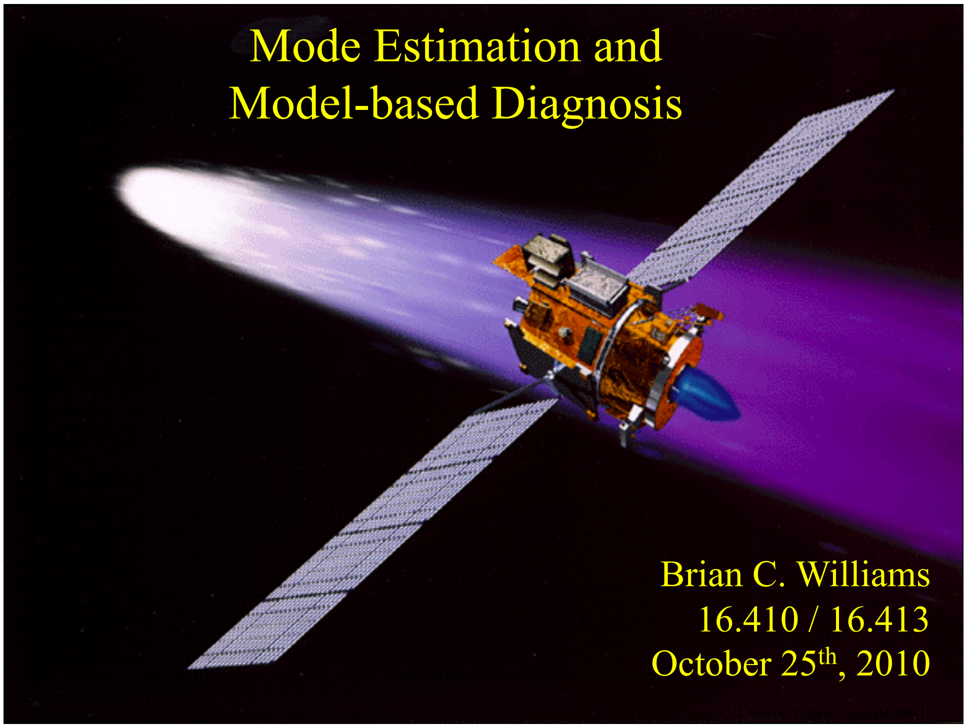


Image credit: NASA.

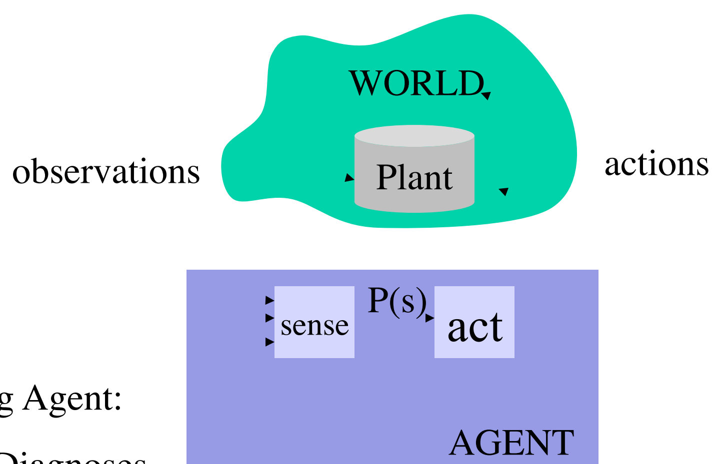
Assignment

- Remember:
  - Problem Set #6 Propositional Logic,
    due this Wednesday, October 27th.
  - 16:413 Project Part 1: Sat-based Activity Planner,
    due Wednesday, November 3rd.
- Reading
  - Today: Johan de Kleer and Brian C. Williams, "Diagnosing Multiple Faults," *Artificial Intelligence*, 32:100-117, 1987.
  - Wednesday: Brian C. Williams, and Robert Ragno, "Conflict-directed A* and its Role in Model-based Embedded Systems," Special Issue on Theory and Applications of Satisfiability Testing, *Journal of Discrete Applied Math*, January 2003.

10/25/10 2

Outline

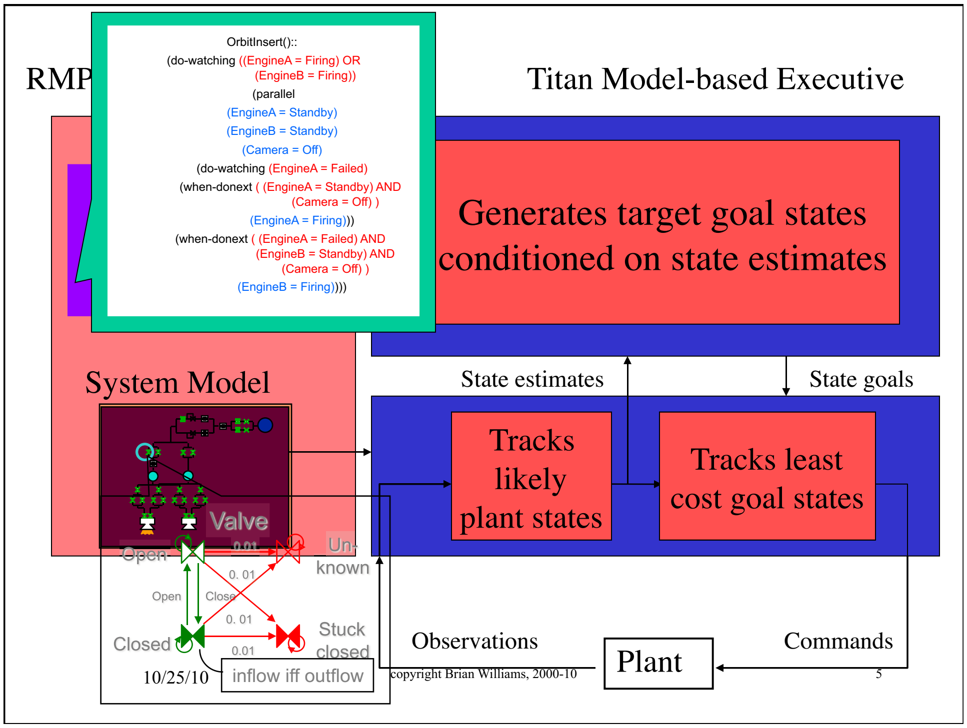
- Self-Repairing Agents
  - Model-based Programming
  - Diagnosis as Conflict-directed Search
- Formulating a Diagnosis
- Diagnosis from Conflicts

