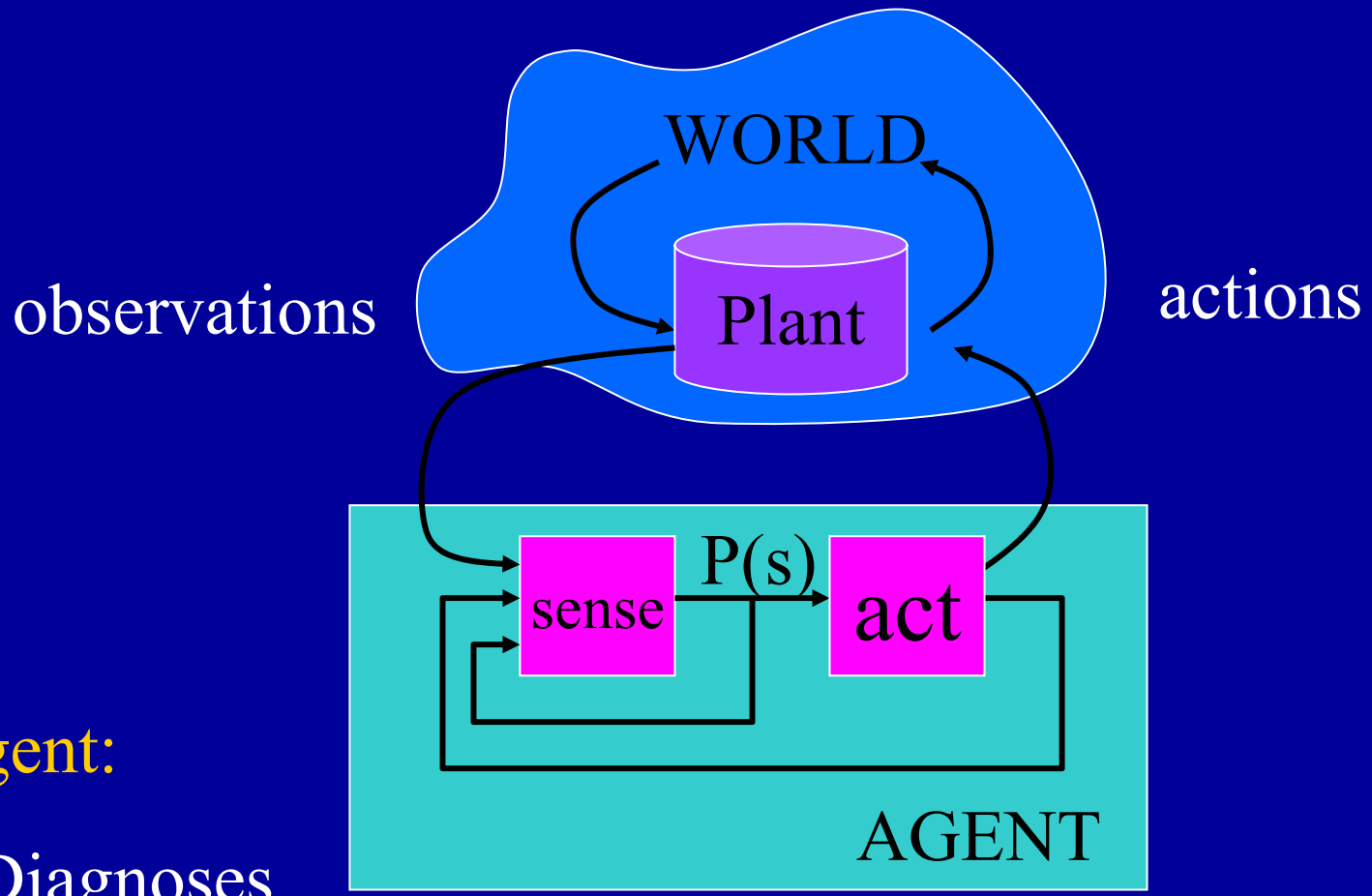
Model-based Diagnosis

Brian C. Williams
16.410-13
November 24th 2004

courtesy of JPL

Assignment

- Reading
  - Model-based Diagnosis – Lecture notes
  - Propositional Logic AIMA Chapter 6
- Problem Set
  - Due Friday, December 3rd



Diagnostic Agent:

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

Symptom-directed

Outline

- Model-based diagnosis
- Defining diagnoses
- Searching for diagnoses
- Appendix

Hidden Failures Require Reasoning from a Model: STS-93

Symptoms:

- **Engine temp sensor high**
- **LOX level low**
- **GN&C detects low thrust**
- **H2 level possibly low**

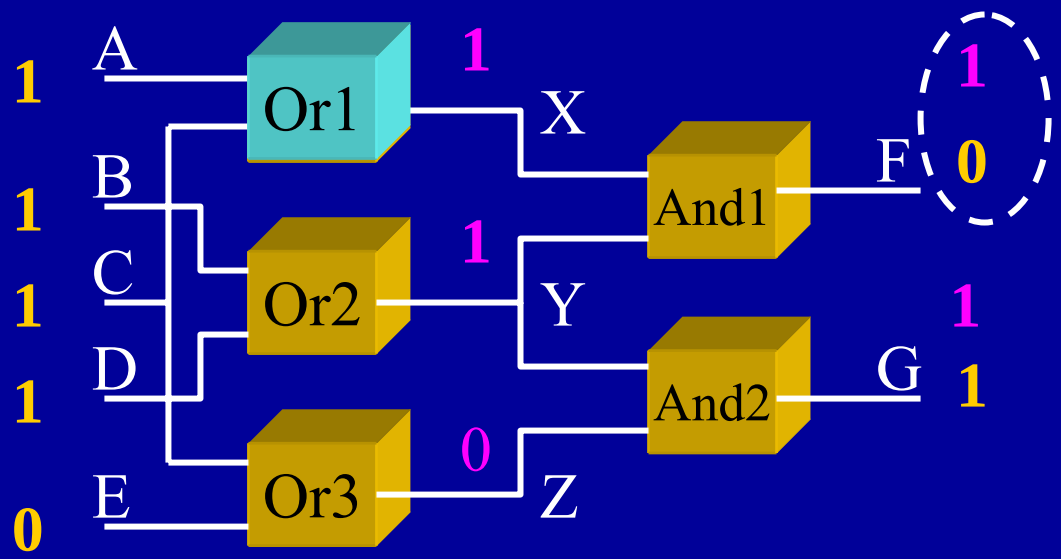
Problem: Liquid hydrogen leak

Effect:

- **LH2 used to cool engine**
- **Engine runs hot**
- **Consumes more LOX**

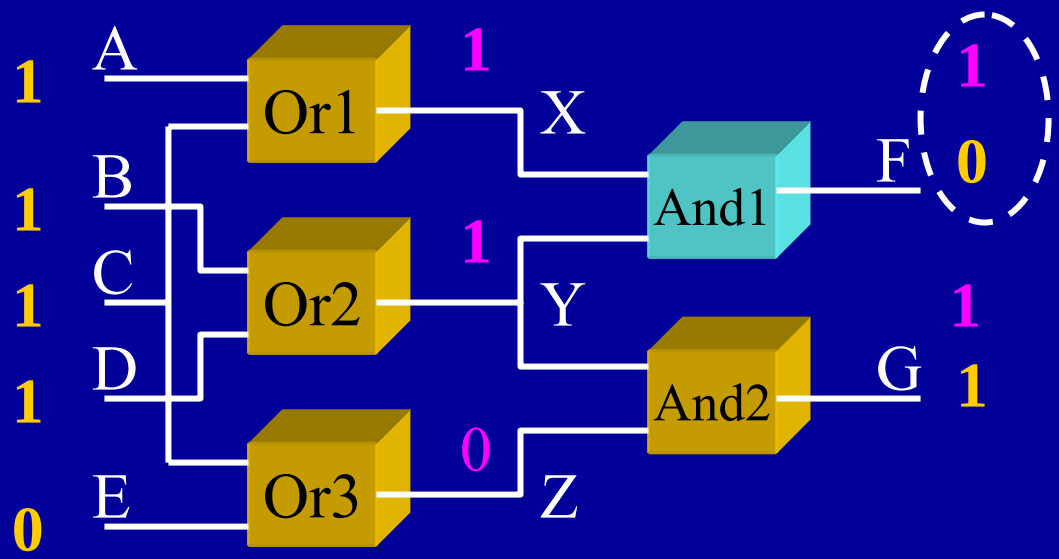
Model-based Diagnosis

Given a system and observations with symptomatic behavior, and a model of the system, find diagnoses that account for the symptoms.



Model-based Diagnosis

Given a system and observations with symptomatic behavior, and a model of the system, find diagnoses that account for the symptoms.



Diagnosis as Hypothesis Testing

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

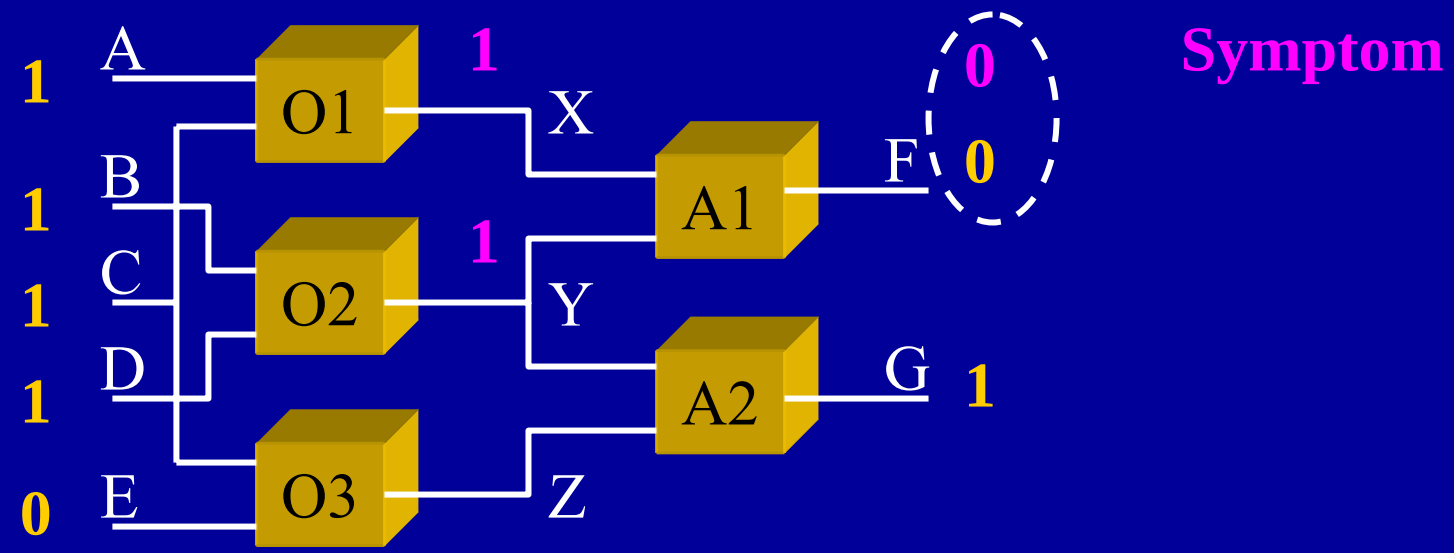
- Set of diagnoses should be complete.
- Set of diagnoses should exploit all available information.

Outline

- Model-based diagnosis
- Defining diagnoses
  - Explaining failures
  - Handling unknown failures
- Searching for diagnoses
- Appendix

How Should Diagnoses Account for Symptoms?

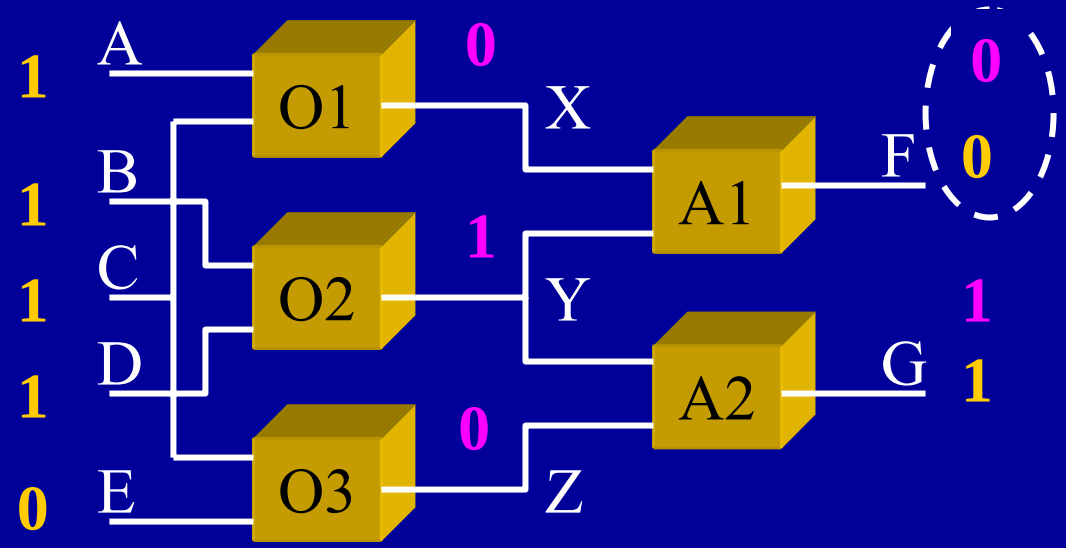
Abductive Diagnosis: Given symptoms, find diagnoses that predict observations.



