

Assumption-Based Constraint Languages

Knowledge Systems Engineering

Administriva

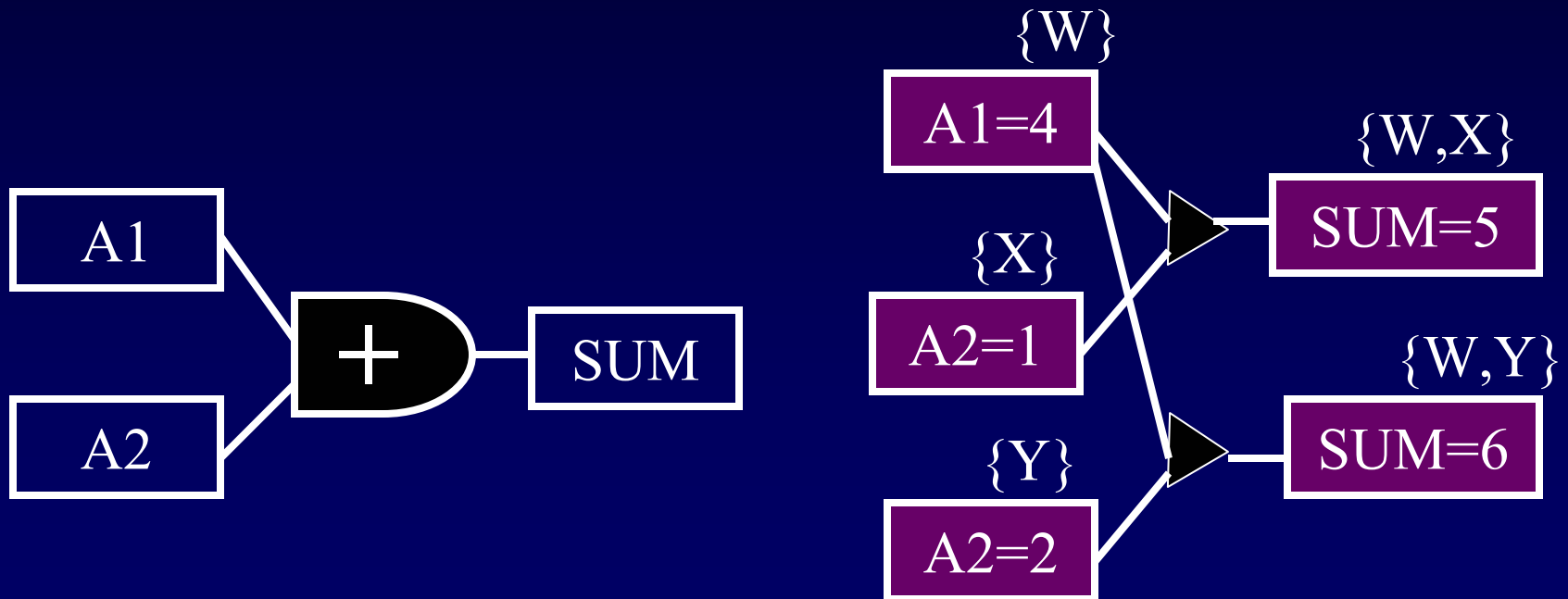
- Homework 5 due Friday

Today: Assumption-based Constraint Languages

- Last time: TCON
 - Modular constraint language
 - Constraints within constraints within...
 - Propagation of values over cells
 - Contradictions handled using contradiction and coincidence handler
 - Diagnosis
 - Given a system with faulty outputs
 - If each constraint represents a component
 - Find potential causes of problem "upstream" from bad output
 - For each potential "bad component", *suspend* its constraint and check against input and output
- Today: Hooking up TCON to the ATMS

Hooking up TCON to the ATMS

- Goal: Make every TCON cell value look like an ATMS node



Hooking up TCON to the ATMS

- Goal: Make every TCON cell *value* look like an ATMS node
 - Part 1: Create TMS nodes to enable and disable constraints
 - Part 2: When cell values are set, TMS nodes must be changed
 - Part 3: When TMS nodes are changed, values must be reset

Part 1: Enabling and disabling constraints

- Additional trigger, "ok", added to constraint's formulae rules
 - "ok" is set only when assumption enabled
- If contradiction, create nogood
 - Uses ATMS notion of contradiction, rather than calling TCON contradiction handler?

Part 1: Create TMS nodes to enable and disable constraints

```
(constraint adder-component ((a1 cell) (a2 cell)
                              (sum cell) (ok assumption))
  (formulae (sum (a1 a2 ok) (+ a1 a2))
            (a1 (sum a2 ok) (- sum a2))
            (a2 (sum a1 ok) (- sum a1))))
```

New data type:
Assumption

Added as trigger node
for all formulae rules,

Note: ASSUME-
CONSTRAINT will
generate ok assumptions
automatically, so use
that.