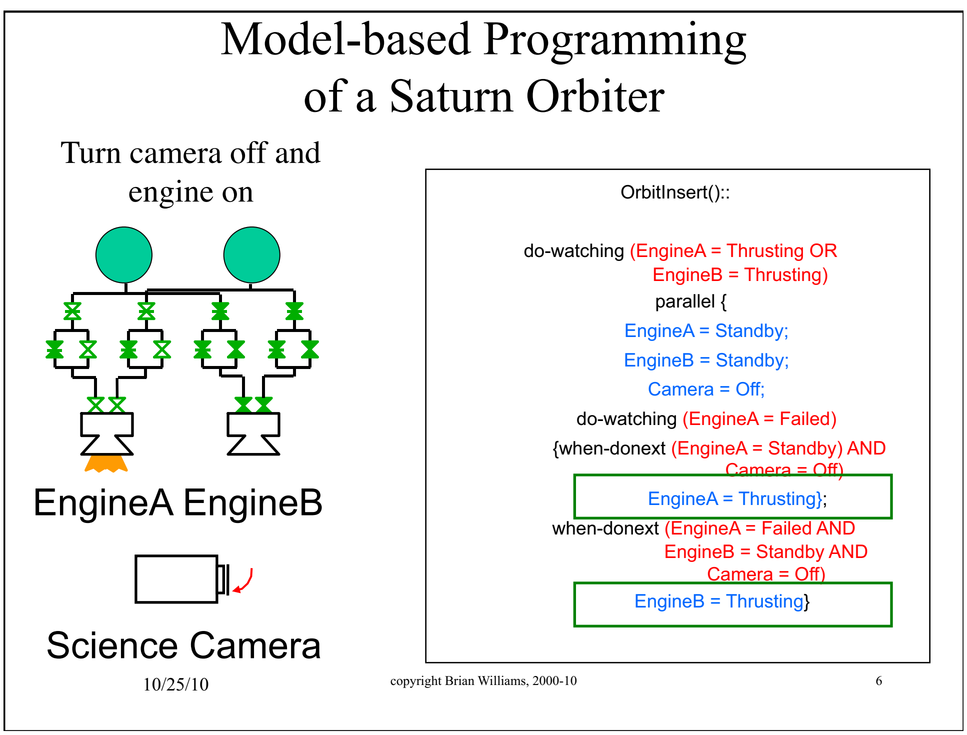


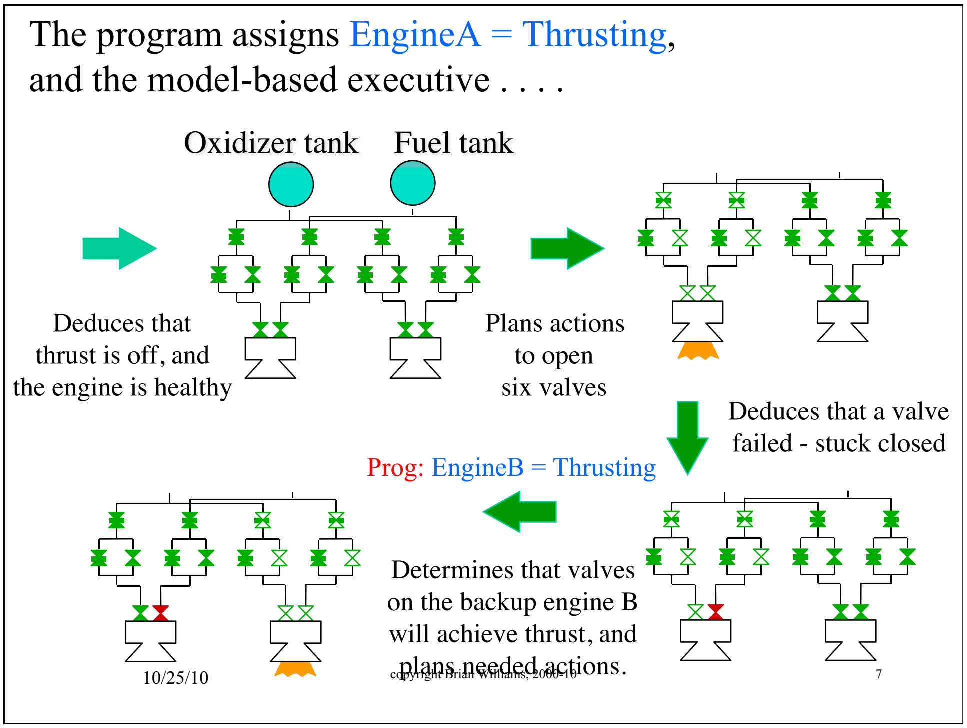
Self-Repairing Agent:

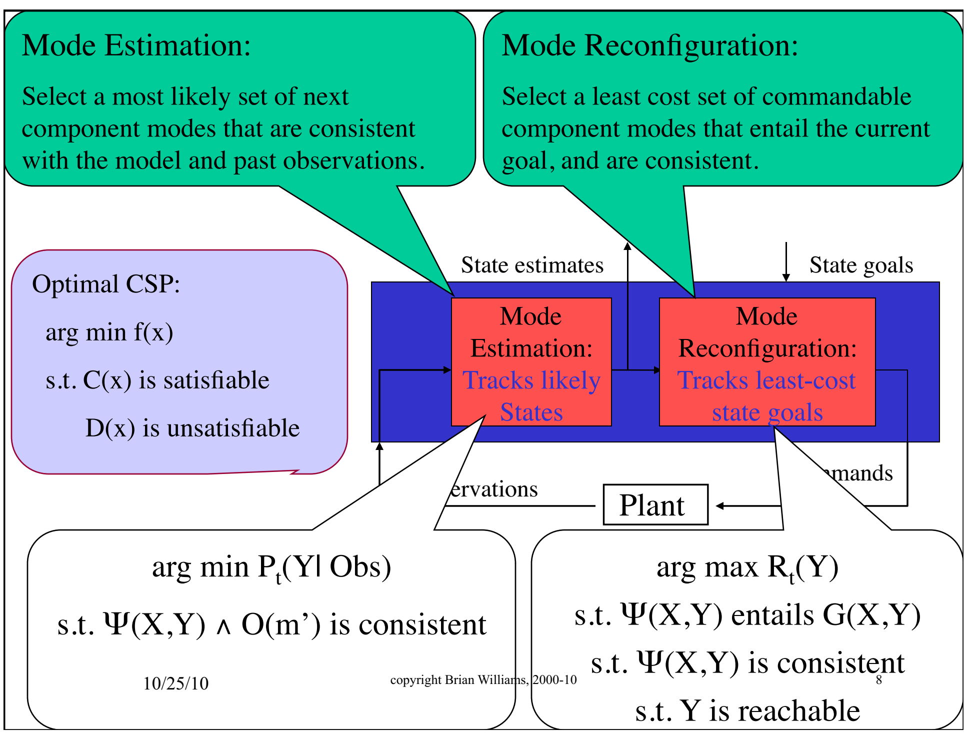
- Monitors & Diagnoses
- Repairs & Avoids
- Probes and Tests

Symptom-directed


RMP
Titan Model-based Executive
OrbitInsert()::
(do-watching ((EngineA = Firing) OR
(EngineB = Firing))
(parallel
(EngineA = Standby)
(EngineB = Standby)
(Camera = Off)
(do-watching (EngineA = Failed)
(when-donext ((EngineA = Standby) AND
(Camera = Off))
(EngineA = Firing)))
(when-donext ((EngineA = Failed) AND
(EngineB = Standby) AND
(Camera = Off))
(EngineB = Firing))))
Generates target goal states conditioned on state estimates
System Model
Valve
Open
Un-
known
0.01
0. 01
Open
Close
0. 01
Stuck
closed
Closed
0.01
inflow iff outflow
State estimates
State goals
Tracks likely plant states
Tracks least cost goal states
Observations
Commands
Plant
10/25/10
copyright Brian Williams, 2000-10
5


Model-based Programming
of a Saturn Orbiter
Turn camera off and
engine on
EngineA EngineB
Science Camera
OrbitInsert()::
do-watching (EngineA = Thrusting OR
EngineB = Thrusting)
parallel {
EngineA = Standby;
EngineB = Standby;
Camera = Off;
do-watching (EngineA = Failed)
{when-donext (EngineA = Standby) AND
Camera = Off)
EngineA = Thrusting};
when-donext (EngineA = Failed AND
EngineB = Standby AND
Camera = Off)
EngineB = Thrusting}
10/25/10
copyright Brian Williams, 2000-10
6


The program assigns EngineA = Thrusting,
and the model-based executive . . . .
Oxidizer tank
Fuel tank
Deduces that
thrust is off, and
the engine is healthy
Plans actions
to open
six valves
Deduces that a valve
failed - stuck closed
Prog: EngineB = Thrusting
Determines that valves
on the backup engine B
will achieve thrust, and
plans needed actions.
10/25/10
copyright Brian Williams, 2000-10
7


Mode Estimation:
Select a most likely set of next
component modes that are consistent
with the model and past observations.
Mode Reconfiguration:
Select a least cost set of commandable
component modes that entail the current
goal, and are consistent.
Optimal CSP:
arg min f(x)
s.t. C(x) is satisfiable
D(x) is unsatisfiable
State estimates
State goals
Mode
Estimation:
Tracks likely
States
Mode
Reconfiguration:
Tracks least-cost
state goals
ervations
mands
Plant
arg min P_t(Y| Obs)
s.t. Ψ(X,Y) ∧ O(m') is consistent
arg max R_t(Y)
s.t. Ψ(X,Y) entails G(X,Y)
s.t. Ψ(X,Y) is consistent
s.t. Y is reachable
10/25/10
copyright Brian Williams, 2000-10
8

Outline

- Self-Repairing Agents
  - Model-based Programming
  - Diagnosis as Conflict-directed Search
- Formulating a Diagnosis
- Diagnosis from Conflicts

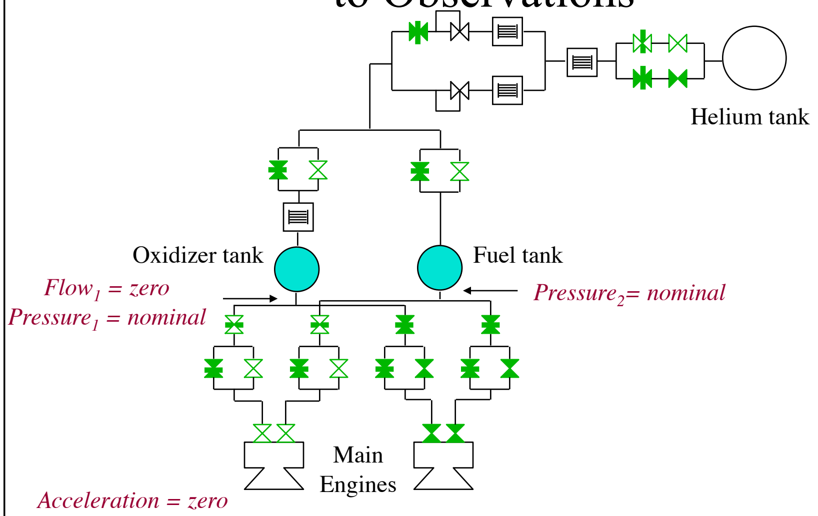
Model-based Diagnosis as Conflict-directed Best First Search

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

- Sherlock Holmes. The Sign of the Four.

1. Generate most likely candidate.
2. Test candidate.
3. If Inconsistent, learn reason for inconsistency (a conflict).
4. Use conflicts to leap over similarly infeasible options to the next best candidate.