Assume Exhaustive Fault Models:

How Should Diagnoses Account for Symptoms?

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

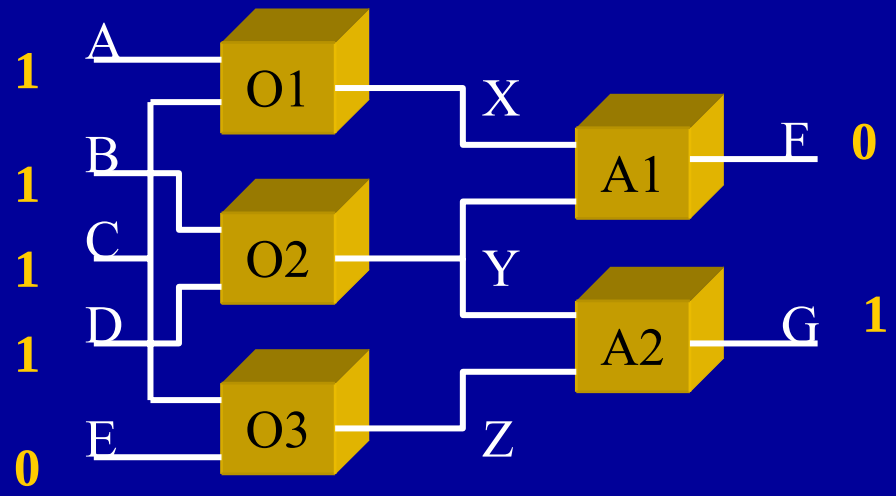


- O1's output is stuck to 0
  - Output shorted to ground

Abductive, Model-based Diagnosis

Or(i):

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

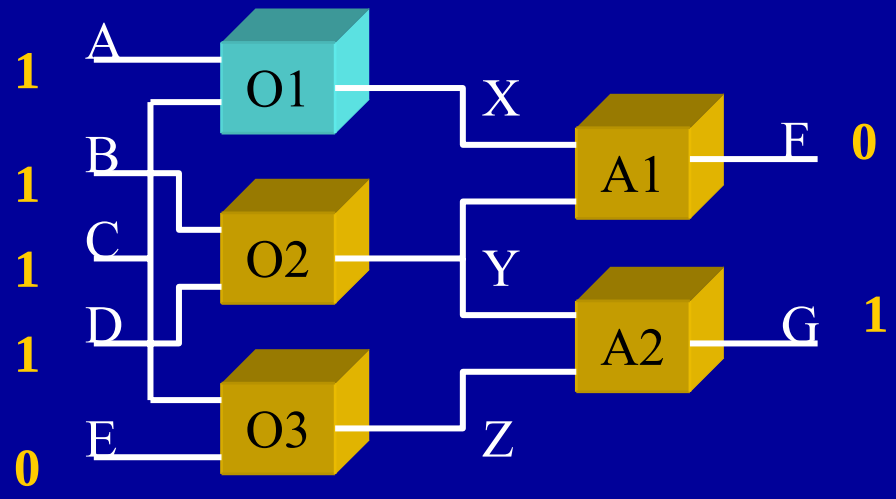


- Model
  - Structure
  - Model of normal behavior for each component
  - Model for **every** component failure mode
- Observations
  - Inputs and Response

Abductive, Model-based Diagnosis

Or(i):

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

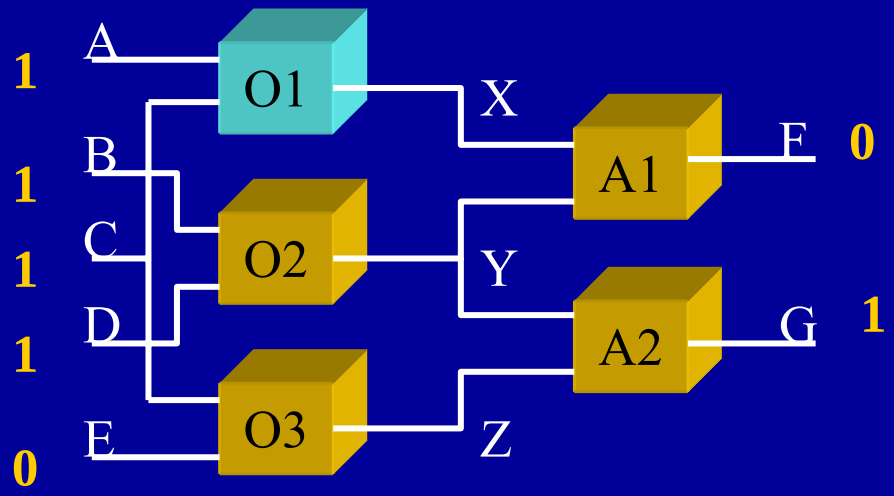


- X — mode variables, one for each component c
- D_c — modes of component c = domain of $x_c \in X$
- Y — model variables
- M(X, Y) — model constraints
- O — observable variables $O \subseteq Y$
  - Partitioned into Inputs I and Responses R

Abductive, Model-based Diagnosis

Or(i):

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



Candidate = {A1=G, A1=G, M1=G, A2=G, M3=G}

Diagnosis = {A1=G, A2=S0, O1=G, M2=G, M3=G}

- Obs <In, Resp>: Assignment to I and R, respectively
- Candidate C_i: Assignment of modes to X
- Diagnosis D_i: A candidate such that
  $D_i \wedge In \wedge M(X,Y)$ predicts (entails) Resp

Abductive Diagnosis by Generate and Test

Given: Correct and exhaustive fault models for each component.

Generate: Consider each mode assignment as a candidate.

Test:

1. Simulate candidate, given inputs.
2. Compare to observations
   - Disagree: Discard
   - Agree: Keep
   - No prediction: Discard
3. Exonerate component if none of its fault modes agree

Problem:

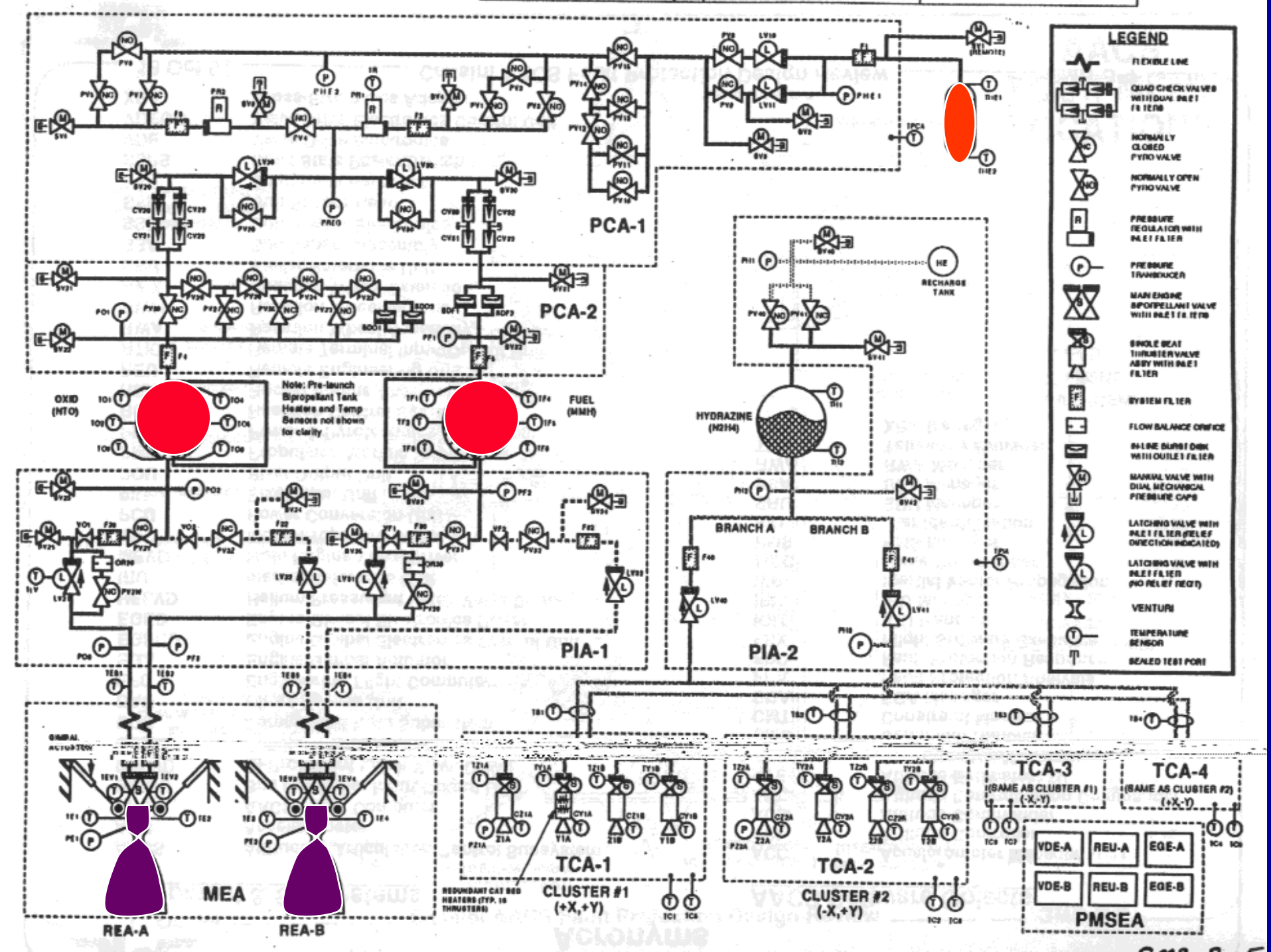
- Fault models are often incomplete
- May incorrectly exonerate faulty components

Outline

- Model-based diagnosis
- Defining diagnosis
  - Explaining failures
  - Handling unknown failures
- Searching for diagnoses
- Appendix

Issue: Failures are Often Novel

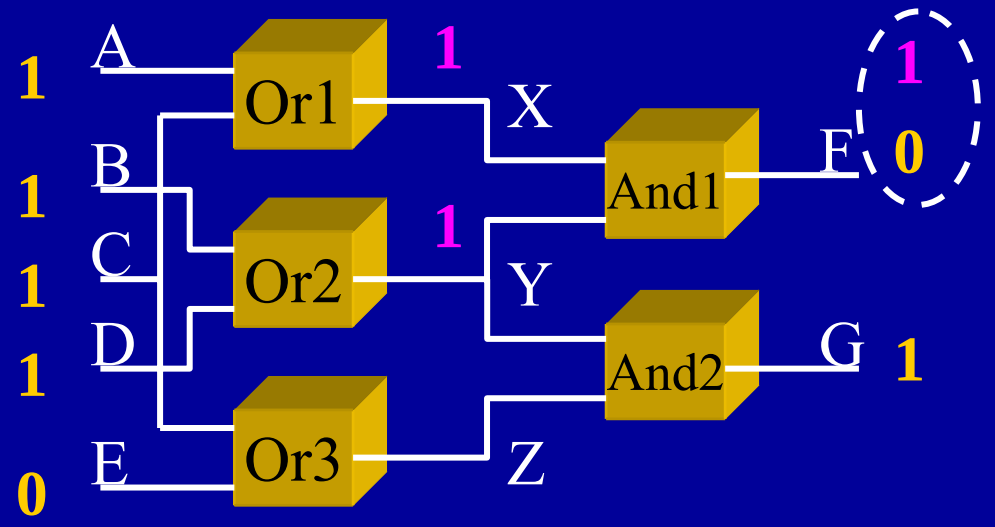
- **Mars Observer**
- **Mars Climate Orbiter**
- **Mars Polar Lander**
- **Deep Space 2**


LEGEND
FLEXIBLE LINE
QUAD CHECK VALVE WITH DUAL INLET FILTER
NORMALLY CLOSED PYRO VALVE
NORMALLY OPEN PYRO VALVE
PRESSURE REGULATOR WITH INLET FILTER
PRESSURE TRANSDUCER
MAIN ENGINE BIPROPELLANT VALVE WITH INLET FILTER
SINGLE SEAT THRUSTER VALVE ASSY WITH INLET FILTER
SYSTEM FILTER
FLOW BALANCE ORIFICE
IN-LINE BURST DISK WITH OUTLET FILTER
MANUAL VALVE WITH DUAL MECHANICAL PRESSURE CAPS
LATCHING VALVE WITH INLET FILTER (RELIEF DIRECTION INDICATED)
LATCHING VALVE WITH INLET FILTER (NO RELIEF REQ'T)
VENTURI
TEMPERATURE SENSOR
SEALED TEST PORT
PCA-1
PCA-2
PIA-1
PIA-2
OXID (NTO)
FUEL (MMH)
HYDRAZINE (N2H4)
RECHARGE TANK
Note: Pre-launch Bipropellant Tank Heaters and Temp Sensors not shown for clarity
BRANCH A
BRANCH B
MEA
REA-A
REA-B
TCA-1
CLUSTER #1 (+X,+Y)
TCA-2
CLUSTER #2 (-X,+Y)
TCA-3
(SAME AS CLUSTER #1) (-X,-Y)
TCA-4
(SAME AS CLUSTER #2) (+X,-Y)
VDE-A
REU-A
EGE-A
VDE-B
REU-B
EGE-B
PMSEA

How Should Diagnoses Account for Novel Symptoms?

