Symmetries in Software

Evan Fortunato

HCSS 2017

May 10th, 2017

This Research was developed with funding from the Defense Advanced Research Projects Agency (DARPA)

Abstraction / Composition is the Computer Science Approach to Managing Complexity

- If Abstraction Holds, System Functions Can Be Reasoned Over Compositionally
  - Allows for scalable engineering, testing and development
  - Allows for fault tolerant cascades of resilient methods
  - Allows for predictable development (budget, schedule, etc.)
- Interfaces are Simple Compared To & Invariant Over Underlying Implementations
  - Abstractions practically hold only over a narrow range of environmental conditions – abstraction doesn't work across a wide range of applications
  - Abstractions practically hold due to wild overprovisioning of resources – abstraction doesn't work well in SWAP (Size, Weight and Power) limited cases
- More Complex Interfaces Allows for More Resilient Abstractions, but Increases the Complexity of Composition
  - Deep Specification allows for formal analysis
  - Exposing more internal implementation details allows for more optimization
  - Exposing more environmental conditions allows for more robustness
- Is There Another Approach for Achieving Resilient & Efficient Compositionality?

We Will Heavily Borrow from the Concept of Symmetries in Physics

- What Do We Mean by Symmetry?
  - Physicists define a symmetry of an action by its conserved quantity
  - An action has a symmetry if there is a property of a state which remains unchanged by the action
  - Lots of well known symmetries (mirror, space, time, energy, momentum, etc.)
  - Direct relationships to mathematical groups and equivalence classes
- Noether’s Theorem From Physics Provides a Useful Roadmap
  - Simple Version: Symmetries Imply Conserved Quantities, Conserved Quantities Imply Symmetries
  - Use: You don’t need to understand all the details – if you can identify a symmetry, then you can deduce a lot from the conserved quantities, even if you don’t know the underlying equations of motion
- The Next Few Slides Walk Through Some Examples To Get Everyone On The Same Page and Show the Relationship between Abstraction and Symmetry
  - Conclusion: Abstraction is a form of Symmetry

How a Physicist "Sees" Bits (and Qubits)

- A Single Two State System: $|T\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ $|F\rangle = \begin{bmatrix}0\\1\end{bmatrix}$

- A Set of Bits is Defined by the Kronecker Product of their States:

$$|F\rangle \otimes |F\rangle \overset{\text{def}}{=} |FF\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix} 0 \cdot \begin{bmatrix}0\\1\end{bmatrix} \\ 1 \cdot \begin{bmatrix}0\\1\end{bmatrix} \end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

$$|FF\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \qquad |FT\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \qquad |TF\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \qquad |TT\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

How a Physicist “Sees” Operators on Bits

- Consider Some Basic 1 Bit Operations:

Identity: $I=\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$ $I|T\rangle = |T\rangle \quad I|F\rangle = |F\rangle$

Not: $X=\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$ $X|T\rangle = |F\rangle \quad X|F\rangle = |T\rangle$

SelectTrue: $S^T=\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}$

SelectFalse: $S^F=\begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}$

- Consider a Two Bit Conditional Not Operator:

```
if bit1 == true
 bit2 = not(bit2)
else
 bit2 = bit2
```

