An Application of Constraint Programming to the Design and Operation of Synthetic Aperture Radars

Michael Holzrichter

Sandia National Laboratories

Sandia National Laboratories

Exceptional service in the national interest

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Presentation Outline

- Fragmentation of logical model of SARs
- Constraint programming paradigm and propagation networks
- SAR Inference Engine
- Current areas of investigation

Synthetic Aperture Radars

- Synthetic aperture radars (SARs) image the earth's surface using microwaves.
- SARs are complex systems
  - Hundreds of quantities: center frequency, beamwidth, scene dimensions, etc.
  - Quantities must obey hundreds of relationships: physics, radar equation, trigonometry, etc.

Logical Model of SARs

- In aggregate, the quantities and relationships
  - Form a large web or network
  - Constitute a logical model of the SAR
- Logical model is dispersed
  - People's minds, documents, software
  - Many partially overlapping subsets
- Inconsistencies invariably creep in
  - Cause degraded performance or faults
  - Incur overhead

SAR Inference Engine

- SAR Inference Engine
  - provides a central, common SAR model
  - uses the constraint programming paradigm.
- Constraints come from
  - physics, geometry, signal processing
  - system engineer design choices.
- A propagation network provides the computational foundation.
  - All propagators derived from constraints.

Sandia National Laboratories

Imperative Code to Propagation Network

- An imperative code may compute a propagating wave's one way travel time using the following code:

```
velocity = frequency * wavelength; // Equation 1
traveltime = distance / velocity;  // Equation 2
```

- Inputs: frequency, wavelength, and distance
- Output: velocity, traveltime

Imperative Code to Propagation Network

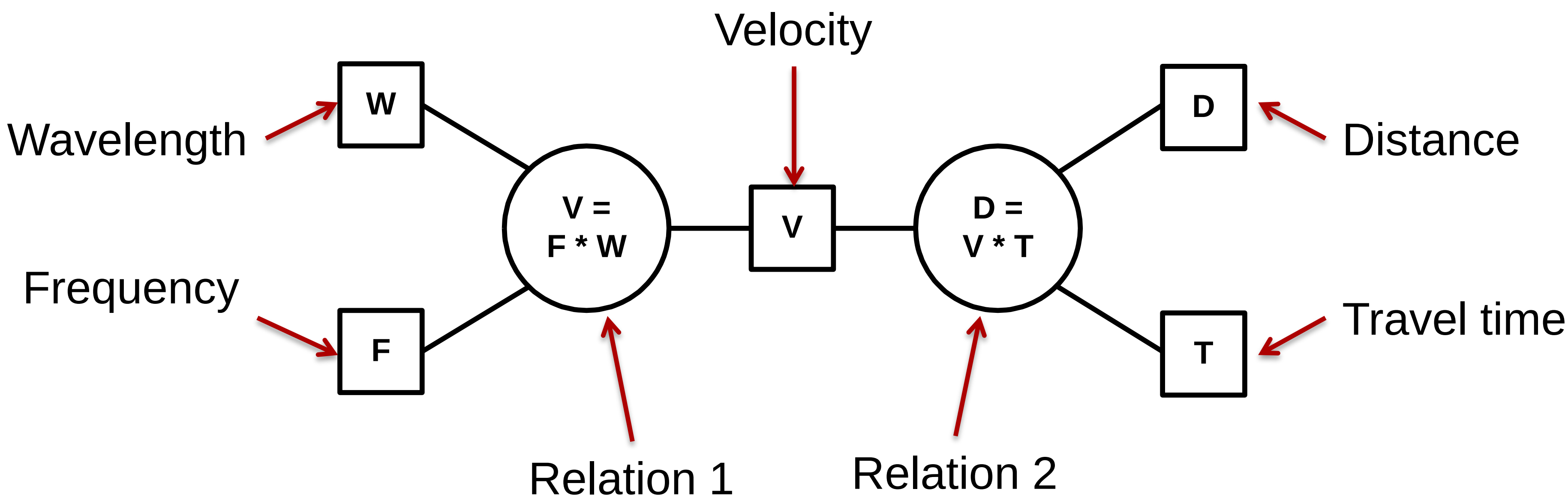
- Alternative arrangements of Equation 1:
  - `velocity = frequency * wavelength;`
  - `wavelength = velocity / frequency;`
  - `frequency = velocity / wavelength;`
- Alternative arrangements of Equation 2:
  - `distance = velocity * traveltime;`
  - `traveltime = distance / velocity;`
  - `velocity = distance / traveltime;`