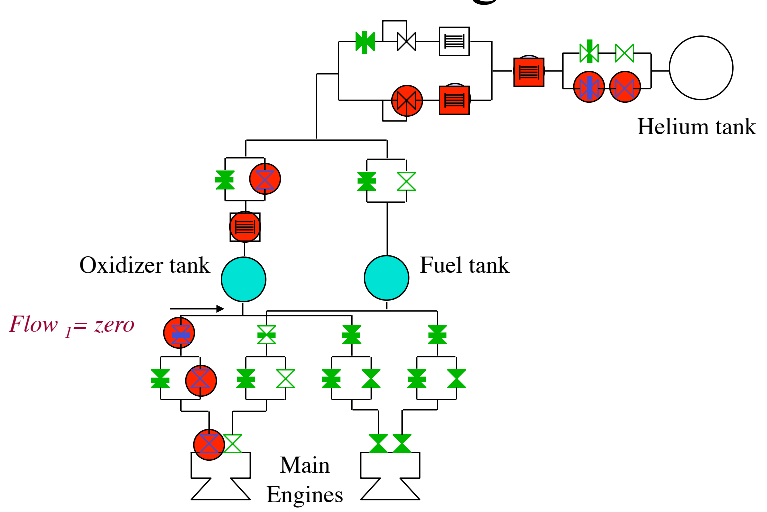
Compare Most Likely Candidate to Observations



It is most likely that all components are okay.

10/25/10 copyright Brian Williams, 2000-10 11

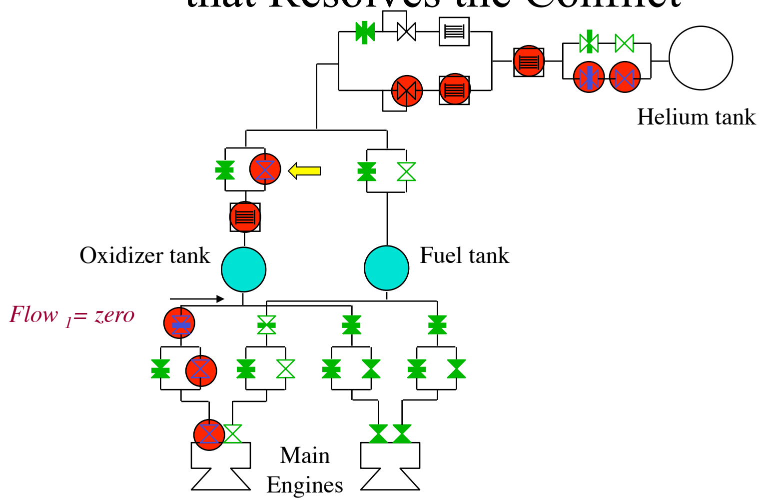
Isolate Conflicting Information



The red component modes *conflict* with the model and observations.

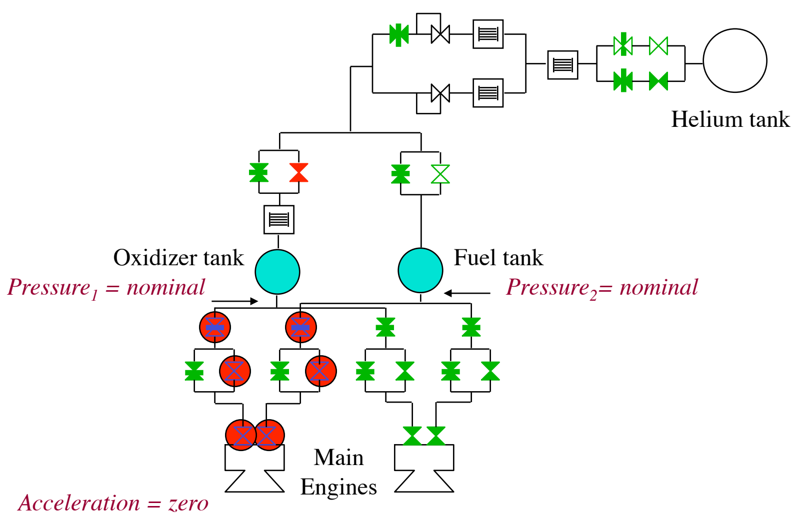
10/25/10 copyright Brian Williams, 2000-10 12

Leap to the Next Most Likely Candidate that Resolves the Conflict



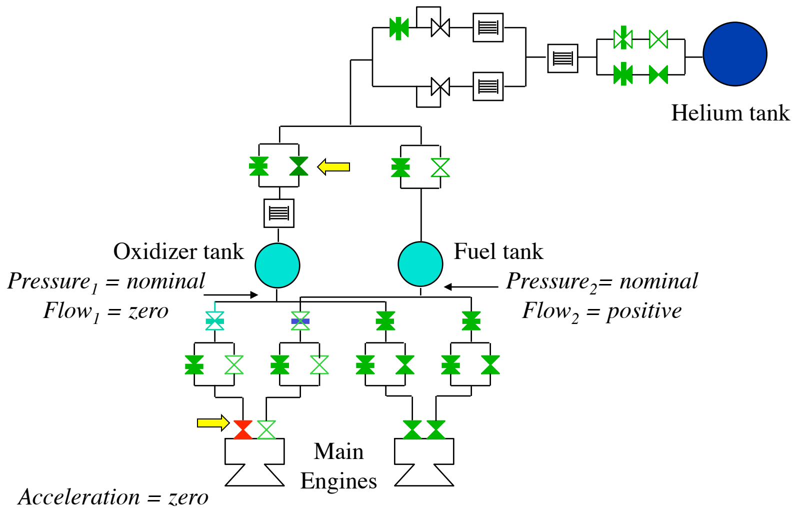
The next candidate must remove the conflict.

New Candidate Exposes Additional Conflicts



Another conflict, try removing both.

Final Candidate Resolves all Conflicts



Implementation: Conflict-directed A* search.

Outline

- Self-Repairing Agents
- Formulating a Diagnosis
- Diagnosis from Conflicts

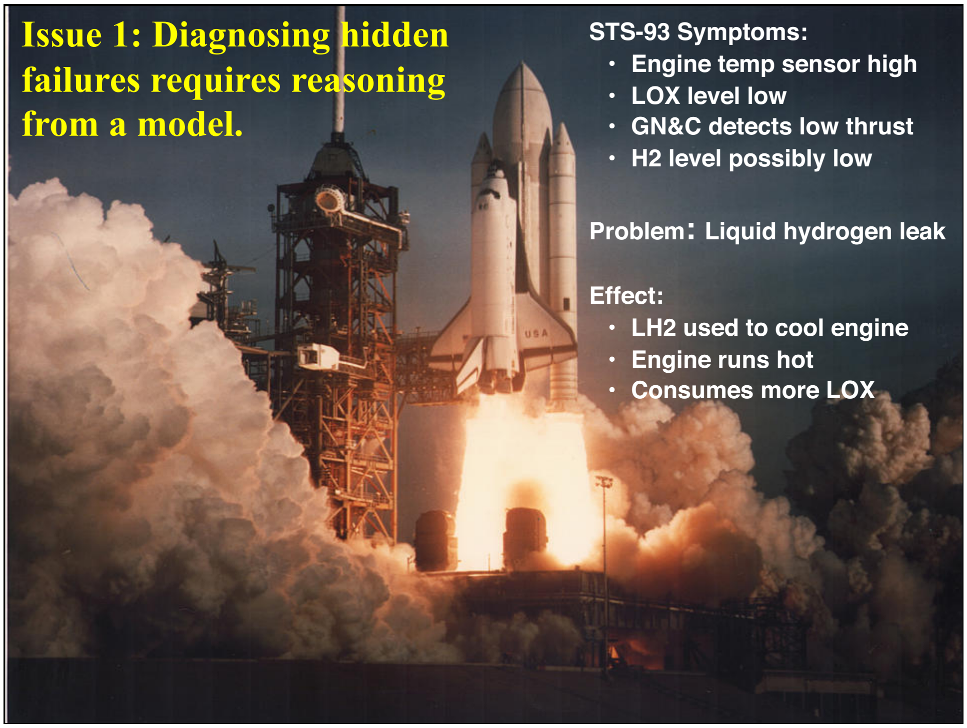
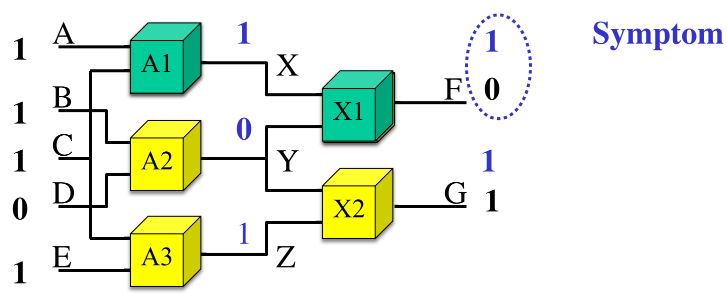


Image credit: NASA.

Model-based Diagnosis

Input: Observations of a system with symptomatic behavior, and a model Φ of the system.

Output: Diagnoses that account for the symptoms.