$$C_1^T Not_2 = S^T{}_1 \otimes X_2 + S^F{}_1 \otimes I_2$$

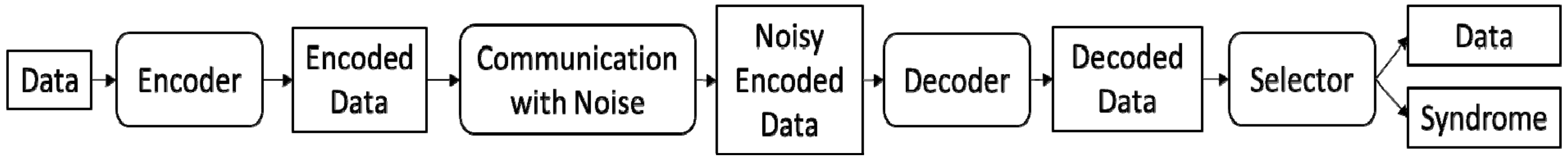
$$= \begin{bmatrix}1 & 0\\0 & 0\end{bmatrix} \otimes \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\0 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}0 & 1 & 0 & 0\\1 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}\boxed{\begin{matrix}0 & 1\\1 & 0\end{matrix}} & \begin{matrix}0 & 0\\0 & 0\end{matrix}\\ \begin{matrix}0 & 0\\0 & 0\end{matrix} & \boxed{\begin{matrix}1 & 0\\0 & 1\end{matrix}}\end{bmatrix} = \begin{array}{|c|c|}\hline X & 0\\ \hline 0 & I\\ \hline\end{array}$$

Some Simple Symmetries

- If the Environment Can Only Apply the I (identity) Operator
  - The State doesn't evolve under the action of the environment
  - All physical bits are conserved – Computation is really easy because we can use our physical bits as our logical bits
  - Note, This represents a perfect abstraction from the environment
- If the Environment Can Apply X (Not) Operator, but Only Rarely
  - Symmetry is broken, but only weakly (because error event is rare). The State will pick up errors at a slow rate
  - Computation is harder now – Can't use the physical bits as our logical bits
  - Can We Restore the Symmetry Dynamically?
    - Yes, via Error Correction
    - With Error Correction, the abstraction is re-established as the effect of the environment is shifted to bits that we don't see…

Normal View of Error Correction – Change the State at Each Step



- Consider the Action of Simple 3 Bit Hamming Code (Triply Redundant Coding with Majority Vote)
  - Starting State: 1 Bit of Data and 2 Auxiliary Bits (both initialized to Zero to Start)
    - True: $|1\rangle \otimes |00\rangle$
    - False: $|0\rangle \otimes |00\rangle$
  - Encoder: $C_1^T Not_2 Not_3 = S_1^T \otimes X_2 \otimes X_3 + S_1^F \otimes \mathrm{I}_2 \otimes \mathrm{I}_3$
  - Environmental Noise: Flip at most 1 Bit
    - $\mathrm{I} \otimes \mathrm{I} \otimes \mathrm{I}$: Flip No Bits
    - $\mathrm{X} \otimes \mathrm{I} \otimes \mathrm{I}$: Flip First Bit
    - $\mathrm{I} \otimes \mathrm{X} \otimes \mathrm{I}$: Flip Second Bit
    - $\mathrm{I} \otimes \mathrm{I} \otimes \mathrm{X}$: Flip Third Bit
  - Decoder: $(Not_1 C_2^T\ C_3^T) \cdot (\ C_1^T Not_2 Not_3)$ (note, right most operation applies first)

Explicitly Running Through the Steps…

	True	False
Starting State (Data)	$\vert 1\rangle$	$\vert 0\rangle$
Add 2 Auxiliary Bits	$\vert 1\rangle\otimes\vert 00\rangle$	$\vert 0\rangle\otimes\vert 00\rangle$
Encode (Copy Code)	$\vert 111\rangle$	$\vert 000\rangle$
Environmental Noise (Bit Flip)	$\vert 111\rangle$ $\vert 011\rangle$ $\vert 101\rangle$ $\vert 110\rangle$	$\vert 000\rangle$ $\vert 100\rangle$ $\vert 010\rangle$ $\vert 001\rangle$
Decode (Majority Vote)	$\vert 100\rangle$ $\vert 111\rangle$ $\vert 110\rangle$ $\vert 101\rangle$	$\vert 000\rangle$ $\vert 011\rangle$ $\vert 010\rangle$ $\vert 001\rangle$
Data ⊗ Syndrome	$\vert 1\rangle\otimes\{\vert 00\rangle\,\vert 11\rangle\,\vert 10\rangle\,\vert 01\rangle\}$	$\vert 0\rangle\otimes\{\vert 00\rangle\,\vert 11\rangle\,\vert 10\rangle\,\vert 01\rangle\}$