Assume-constraint handles the OK assumptions for us

```
(assume-constraint adder-component
                  ((a1 cell) (a2 cell)
                   (sum cell))
 (formulae (sum (a1 a2) (+ a1 a2))
           (a1 (sum a2) (- sum a2))
           (a2 (sum a1) (- sum a1))))
```

When constraint is created, assumption "cells" are created as well

```
(defun create (name type &optional supplied-parts (atcon atcon*))  
  (create1 name type supplied-parts atcon :USER))
```

```
(defun create1 (name type supplied-parts atcon owner &aux cell)  
  (case type  
    (CELL (create-cell name atcon supplied-parts  
      (ASSUMPTION  
        (setq cell (create-cell name atcon nil owner))  
        (assume-node (lookup-node cell  
          (if (symbolp owner)  
              name  
              (constraint-name owner))))  
          cell)  
      (t (create-prototype name type supplied-parts atcon owner))))
```

Create cell, just like any other cell...

Build ATMS node for cell.

Some notes:

- Remember diagnosis via constraint suspension?
 - Suspending a constraint required:
 - Unwiring the constraint from the network
 - Disabling cell values
 - Propagating new values
- Here, just retract the enabling assumption for that constraint
 - Much, much simpler!

Part 2: When cell values are set, TMS nodes must be changed

- (set-parameter <cell> <value>)
 - Creates a premise node (i.e., it is valid in all environments).
- (assume-parameter <cell> <value> <str>)
 - Set cell to value, tied to assumption node with name as <str>.

A demonstration of parameter-setting

- Try this:

```
(create 'a 'adder-component)
```

```
(assume-parameter (>> a1 a) 1 "a1=1")
```

```
(assume-parameter (>> a2 a) 10 "a2=10")
```

```
(assume-parameter (>> sum a) 100  
  "sum=100")
```

```
(show-network *atcon*)
```

Constraints that invalidate premise cell values are retracted

- Now try:
 - (set-parameter (>> a1 a) 1)
 - (set-parameter (>> a2 a) 10)
 - (show-network *atcon*)
- Since sum=100 doesn't fit the premises, it is made a nogood and removed from network.

How does this work?

- *Brief explanation:*
 - Cell struct contains list of its tms-nodes
 - Remember: one TMS node per value
 - When parameter assumed, node sought:

```
(defun assume-parameter (cell expression &optional string &aux node)
  (setq node (lookup-node cell expression))
  (setf (value-string (tms-node-datum node)) string)
  (assume-node node)
  (fire-constraints (cell-atcon cell)))
```

Part 3: When TMS nodes are changed, values must be reset

- Trick question
- When environments are made nogood, no constraint cells need to be changed
 - Cell values are always contingent on the given environment in ATCON
- Note: rules fire when ATMS nodes become :IN (queued on nodes)

Summary: Building ATCON

- Enable and disable individual constraints by enabling or retracting "ok" assumptions
- Cell values given relative to environment, saved as TMS node set
- Assume-parameter will set a cell value in a given environment
- Contradictions (nogoods) remove cell values from network

Other differences between TCON and ATCON

- ATCON doesn't recompute identical values, TCON does
- ATCON uses a *lot* more space
 - Cell values never go away

Performing search over ATCON network

- Remember INTERPRETATIONS?
 - DDSearch-ish
 - Found sets of consistent environments
- Can perform interpretations search over ATCON's ATMS to find consistent labelings

Try this:

```
(progn
  (setq *atcon* (create-atcon "Interp test" :debugging t))
  (create 'a 'adder-component)
  (assume-parameter (>> a1 a) 1 "a1=1")
  (assume-parameter (>> a2 a) 12 "a2=12")
  (assume-parameter (>> sum a) 10 "sum=10")
  (assume-parameter (>> sum a) 100 "sum=100")
  (show-network *atcon*)
)
```

Now, run interpretation search

- (trace interpretations)
- (solutions)
- (print-solutions)

Finding consistent sets via output labels

- Adder circuit

Code for adder circuit

```
(defun adder1 ()
  (let ((atcon (create-atcon "Adder1 Test" :debugging T)))
    (setq *atcon* atcon)
    (with-network atcon
      (bps-load-file (make-bps-path "gde") "condef"))
    (create 'add '2-bit-adder)
    (assume-parameter (>> a bit0 add) 1 "a0=1")
    (assume-parameter (>> a bit0 add) 0 "a0=0")
    (assume-parameter (>> b bit0 add) 1 "b0=1")
    (assume-parameter (>> b bit0 add) 0 "b0=0")
    (assume-parameter (>> a bit1 add) 1 "a1=1")
    (assume-parameter (>> a bit1 add) 0 "a1=0")
    (assume-parameter (>> b bit1 add) 1 "b1=1")
    (assume-parameter (>> b bit1 add) 0 "b1=1")
    (assume-parameter (>> ci bit0 add) 1 "ci=1")
    (assume-parameter (>> ci bit0 add) 0 "ci=0")
  ))
```

Let's assume that:

- Inputs are all 1's
- Carry-out is 0
- Which inputs might really be low?