Imperative Code to Propagation Network

- Relation 1:
  - `velocity = frequency * wavelength`
- Relation 2:
  - `distance = velocity * traveltime`
- Observations:
  - Given any two quantities in a relation the third quantity can be calculated.
  - Both relations have the form "A = B * C" (more on this later).

Imperative Code to Propagation Network

- Relations 1 and 2 depicted as a graph:



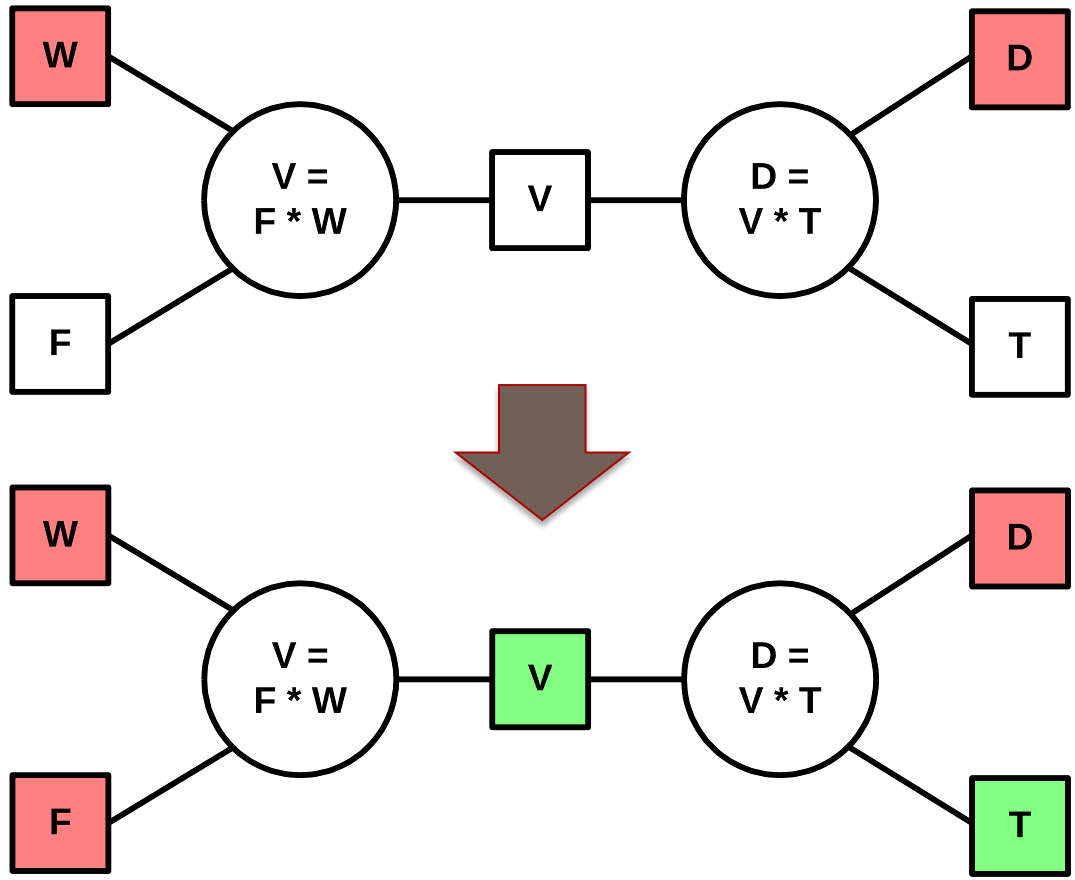
Note: The graph depicts a simple propagation network.

