigates the high simulation cost in the long run. While compact compiled behavioral rules can eliminate the need of large output tables generated by learning, they require an exact understanding of component behavior.
Incorporating the ideas discussed in this paper in a large, efficient expert system suggests using object oriented and frame based systems: the structure and properties of the system under diagnosis could be mapped to objects holding relevant properties in slots where these slots are associated with special sensitive-to-value-change procedures performing the property updates (property daemons). Object oriented programming in this case would ease simulation inference. Moreover parallel processing might greatly cut down simulation cost.

Future work for topics not covered in this work include:

1. Tuning of quantization by proper selection of landmarks on property spaces.
2. Qualitative modeling is rather rigid and if compact readily-compiled behavioral rules are to be largely used, a need exists for deeper qualitative modeling and simulation.
3. Deriving the schema of qualitative behavioral nets automatically rather than predefining them, seems a challenging topic worth exploration.
4. In our prototype system, quantitative simulation uses a kind of constraint propagation where a delay time of a primitive is indicated, thus actually restricting the applicability of the prototype to suitable systems under diagnosis. For example it will be difficult to model and reason about say, flow of heat using the quantitative knowledge in it's present implementation. Representation by mathematical equations, for example, may increase the scope of the simulator.

REFERENCES