(progn ;; Takes long time

 (assume-parameter (>> a bit0 add) 1 "a0=1")

 (assume-parameter (>> b bit0 add) 1 "b0=1")

 (assume-parameter (>> a bit1 add) 1 "a1=1")

 (assume-parameter (>> b bit1 add) 1 "b1=1")

 (assume-parameter (>> ci bit0 add) 1 "ci=1")

 (set-parameter (>> co bit1 add) 0))

(solutions *atcon*)

A new diagnosis engine using ATCON

- Review of constraint suspension

Essentials of diagnosis via constraint suspension

- Given set of valid inputs, and tested outputs:
 - Find invalid by comparing expected against actual outputs
 - Find culprit constraints for each bad output
 - Take intersection of all culprit sets
- Test each culprit by suspending constraint
 - Unwire constraint from network
 - Set outputs and inputs
 - See if contradiction results: Yes=not cause, No=cause

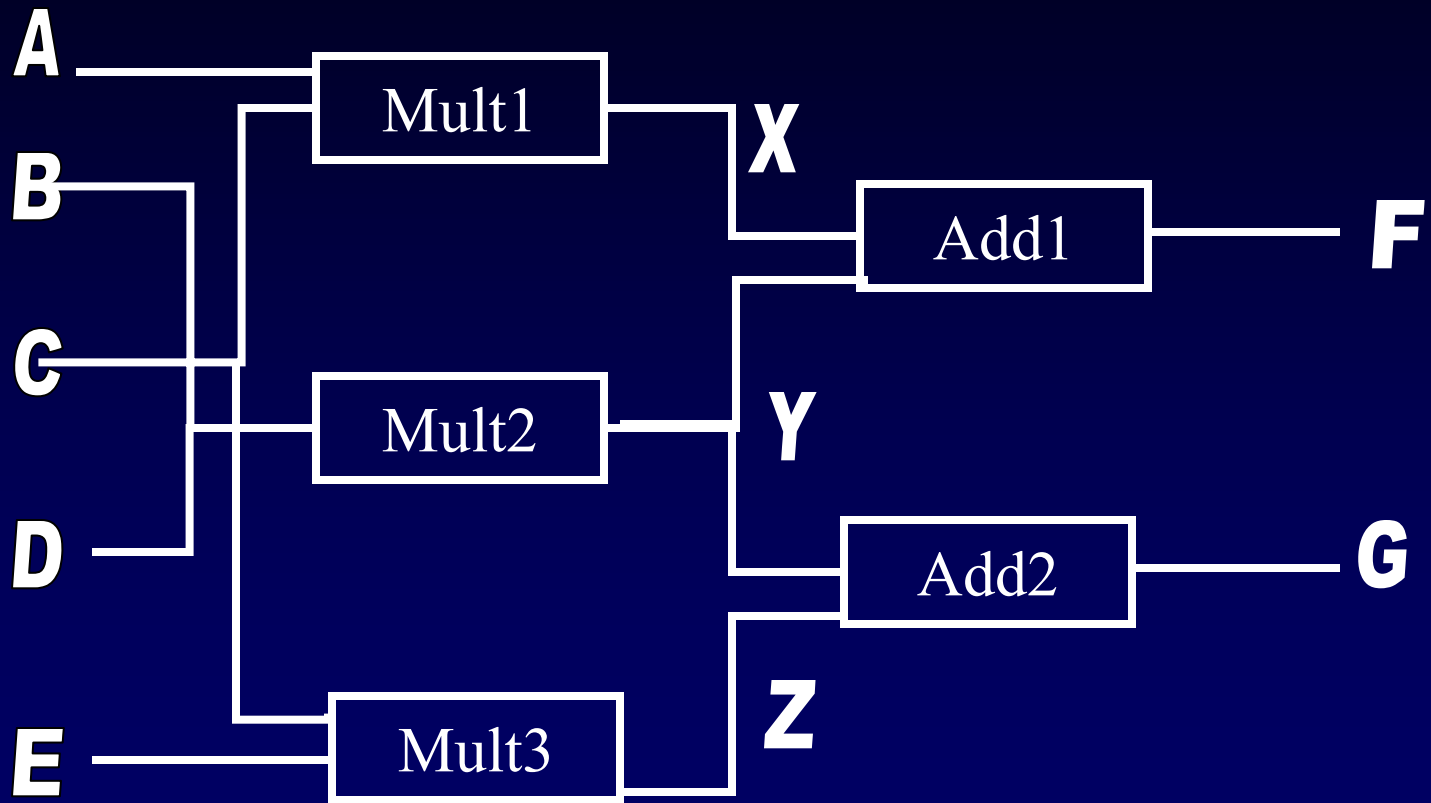
Notes on constraint suspension

- Assumes
 - single cause or culprit
 - bug doesn't change between measurements
 - Add inputs and outputs known
- Advantage of GDE, in contrast
 - Can handle multiple faults (somewhat)
 - Can do sequential diagnosis
 - Directs which measurements to take when