Our Data Is Preserved! …As long as the Noise Doesn't Get Any Stronger

Alternate View of Error Correction – Dynamically Generate a Conditional Symmetry

Expand out & Partially Evaluate Sequence for Each Environmental Actions:

– (Decoder ⊗ Environment ⊗ Encoder)$\cdot|?\rangle\otimes|00\rangle$

Environment: $\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I}$: $(Not_1 C_2^T\ C_3^T)(\ C_1^T Not_2 Not_3)(\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I})(\ C_1^T Not_2 Not_3)\cdot|?\rangle\otimes|00\rangle$

$= (Not_1 C_2^T\ C_3^T)(\ C_1^T Not_2 Not_3)(\ C_1^T Not_2 Not_3)(\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I})\cdot|?\rangle\otimes|00\rangle$

$= (\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I})(Not_1 C_2^T\ C_3^T)\cdot|?\rangle\otimes|00\rangle = (\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I})\cdot|?\rangle\otimes|00\rangle$

Environment: $\mathrm{X}\otimes\mathrm{I}\otimes\mathrm{I}$: $(Not_1 C_2^T\ C_3^T)(\ C_1^T Not_2 Not_3)(\mathrm{X}\otimes\mathrm{I}\otimes\mathrm{I})(\ C_1^T Not_2 Not_3)\cdot|?\rangle\otimes|00\rangle$

$=(Not_1 C_2^T\ C_3^T)(\mathrm{X}\otimes\mathrm{I}\otimes\mathrm{I})(\ C_1^F Not_2 Not_3)(\ C_1^T Not_2 Not_3)\cdot|?\rangle\otimes|00\rangle$

$=(Not_1 C_2^T\ C_3^T)(\mathrm{X}\otimes\mathrm{I}\otimes\mathrm{I})(\mathrm{I}_1 Not_2 Not_3)\cdot|?\rangle\otimes|00\rangle$

$=(Not_1 C_2^T\ C_3^T)(\mathrm{X}\otimes\mathrm{X}\otimes\mathrm{X}) = (Not_1 C_2^T\ C_3^T)(\mathrm{X}\otimes\mathrm{X}\otimes\mathrm{X})\cdot|?\rangle\otimes|00\rangle$

$=(\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{X})(Not_1 C_2^T\ C_3^T)\cdot|?\rangle\otimes|00\rangle = (\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{X})\cdot|?\rangle\otimes|00\rangle$

Environment: $\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{I}$: $(Not_1 C_2^T\ C_3^T)(\ C_1^T Not_2 Not_3)(\mathrm{I}\otimes\mathrm{X}\otimes I)(\ C_1^T Not_2 Not_3)\cdot|?\rangle\otimes|00\rangle$

$= (Not_1 C_2^T\ C_3^T)(\ C_1^T Not_2 Not_3)(\ C_1^T Not_2 Not_3)(\mathrm{I}\otimes X\otimes I)\cdot|?\rangle\otimes|00\rangle$

$= (Not_1 C_2^T\ C_3^T)(\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{I})\cdot|?\rangle\otimes|00\rangle = (\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{I})(Not_1 C_2^T\ C_3^F)\cdot|?\rangle\otimes|00\rangle$

$= (\mathrm{I}\otimes\mathrm{X}\otimes I)\cdot|?\rangle\otimes|00\rangle$

Environment: $\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{X}$: $(Not_1 C_2^T\ C_3^T)(\ C_1^T Not_2 Not_3)(\mathrm{I}\otimes\mathrm{X}\otimes I)(\ C_1^T Not_2 Not_3)\cdot|?\rangle\otimes|00\rangle$

$= (\mathrm{I}\otimes I\otimes X)\cdot|?\rangle\otimes|00\rangle$

Dynamically Generate a Conditional Symmetry Just the Results

Expand out the Sequence for Each of the Environmental Actions:

- (Decoder ⊗ Environment ⊗ Encoder)$\cdot|?\rangle\otimes|00\rangle$ vs. Environment$\cdot|?\rangle\otimes|00\rangle$

Environment: $\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I}$:	$(\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I})\cdot|?\rangle\otimes|00\rangle$	vs.	$(\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{I})\cdot|?\rangle\otimes|00\rangle$
Environment: $\mathrm{X}\otimes\mathrm{I}\otimes\mathrm{I}$:	$(\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{X})\cdot|?\rangle\otimes|00\rangle$	vs.	$(\mathrm{X}\otimes\mathrm{I}\otimes\mathrm{I})\cdot|?\rangle\otimes|00\rangle$
Environment: $\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{I}$:	$(\mathrm{I}\otimes\mathrm{X}\otimes I)\cdot|?\rangle\otimes|00\rangle$	vs.	$(\mathrm{I}\otimes\mathrm{X}\otimes\mathrm{I})\cdot|?\rangle\otimes|00\rangle$
Environment: $\mathrm{I}\otimes\mathrm{I}\otimes\mathrm{X}$:	$(\mathrm{I}\otimes I\otimes X)\cdot|?\rangle\otimes|00\rangle$	vs.	$(\mathrm{I}\otimes\mathrm{I}\otimes X)\cdot|?\rangle\otimes|00\rangle$

Result: Generates a Conditional Symmetry
Identity Operator on Bit 1
If Bits 2,3 = False (which they are by definition)

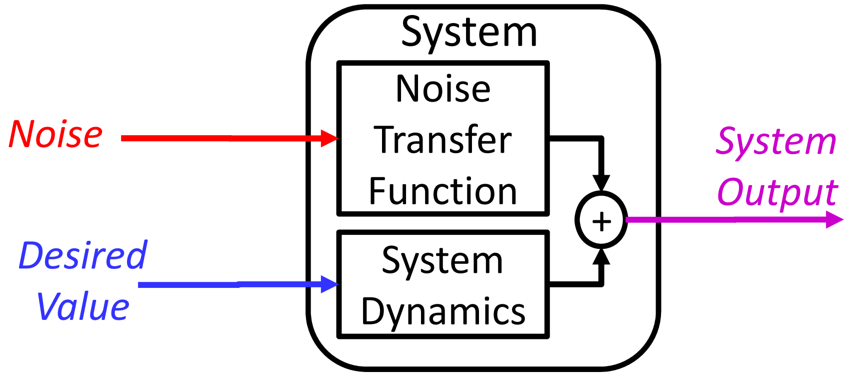
Nice View, But How Does This Really Help?

- Consider a Different Environmental Noise
  - Strong, Frequent Far Field Noise – Flips all of the bits in the channel a large number of times, but always flips all the bits together
  - Noise Operation is $(\mathrm{X} \otimes \mathrm{X} \otimes \mathrm{X} \otimes \mathrm{X} \otimes \mathrm{X} \otimes \mathrm{X} \otimes \mathrm{X} \ldots)^n$
- There is No Possible Finite Distance Error Correcting Code for this Noise
  - But there is a Symmetry: All Bits are Affected the Same, So the Environmental Noise Is Symmetric to Bit Swaps
  - Therefore, there must be a conserved quantity / invariant (parity in this case) that is unchanged by the action of the noise
  - Consider the two bit case:
    - Starting State: $|00\rangle$ -> $|11\rangle$ -> $|00\rangle$ -> $|11\rangle$ -> $|00\rangle$ …
    - Starting State: $|10\rangle$ -> $|01\rangle$ -> $|10\rangle$ -> $|01\rangle$ -> $|10\rangle$ …
  - Parity States are Now the Logical True and False States for Computation and are completely robust to all effects of this environmental noise