Solution: Diagnosis as Hypothesis Testing

1. Generate candidates, given symptoms.
2. Test if candidates account for all symptoms.

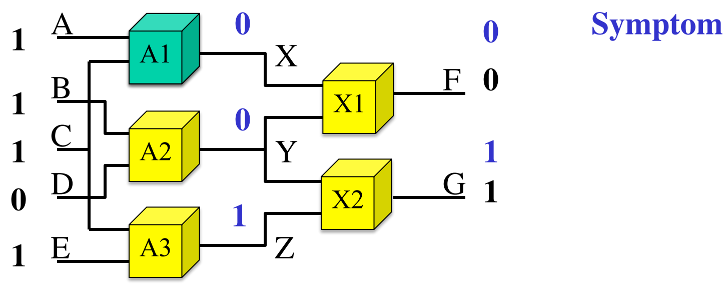
- Set of diagnoses should be complete.
- Set of diagnoses should exploit all available information.

Outline

- Self-Repairing Agents
- Formulating Diagnosis
  - Explaining failures
  - Handling unknown failures
  - Multiple faults
  - Partial explanation
  - Execution monitoring
- Diagnosis from Conflicts

How Should Diagnoses Account for Symptoms?

Abductive Diagnosis: Given symptoms, find diagnoses that predict observations.

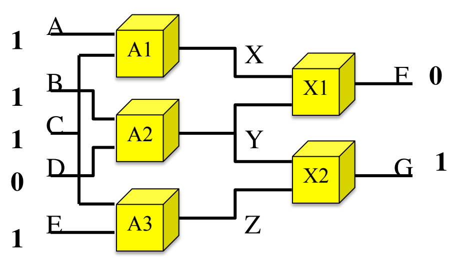


- Fault Model: A1's output is stuck at 0.
- Abductive diagnosis needs exhaustive fault models.

10/25/10 21

Input: Abductive, Model-based Diagnosis

Xor(i):
- G(i):
  Out(i) = In1(i) xor In2(i)
- Stuck_0(i):
  Out(i) = 0

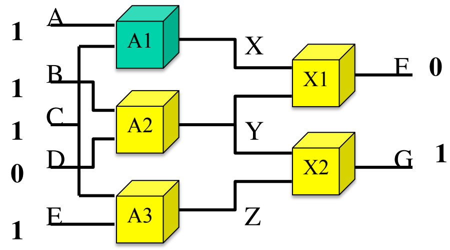


- Model Φ
  - Structure.
  - Model of normal behavior for each component.
  - Model for **every** component failure mode.
- Observations Obs
  - Inputs and Response.

10/25/10 22

Model: Abductive, Model-based Diagnosis

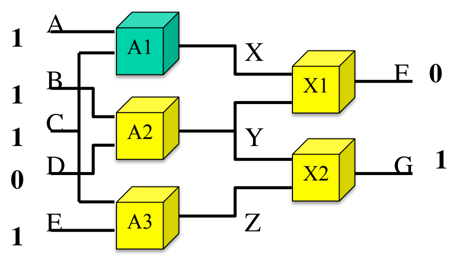
Xor(i):
- G(i):
  Out(i) = In1(i) xor In2(i)
- Stuck_0(i):
  Out(i) = 0



- X — mode variables, one for each component c.
- D_c — modes of component c = domain of $m_c \in M$.
- Y — state variables, with domains D_Y.
- $\Phi(X, Y)$ — model constraints.
- O — observed variables $O \subseteq M \cup Y$.
  - » Partitioned into Input I and Response R variables.

Output: Abductive, Model-based Diagnosis

Xor(i):
- G(i):
  Out(i) = In1(i) xor In2(i)
- Stuck_0(i):
  Out(i) = 0



Candidate = {X1=G, X2=G, A1=G, A2=G, A3=G}

Diagnosis = {X1=G, X2=G, A1=S0, A2=G, A3=G}

- Obs = <Inp; Rsp> — Assignment to I and R, respectively.
- Candidate C_i: — Assignment of modes to X.
- Diagnosis D_i: — A candidate such that $D_i \wedge \text{Inp} \wedge \Phi$ entails Rsp.

Abductive Diagnosis by Generate and Test

Given: exhaustive fault models, structure and observations.

Generate: candidate mode assignment C_i.

Test: C_i as an abductive diagnosis:

1. Find Rsp entailed by C_i, given Inp.
2. Compare observed and predicted Rsp:
   - Disagree: Discard
   - Agree: Keep
   - No prediction: **Discard**

Exonerate: component if none of its fault models agree.

Problem:

- Fault models are typically incomplete.
- May incorrectly exonerate faulty components.

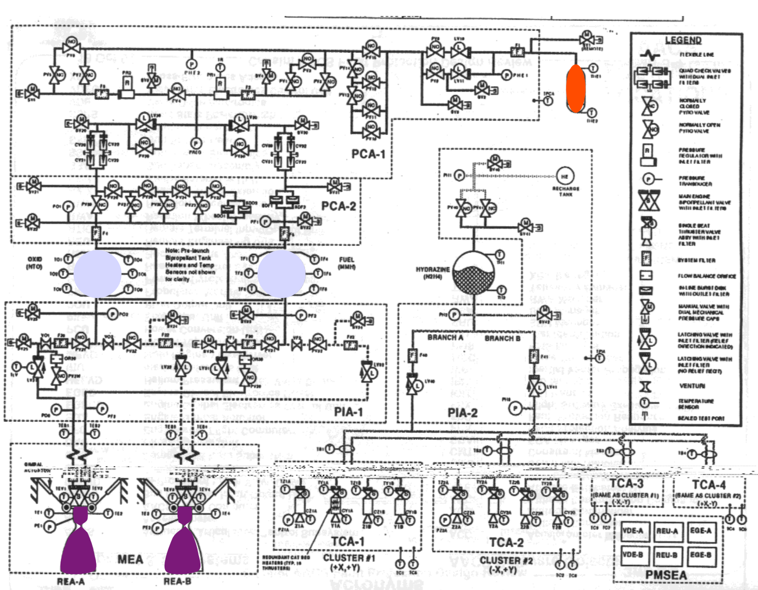
Outline

- Self-Repairing Agents
- Formulating Diagnosis
  - Explaining failures
  - Handling unknown failures
  - Multiple faults
  - Partial explanation
  - Execution monitoring
- Diagnosis from Conflicts

Issue 2: Failures are Often Novel

Image credit: NASA/JPL.

- Mars Observer
- **Mars Climate Orbiter**
- **Mars Polar Lander**
- **Deep Space 2**

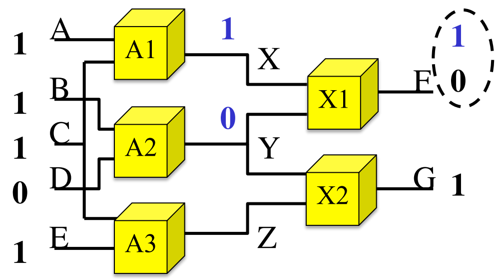


Failure models are never completely known.

How Should Diagnoses Account for Novel Symptoms?

Consistency-based Diagnosis: Given symptoms, find diagnoses that are consistent with symptoms.

Suspending Constraints: For novel faults, make no presumption about faulty component behavior.



Symptom

[deKleer & Brown, 83]

[Davis, 84]

[Geneserth, 84]

Outline

- Self-Repairing Agents
- Formulating Diagnosis
  - Explaining failures
  - Handling unknown failures
  - Multiple faults
  - Partial explanation
  - Execution monitoring
- Diagnosis from Conflicts

Issue 3: Multiple Faults Occur

Image source: NASA.

- Three shorts, tank-line and pressure jacket burst, and panel flies off.

➔ Diagnosis = mode assignment.

➔ Solve by divide & conquer:

1. Diagnose each symptom.
2. Summarize conflicts.
3. Combine diagnoses.

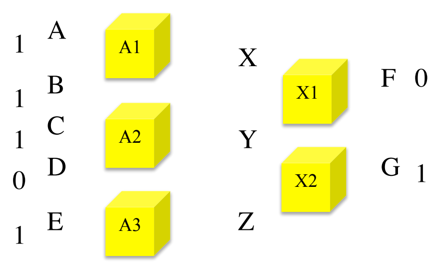
APOLLO 13

10/25/10 copyright Brian Williams, 2000-10 31

Solution: Identify all Combinations of Consistent "Unknown" Modes

And(i):

- G(i):
  Out(i) = In1(i) AND In2(i)
- U(i):



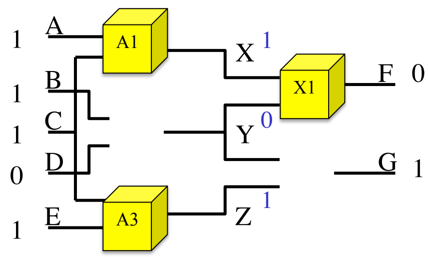
Candidate = {A1=G, A2=G, A3=G, X1=G, X2=G}

- Candidate: Assignment of G or U to each component.

10/25/10 copyright Brian Williams, 2000-10 32

Solution: Identify all Combinations of Consistent Unknown Modes

And(i):
- G(i):
  Out(i) = In1(i) AND In2(i)
- U(i):



Diagnosis = {A1=G, A2=U, A3=G, X1=G, X2=U}

- Candidate: Assignment of G or U to each component.
- Diagnosis: Candidate consistent with model and observations.