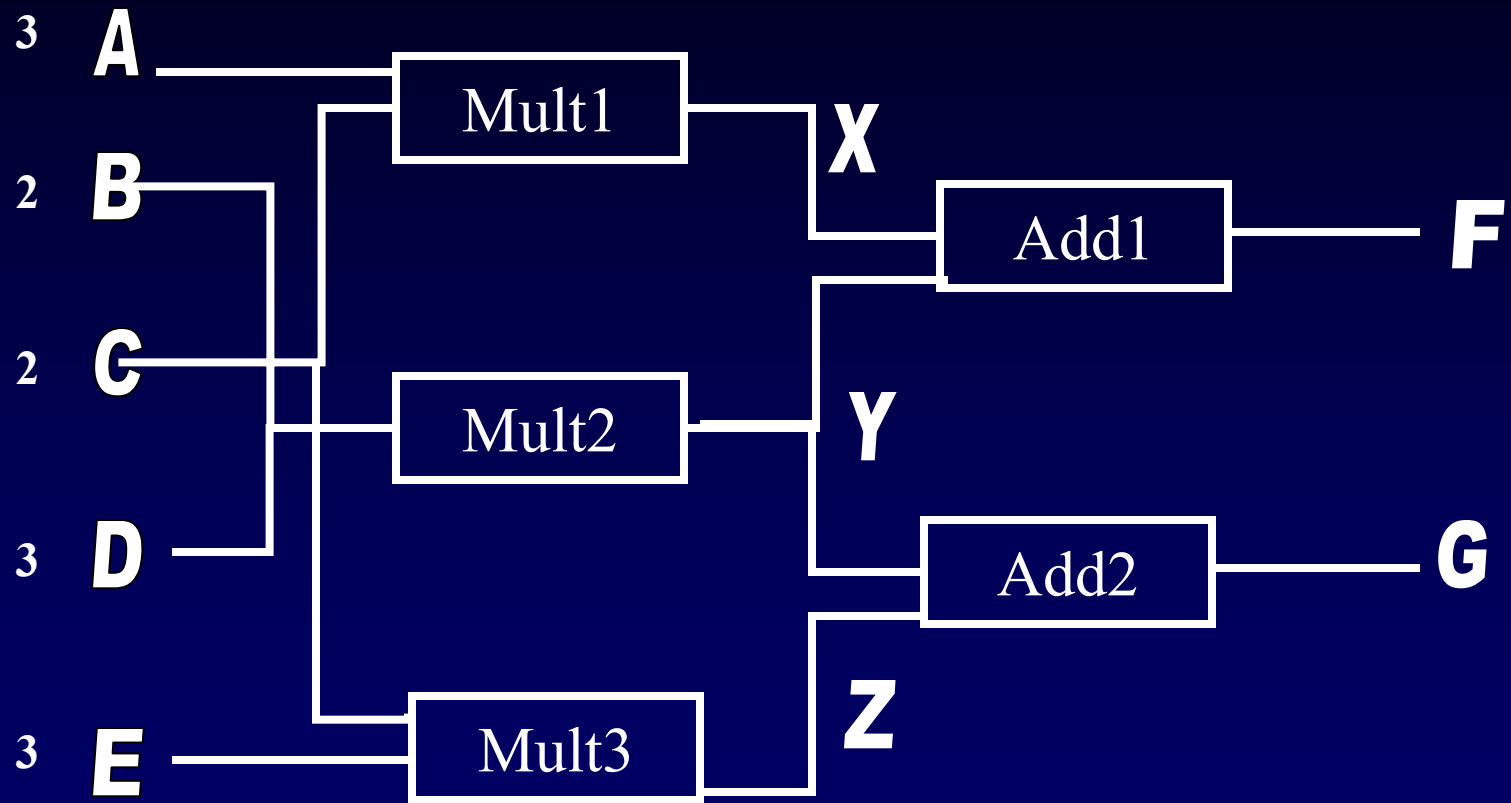
General Diagnosis Engine (Tiny Version)