Sandia National Laboratories

Scenario 1: Determine Travel Time

Given a state where wavelength and distance are set…

setting the value of frequency fixes the value of velocity and travel time also.



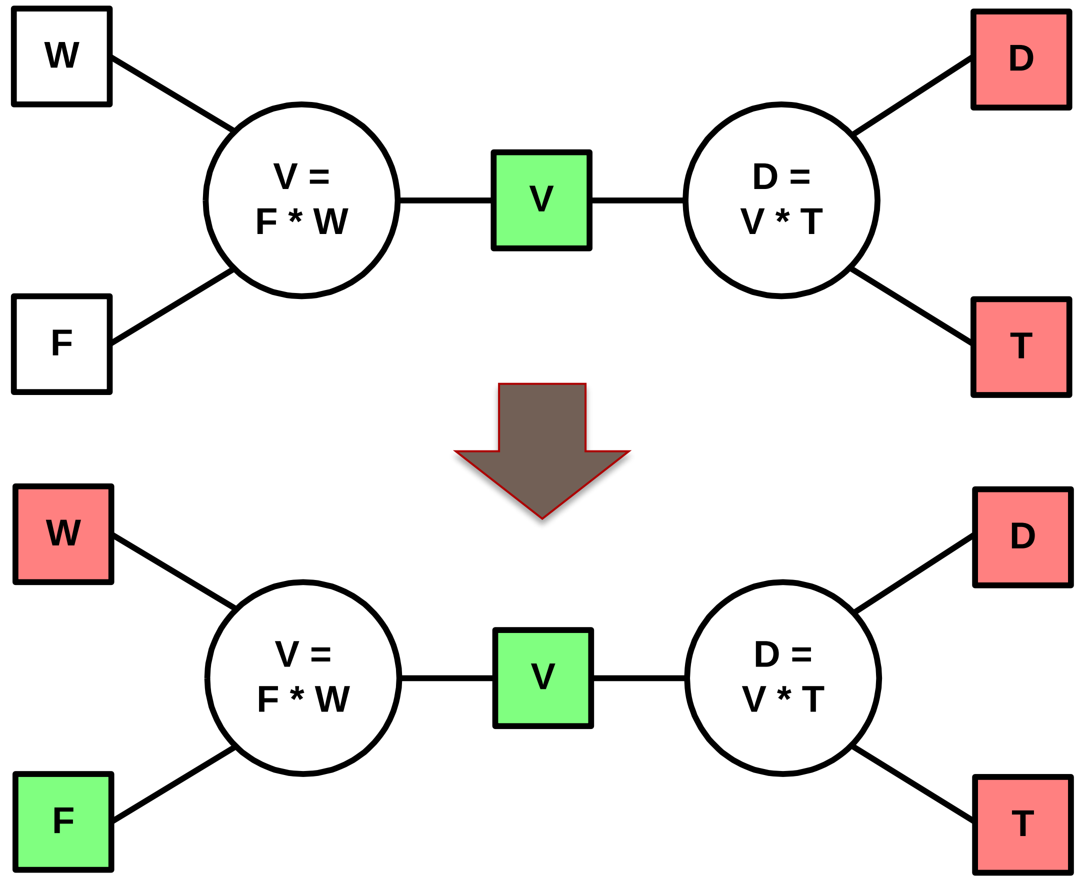
Value undetermined

Value set by “fiat”

Value inferred from constraints

Scenario 2: Determine Frequency

Given a state where distance and travel time are set…



setting the value of wavelength fixes the value of frequency also.