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

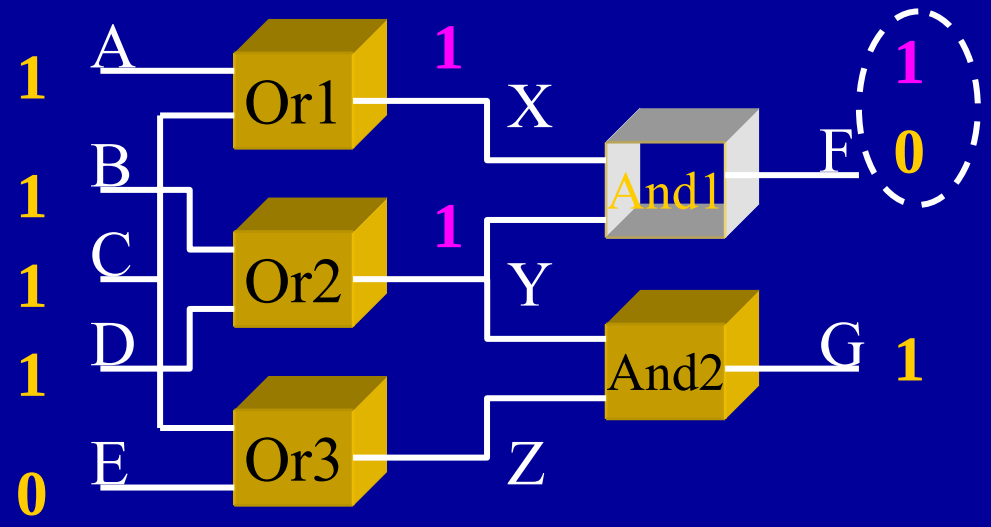
Suspending Constraints: For novel faults make no presumption about faulty component behavior.



How Should Diagnoses Account for Novel Symptoms?

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

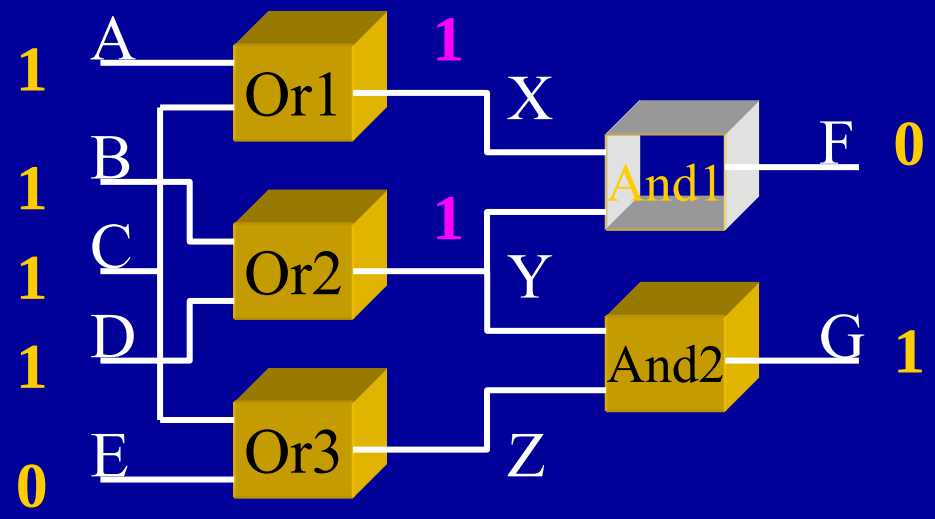
Suspending Constraints: For novel faults make no presumption about faulty component behavior.



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.

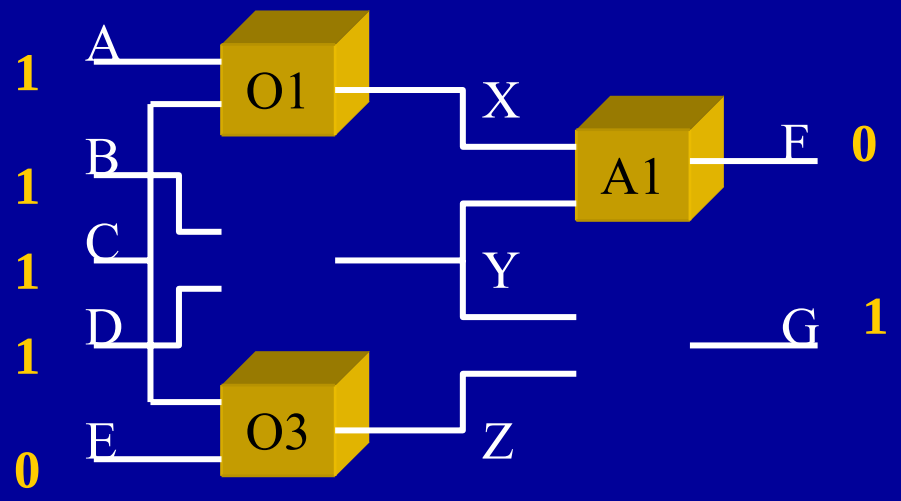


Consistency-based Diagnosis

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 C



Diagnosis = {A1=G, A2=U O1=G, O2=U, O3=G}

- Obs: Assignment to O
- Candidate C_i: Assignment of modes to X
- Diagnosis D_i: A candidate such that
  $D_i \wedge Obs \wedge C(X,Y)$ is satisfiable.

Testing Consistency

→ Propositional Logic

- DPLL
- Just unit propagation (incomplete)

•Finite Domain Constraints

- Backtrack w forward checking
- Waltz constraint propagation (incomplete)

• Algebraic Constraints

- Gaussian Elimination
- Sussman/Steele Constraint Propagation (incomplete)
  - Propagate newly assigned values through equations mentioning variables.
  - To propagate, use assigned values of constraint to deduce unknown value(s) of constraint.

Encoding Models In Propositional Logic

And(i):

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

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

- U(i):

$\neg(i=G) \vee \neg(In1(i)=1) \vee \neg(In2(i)=1) \vee Out(i)=1$

Or(i):

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

$\neg(i=G) \vee \neg(In1(i)=1) \vee Out(i)=1$
$\neg(i=G) \vee \neg(In2(i)=1) \vee Out(i)=1$

- U(i):

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

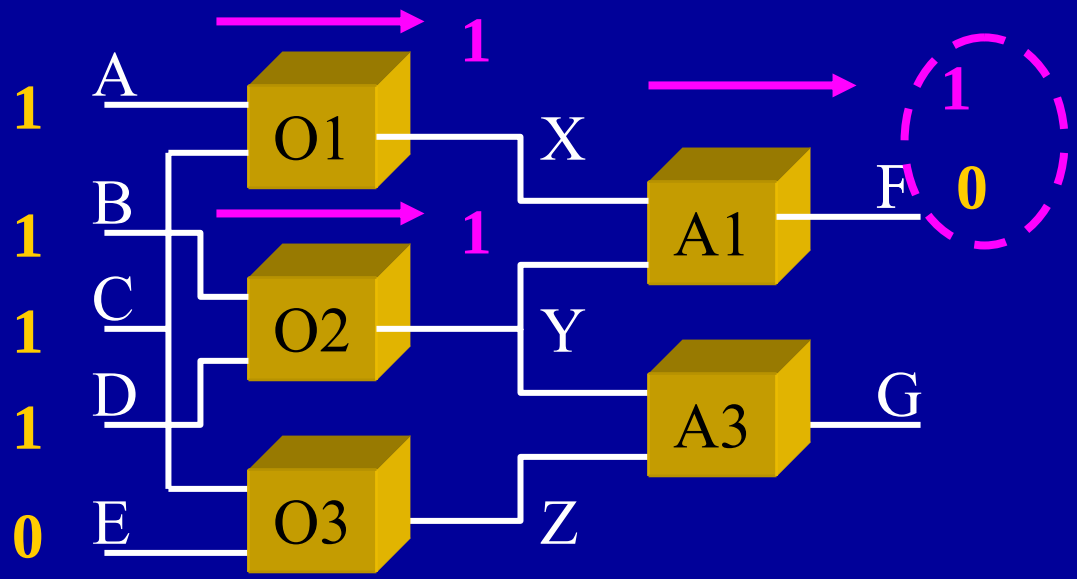
$X \in \{1,0\}$ $\quad X=1 \vee X=0$

$\neg X=1 \vee \neg X=0$

Outline

- Model-based diagnosis
- Defining diagnosis
  - Explaining failures
  - Handling unknown failures
- Searching for diagnoses
  - Conflict learning
  - Single-fault diagnosis
- Appendix
  - Multiple-fault diagnosis

Learning Conflicts From Symptoms



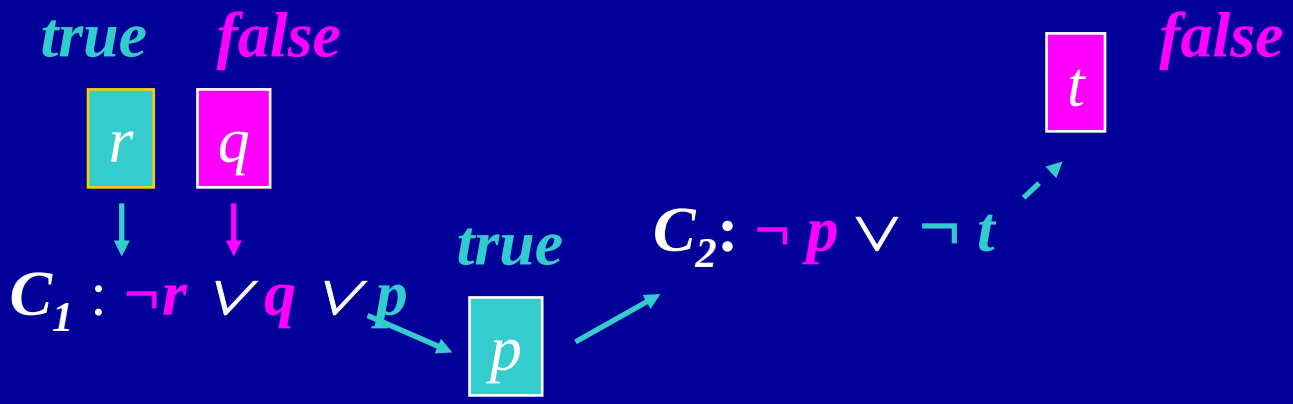
Symptom:

F is observed 0, but should be **1** if O1, O2 and A1 are okay.

