Fundamentals of Model-based Reasoning

Outline

• Introduction & Overview
• Model-based Programming
• Execution of Model-based Programs
• Fundamentals of Model-based Reasoning
  – Consistency-based diagnosis (GDE)
  – Pre-compilation & Probing
  – Livingstone/Titan
  – Conflict-Directed A*
• Modeling via State Analysis
• Advanced Methods
• Conclusion
Model-based Diagnosis

- Discrepancy between predicted and observed behavior indicates a fault.
- Structural discrepancy allows us to build fault candidates.
- Sort fault candidates in order of probability and perform additional tests to reject high probability candidates.

Simple Circuit with Multipliers & Adders

X, Y, and Z are not directly observable
Simple Circuit with Multipliers & Adders

Constraints: X=A*C
Y=B*D
Z=C*E
F=X+Y
G=Y+Z

Predictions: X=6
Y=6
Z=6
F=12
G=12

Components

• Language for modeling the components and their structure (connections)
• Predictive inference engine (propagation)
• Diagnostic Engine
• Approach:
  – Find a sorted set of fault candidates.
  – Perform a sequence of tests that refine the fault candidate set.
  – Tests are expensive so select tests carefully.
Candidates: [A1], [M1], [A2, M2], [M2, M3], …

Test: X=6  Candidates: [A1], [A2, M2], [M2, M3], …
Test: Y=4  Candidates: [A2, M2], [M2, M3], …
Test: Z=8  Candidates: [M2, M3], …
Propagation

[A=3, {}] [B=2, {}] [C=2, {}] [D=3,{}] [E=3, {}]
[F=10, {}] [G=12, {}]
[F=12, {A1, M1, M2}]
[X=4, {A1,M2}{A1,A2,M3}
  6, {M1}]
[Y=4, {A1,M1}]
  6, {M2}{A2,M3}]
[Z=8, {A1, A2, M1}]
  6, {M3}{A2,M2}]
[G=10, {A1,A2,M1,M3}]
  12, {A2,M2,M3}]

Two Conflicts

Possible Faulty Components

[M1,M2,M3,A1,A2]

[M1,M2,M3,A1] [M1,M2,M3,A2] [M1,M2,A1,A2] …

[M1,M2,M3] [M1,M2,A1] [M1,M2,A2] [M1,M3,A1] …

[M1,M2] [M1,M3] [M1,A1] [M2,M3] [M1,A2] [M2,A1] …

[M1] [M2] [M3] [A1] [A2]
Kernel Diagnosis

Sub Diagnoses:

\[ F = 12, \{[A1], [M1], [M2]\} \]
\[ G = 10, \{[A1], [A2], [M1], [M3]\} \]

Kernel Diagnoses:

\[ [A1] \]
\[ [M1] \]
\[ [M2,A2] \]
\[ [M2,M3] \]
\[ [A1] [M1] [M2,A2] [M2,M3] \]

Simple Circuit with Multipliers & Adders

Candidate Generation: [] (everything working)
\[ <[A1], [M1], [M2]> \text{ (from conflict } F = 10) \]
\[ <[A1], [M1], [M2, A2], [M2, M3]> \text{ (G = 12)} \]

Note: [M2, M1] and [M2, A1] not included because they are supersets of [M1] and [A1].
Notes on Incremental Candidate Generation

• New measurements may increase or decrease the number of minimal candidates.
• Once a candidate is eliminated it can never reappear.
• Eliminated minimal candidates are replaced by larger candidates.
• If a component appears in every minimal candidate, that component is necessarily faulted.

Scalability

• For large systems the number of candidates can grow very large.
• We manage this using various techniques including:
  – Representing only the minimal candidates.
  – Restricting candidate generation to only consider ‘n’ faults.
• Push the hard work back to compile time to reduce runtime cost.
Pre-compilation Phase

Idea: Pre-compile Test Results
1. Solve diagnosis sub-problems at compile-time, by generalizing GDE's conflict recognition.
2. Create run-time rules mapping observations to sub-diagnoses.
3. Given observations, synthesize likely global diagnoses.

Features:
- Shifts an NP-hard problem to compile-time.
- Sub-diagnoses tend to be small.
- Avoids generating large set of unlikely global diagnoses.
- Viewing decomposed rules aids engineering analysis.

Active Probing/Measurements

- Select measurements that maximize the elimination of high probability candidates.
- Continue to select new measurements (tests) until all high probability faults have been found or until no more useful measurements can be taken.
- Objective: Find the correct candidate with the minimum total test cost.
  - The best next measurement is the one that minimizes the expected entropy of candidate probabilities resulting from the measurement.