- Construction
- Example: Polybox
- Example: 2-bit adder circuit

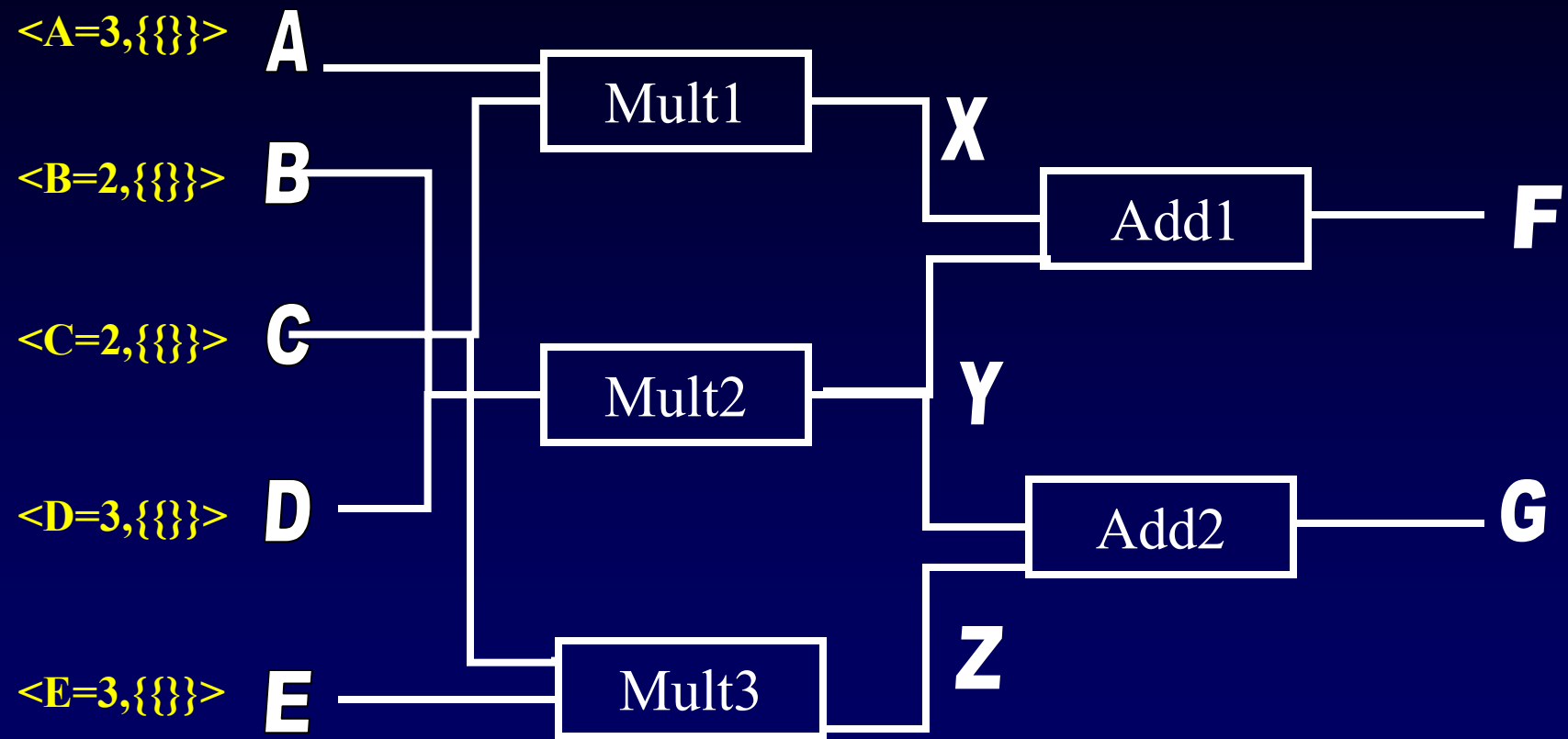
Polybox



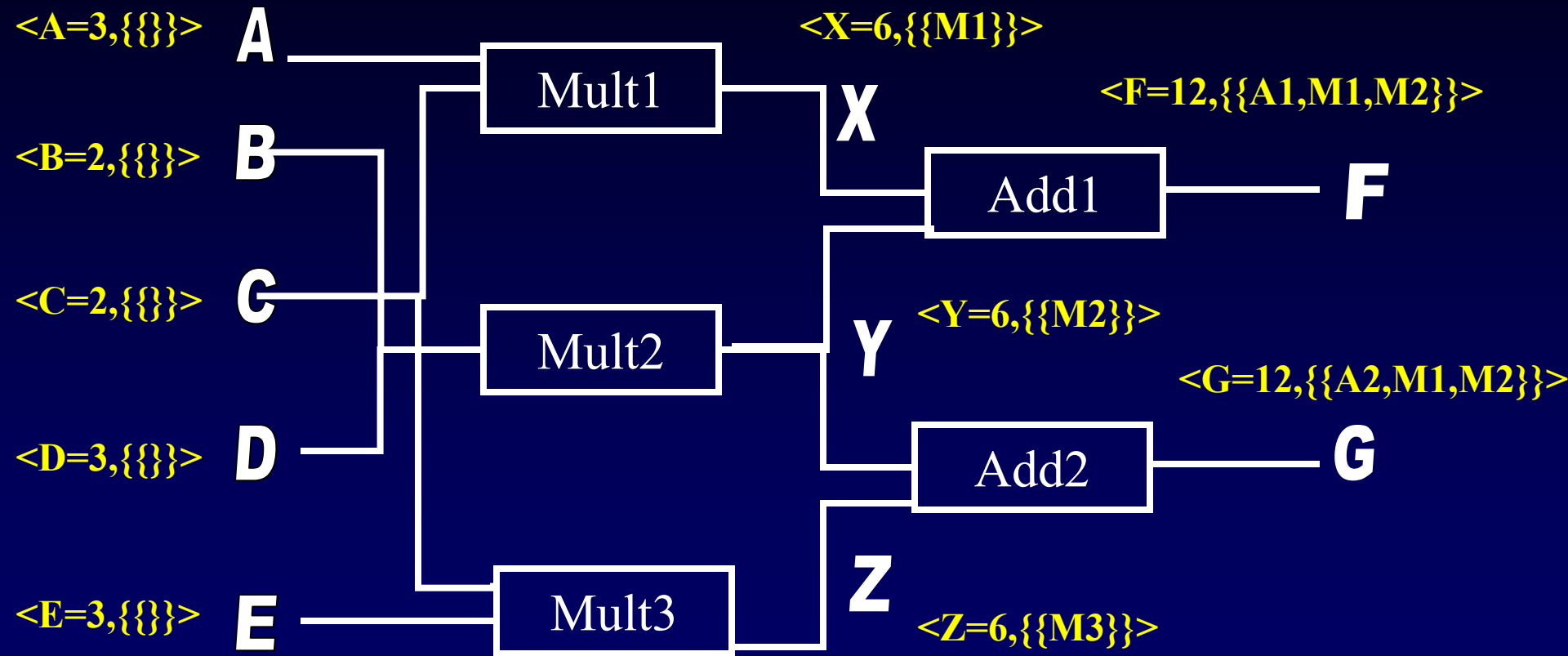