Conflict: {A1=G, O1=G, O2=G} is inconsistent

→ At least A1=U, O1=U, or O2=U

Find Symptom Using Unit Propagation



procedure *propagate*(***C***) // ***C*** is a clause

if all literals in ***C*** are false except **l**, and **l** is unassigned

then assign true to ***l*** and

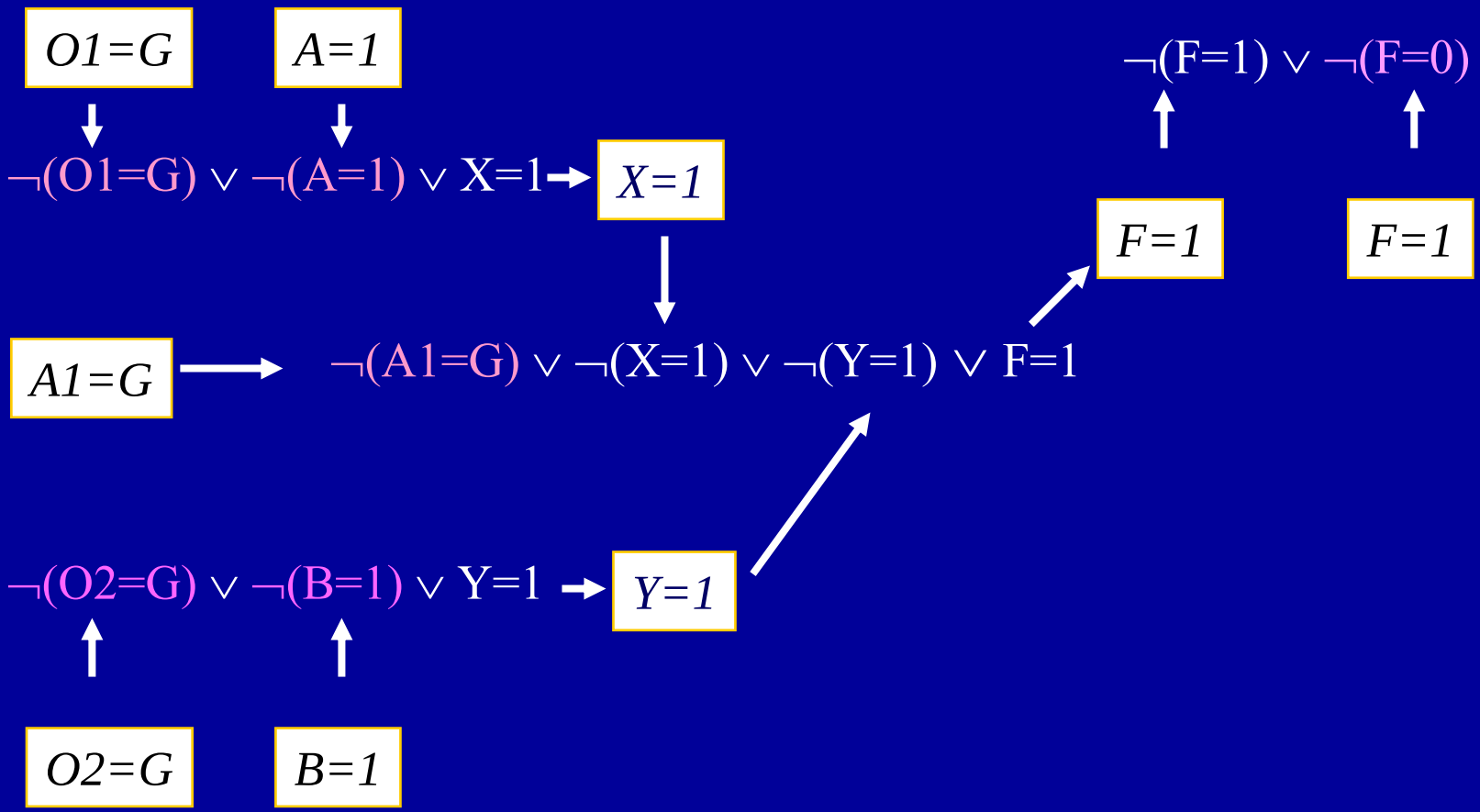
record **C** as a support for **l** and

for each clause ***C'*** mentioning "not **l**",

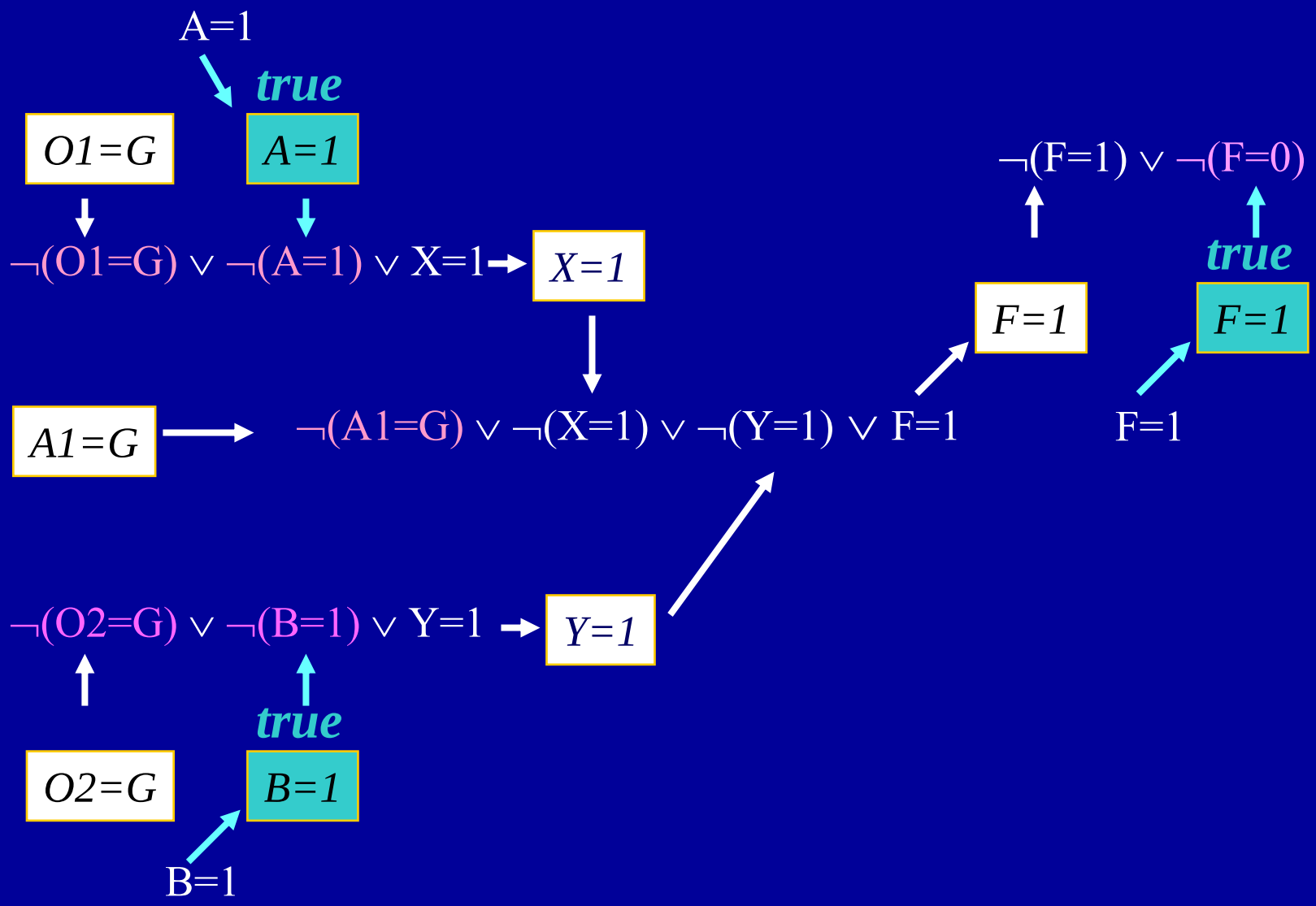
propagate(***C'***)