De Kleer and Williams AIJ 1987 “Diagnosing Multiple Faults”
So far we have considered stateless components.
- Actually our components have had two states:
  1. Working (with constraints)
  2. Faulted (no constraints)

Many components have multiple working states in addition to faulted state(s) for example:
- Valves, Switches, Sockets, etc.

We need to:
- Represent the different working states along with their corresponding constraints.
- Estimate what state our components are in given our measurements (when the state is not directly/completely observable).
Example: Propulsion Subsystem

Models:
- **Solenoid Valve (V)**
  - Modes – \{O, C, U\}*
  - O(V) \iff ((P1 = nom) \land (P2 = nom)) \land ((P1 = low) \land (P2 = low))
  - C(V) \iff (P2 = low)
  - U(V) \iff ( )
- **Catalyst Bed (C)**
  - Modes – \{G, B, U\}*
  - G(C) \iff ((P2 = nom) \land (TH = on)) \land ((P2 = low) \land (TH = off))
  - B(C) \iff (TH = off)
  - U(C) \iff ( )
- **Pressure Transducer (T)**
  - Modes – \{G, SH, SL, U\}*
  - G(T) \iff ((TP = nom) \land (P1 = nom)) \land ((TP = low) \land (P1 = low))
  - SH(T) \iff (P1 = nom)
  - SL(T) \iff (P1 = low)
  - U(T) \iff ( )

* All modes have an associated probability.

Variables:
- **Observed**
  - Pipe 1 Pressure (P1) – \{nom, low\}
  - Engine Thrust (TH) – \{on, off\}
- **Hidden**
  - Tank Pressure (TP) – \{nom, low\}
  - Pipe 2 Pressure (P2) – \{nom, low\}

Pre-compile Possible Conflicts (Dissents)

- **Identify Dissents:**
  - Dissent maps observations to conflicting modes:
    - (P1 = low) \land (TH = on) \Rightarrow \neg (G(T) \land O(V) \land G(C))

- **Generate Partial Diagnosis Rules:**
  - Replace conflicts with sub-diagnoses:
    - (P1 = low) \land (TH = on)
    \Rightarrow (SH(T) \lor SL(T) \lor U(T) \lor C(V) \lor U(V) \lor B(C) \lor U(C))

- **Compilation method:**
  - Identify dissents by generating Prime Implicates containing only OBS and Modes.
• Monitors generate discrete data
  – Value : Sensor Voltage = 23 V               Sensor Voltage = nominal
• Monitors trigger Rules, … which produce sub-diagnoses
• \((P1 = \text{nom}) \land (T = \text{off}) \Rightarrow SH(T) \lor SL(T) \lor U(T) \lor C(V) \lor U(V) \lor B(C) \lor U(C)\)
• \((P1 = \text{nom}) \Rightarrow G(T) \lor SH(T) \lor U(T)\)
• \((P1 = \text{low}) \Rightarrow G(T) \lor SL(T) \lor U(T)\)
• …

Search for most likely Kernel Diagnoses
• Find most likely covering of sub-diagnoses
• Guide set covering by A* search
Example: Propulsion Subsystem

- Observations
  - $P_1 = \text{low}$
  - $TH = \text{on}$

- Triggered Partial Diagnoses
  - $G(C) \lor U(C)$
  - $GV(U) \lor U(V)$
  - $G(T) \lor SL(T) \lor U(T)$
  - $SH(T) \lor SL(T) \lor U(T) \lor C(V) \lor U(V) \lor B(C) \lor U(C)$

- Most-likely Diagnosis
  - $0.865 \text{ C(C)}$
  - $0.0009 \text{ U(C)}$

Example: Propulsion Subsystem
Example: Propulsion Subsystem

- Observations
  - $P_1 = \text{low}$
  - $TH = \text{on}$

- Triggered Partial Diagnoses
  - $G(C) \lor U(C)$
  - $O(V) \lor U(V)$
  - $G(T) \lor SL(T) \lor U(T)$
  - $SH(T) \lor SL(T) \lor U(T) \lor C(V) \lor U(V) \lor B(C) \lor U(C)$

- Most-likely Diagnosis
• Observations
  • $P_1 = \text{low}$
  • $T = \text{on}$

• Triggered Partial Diagnoses
  - $G(C) \lor U(C)$
  - $O(V) \lor U(V)$
  - $G(T) \lor SL(T) \lor U(T)$
  - $SH(T) \lor SL(T) \lor U(T) \lor C(V) \lor U(V) \lor B(C) \lor U(C)$

• Most-likely Diagnosis

Example: Propulsion Subsystem

Example: Propulsion Subsystem
**Conflict-directed, best-first, deductive kernel**

- Tasks & models compiled into propositional logic queries
- ITMS efficiently tracks state changes in truth assignments
- Conflicts dramatically focus search
- Careful enumeration grows agenda linearly

---

**Conflict-directed A***

When you have eliminated the impossible, whatever remains, however improbable, must be the truth.