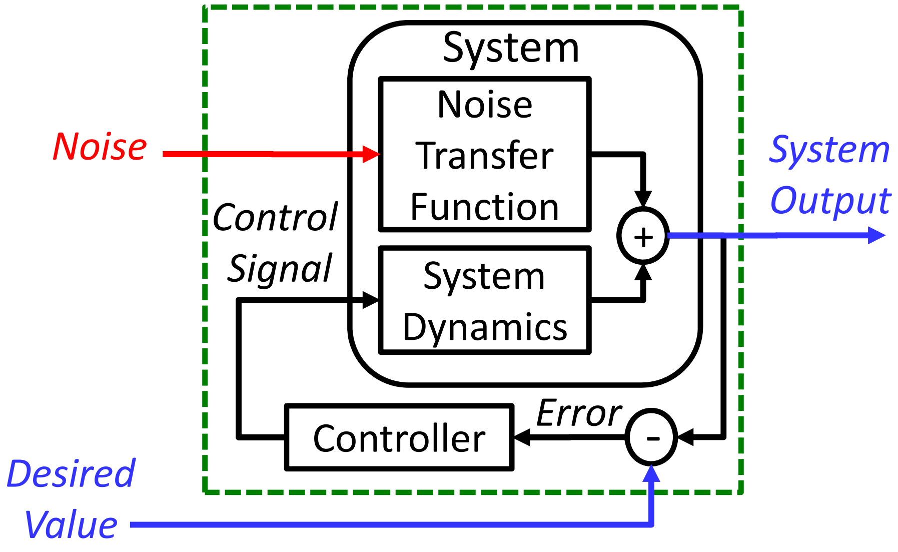
Result: Looking at the Symmetries Allows Efficient Abstraction from the Details of the Noise – Just Need to Compile Code for the Logical States

Control Theory Is Another Form of Dynamically Generated Symmetry

- Consider a Simple Relay System with Noise
  - System output represents sum of noise and system state
  - For small disturbances, system operates approximately correctly
  - Weak symmetry as the system output will naturally drift over time (non-constant)
  - Can't Abstract the System from the Environment

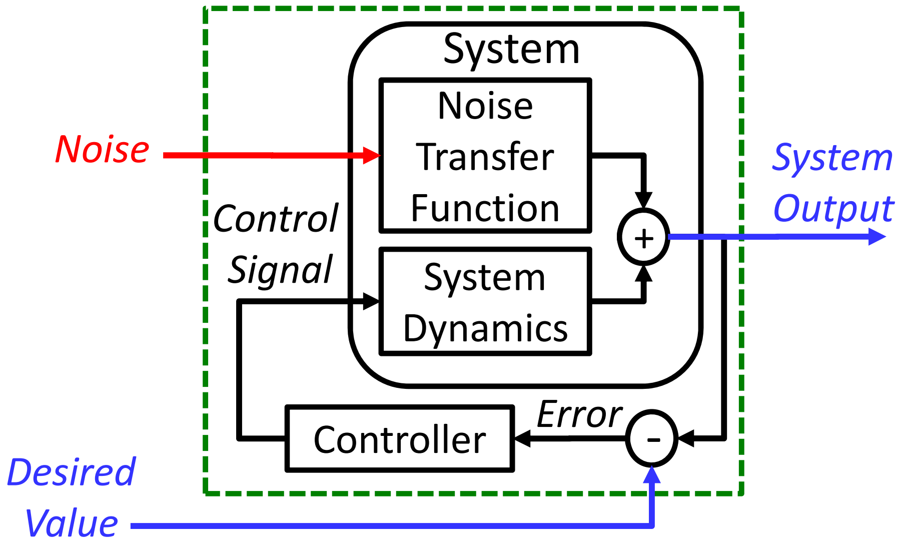


- Consider a simple feedback control loop
  - Controller measures error signal (system output – desired value) and then actuates system to bound error
  - Result: System Output is invariant to noise, therefore a symmetry (identity) is being dynamically generated
  - Can Now Abstract the System from the Environment



There is a Relationship between Error Correction and Control Theory

- Error Correction (EC) is a Form of Closed Loop Control (CLC)
  - CLC: Measure the state, actuate the system to reject the noise
  - EC: Pre-process the state, measure the state, correct the state to reject noise
  - Control systems are typically stuck in the physical basis, they can't cheat in a logical basis
  - Error correction uses an extra degree of freedom (pre-processing the state) to construct an equivalent but more controllable system
- Controllers of Software (not Controllers Implemented in Software) Can Pre-Process Their State
  - What does this allow?