end *propagate*

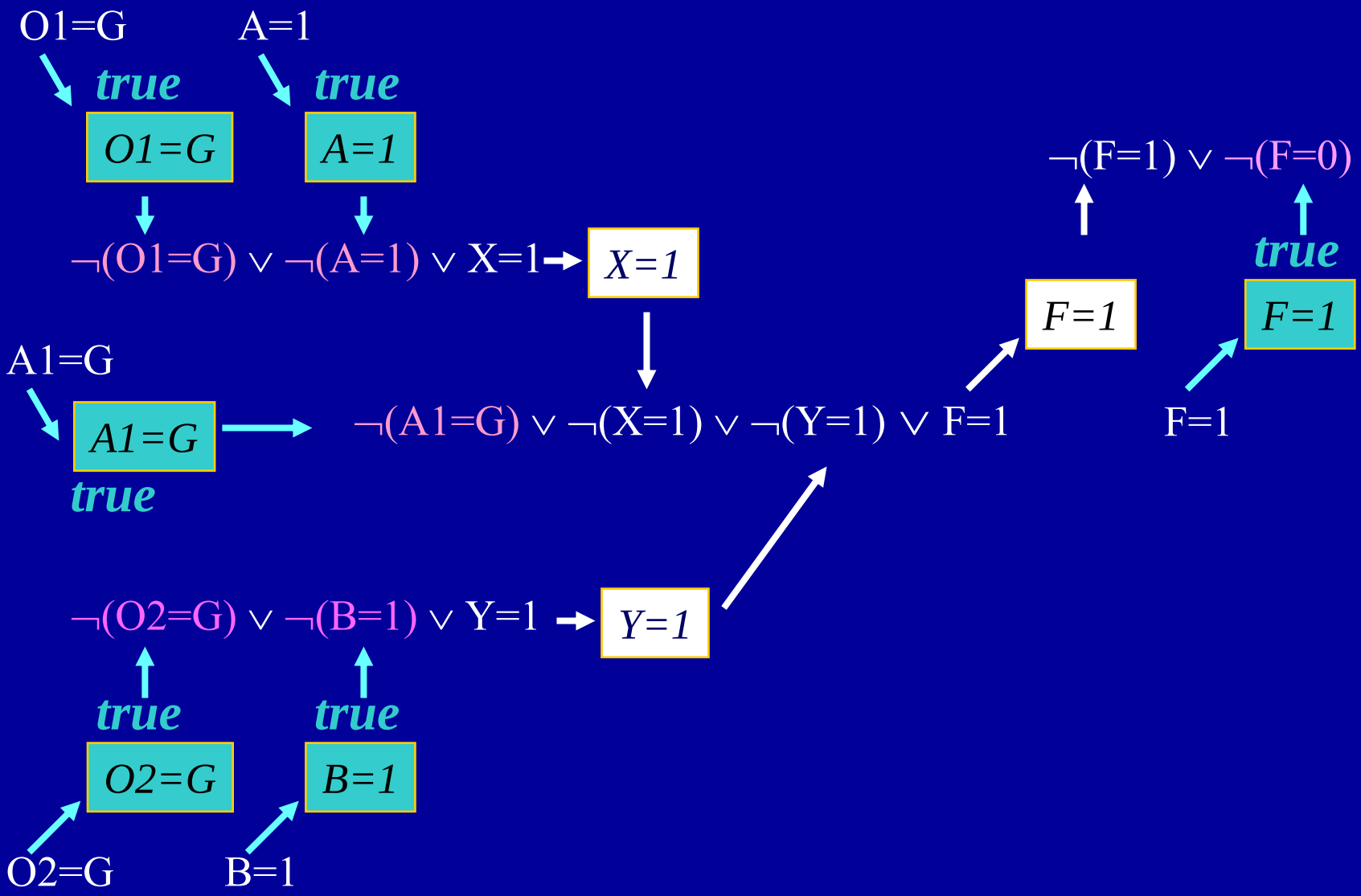
Find Symptom Using Unit Propagation



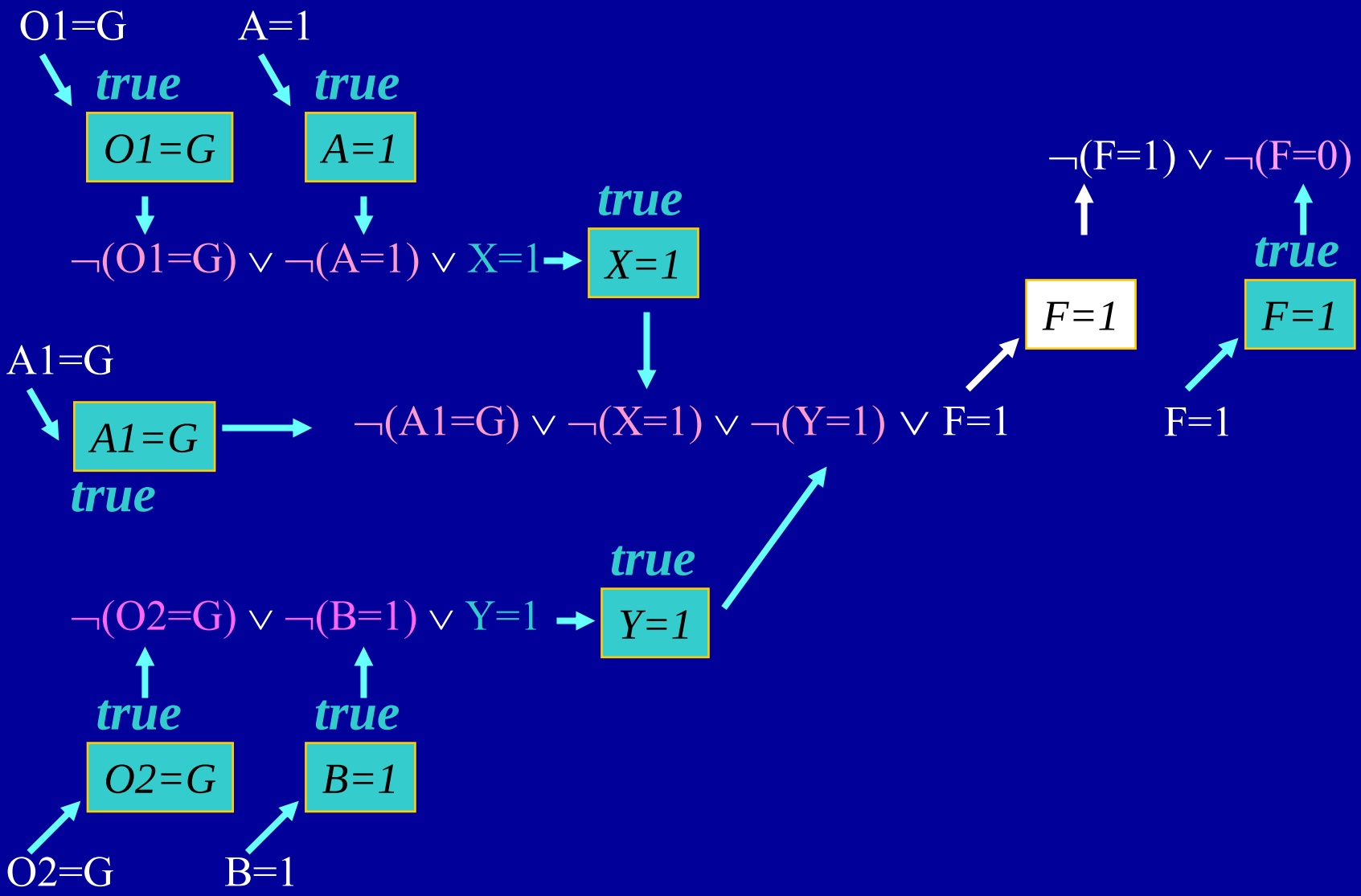
Find Symptom Using Unit Propagation



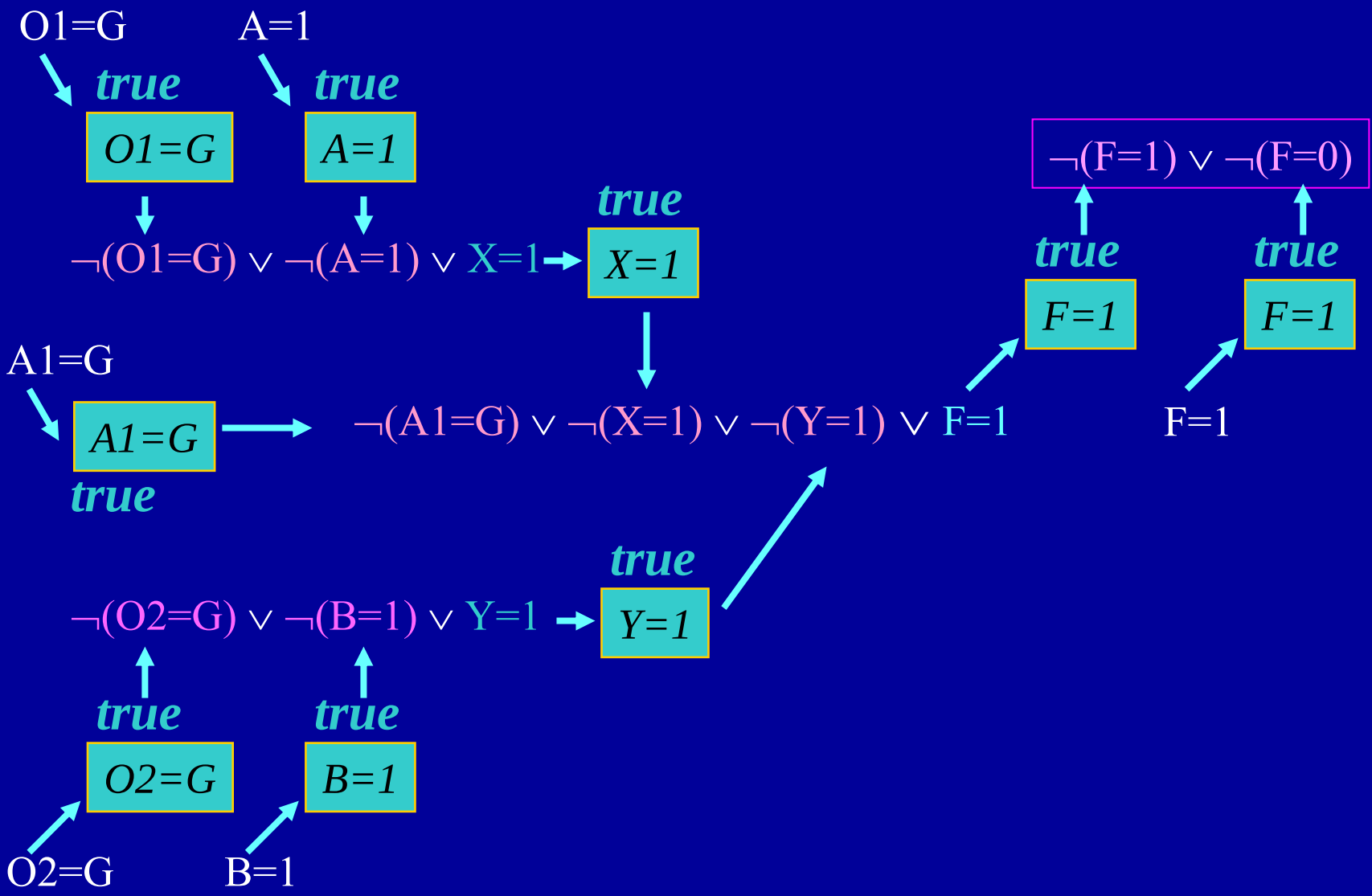
Find Symptom Using Unit Propagation



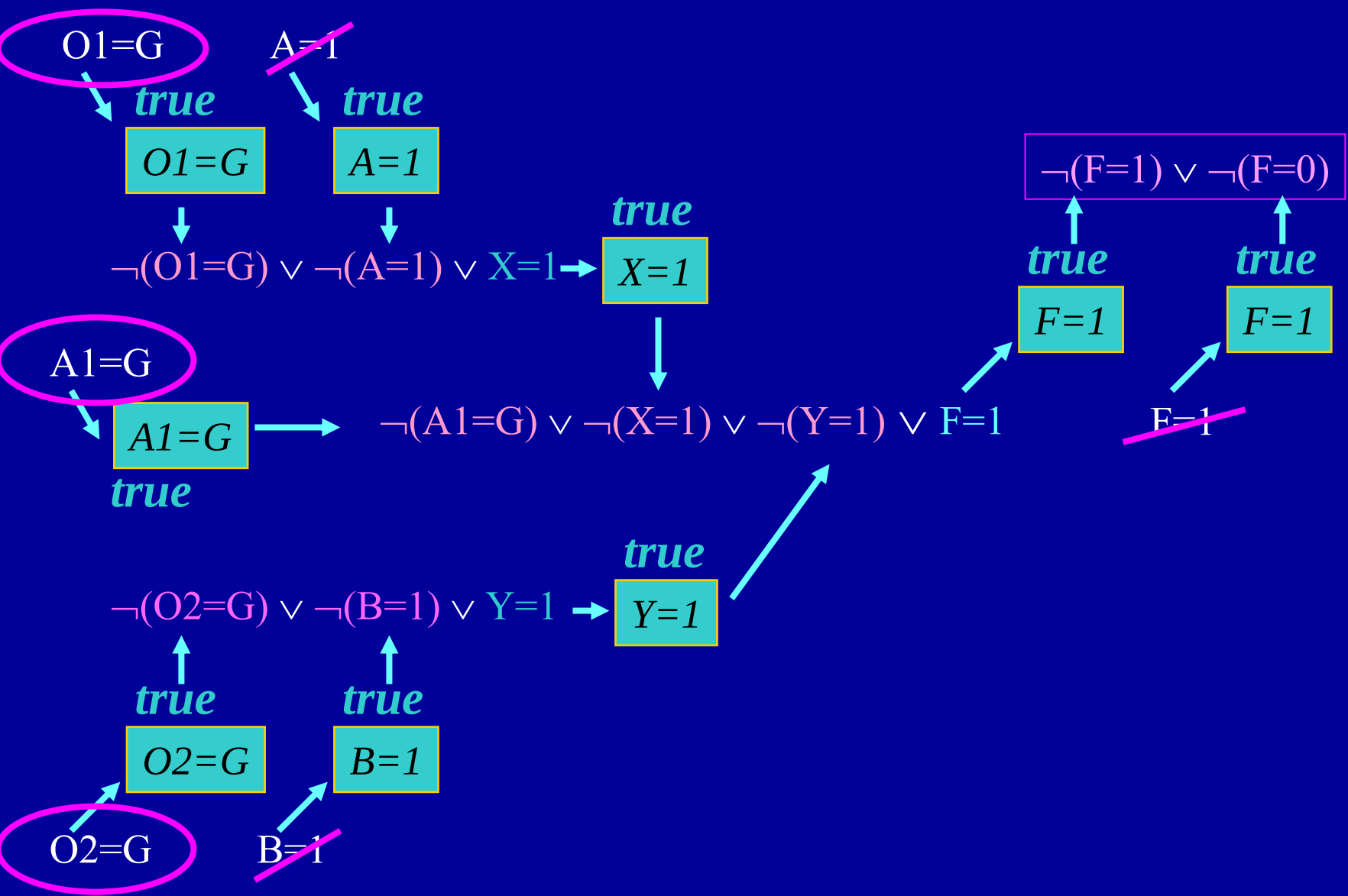
Find Symptom Using Unit Propagation



Find Symptom Using Unit Propagation



Extract Conflict by Tracing Support



Extract Conflict by Tracing Support

procedure *Conflict*(***C***) // ***C*** is an inconsistent clause
  for each literal l in C
    union Support-Conflict(l, support(l))
  end *Conflict*

procedure *Support-Conflict*(l,S)
  If unit-clause?(C)
    If mode-assignment?(literal (C))
      Then { literal(C) }
      Else { }
  Else for each literal l1 in C, other than l
    Union Support-Conflict(l1, support(l1))
  end *Support-Conflict*

Candidate Test with Conflict Extraction

procedure *Test_Candidate(c,M,obs)*

1. Assert candidate assignment c
2. Propagate obs through model M using unit propagation.
3. If inconsistent clause return Conflict(c)
4. Else search for satisfying solution using DPLL
   - If inconsistent **return c** as a conflict.
   - Else **return** "**consistent**"

Outline

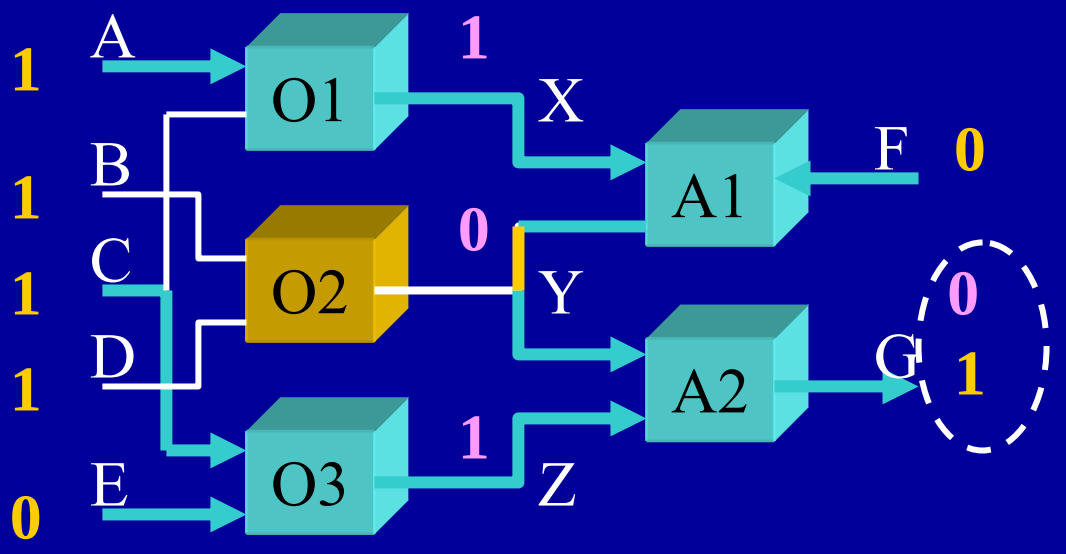
- Model-based diagnosis
- Defining diagnosis
- Searching for diagnoses
  - Conflict learning
  - Single-fault diagnosis
- Appendix

Single Fault Diagnosis w Conflicts: Generate Candidates From Symptom

Single_Fault_w_Conflicts(M, X, Obs)
\\ *Model M, Mode variables X, Observation Obs*

1. Assume all components okay,
   All_Good = { x=G | x ∈ X}
2. Conflict ← *Test_Candidate(All_Good, M, Obs)*
3. **If** Conflict = "consistent" **return** All_Good
4. Generate single fault candidates
   Cands ← {{x=U}∪Z=G | x=G ∈ Conflict, Z=X-{x}}
5. Test_Candidates(Cands, M, Obs)

Generate Candidates From Symptom

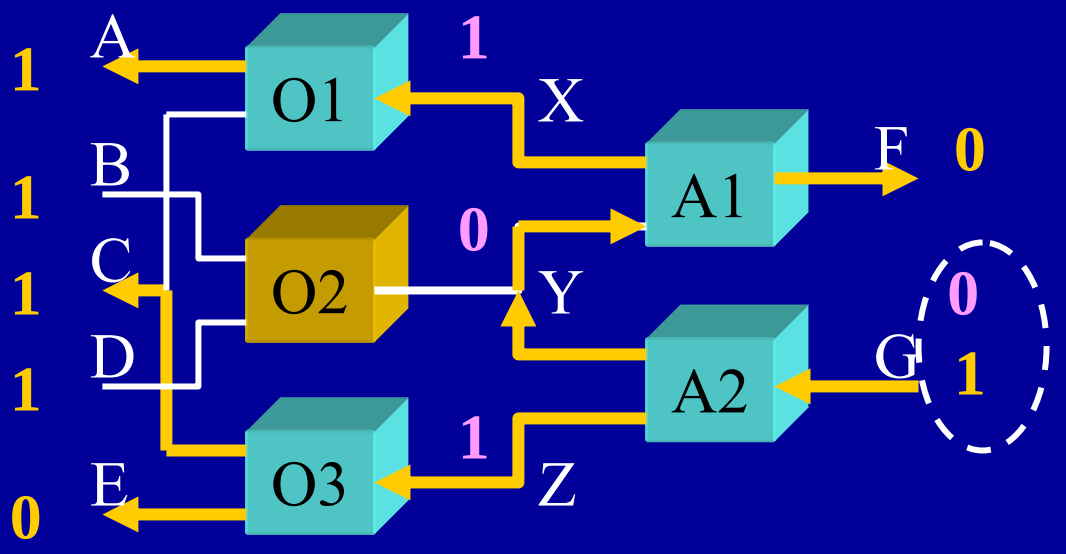


Symptom: F is observed 0, but should be 1

Conflict: {O1=G, O3=G, A1=G, A2=G} is inconsistent

Candidates: {O1=U, O3=U, A1=U, A2=G}

Generate Candidates From Symptom



Symptom: F is observed 0, but should be 1

Conflict: {O1=G, O3=G, A1=G, A2=G} is inconsistent

Candidates: {{O1=U…}, {O3=U…}, {A1=U...}, {A2=G…}}

Single Fault Diagnosis w Conflicts: Test Candidates, Collecting Conflicts

Single_Fault_Test_Candidates(C,M, Obs)

\\ *Candidates C, Model M, Observation Obs*

Diagnoses ← {}, Conflicts ← {}

For each c in C

If c is a superset of some conflict in Conflicts

Then inconsistent candidate, ignore.