- Sherlock Holmes. The Sign of the Four.

1. Test Hypothesis
2. If inconsistent, learn reason for inconsistency (a Conflict).
3. Use conflicts to leap over similarly infeasible options to next best hypothesis

Cd A* [Williams & Ragno, JDAM 05]
Livingstone [Williams & Nayak, AAAI 95]
Sherlock [de Kleer & Williams, IJCAI 89]
DDB [Stallman & Sussman, 77]
Function Conflict-directed-A*(OCSP)
returns the leading minimal cost solutions.
Conflicts[OCSP] ← {} 
OCSP ← Initialize-Best-Kernels(OCSP) 
Solutions[OCSP] ← {} 
loop do 

decision-state ← Next-Best-State-Resolving-Conflicts(OCSP) 
if no decision-state returned or Terminate?(OCSP) 
then return Solutions[OCSP] 
if Consistent?(C[OCSP], decision-state) 
then add decision-state to Solutions[OCSP] 
new-conflicts ← Extract-Conflicts(CSP[OCSP], decision-state) 
Conflicts[OCSP] ← Eliminate-Redundant-Conflicts (Conflicts[OCSP] ∪ new-conflicts)
end

Conflict-directed A*

• Feasible subregions described by kernel assignments.
▷ Approach: Use conflicts to search for kernel assignment containing the best cost candidate.
**Next-Best-State-Resolving-Conflicts**

function Next-Best-State-Resolving-Conflicts(OCSP)

\[
\text{best-kernel} \leftarrow \text{Next-Best-Kernel}(\text{OCSP})
\]

if best-kernel = failure

then return failure

else return Kernel-Best-State(problem)(best-kernel)

end

function Kernel-Best-State(kernel)

unassigned \leftarrow \text{all variables not assigned in kernel}

return kernel \cup \{\text{Best-Assignment}(v) \mid v \in \text{unassigned}\}

End

{M2=U}

{M1=G, M2=U, M3=G, A1=G, A2=G}

See [Williams & Ragno, JDAM 05] to find multiple leading solutions

**Candidate Lattice**

Edge between two nodes that differ by a single mode assignment

Double faults

Single faults

All nominal modes
Best First Search

Conflict-Directed
Best First Search

A conflict is an assignment to a subset of the variables that is inconsistent with the model and observations.
Optimizing the Agenda

Frame Mode Estimation & Mode Reconfiguration as OCSPs

OCSP = \langle X, D_X, g_X, Y, D_Y, C(X, Y) \rangle
- Decision variables \( X \) with domain \( D_X \)
- Utility function \( g_X(X): D_X \rightarrow \mathbb{R} \)
- State variables \( Y \) with domain \( D_Y \)
- Constraint \( C(X, Y): D_X \times D_Y \rightarrow \{True, False\} \)

Find Leading \( arg \ max \ g(X) \)
\[ X \in D_X \]
\[ \text{s.t. } \exists Y \in D_Y \text{ s.t. } C(X, Y) \text{ is True} \]
\( g_X() \) is a multi-attribute utility function that is preferentially independent.
Mutual Preferential Independence

Assignment $\delta_1$ preferred over $\delta_2$ if $g(\delta_1) < g(\delta_2)$

For $W \subseteq X$, the preference between two assignments to $W$ is independent of the assignment to the remaining variables, $W - X$.

Example: Diagnosis: $g(X) = G(g_1(x_1), g_2(x_2), \ldots)$

$$g_i(x_i = \text{mode}_i) = P(x_i = \text{mode}_i)$$

$$G(u_1, u_2) = u_1 \times u_2$$

If $M1 = G$ is more likely than $M1 = U$,
to $\{M1 = U, M2 = G, M3 = U, A1 = G, A2 = G\}$

Summary

- Early model-based diagnosis systems, like GDE, used consistency-based diagnosis to produce a set of feasible candidate solutions.

- State-of-the-art model-based executives, like Livingstone and Titan, leverage the Conflict-Directed A* algorithm to produce solutions in best-first order:
  - Probability-ordered for ME;
  - Cost-ordered for MR (GI).

- Pre-compilation of the most computationally expensive operations allows for improved online reactivity.
• Introduction to State Analysis
  – An overview of a Model-based Systems Engineering Methodology that is compatible with Model-based Programming
  – A discussion of how Model-based Programming fits into the project lifecycle