Polybox



Polybox



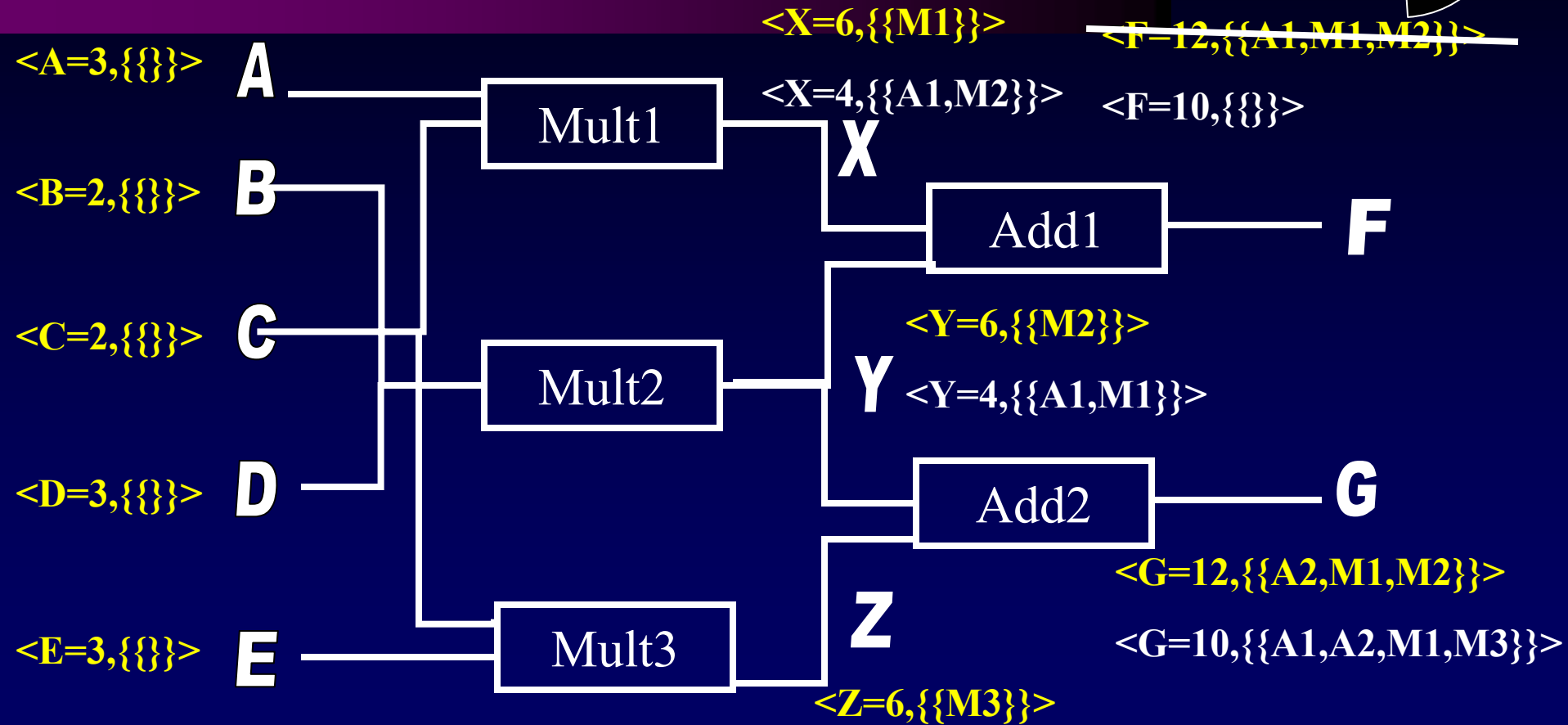
Polybox



1. Probe: $F=10$.

Nogood: $\{\{A1, M1, M2\}\}$

Where should next probe be?