Else Conflict = *Test_Candidate(c, M, Obs)*

If Conflict = "consistent"

Then add c to Diagnoses

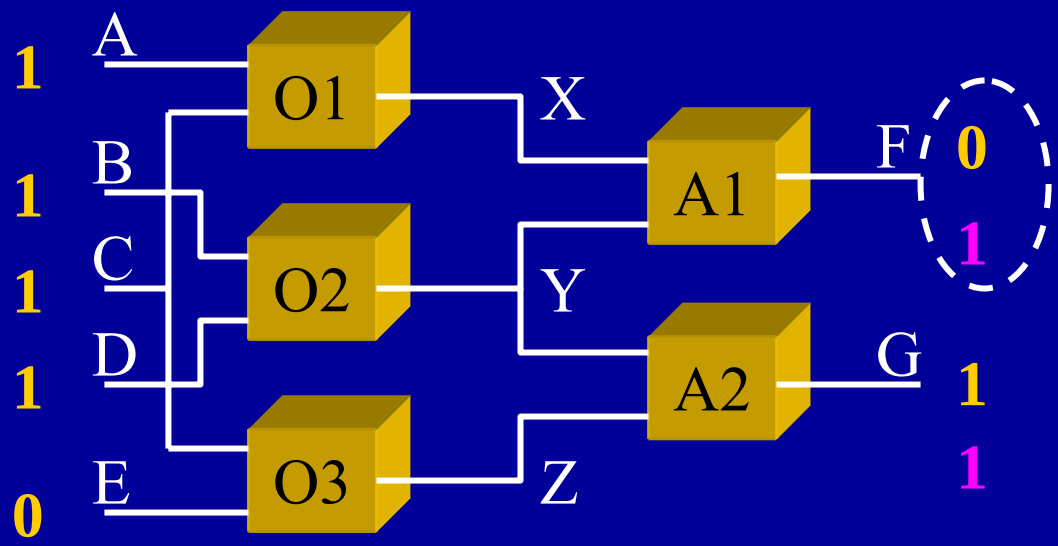
Else add Conflict to Conflicts

return Diagnoses

Test Candidates, Collecting Conflicts

Candidates: {{O1=U…}, {O3=U…}, {A1=U…}, {A3=U…}}

Diagnoses: {}

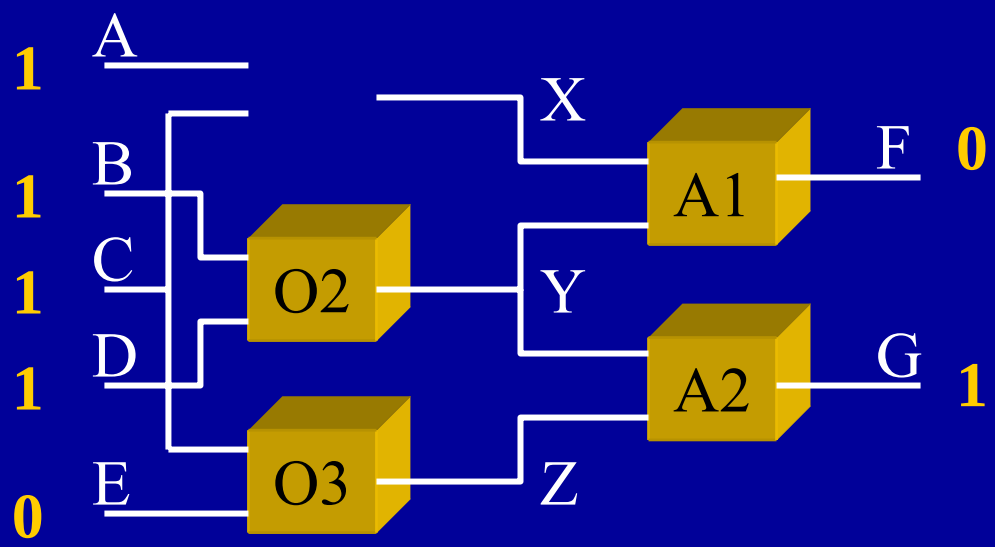


- First candidate {O1=U, …}

Test Candidates, Collecting Conflicts

Candidates: {{O1=U…}, {O3=U…}, {A1=U…}, {A3=U…}}

Solutions: {}

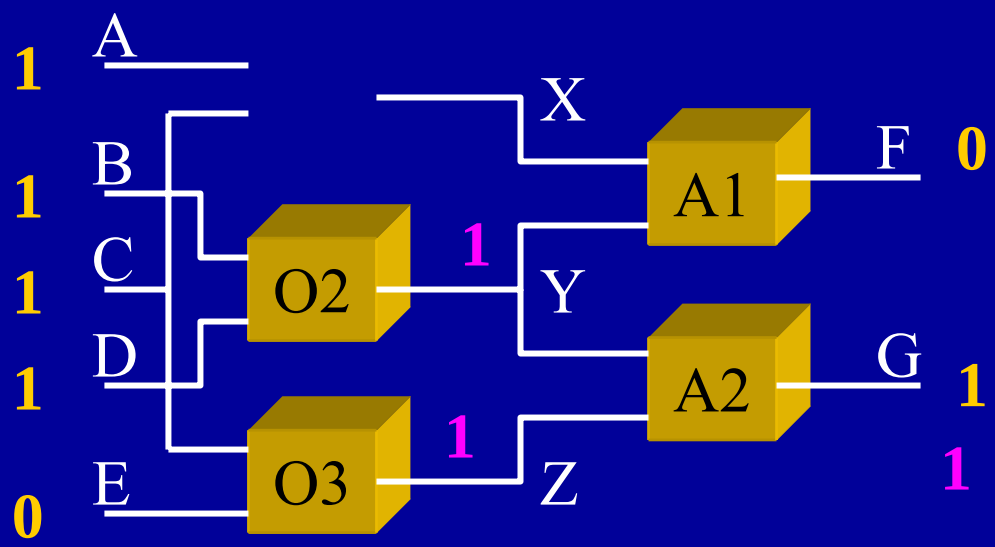


- First candidate {O1=U, …}
- Suspend O1's constraints

Test Candidates, Collecting Conflicts

Candidates: {{O1=U…}, {O3=U…}, {A1=U…}, {A3=U…}}

Solutions: {}

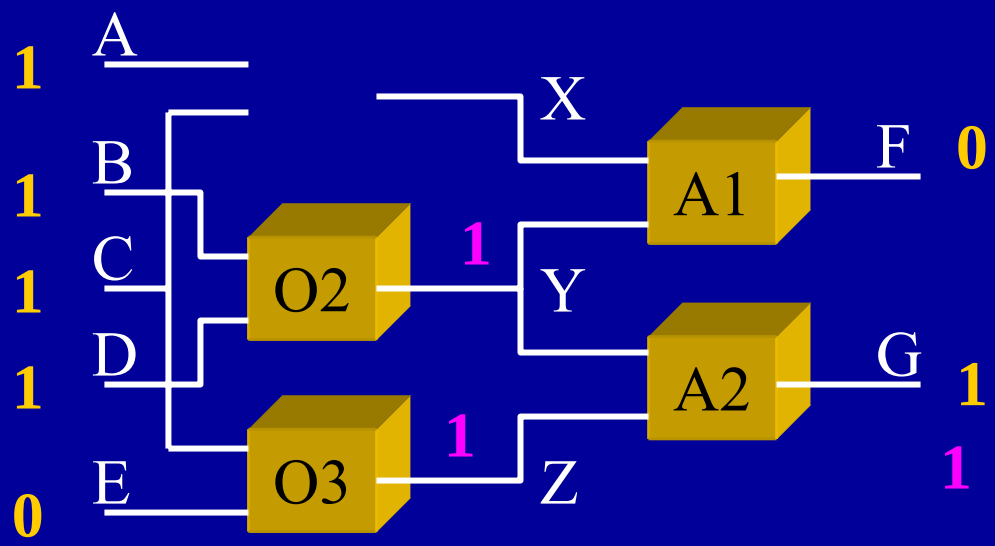


- First candidate {O1=U, …}
- Suspend O1's constraints
- Test consistency

Test Candidates, Collecting Conflicts

Candidates: {{O3=U…}, {A1=U…}, {A3=U…}}

Solutions: {{O1=U…}}

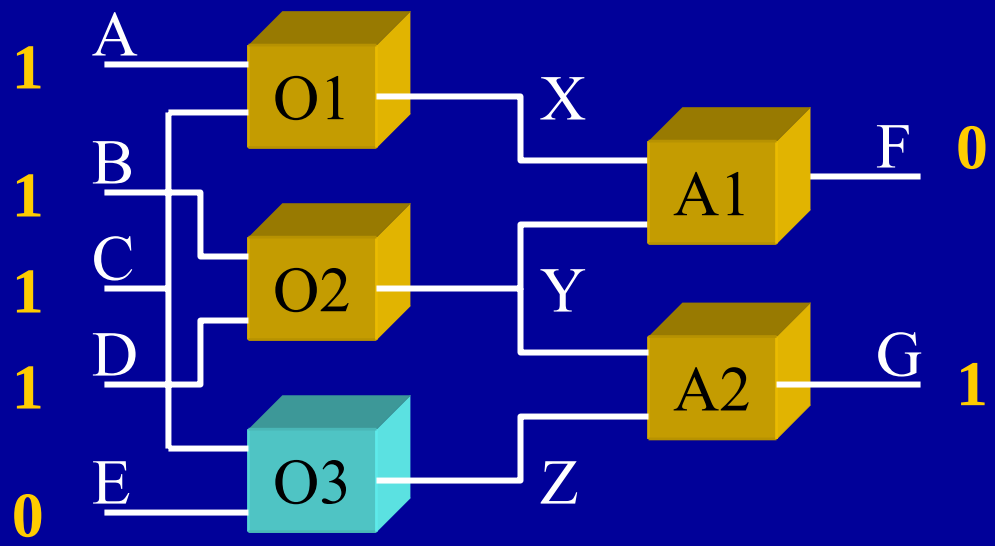


- First candidate {O1=U, …}
- Suspend O1's constraints
- Test consistency → Consistent: Add to solutions

Test Candidates, Collecting Conflicts

Candidates: {{O3=U…}, {A1=U…}, {A2=U…}}

Solutions: {{O1=U…}}

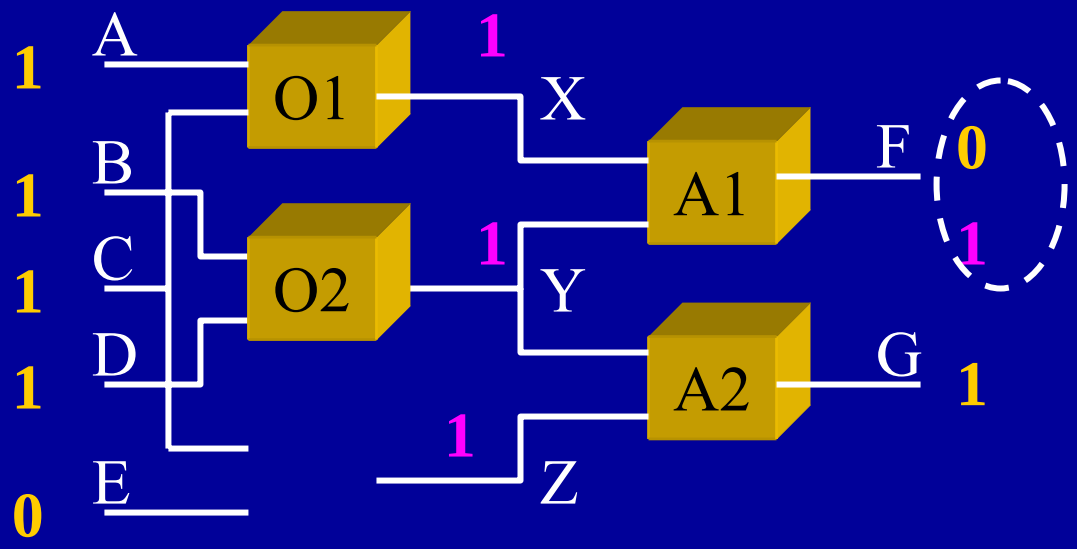


- Second candidate {O3=U, …}
- Suspend O3's constraints
- Test consistency

Test Candidates, Collecting Conflicts

Candidates: {{O3=U…}, {A1=U…}, {A2=U…}}

Solutions: {{O1=U…}}



- Second candidate {O3=U, …}
- Suspend O3's constraints
- Test consistency⟶ Inconsistent