Software Symmetries Drives a New Approach

- Express the architecture as a set of symmetries that reflect the nature of the problem being solved
  - Static Symmetries: Traditional APIs
  - Dynamic Symmetries: OpAmp Analogy (Fixed Gain with Cheap Components Because Resisters are Cheap and Feedback Stabilize Performance)
- Compiler generates instances of the solution (consistent with the symmetries) based on what is needed at the moment
  - Domain Specific Just In Time (DS-JIT) compilation allow this to happen at run time by creating closed loop controller during compilation
  - New programming construct allow us to automate implementation of the JIT (controller) resulting in
    - Faster and cheaper development; better run time performance
    - Improved resiliency to different compositions & environments
    - All at the same time

Example: Preserving Info on a Comms Line

Consider Two Types of Noise Often Found in a Communication Line

1. Strong Far Field: Correlated bits flip, arbitrarily often –
   No Finite Distance Error Code can work but exhibits a swap symmetry
2. Voltage Drift: If you send many 0s, the line reads 0 when 1 is sent –
   Must Balance the number of 1s & 0s sent, line balance symmetry

How is it Done today?

- Parity Encode the Data to Handle Far Field Noise; Often Implemented via Twisted Pair

$$\begin{tikzcd}
0 \arrow[r, "\text{Encode}"] & 00 \arrow[r, "\text{Noise}"] & \{00, 11\} \arrow[r, "\text{Decode}"] & 0 \\
1 \arrow[r, "\text{Encode}"] & 10 \arrow[r, "\text{Noise}"] & \{10, 01\} \arrow[r, "\text{Decode}"] & 1
\end{tikzcd}$$

- 8b10b Encode via Extra Bits to Signal Bits Flips to Balance Line (~25% performance hit)
  - Flip data bits when needed for line balance, use extra two bits to denote data bit flips

How Would It Look Via its Symmetries?

The Parity Code on the Previous Page is the Obvious One, but is it the ONLY One?

$$0 \xrightarrow{\text{Encode}} 00 \xrightarrow{\text{Noise}} \{00, 11\} \xrightarrow{\text{Decode}} 0$$

$$1 \xrightarrow{\text{Encode}} 10 \xrightarrow{\text{Noise}} \{10, 01\} \xrightarrow{\text{Decode}} 1$$

Nope! Here is another Equally Valid Parity Encoding

$$0 \xrightarrow{\text{Encode'}} 11 \xrightarrow{\text{Noise}} \{00, 11\} \xrightarrow{\text{Decode}} 0$$

$$1 \xrightarrow{\text{Encode'}} 01 \xrightarrow{\text{Noise}} \{10, 01\} \xrightarrow{\text{Decode}} 1$$

- Note the difference: Flip the bits, so switch the number of 1s & 0s used to encode the same information
- Note that both use the same decoder!
- Therefore, switching between these encoders can balance the line with no additional bits!
  - ~20% greater throughput (just need a DS-JIT to manage the switching)
- Goal is to have the compiler generate instantiations from the conserved properties of the symmetries (parity, line balance) not just selecting among manually generated instantiations.

A More Complex (and Less Obvious) Example of Symmetries

From the SoSITE Open BAA:

The goal of the System of Systems (SoS) Integration Technology and Experimentation (SoSITE) program is to develop SoS architectures to maintain US air superiority in contested environments, demonstrate rapid integration of mission systems into architectures, and demonstrate the combat effectiveness and robustness of those architectures.