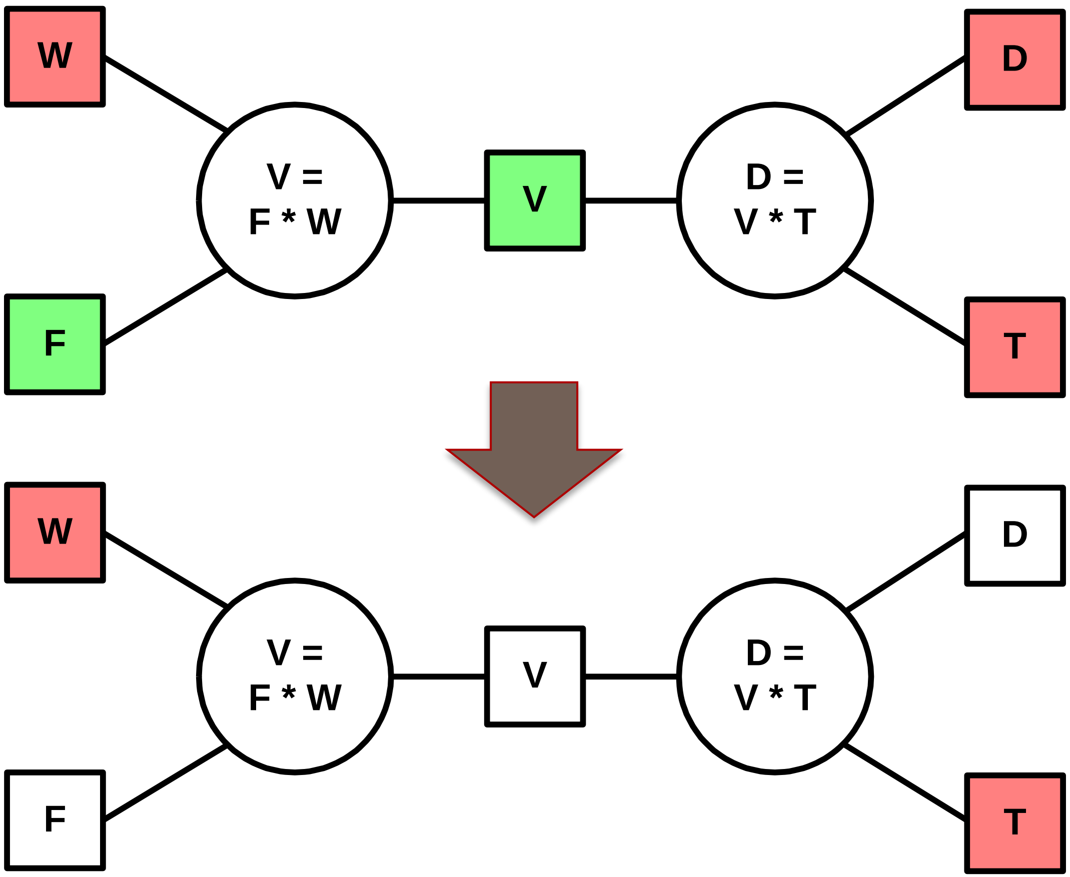
Propagation Network Observations

- Many different sets of inputs are possible.
- Relationships can also be inequalities.
- Operation is progressive.
- Useful to record dependencies of “inferred” values on values determined by “fiat”
  - Enables backtracking
  - Enables determining why a quantity is in its current state.