Outline

- Self-Repairing Agents
- Formulating Diagnosis
  - Explaining failures
  - Handling unknown failures
  - Multiple faults
  - Partial explanation
  - Execution monitoring
- Diagnosis from Conflicts

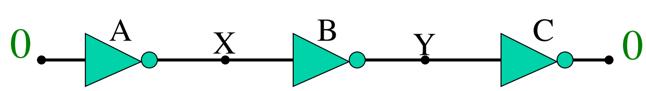
Issue 4: The cause of failure is often needed to plan a recovery strategy (*Partial Explanation*).

Issue 5: Component mode estimates are needed to confirm correct behavior (Execution Monitoring).

courtesy of NASA

Incorporating Failure Modes: Mode Estimation

Sherlock
[de Kleer & Williams, IJCAI 89]



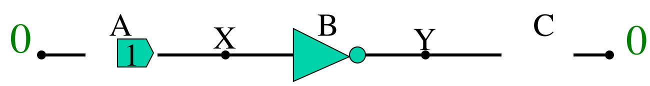
Inverter(i):

- G(i): Out(i) = not(In(i))
- S1(i): Out(i) = 1
- S0(i): Out(i) = 0
- U(i):

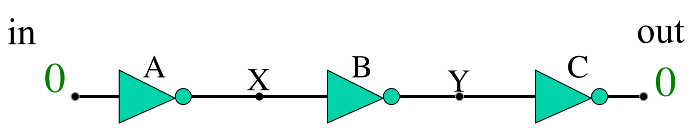
- Isolates unknown.
- Explains.

Nominal, Fault and Unknown Modes

Example Diagnoses



Diagnosis: [S1(A),G(B),U(C)]



Diagnoses: (42 of 64 candidates)

Fully Explained Failures
- [G(A),G(B),S0(C)]
- [G(A),S1(B),S0(C)]
- [S0(A),G(B),G(C)]

. . .

Partial Explained
- [G(A),U(B),S0(C)]
- [U(A),S1(B),G(C)]
- [S0(A),U(B),G(C)]

. . .

Fault Isolated, But Unexplained
- [G(A),G(B),U(C)]
- [G(A),U(B),G(C)]
- [U(A),G(B),G(C)]

Mode Estimation

- Mode, State, Observation Variables: X, Y, O
- Model: $\Phi(X,Y)$ = components + structure

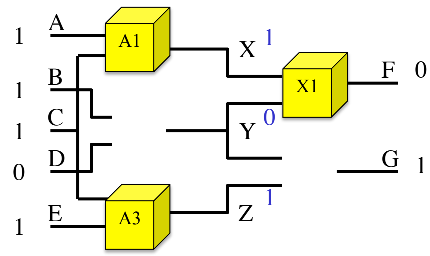
And(i):

G(i):

Out(i) = In1(i) AND In2(i)

U(i):

ALL components have "unknown Mode" U, whose assignment is never mentioned in any constraint.



Diagnosis = {A1=G, A2=U A3=G, X1=G. X2=U}

- Candidate C_i: Assignment of modes to X.
- Obs: Assignment to O.
- Diagnosis D_i: Candidate consistent with Model and Obs: $D_i \wedge \text{Obs} \wedge \Phi(X,Y)$ is satisfiable.

10/25/10 copyright Brian Williams, 2000-10 39

Mode Estimation

Given:

- Mode, State, Observation Variables: X, Y, O
- Model: $\Phi(X,Y)$ = components + structure

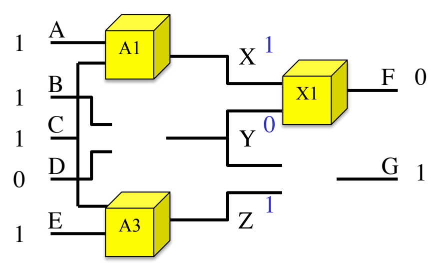
And(i):

G(i):

Out(i) = In1(i) AND In2(i)

U(i):

- All behaviors associated with modes.
- ALL components have "unknown Mode" U, whose assignment is never mentioned in any constraint.



Return:

$$D_{\Phi,obs} \equiv \{X \in D_X \mid \exists Y \in D_X \text{ st Obs} \wedge \Phi(X,Y)\}$$

10/25/10 copyright Brian Williams, 2000-10 40

Constraint Modeling and Consistency Testing

→ Propositional Logic:
- Complete: DPLL. (Titan)
- Incomplete: Unit propagation. (Livingstone/DS1)

- Finite Domain Constraints:
  - Complete: Backtracking with forward checking.
  - Incomplete: AC-3 / Waltz constraint propagation.

- Algebraic Constraints: (GDE/Sherlock/GDE+/XDE)
  - Complete: Gaussian Elimination.
  - Incomplete: Sussman/Steele Constraint Propagation.
    - Propagate newly assigned values through equations that mention the newly assigned variables.
    - To propagate, use assigned values of constraint to deduce unknown value(s) of constraint.

Models in Propositional State Logic

And(i):
- G(i):
  Out(i) = In1(i) AND In2(i)
- U(i):

$i{=}G \Rightarrow \{[In1(i){=}1 \wedge In2(i){=}1] \text{ iff } Out(i){=}1\}$

Or(i):
- G(i):
  Out(i) = In1(i) OR In2(i)
- U(i):

$i{=}G \Rightarrow \{[In1(i){=}1 \vee In2(i){=}1] \text{ iff } Out(i){=}1\}$

$X \in \{1,0\}$ $X{=}1 \vee X{=}0$

$\neg[X{=}1 \wedge X{=}0]$

$\neg(i{=}G) \vee \neg(In1(i){=}1) \vee Out(i){=}1$

$\neg(i{=}G) \vee \neg(In2(i){=}1) \vee Out(i){=}1$

$\neg(i{=}G) \vee \neg(In1(i){=}0) \vee \neg(In2(i){=}0) \vee Out(i){=}0$

Solution: Diagnosis as Hypothesis Testing

1. Generate candidates C_i, given symptoms.
   – Use Backtrack Search over mode variables X.
2. Test if candidates account for all symptoms.
   – Use DPLL to find assignment to Y such that $C_i \wedge Obs \wedge \Phi(X,Y)$ is satisfiable.

- Set of diagnoses should be complete.
- Set of diagnoses should exploit all available information.

Outline

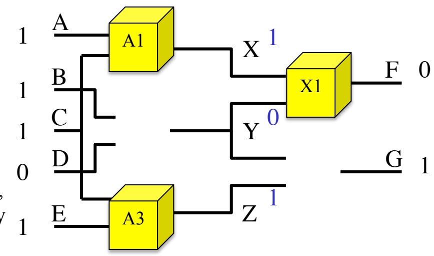
- Self-Repairing Agents
- Formulating Diagnosis
- Diagnosis from Conflicts
  – Kernels
  – Conflicts
  – Candidate Generation
  – Conflict Recognition

Mode Estimation

- Mode, State, Observation Variables: X, Y, O
- Model: Φ(X,Y) = components + structure

And(i):
G(i):
Out(i) = In1(i) AND In2(i)
U(i):

ALL components have "unknown Mode" U, whose assignment is never mentioned in any constraint.



$$D_{\Phi,obs} \equiv \{X \in D_X \mid \exists Y \in D_Y \text{ st } \text{Obs} \wedge \Phi(X,Y)\}$$

As more constraints are relaxed, candidates are more easily satisfied.

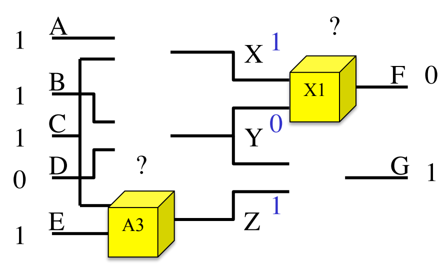
➔Typically an exponential number of diagnoses (mode estimates).

How do we encode solutions compactly?

Partial Diagnoses

Partial Diagnosis

{A1=U, A2=U, X2=U}



Partial Diagnosis:

A partial mode assignment M,
all of whose full extensions are diagnoses.

- M "removes all symptoms."

Extensions (Diagnoses):

{A1=U, A2=U, A3=G, X1=G, X2=U}

{A1=U, A2=U, A3=G, X1=U, X2=U}

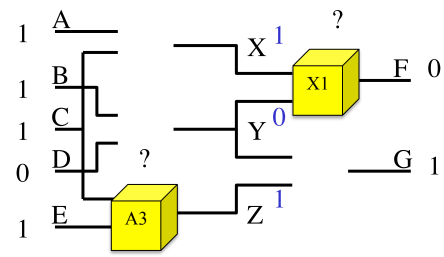
{A1=U, A2=U, A3=U, X1=G, X2=U}

{A1=U, A2=U, A3=U, X1=U, X2=U}

Partial Diagnoses

Partial Diagnosis

{A1=U, A2=U, X2=U}



Partial Diagnosis:

A partial mode assignment M, all of whose full extensions are diagnoses.