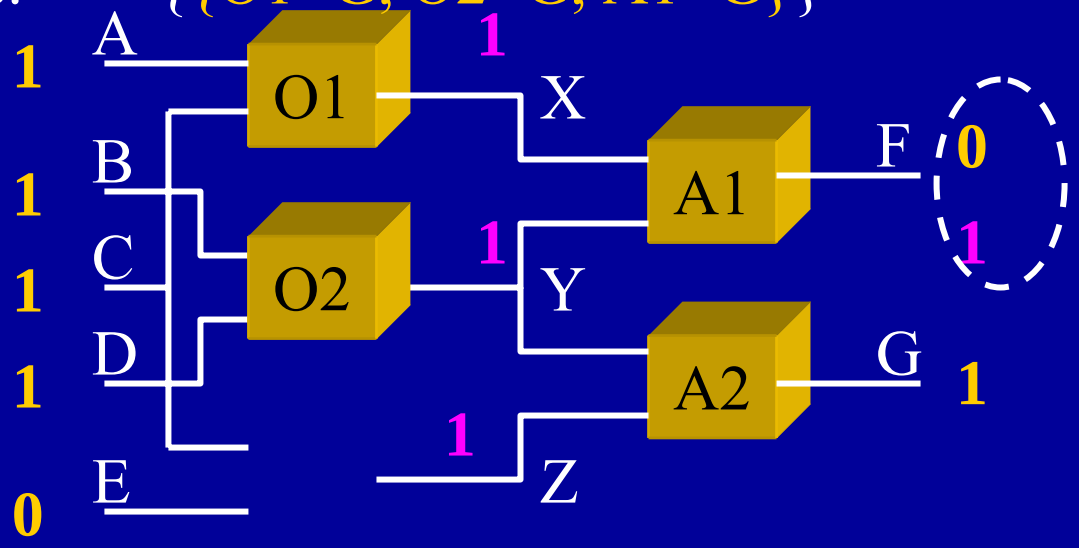
Testing Consistency ≠ Forward Prediction

Test Candidates, Collecting Conflicts

Candidates: {{O3=U…}, {A1=U…}, {A2=U…}}

Solutions: {{O1=U…}}

Conflicts: {{O1=G, O2=G, A1=G}}



- Second candidate {O3=U, …}
- Suspend O3's constraints
- Test ⟶ Inconsistent

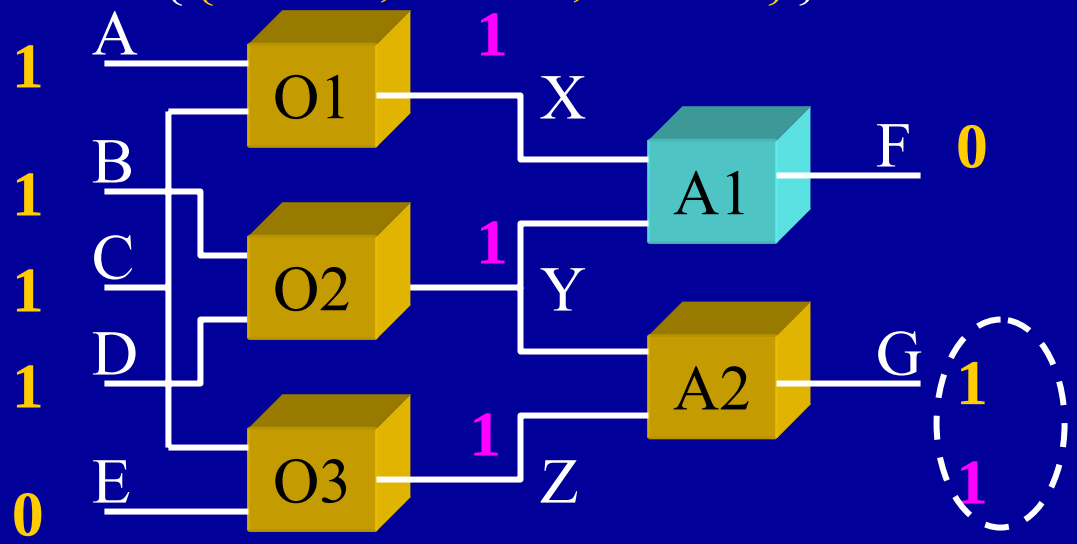
- Extract Conflict: {O1=G, O2=G, A1=G}
- Use to prune candidates

Test Candidates, Collecting Conflicts

Candidates: {{A1=U…}, {A2=U…}}

Solutions: {{O1=U…}}

Conflicts: {{O1=G, O2=G, A1=G}}



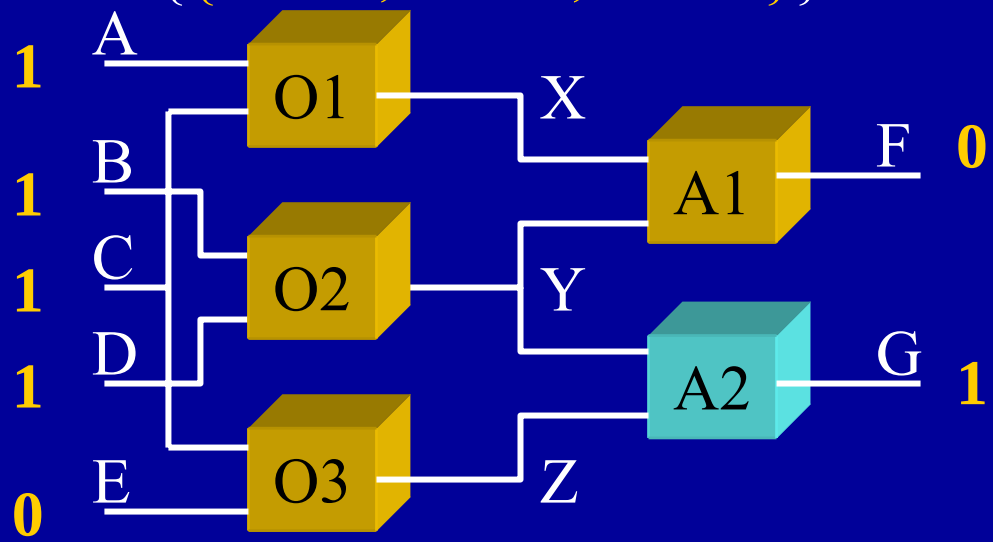
- Third candidate {A1=U, …}
- Subsumed by conflict? → No, since A1 = U, not A1=G
- Suspend A1's constraints
- Test → Consistent

Test Candidates, Collecting Conflicts

Candidates: {{A2=U…}}

Solutions: {{O1=U…}, {A1=U…}}

Conflicts: {{O1=G, O2=G, A1=G}}



- Fourth candidate {A2=U, …}
- Subsumed by conflict? → Yes, since O1=G,O2=G and A1=G
- Eliminate candidate

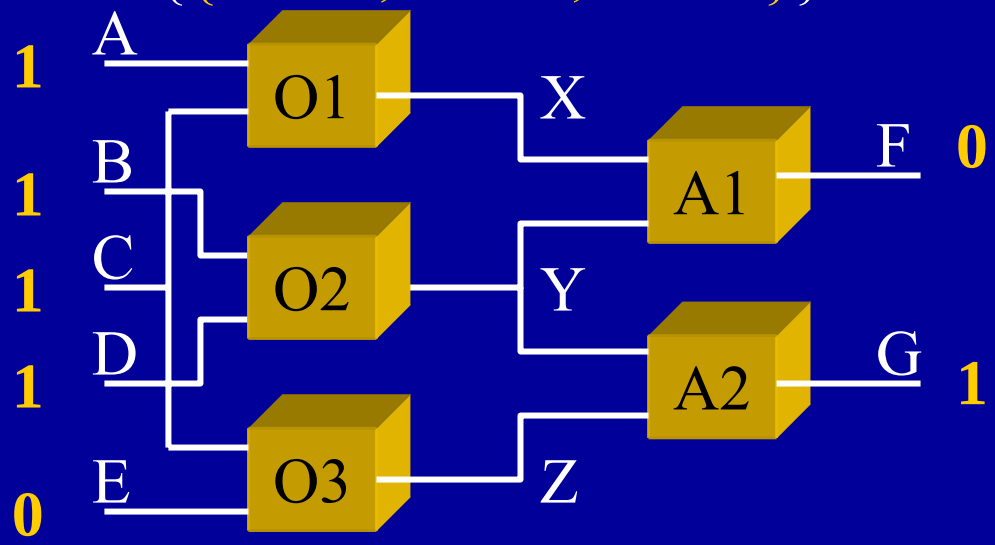
Consistent

Test Candidates, Collecting Conflicts

Candidates: {}

Solutions: {{O1=U…}, {A1=U…}

Conflicts: {{O1=G, O2=G, A1=G}}



- Return Solutions → O1 or A1 broken

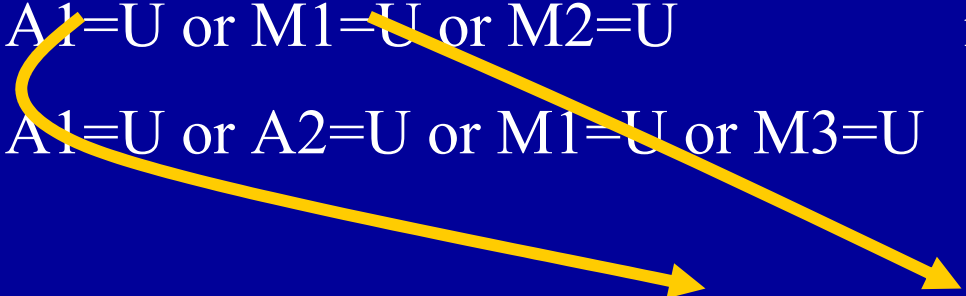
Single Fault Diagnoses are the Intersection of All Conflicts

{A1=G, M1=U, M2=U} conflict 1.

{A1=U, A2=U, M1=U, M3=U} conflict 2

A1=U or M1=U or M2=U removes conflict 1.

A1=U or A2=U or M1=U or M3=U removes conflict 2



Single Fault Diagnoses = {{A1=U..}, {M1=U..}}

Outline

- Model-based diagnosis
- Defining diagnosis
  - Explaining failures
  - Handling unknown failures
- Searching for diagnoses
  - Conflict learning
  - Single-fault diagnosis
- Appendix
  - Multiple-fault diagnosis

Summary: Model-based Diagnosis

- A failure is a discrepancy between the model and observations of an artifact.
- Diagnosis is symptom directed
- Symptoms identify conflicting components as initial candidates.
- Test novel failures by suspending constraints and testing consistency.
- Newly discovered conflicts further prune candidates.

Multiple Faults Occur

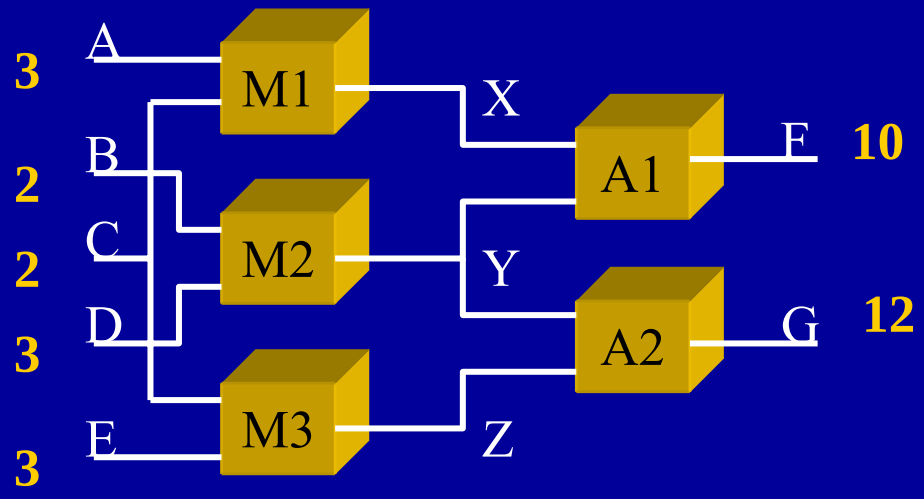
- Quintuple fault occurs (three shorts, tank-line and pressure jacket burst, panel flies off).
- Power limitations too severe to perform new mission..
- Novel reconfiguration identified, exploiting LEM batteries for power.
- Swaggert & Lovell work on Apollo 13 emergency rig lithium hydroxide unit.

APOLLO 13

Diagnosis identifies consistent modes

Adder(i):

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



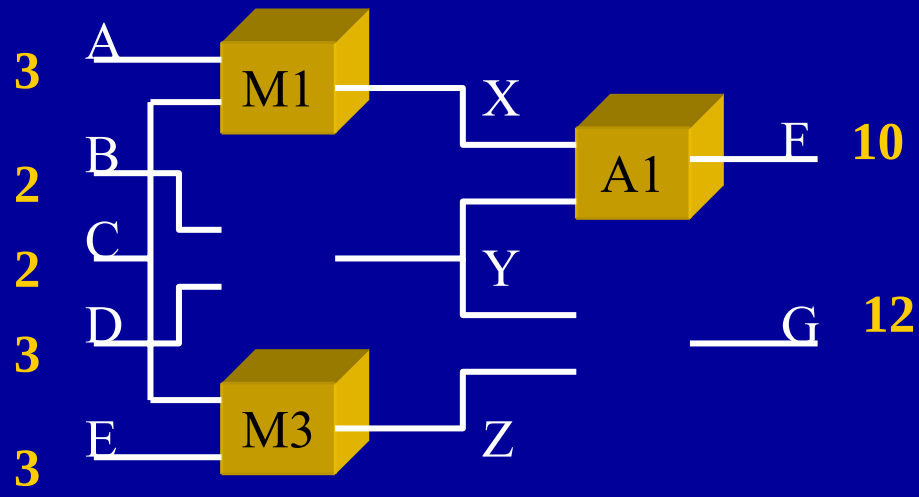
Candidate = {A1=G, A2=G, M1=G, M2=G, M3=G}

- Candidate: Assignment to all component modes.

