Basic Definitions

Definition 1.1 (System) A system is a tuple \((SD, COMP \cup LEAF)\) where \(SD\) denotes the system description, \(COMP\) the set of components, and \(LEAF\) the set of components not connected to the outputs of another component.

\(LEAF\) components can be considered to be the input ports of the system and we assume they always behave correctly.

- Every component (except leaves) has an associated function:
  \[
  \text{func} : COMP \rightarrow Func \text{ with arity}
  \]
  \[
  \text{arity} : Func \rightarrow \mathbb{N}
  \]  
  (For brevity we will write \(\text{arity}(C)\) instead of \(\text{arity}(\text{func}(C))\).)

- Input ports: \(in(C)\)

- Output port: \(out(C)\).

- Ports have values: \(val : PORTS \rightarrow VALUES\).
  (We write \(val(C)\) instead of \(val(out(C))\)).

System description SD

- Components:
  
  \[
  \text{ok}(C) \Rightarrow (val(C) = \text{func}(C)(\text{val}(in_1(C)), \ldots, \text{val}(\text{in}_{\text{arity}(C)}(C)))) \]  

- Connections:
  
  \[
  out(C) = in(C').
  \]
  
  Every connection allows a value propagation:

  \[
  out(C) = in(C') \Rightarrow val(C) = val(in(C')).
  \]

Definition 1.2 (Diagnosis Problem) Let \((SD, COMP \cup LEAF)\) be a system and OBS be observations. The tuple \((SD, COMP \cup LEAF, OBS)\) is said to be a diagnosis problem. We assume that all elements of \(LEAF\) have an associated observation.

The observations are given in terms of a function returning the observed value of the component:

\[
\text{observed} : COMP \cup LEAF \rightarrow VALUES.
\]

A system is correct if the observed value is equal to the derived value for every component. We therefore introduce the rule

\[
\text{observed}(C) = \text{val}(C).
\]

Diagnoses are defined in the usual manner.

Definition 1.3 (Diagnosis) \(\Delta\) a subset of \(COMP\) is a diagnosis for \((SD, COMP \cup LEAF, OBS)\) iff \(SD \cup OBS \cup \{\text{ok}(C) \in COMP \setminus \Delta \cup \{-\text{ok}(C) | C \in \Delta\}\}\) is consistent.

- Top components:
  
  \(\top_{SD}(SD, COMP \cup LEAF)\) includes all components that are not used as input of other components.

- Path:
  
  Let \(C_1, C_2\) be two components. A path connecting \(C_1\) with \(C_2\) is a sequence of components \([C_1, \ldots, C_n]\) where each \(C_{i+1}\) is connected to an input of \(C_i\):

  \[
  \text{in}(C_i) = \text{out}(C_{i+1}) \in SD \Rightarrow \text{path}(C_1, C_2) = [C_1, C_2]
  \]

  \[
  \text{in}(C_1) = \text{out}(C_2) \in SD \land \text{path}(C_1, C_2) = [C_2, \ldots, C_n] \Rightarrow \text{path}(C_1, C_2) = [C_1, C_2, \ldots, C_n]
  \]
Tree-structured Systems

- Set of paths:
  \[ \text{path}(C_1, C_2) = \{ p | p = \text{path}(C_1, C_2) \} \].

Tree structured systems have only one top component and from every component to the top component there exists only one path. Figure 7.7 (a) shows a tree structured system, and (b) an acyclic one.

**Definition 1.4 (Tree Structured Systems)** A system \((SD, \text{COMP} \cup \text{LEAF})\) is tree structured iff

\[
\text{tops}(SD, \text{COMP} \cup \text{LEAF}) = \{ C \} \land \forall C' \in \text{COMP} \cup \text{LEAF} : \text{path}(C, C') = 1.
\]

**Corollary 1.1** Every tree structured system is an acyclic system.

---

**Examples**

- Tree-structured system:

![Tree-structured system diagram]

- Acyclic system:

![Acyclic system diagram]

---

**TREE: Basic Idea**

Consider a component \(C\)

- **Observed value of** \(C \neq \) computed output of \(C\)
  
  \(\Rightarrow\) There must be an incorrect component in the system

  1. \(C\) behave incorrectly
  2. \(C\) behave correctly but another component connected to an input of \(C\) behave incorrectly

- The inputs of \(C\) must have values leading to the observed output value.

- **Introduce a function** \(\text{INPUTs}\): Compute all input values producing the observed value.

- **Call the diagnosis function recursively on** the input components using input values.

- All diagnoses must be combined to give the overall diagnosis.

- Do not compute all input tuples

  Example: And gate (input \((1,1)\), observed output \(0\))

  In this case we use the tuples \((1,0)\) and \((0,1)\) for computing (minimal) diagnoses.

---

**Some Definitions**

- **Combining diagnoses**

  1. \(S \times \{\} = \{\} \times S = \{\}
  2. \(S_1 \times S_2 = \bigcup_{x_1 \in S_1} \{x_1 \cup x_2\}

  1. \(x_{\text{top}}S = S_1 \times (x_{\text{top}}S_2)
  2. \(x_{\text{top}}S = S_n

- **Restricting diagnoses by size**

  For a diagnosis set \(S\) and \(ds \in \mathbb{N}\), \(S|ds = \{D|D \in S \land |D| \leq ds\}

- **Function** checked **answering whether a component has been used for diagnosis or not**

  Before executing **TREE_DIAG**, checked answers false for all components.

- **Global variable**

  \(\text{diags}\), to collect diagnoses as they are constructed (initialized to \(\{\}\))

- **Diagnosis algorithm calg**

  **TREE_DIAG**(top(SD, COMP))
Algorithm TREE_Diag(C)

Evaluates every component, i.e., computes their output values and calls the diagnosis algorithm if the computed value is not equal to an observed value.

C: current Component, initially the top component
Global constants: SD, CMF, LEAF, O RS
Global variables: diagSize (Diagnosis size) and diag (Diagnosis set, initialized to [])

IF C ∈ LEAF THEN
  val(C) ← observed(C));
  checked(C) ← true
ELSE
  FOR i ∈ {1, ..., arity(C)} DO
    Vᵢ ← TREE_Diag(tᵢ(C));
  END FOR;
  V ← func(C)(V₁,...,Vᵣ(C));
  IF exists(known(C)) AND
    V ≠ observed(C) THEN
    diag ← diag ∪
    DIAG(known(C)) | diagSize;
    val(C) ← observed(C)
  ELSE
    val(C) ← V
  END IF
END IF

Example

![Example Diagram](attachment:image.png)

Practical Results

Performance TREE_Diag, Reiter, and Dechter

![Performance Graph](attachment:image.png)
**TREE_DIAG** and Dechter

Systems with up to 10,000 diagnosis components.

Influence of input number on **TREE_DIAG**

$|COMPS| = 100$

---

**Conclusion**

- **TREE_DIAG** is very fast

- Can be used for Software Debugging of Functional Languages

- Can be extended (Fault modes, . . .)