Technical Area 2: Integration Technology Development. This area develops tools to compose distributed architectures and quickly integrate heterogeneous mission systems onto platforms with widely different capabilities.

It Might Not Be Obvious How Symmetries Helps Solve This Problem!

Note, A Detailed Distribution A Presentation Available (If you Want More Details) at
http://www.dtic.mil/ndia/2016/systems/18869_Fortunato_SoSITE_STITCHES_Overview_Long_9Sep2016_.pdf

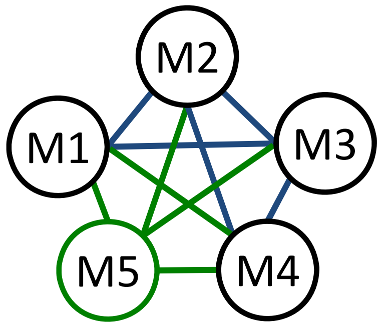
The Goal: Composing Systems That Keep Up With The Times

- DoD has long assumed that homogeneous, fixed-configuration weapon systems are the only way to meet their goals of a superior military force
  - Must last a long time, so requirements are developed for 30+ years out.
  - Meeting 30 year out requirements with today's technology is hard
  - **Result is the best design possible with 20-30 year old technology** and updates are not efficient with respect to time or cost…
- Open Architectures Try to Solve this Problem
  - Requires enormous effort to reach a "global" consensus on the system architecture,
    - Even then, it is only a "local" version of "global"
    - Global standards have to work for everyone, so aren't optimized for your application
  - Result is **heterogeneous components in a homogeneous architecture** – which doesn't work because the architecture needs to evolve with the technology

- What if Global Interoperability Didn't Require a Common Interface at ALL?

Understanding the Trade between Local and Global Message Standards…

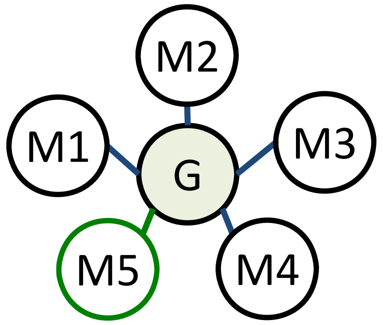
- Local Message Standards
  - Flexible – You Can Add New Messages Easily
  - Inefficient - Require N^2 Transforms (all pairs) for Interoperability



M# = Message #

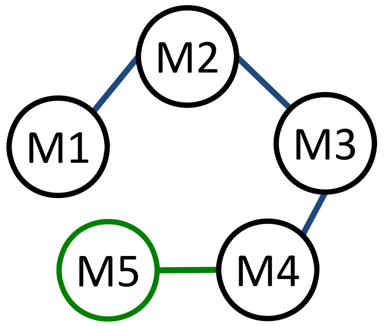
Transform M2 <- M1:
M2 = T21(M1)
Transform M5 <- M1:
M5 = T51(M1)

- Global Open Standards
  - Efficient – N Transforms to/from the Global Standard
  - Not Flexible – Can't change without tremendous effort



Transform M2 <- M1:
M2 = T2G(TG1(M1))
Transform M5 <- M1:
M5 = T5G(TG1(M1))

- Incremental Standards
  - Efficient – ~N Transforms for Interoperability
  - Flexible – You Can Add New Messages Easily