Example of Selective Backtracking

Before invalidating distance

After invalidating distance



Hello World Example

```
#include <stdio.h>

#include "InferenceEngine.h"

#include "PropWaveConstraint.h"

#include "CenterFrequencyQuantity.h"
#include "CenterWavelengthQuantity.h"
#include "SpeedOfLightQuantity.h"

int main (int argc, char * const argv[]) {

   //  Create Inference Engine and initialize with constraints
   InferenceEngine *inferenceEngine = new InferenceEngine ();
   inferenceEngine->addConstraint (PropWaveConstraint::getInstance ());
   inferenceEngine->concludeInitialization ();

   //  Assign values to center frequency and speed of light quantities
   inferenceEngine->assignQuantityValue (SPEED_OF_LIGHT_NAME,   2.99739141e+008, 0);
   inferenceEngine->assignQuantityValue (CENTER_FREQUENCY_NAME, 1.00000000e+008, 0);

   //  Get value of wavelength
   double centerWavelengthValue = Quantity::getInstance (CENTER_WAVELENGTH_NAME)->getValue ();

   //  Print out result
   fprintf (stdout, "CenterWavelength = %.17e\n", centerWavelengthValue);

   return 0;
}
```

Step 1: Include header files of
- Inference Engine
- Constraints
- Quantities

Step 2: Create Inference Engine and populate it with constraints

Step 3a: Enter values for inputs

Step 3b: Extract values of outputs

Sandia National Laboratories

Scaling Up: Comprehensive SAR Model

- Can a SAR be modeled?
- Full-scale SAR model:
  - Port of existing Matlab model that was used to develop two generations of SARs
  - Over 300 Quantities of "physical interest"
  - Over 250 Constraints of "physical interest"
  - Example of diverse quantities included: SNR, geometry, hardware delays, resolution

More on Constraints

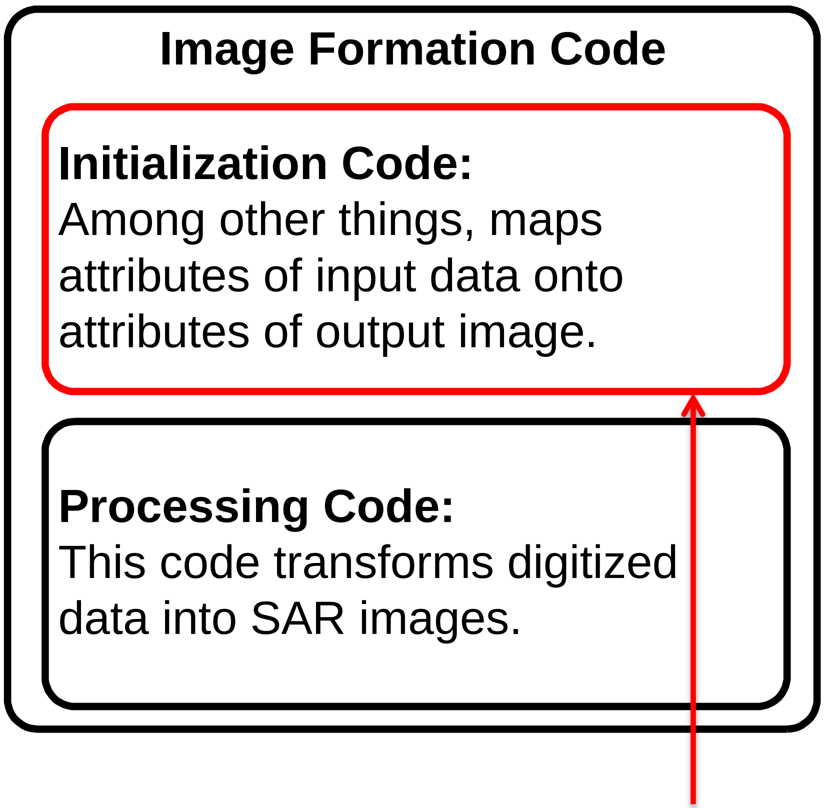
- SAR Inference Engine’s modeling:
  - Just 11 types of constraints
  - Low-level, i.e. minimal semantic content
  - High-level constraints expressed as multiple low-level ones
- Most important types of constraints:
  - A = B op C where “op” is either “+” or “*”
  - “Triangle Constraint” among 3 angles and 3 lengths making up a triangle

Sandia National Laboratories

Usage Throughout the SAR Lifecycle

- Design phase
  - Batch mode: generate performance curves
  - Interactive mode: explore design space
- Mission planning
  - Handle unanticipated conops
- Radar operation
  - Radar operator interface
  - Embedded in radar

Example: Insertion Into Image Formation



Replace mapping logic with calls to SAR Inference Engine.