- M "removes all symptoms."
- M ∧ Φ ∧ Obs is consistent.
- M entails Φ ∧ Obs. *(implicant)*

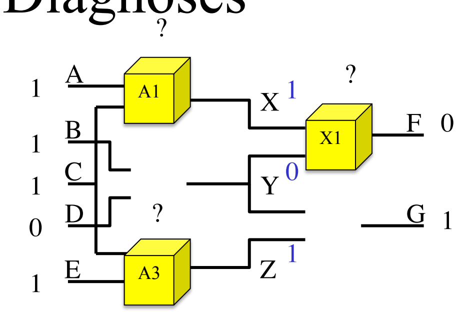
Extensions (Diagnoses):

{A1=U, A2=U, A3=G, X1=G, X2=U}
{A1=U, A2=U, A3=G, X1=U, X2=U}
{A1=U, A2=U, A3=U, X1=G, X2=U}
{A1=U, A2=U, A3=U, X1=U, X2=U}

Kernel Diagnoses

Kernel Diagnosis

{A2=U, X2=U}



Partial Diagnosis:

A partial mode assignment M, all of whose full extensions are diagnoses.

- M entails Φ ∧ Obs *(implicant)*

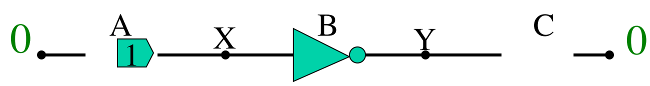
Kernel Diagnosis:

A partial diagnosis K, no subset of which is a partial diagnosis.

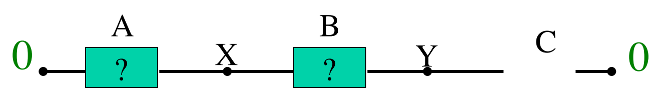
- K is a *prime implicant* of Φ ∧ Obs

Example Diagnoses

Sherlock
[de Kleer & Williams, IJCAI 89]



Diagnoses: [S1(A),G(B),U(C)] (42 total)



Kernel Diagnoses: [U(C)]
[S0(C)]
[U(B),G(C]
[S1(B),G(C)]
[U(A),G(B),G(C)]
[S0(A),G(B),G(C)]

Outline

- Self-Repairing Agents
- Formulating Diagnosis
- Diagnosis from Conflicts
  - Kernels
  - Conflicts
  - Candidate Generation
  - Conflict Recognition

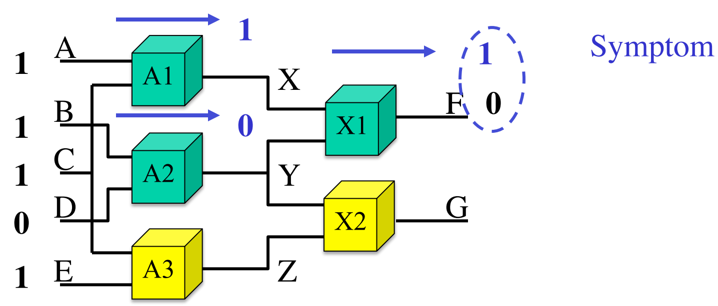
Diagnosis by Divide and Conquer

Given model Φ and observations Obs

1. Find all symptoms.
2. Diagnose each symptom separately
   (each generates a conflict).
3. Merge diagnoses
   (set covering → kernel diagnoses).

General Diagnostic Engine
[de Kleer & Williams, AIJ 87]
[Reiter AIJ 87]

Conflicts Explain How to Remove Symptoms



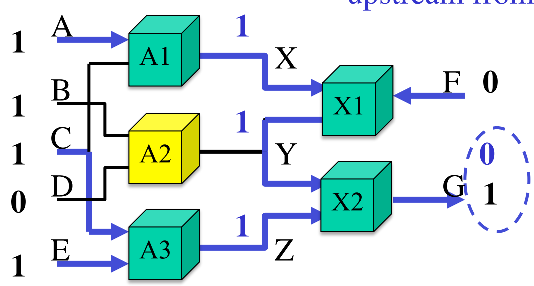
Symptom:
F is observed 0, but predicted to be 1 if A1, A2 and X1 are okay.

Conflict 1: {A1=G, A2=G, X1=G} is inconsistent.
→One of A1, A2 or X1 must be broken.

Conflict: An inconsistent partial assignment to mode variables X.

Second Conflict

Conflicting modes aren't always upstream from symptom.



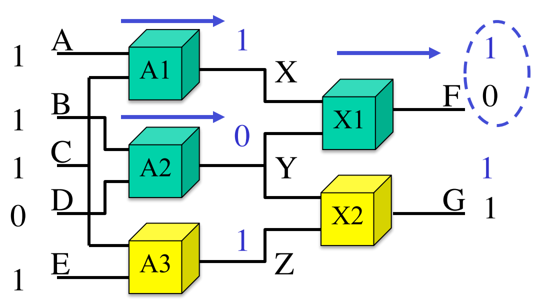
Symptom

Symptom: G is observed 1, but predicted **0.**

Conflict 2: {A1=G, A3=G, X1=G, X2=G} is inconsistent.

$\rightarrow$ One of A1, A3, X1 or X2 must be broken.

Summary: Conflicts



Symptom

Conflict: A partial mode assignment M that is inconsistent with the model and observations.

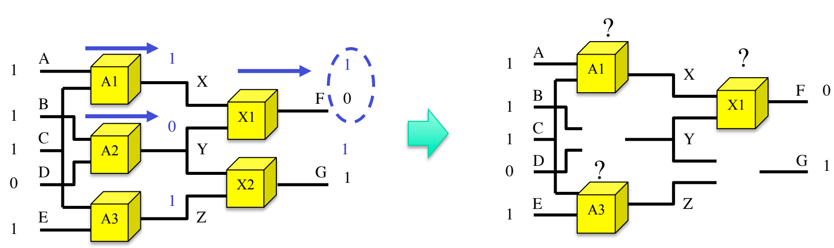
Properties:

- Every superset of a conflict is a conflict.
- Only need conflicts that are minimal under subset.
- $\Phi \wedge Obs \succ \neg M$

Outline

- Self-Repairing Agents
- Formulating Diagnosis
- Diagnosis from Conflicts
  - – Kernels
  - – Conflicts
  - – Candidate Generation
  - – Conflict Recognition

From Conflicts to Kernels



Constituent Kernel: An assignment a that "resolves" one conflict C_i.

{A2=U} resolves {A1=G, A3=G, X1=G, X2=G}.

a entails $\neg$ C_i.

Mapping Conflicts to Constituent Kernels

Conflict: {A1=G, A2=G, X1=G}

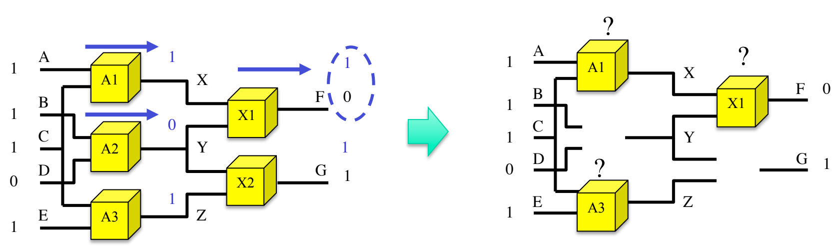
$\neg$(A1=G $\wedge$ A2=G $\wedge$ X1=G)

A1=U $\vee$ A2=U $\vee$ X1=U

Constituent Kernels: {A1=U, A2=U, X1=U}

$$Constituent\_Kernels(c) \equiv \{\neg l_i \mid c \equiv \neg(\wedge l_i)\}$$

From Conflicts to Kernels



Constituent Kernel: An assignment a that “resolves” one conflict C_i.

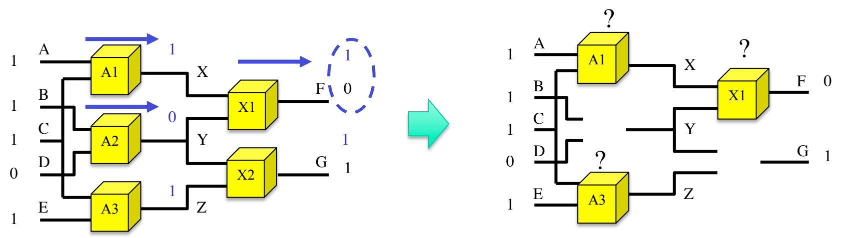
{A2=U} resolves {A1=G, A3=G, X1=G, X2=G}.

Kernel: A minimal set of assignments A that “resolve” all conflicts C.

{A2=U, X2=U} resolves {A1=G, A3=G, X1=G, X2=G}, and

{A2=U, X2=U} resolves {A1=G, A2=G, X1=G}.

From Conflicts to Kernels



Constituent Kernel: An assignment a that "resolves" a conflict C_i.
a entails $\neg$ C_i.

Kernel: A minimal set of assignments A that "resolves" all conflicts C.
A entails $\neg$ C_i for all C_i in C.

⇨ Map constituent kernels to kernels by minimal set covering.