1. Probe: F=10.

2. Probe: G=12.

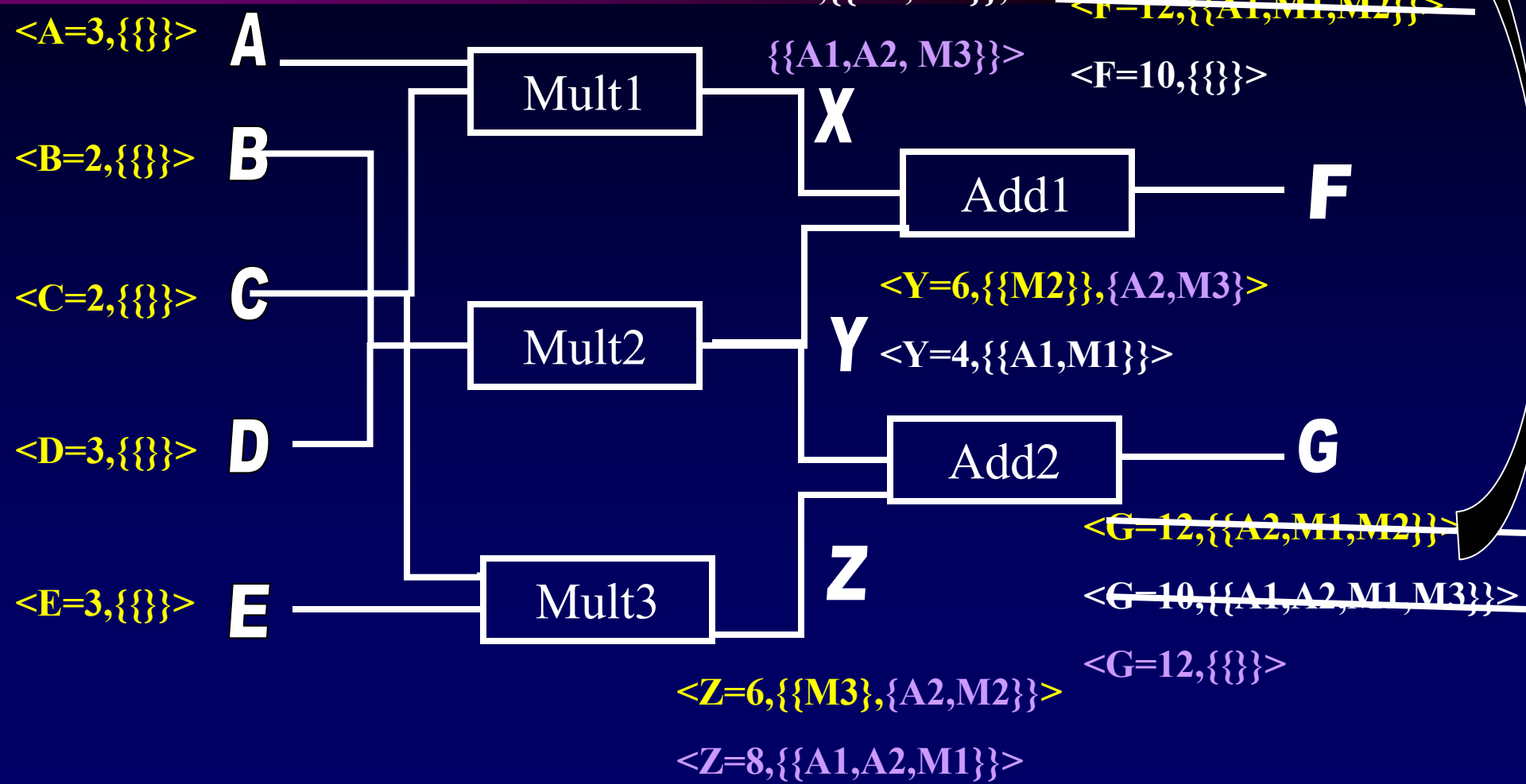
Nogood: $\{\{A1, M1, M2\}\}$.

Nogood: $\{\{A1, A2, M1, M3\}\}$

$\langle X=6, \{\{M1\}\} \rangle$

$\langle X=4, \{\{A1, M2\}\},$

~~$\langle F=12, \{\{A1, M1, M2\}\} \rangle$~~



1. Probe: F=10.

2. Probe: G=12.

Nogood: $\{\{A1, M1, M2\}\}$.

Nogood: $\{\{A1, A2, M1, M3\}\}$.