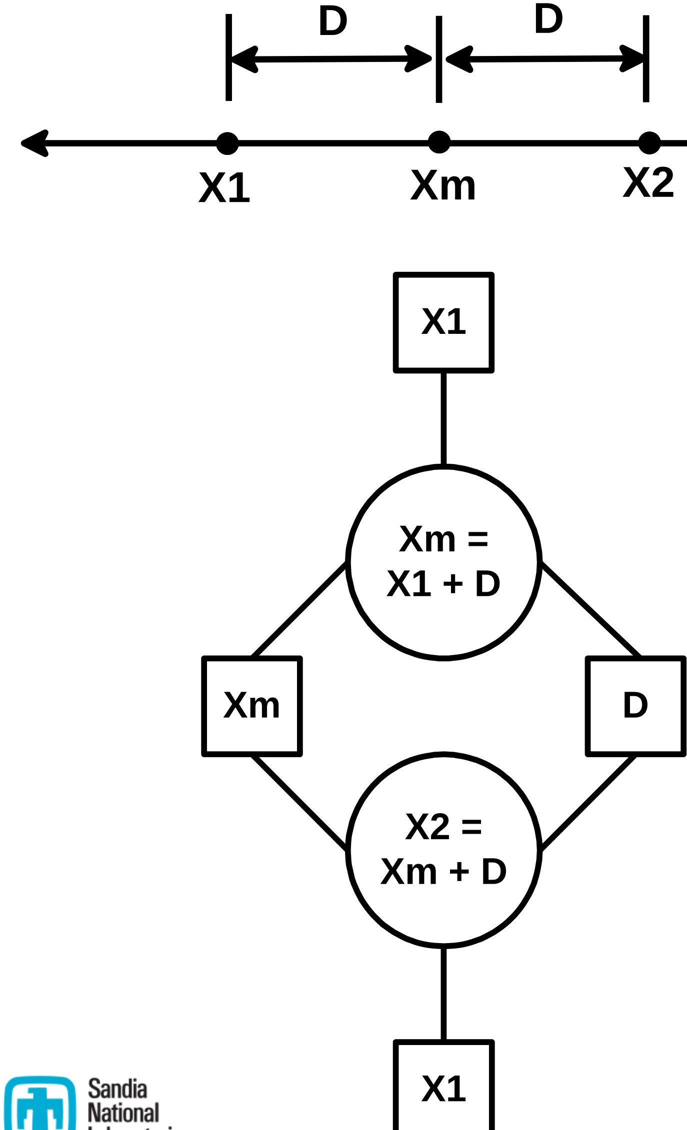
- Code has two phases
  - Setup / initialization
  - Data processing
- Setup phase maps data attributes to image attributes.
- Mapping process intimately tied to logical model of SAR.
  - Generation of data involved the inverse mapping.
- Machine generated code solves speed issue.

Sandia National Laboratories

Current Areas of Investigation

- Multivalued solutions
  - E.g. ambiguous case of Law of Sines
- Completeness: Are there cases where inferences could be made but aren't?
  - E.g. set of triangles to fully define geometry
- Constraint Satisfaction Problem issues:
  - Constraint propagation is weaker than CSP but many issues are common to both.
  - global and local consistency, relaxation