Diagnosis identifies All sets of consistent modes

Adder(i):

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

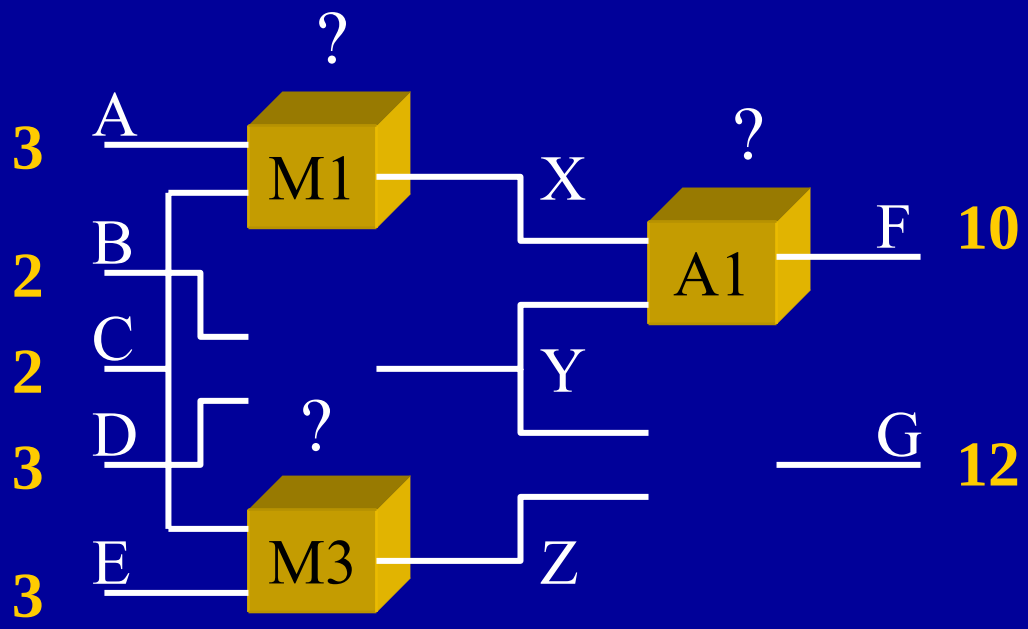


Diagnosis = {A1=G, A2=U, M1=G, M2=U, M3=G}

- Diagnosis D: Candidate consistent with model Phi and observables OBS.
- As more constraints are relaxed, candidates are more easily satisfied.

➔ Typically an exponential number of candidates.

Representing Diagnoses Compactly: Kernel Diagnoses



Kernel Diagnosis = {A2=U, M2=U}

"Smallest" sets of modes that remove all symptoms

Every candidate that is a subset of a kernel diagnosis is a diagnosis.

Generate Kernels From Conflicts

{A1=G, M1=U, M2=U} conflict 1.

{A1=U, A2=U, M1=U, M3=U} conflict 2

A1=U or M1=U or M2=U removes conflict 1.

A1=U or A2=U or M1=U or M3=U removes conflict 2

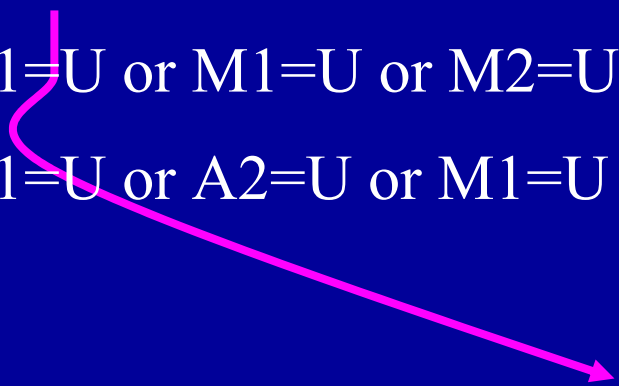
Kernel Diagnoses =

"Smallest" sets of modes that remove all conflicts

Generate Kernels From Conflicts

A1=U or M1=U or M2=U removes conflict 1.

A1=U or A2=U or M1=U or M3=U removes conflict 2



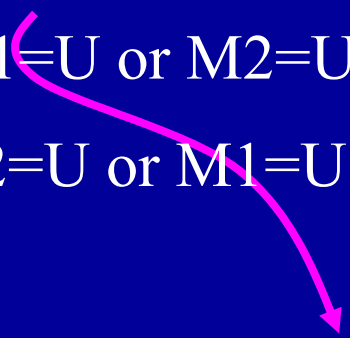
Kernel Diagnoses = {A1=U}

"Smallest" sets of modes that remove all conflicts

Generate Kernels From Conflicts

A1=U or M1=U or M2=U removes conflict 1.

A1=U or A2=U or M1=U or M3=U removes conflict 2



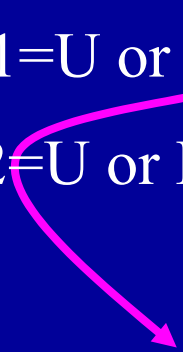
Kernel Diagnoses = {M1=U}
{A1=U}

"Smallest" sets of modes that remove all conflicts

Generate Kernels From Conflicts

A1=U or M1=U or M2=U removes conflict 1.

A1=U or A2=U or M1=U or M3=U removes conflict 2



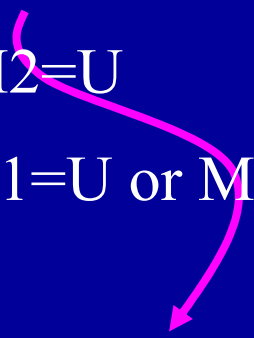
Kernel Diagnoses = {A2=U, M2=U}
{M1=U}
{A1=U}

"Smallest" sets of modes that remove all conflicts

Generate Kernels From Conflicts

A1=U or M1=U or M2=U removes conflict 1.

A1=U or A2=U or M1=U or M3=U removes conflict 2



Kernel Diagnoses = {M2=U, M3=U}
{A2=U, M2=U}
{M1=U}
{A1=U}

"Smallest" sets of modes that remove all conflicts

Diagnosis by Divide and Conquer

Given model Phi and observations OBS

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

General Diagnostic Engine
[de Kleer & Williams, 87]

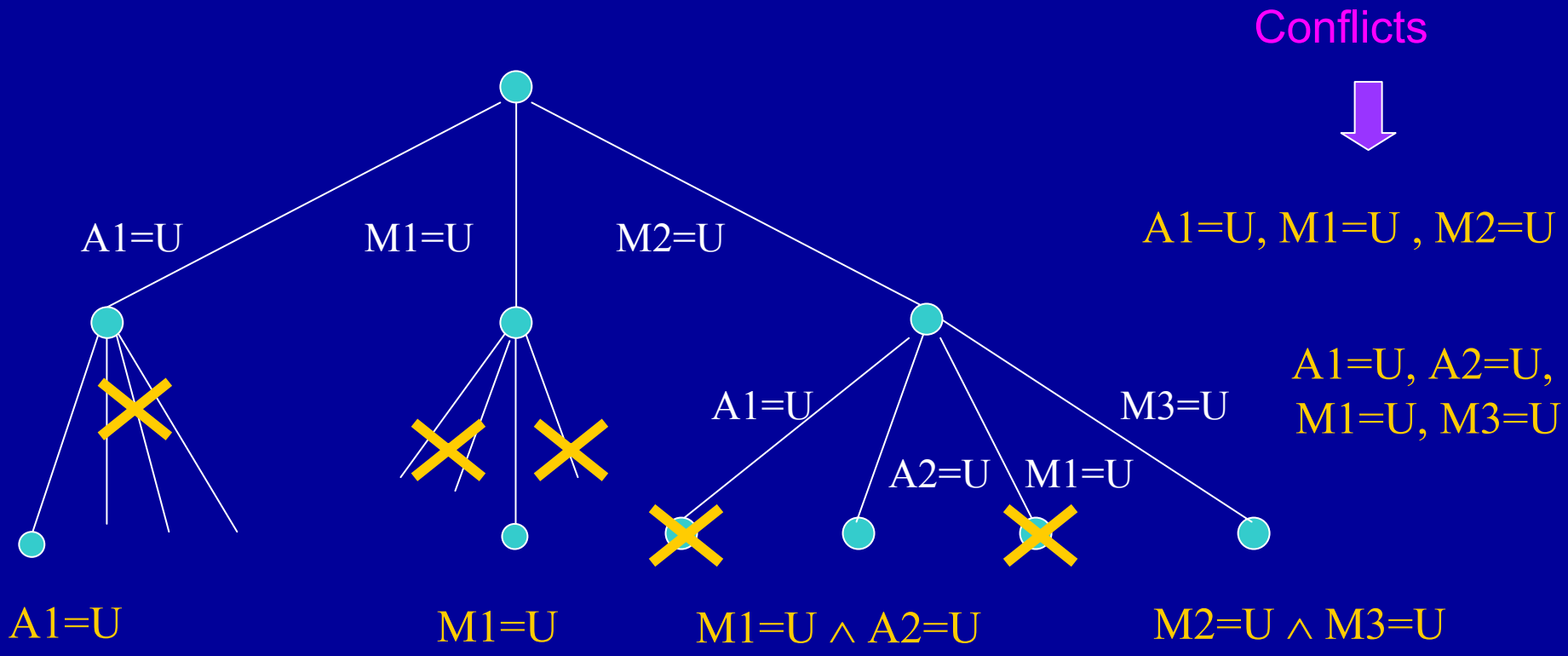
Exploring the Improbable

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

- Sherlock Holmes.

The Sign of the Four.

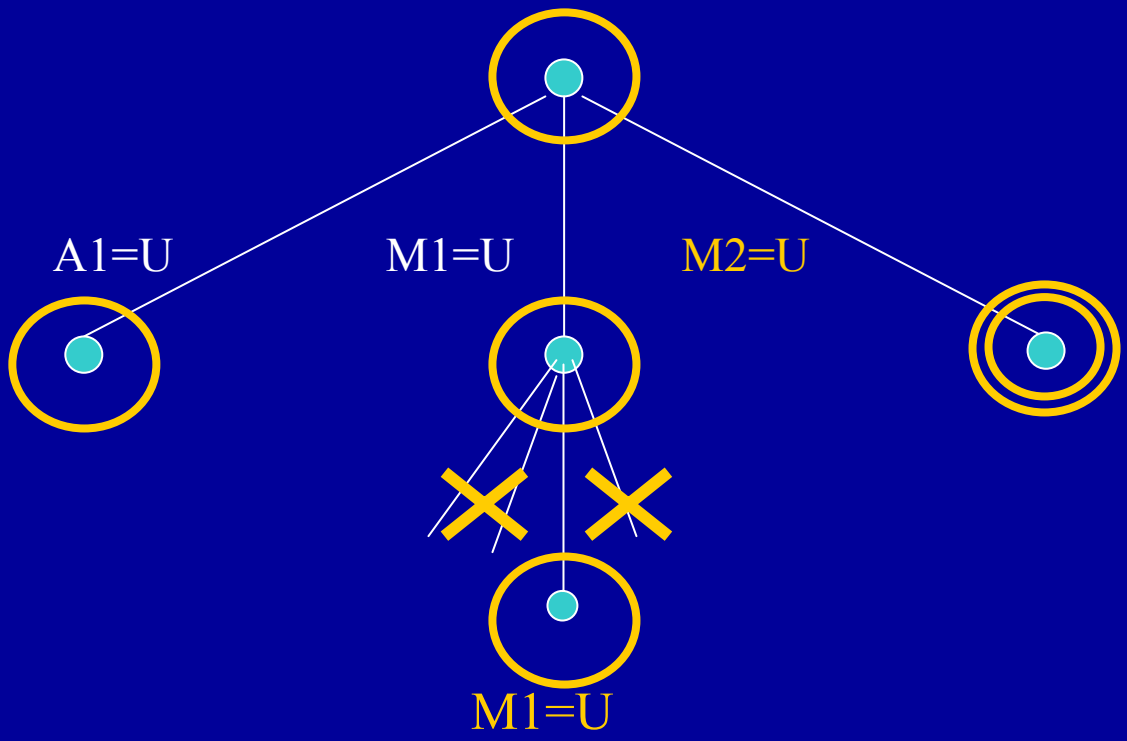
Conflict-Directed A*: Generating The Best Kernel



Insight:

- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.

Conflict-Directed A*: Generating The Best Kernel



Conflicts

A1=U, M1=U , M2=U

A1=U, A2=U,
M1=U, M3=U

Insight:

- Kernels found by minimal set covering
- Minimal set covering is an instance of breadth first search.
- To find the best kernel, expand tree in best first order.