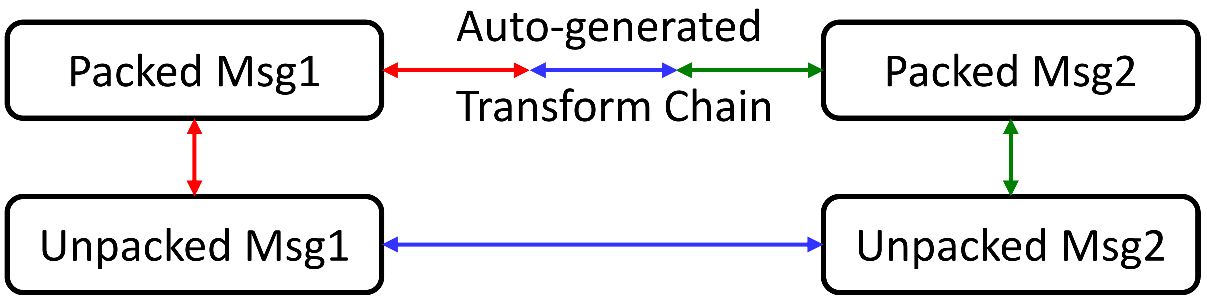
Transform M2 <- M1:
M2 = T21(M1))
Transform M5 <- M1:
M5 = T54(T43(T32(T21(M1))))

Key Innovation:
Field and Transform Graph (FTG)

- Fields are Nodes in the Graph and Contain:
  - A set of subfields (which are defined by other nodes in the graph)
  - A set of properties (mathematically precise specification of node properties)
  - Note: All node information is defined locally, no coordination required!
- Nodes are Connected by Links That Define the Transform from Source to Destination Nodes
  - Each link requires a pair wise human coordination between the source and destination
  - Transforms Expressed in a Domain Specific Language Built for this Purpose
  - Graph algorithms determine a composition of transforms (path through the FTG) that produce the destination message given a source message
- No Global Coordination Required to Update or Evolve Data in the FTG

Handling Packed Representations

- Many real systems mix their interface definition with their implementation
  - Result is a serialized (Packed) form of the interface that can represent multiple different interface messages (e.g., STANAG 4607) with descriptor words for run time resolution
  - Packed messages are often used for run-time efficiency - they tend to be the big / high rate messages in the system. So don't want to unpack if not necessary
- Mirrored Unpacked Nodes Provide an Effective and Efficient Solution
  - Create a Second Unpacked Node that Contains a Structured Version of the Interface
  - Create Transforms between the Unpacked and Packed Nodes
  - Interact with other Interfaces via their Unpacked Representations
  - Auto-generate the Desired (high performance) Packed-to-Packed Transforms



Optimized Performance:

[Packed → Unpacked → Unpacked → Packed] vs. [Packed → Packed]

Connection	PUUP vs PP	Java		C++	
		Speed Mbps	Latency ms	Speed Mbps	Latency ms
R1 -> T1 (No Transform)	PUUP	3000±35	1.1±0.1	2889±52	0.7±0.1
R1 -> T1 (No Transform)	PP	3005±18	1.0±0.1	2897±38	0.7±0.1
R1 -> T2 (Only Change Time)	PUUP	1972±18	1.1±0.1	2891±38	0.7±0.1
R1 -> T2 (Only Change Time)	PP	1967±22	1.2±0.1	2889±53	0.7±0.1
R1 -> T3 (Switch Order Lat, Lon)	PUUP	1100±9	1.5±0.1	1035±32	1.1±0.1
R1 -> T3 (Switch Order Lat, Lon)	PP	1058±9	1.6±0.1	1042±25	1.2±0.1
R1 -> T4 (Change All Fields)	PUUP	685±5	2.0±0.1	963±23	1.2±0.1
R1 -> T4 (Change All Fields)	PP	755±7	1.9±0.1	898±21	1.3±0.05

All interactions via localhost, so no network latencies are involved
Data Gathered on a Standard Quad Core Workstation

Symmetries in Software

- Abstraction Has Been a Critical Technique for Managing Complexity in Software Systems
  - But it has some limitations in its current form
  - Hard to get optimized resilient solutions that have manageable complexity
- Symmetries Provides an Alternate Way to Manage Complexity
  - Benefits of Abstraction (Static Symmetries)
  - Unification with Control Theory (Dynamic Symmetries)
- Significant Benefits Seem Achievable
  - Generating Interoperability without Common Interfaces
  - Optimizations that Don’t Create Maintainability or Resiliency Issues

Questions?