10/25/10 copyright Brian Williams, 2000-10 59

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses =

"Smallest" sets of modes that remove all conflicts. 60

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses = {A1=U}

1. Compute cross product.
2. Remove supersets.
   - New superset Old.
   - Old superset New.

"Smallest" sets of modes that remove all conflicts.

61

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses = ~~{A1=U, A3=U}~~
{A1=U}

1. Compute cross product.
2. Remove supersets.
   - New superset Old.
   - Old superset New.

"Smallest" sets of modes that remove all conflicts.

62

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses = ~~{A1=U, X1=U}~~
{A1=U}

1. Compute cross product.
2. Remove supersets.
   - New superset Old.
   - Old superset New.

"Smallest" sets of modes that remove all conflicts.

63

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses = ~~{A1=U, X2=U}~~
{A1=U}

1. Compute cross product.
2. Remove supersets.
   - New superset Old.
   - Old superset New.

"Smallest" sets of modes that remove all conflicts.

64

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses =
{A2=U, X2=U}
{A2=U, X1=U}
{A2=U, A3=U}
~~{A2=U, A1=U}~~
{A1=U}

1. Compute cross product.
2. Remove supersets.
   - New superset Old.
   - Old superset New.

"Smallest" sets of modes that remove all conflicts.

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses =
{X1=U}
~~{X1=U, A3=U}~~
~~{X1=U, A1=U}~~
{A2=U, X2=U}
~~{A2=U, X1=U}~~
{A2=U, A3=U}
{A1=U}

1. Compute cross product.
2. Remove supersets.
   - New superset Old.
   - Old superset New.

Generate Kernels From Conflicts

{A1=G, A2=G, X1=G} Conflict 1.

{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Kernel Diagnoses = {X1=U}
{A2=U, X2=U}
{A2=U, A3=U}
{A1=U}

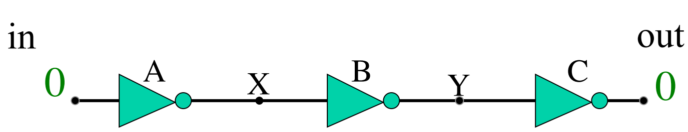
1. Compute cross product.
2. Remove supersets.
   - New superset Old.
   - Old superset New.

"Smallest" sets of modes that remove all conflicts.


```
Candidate-Generation(Conflicts){
 // Compute all minimal coverings of Conflicts
  Next_Kernels = {};
  For each c in Conflicts{
    Kernels = Next_Kernels;
    Next_Kernels = {};
    For each c' in Constituent_Kernels(c) {
      For each k in Kernels {
        Next_Kernels
          = Add_Kernel(c'∪k, Next_Kernels)
  return Next_Kernels}}}
}
```

Add-Kernel(Kernel, Kernels){
// *Add Kernel to Kernels while preserving minimality.*
If $\exists k \in Kernels.\ k \subseteq Kernel$
Then return Kernels
Else {
New_Kernels = {};
For each k in Kernels{
Unless $Kernel \subseteq k$
Add_To_End(k, New_Kernels)};
return $\{Kernel\} \cup New\_Kernels$
}}

10/25/10 copyright Brian Williams, 2000-10 69



Diagnoses: (42 of 64 candidates)

Fully Explained Failures
- [G(A),G(B),S0(C)]
- [G(A),S1(B),S0(C)]
- [S0(A),G(B),G(C)]

. . .

Partial Explained
- [G(A),U(B),S0(C)]
- [U(A),S1(B),G(C)]
- [S0(A),U(B),G(C)]

. . .

Fault Isolated, But Unexplained
- [G(A),G(B),U(C)]
- [G(A),U(B),G(C)]
- [U(A),G(B),G(C)]

10/25/10 copyright Brian Williams, 2000-10 70

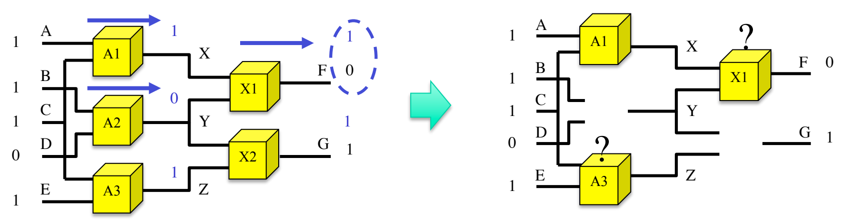
Generate Kernels from Conflicts

- [G(C), S0(C), U(C)]
- [G(B), S1(B), U(B), S1(C), S0(C), U(C)]
- [G(A), S0(A), U(A), S1(B), S0(B), U(B), S1(C), S0(C), U(C)]
- [S1(A), S0(A), U(A), S1(B), S0(B), U(B), S1(C), S0(C), U(C)]

- [U(C)]
- [S0(C)]
- [U(B),G(C]
- [S1(B),G(C)]
- [U(A),G(B),G(C)]
- [S0(A),G(B),G(C)]

10/25/10 copyright Brian Williams, 2000-10 71

Summary: Mapping Conflicts to Kernels



Conflict C_i: A partial mode assignment, to X, that is inconsistent with model Φ and obs.

$C_i \wedge \Phi \wedge$ obs is inconsistent $\Phi \wedge$ obs entails $\neg C_i$

Constituent Kernel: An assignment a that resolves one conflict C_i.

a entails $\neg$ C_i

Kernel: A minimal partial assignment that resolves all conflicts C.

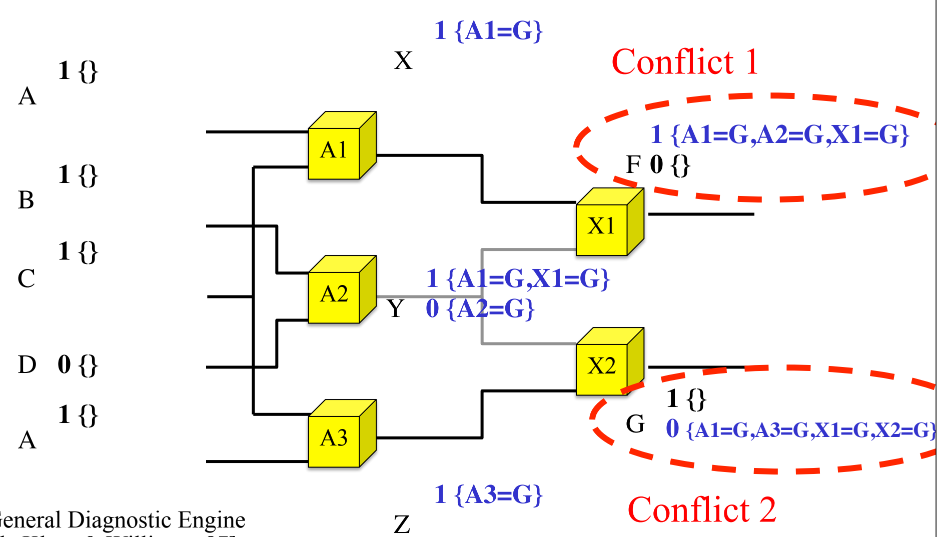
A entails $\neg$ C_i for all C_i in C

72

Outline

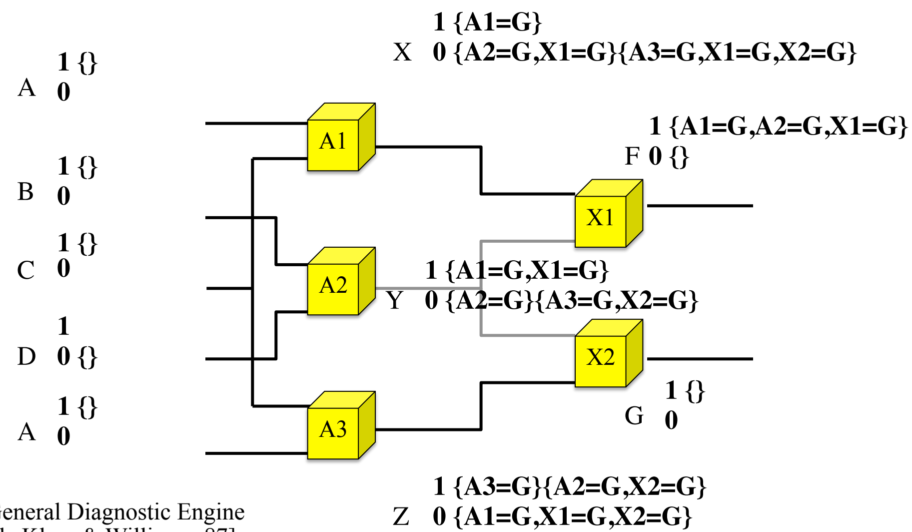
- Self-Repairing Agents
- Formulating Diagnosis
- Diagnosis from Conflicts
  - Kernels
  - Conflicts
  - Candidate Generation
  - Conflict Recognition

Recognizing Conflicts within GDE



General Diagnostic Engine
[de Kleer & Williams, 87]

Recognizing Conflicts within GDE



General Diagnostic Engine
[de Kleer & Williams, 87]

Summary: Mode Estimation

- A failure is a discrepancy between the model and observations of an artifact.
- Mode estimation supports diagnosis of unknown failures, multiple faults, partial explanation and execution monitoring.
- Mode estimates are encoded compactly using kernels.
- Symptoms are used to recognize conflicts, which are merged to produce kernels.
- Conflict-directed search is at the foundation of fast satisfiability and optimization.