$\langle X=6, \{\{M1\}\} \rangle$

$\langle X=4, \{\{A1, M2\}\} \rangle$

~~$\langle F=12, \{\{A1, M1, M2\}\} \rangle$~~

$\langle A=3, \{\{\}\} \rangle$

A

$\langle B=2, \{\{\}\} \rangle$

B

$\langle C=2, \{\{\}\} \rangle$

C

$\langle D=3, \{\{\}\} \rangle$

D

$\langle E=3, \{\{\}\} \rangle$

E



$\{\{A1, A2, M3\}\}$

X



F

$\langle F=10, \{\{\}\} \rangle$

$\langle Y=6, \{\{M2\}\}, \{A2, M3\} \rangle$

Y

$\langle Y=4, \{\{A1, M1\}\} \rangle$



G

~~$\langle G=12, \{\{A2, M1, M2\}\} \rangle$~~

~~$\langle G=10, \{\{A1, A2, M1, M3\}\} \rangle$~~

$\langle G=12, \{\{\}\} \rangle$

Z

$\langle Z=6, \{\{M3\}\}, \{A2, M2\} \rangle$

$\langle Z=8, \{\{A1, A2, M1\}\} \rangle$

Try the code

Constructing a minimal diagnosis

- For constraint suspension, assumed 1 fault
 - Won't do that here
- Instead, will assume good explanations are minimal
 - Minimum number of culprit constraints
 - $\{\{A1, M1\}\}$ better than $\{\{A1, A2, M2\}\}$.
 - No culprit subsumes another
 - $\{\{A1\}\}$ subsumes $\{\{A1, A2\}\}$.

How to construct minimal diagnoses

- Turn problem on head
 - Instead of constructing minimal nogood sets, construct maximal consistent sets
 - Interpretations mechanism will do nicely
- To construct maximal consistent set
 - Give interpretations the OK assumption for every constraint

Diagnosis algorithm

- Find maximally consistent sets via interpretations mechanism
- If only one, we're done
- If many, rank by score (we'll get to this)
- If several with highest score, have user choose one
- Have user place value
- Repeat until only one interpretation remains

```
(defun diagnose (&aux diagnoses probes result probe)
  (loop
    (setq diagnoses (smallest-diagnoses))
    (when (null (cdr diagnoses)) ;; Only 1 diagnosis?
      (format T "~%Correct diagnosis is: ~A ` diagnoses)
      (return nil)) ;; Done!
    ;; If many diagnoses, have user choose one.
    (setq probes (score-measurements diagnoses))
    (setq probe (user-choose-probe probes))
    ;; Have user test value of probe point.
    (format T "~%Please enter result of measuring ~A:"
      (car probe))
    (setq result (read))
    (set-parameter (car probe) result)))
```

Choosing the smallest nogood sets (i.e., the largest consistent sets)

```
(defun smallest-diagnoses (&aux atms diagnoses (smallest-size 0))
  ;; Find maximal consistent set
  (setq atms (atcon-atms *atcon*)
        diagnoses
          (interpretations atms nil (atms-assumptions atms)))
  ;; Find set of environments that contain the most environment
  ;; (which is the same as those with the smallest nogood sets).
  (unless diagnoses (return-from smallest-diagnoses nil))
  (dolist (diagnosis diagnoses)
    (if (> (env-count diagnosis) smallest-size)
        (setq smallest-size (env-count diagnosis))))
  (remove-if #'(lambda (env) (< (env-count env) smallest-size))
            diagnoses))
```

Computing the cost

- Cost heuristics
 - Assume that each cell has k outcomes
 - To discriminate between n diagnoses, takes at least $\log_k n$ measurements
 - Heuristic: for set of node values, c_i :

$$\sum \frac{c_i}{n} \ln k c_i$$

```
(defun compute-cost (cell diagnoses
                     &aux (cost 0) count (misses 0) (base 0))
  (when (> (length (cell-domain cell)) 0)
    (dolist (d diagnoses)
      (unless (dolist (node (cell-nodes cell))
                      (if (in-node? node d) (return T)))
        (incf misses)))
      (setq base (/ misses (float (length (cell-domain cell)))))
    (dolist (node (cell-nodes cell))
      (setq count base)
      (dolist (diagnosis diagnoses)
        (if (in-node? node diagnosis) (incf count)))
      (unless (= count 0) (incf cost (* count (log count 2)))))
    cost)
```

Running the code

- Example: standard-poly, diagnose
- Example: standard-2bit

Standard 2-bit

- See text

Summary: GDE

- Using multiple environments in ATCON makes GDE more powerful than constraint suspension
 - No need to assume only a single bug
 - No need to unwire and rewire constraints
- Basis of GDE is the interpretations mechanism, turned on head
 - Find maximally consistent environments
 - Choose next probe point using heuristic for which values will differentiate between the most alternative diagnoses

Next time: Symbolic relaxation

- Read Waltzer chapter in BPS
- Get copy of Charniak article from ILS or from Web
- Good luck on projects