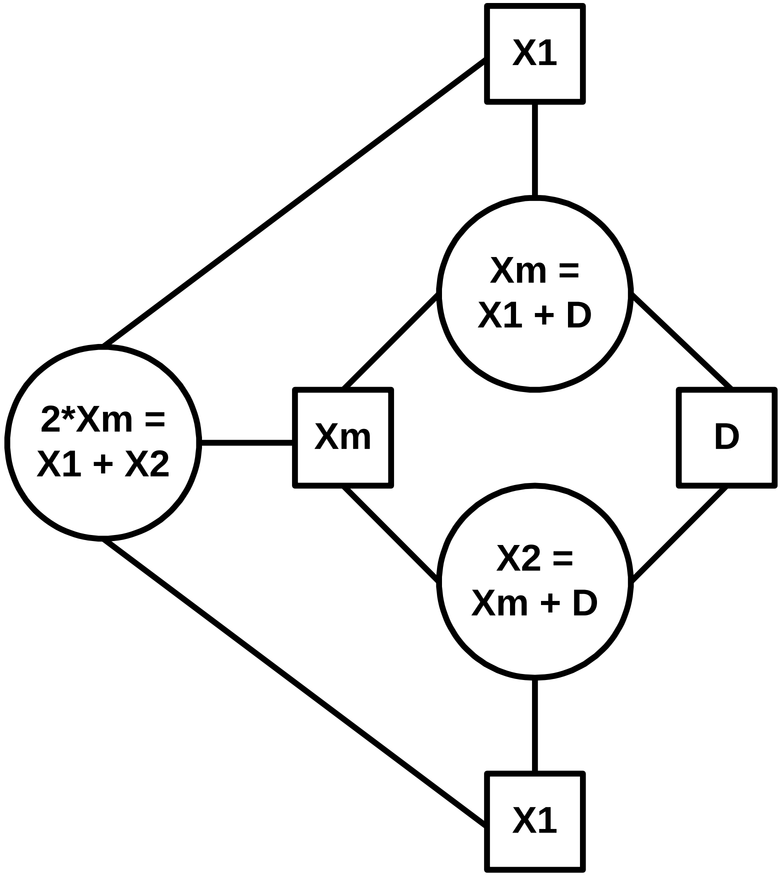
Example Propagation Network Problem



- Line segment mid-point:
  - **`Xm = X1 + D`**
  - **`X2 = Xm + D`**
- X1 and Xm given then d and X2 easily calculated
- Local propagation fails when X1 and X2 given.
- Solutions:
  - “relaxation”
  - modify network

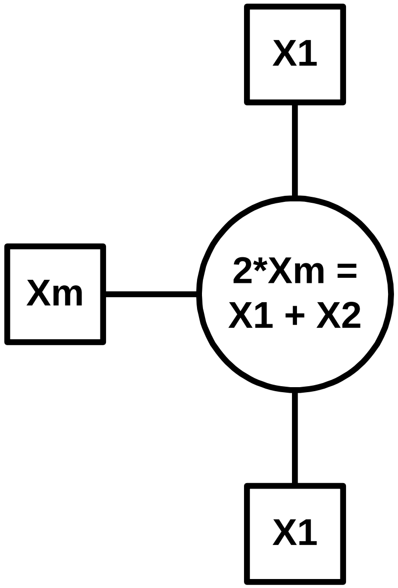
Sandia National Laboratories

Solution 1: Augment Network



- Address problem by augmenting network.
- Add a new constraint
  - `2*Xm = X1 + X2`
- New constraint prevents local propagation from getting stuck
- New constraint mechanically derivable from existing constraints

Solution 2: Simplify Network



- Or address problem by simplifying network.
- Remove D and combine constraints involving D.
- Decision to remove D not suitable for mechanization.
- Simple solution in this case but other cases…

Conclusion

- SAR Inference Engine
  - A implements a single logical model
    - Avoids inconsistencies arising from multiple fragmented models
    - Focuses refinement and maturation efforts.
  - Is usable throughout lifecycle of radar
    - Design through deployed operation
  - Diverse uses and comprehensive scope enabled by
    - Constraint programming paradigm
    - Constraint propagation network