Outline

- Self-Repairing Agents
- Formulating Diagnosis
- Diagnosis from Conflicts
- Appendix: Single Fault Diagnosis

Single Fault Diagnosis

The single fault diagnoses are the intersections of the conflict constituent kernels.

{A1=G, A2=G, X1=G} Conflict 1.

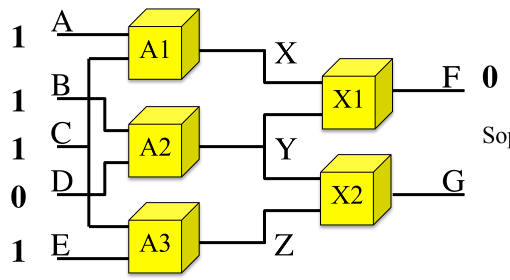
{A1=G, A3=G, X1=G, X2=G} Conflict 2.

{A1=U, A2=U, X1=U} constituents of Conflict 1.

{A1=U, A3=U, X1=U, X2=U} constituents of Conflict 2.

Single Fault Diagnoses = {A1=U} {X1=U}

Finding Single Fault Diagnosis



HT [Davis & Shrobe]
Dart [Genesereth]
Sophie [de Kleer & Brown]
Early 80's

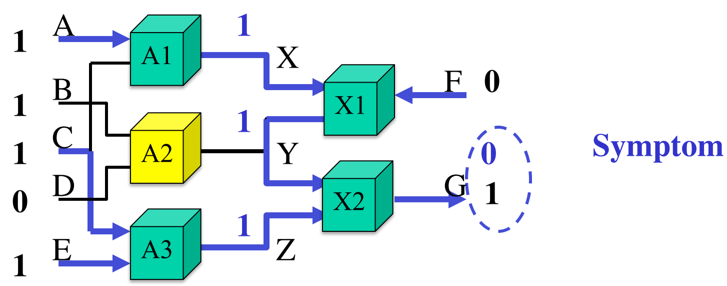
1. Generate initial candidates:
   - Assume all components okay and test consistency.
   - If inconsistent, conflict kernels denote single fault candidates.
2. Check consistency of each candidate:
   - Prune candidate if superset of a conflict.
   - Else check consistency and record conflict if inconsistent.

Procedure Single_Fault_w_Conflicts(Md, M, Obs)

Input: A model Md, Mode variables M, and observations Obs.
Output: A set of consistent, single fault diagnoses.

All_Good ← { M_i=G | $M_i \in$ M}; Assume all components are okay,
Conflict ← *Test_Candidate(All_Good, Md, Obs)*
If Conflict = Consistent
  Return All_Good
Else
  Cands Generate single fault candidates
    ← {{M_i=U} ∪ Z=G | M_i=G ∈ Conflict, Z = M - {M_i} };
  Diagnoses ← Test_Candidates(Cands, Md, Obs)
  Return Diagnoses

Generate Candidates From Symptom

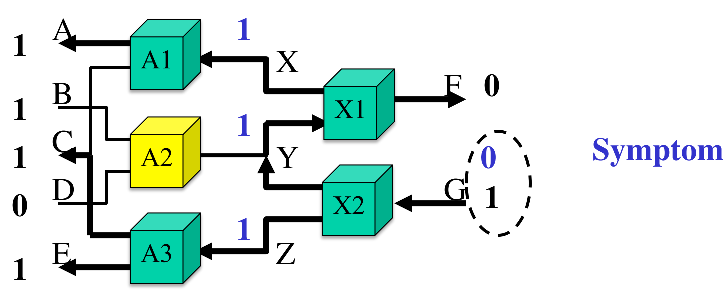


Symptom: G is observed 1, but predicted **0**

Conflict: {A1=G, A3=G, X1=G, X2=G} is inconsistent

Candidates: {{A1=U…}, {A3=U…}, {X1=U...}, {X2=U…}}

Generate Candidates From Symptom



Symptom: G is observed 1, but predicted **0**

Conflict: {A1=G, A3=G, X1=G, X2=G} is inconsistent

Candidates: {{A1=U…}, {A3=U…}, {X1=U...}, {X2=U…}}

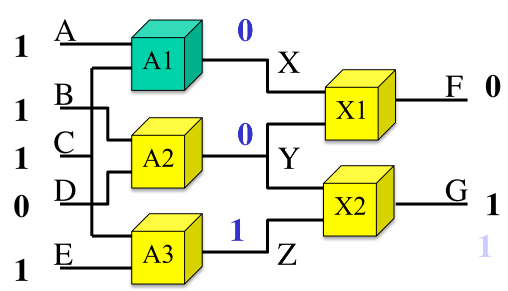
Procedure Single_Fault_Test_Candidates(C,M, Obs)

Input: Candidates C, Model Md, Observation Obs
Output: The set of consistent single-fault diagnoses.

Diagnoses ← {}, Conflicts ← {}
For each C_i in C
  If C_i is a superset of some Conflict_j in Conflicts
    Then inconsistent candidate C_i, ignore.
  Else Conflict_i = *Test_Candidate(C_i, M, Obs)*
    If Conflict_i = Consistent
      Then add C_i to Diagnoses
    Else add Conflict_i to Conflicts
  return Diagnoses

Test Candidates, Collect Conflicts

Candidates: {{~~A1=U…~~}, {A3=U…}, {X1=U…}, {X2=U…}}
Diagnoses: {{A1=U…}}



- First candidate {A1=U, …}
- Suspend A1's constraints
- Test consistency → consistent

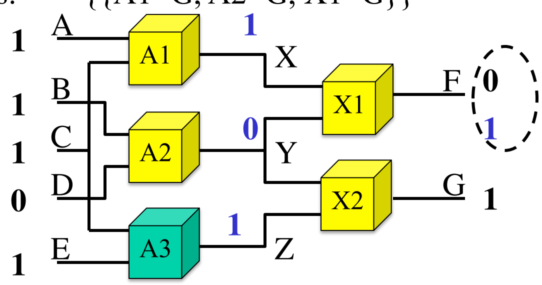
- Add to diagnoses

Test Candidates, Collect Conflicts

Candidates: {{~~A3=U…~~}, {X1=U…}, {X2=U…}}

Diagnoses: {{A1=U…}}

Conflicts: {{A1=G, A2=G, X1=G}}



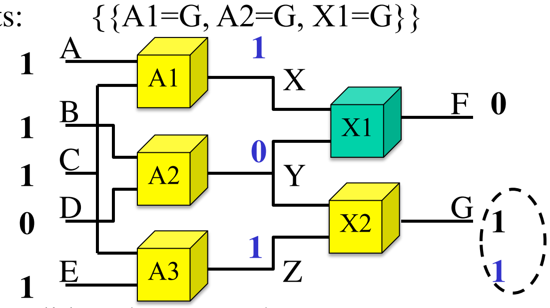
- Second candidate {A3=U, …}
- Suspend A3's constraints
- Test consistency → inconsistent
- Extract conflict
  {A1=G, A2=G, X1=G}
- Intersect candidates

Test Candidates, Collect Conflicts

Candidates: {{~~X1=U…~~}, {X2=U…}}

Diagnoses: {{A1=U…}}

Conflicts: {{A1=G, A2=G, X1=G}}



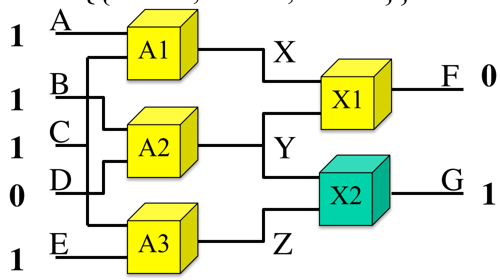
- Third candidate {X1=U, …}
- Superset of conflict? → No, since X1 = U, not X1=G
- Suspend X1's constraints
- Test consistency → consistent

Test Candidates, Collect Conflicts

Candidates: {{~~X2=U~~…}}

Diagnoses: {{A1=U…}, {X1=U…}}

Conflicts: {{A1=G, A2=G, X1=G}}



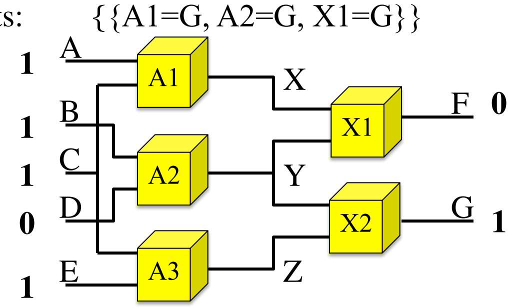
- Fourth candidate {X2=U, …}
- Superset of conflict? ⟶ Yes, since A1=G, A2=G and X1=G
- Eliminate candidate

Test Candidates, Collect Conflicts

Candidates: {}

Diagnoses: {{A1=U…}, {X1=U…}}

Conflicts: {{A1=G, A2=G, X1=G}}



- Return diagnoses→ A1 or X1 broken