# Transforming Cyber-Physical System Models

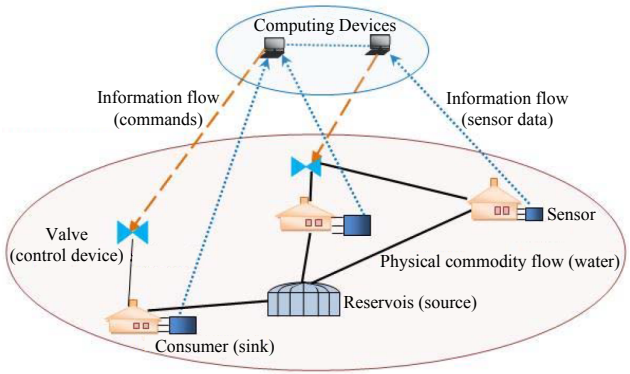
Nathan Jarus
Ph.D. Candidate

Department of Electrical and Computer Engineering
Advisors: Dr. Sahra Sedigh Sarvestani and Dr. Ali Hurson
ISC Graduate Research Symposium

26 April 2017


# Introduction

- *Cyber-physical systems* (CPSs) are characterized by tight integration between a physical network and a cyber network that monitors and controls the physical network.
- Can be used to build sustainable, dependable infrastructure:
  - Make existing physical networks more dependable.
  - Reduce the physical resources needed to build new infrastructure.
  - Incorporate information about the whole system in decision making.
- Examples:
  - Smart grids
  - Intelligent water distribution networks
  - Intelligent transportation systems



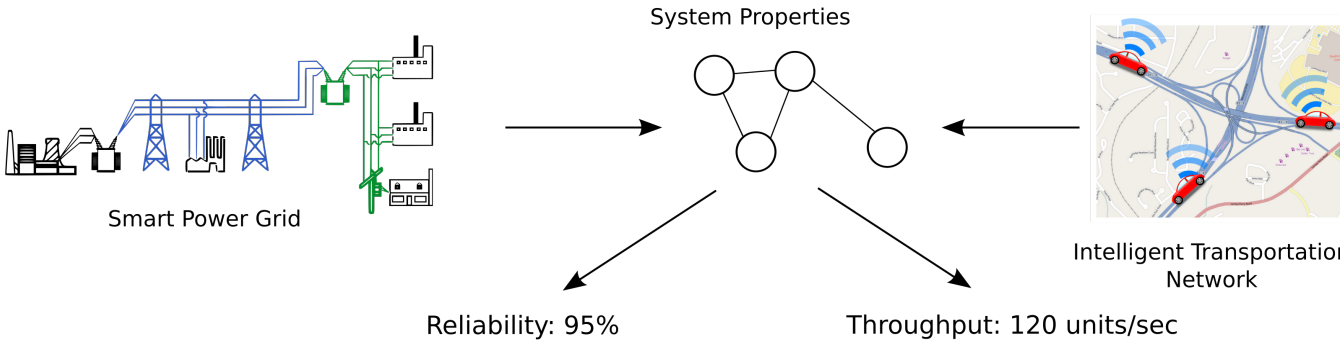
# Motivation

- Designing cyber-physical systems requires constructing multiple models of each design:
  - Performance models
  - Dependability models (reliability, resilience, survivability, etc.)
- Modeling challenges:
  - How do you avoid the need to reconstruct each model every time the system design changes?
  - How do you make sure each model is derived from the same system attributes?
- Identifying and understanding relationships between system attributes improves our understanding of complex system behavior and improves the accuracy of the models.

# Objective

## Create a metamodeling framework that enables *provably correct* system model transformation.

Model transformation converts a model of one type, e.g., a survivability model, to a model of another type, e.g., a reliability model.




Model transformation techniques:

- Reduce the *effort* required to model complex systems.
- Reduce *errors* in the modeling process.
- Enable *cross-domain application* of modeling approaches.

# Related Work: Model Transformation

- *Graph transformation:* Models are represented by graphs and transformed through graph rewriting.
  - Projects: AToM$^3$ (McGill, 2002), CHESS (Intecs, Italy, 2016), CONCERTO (Intecs, Italy, 2015)
- *Class inheritance transformation:* Models are instances of classes in an object-oriented class hierarchy.
  - Projects: OsMoSys (University of Napoli, 2007), SIMTHESys (University of Napoli, 2016)
- *Coalgebraic transformation:* Models are coalgebras in a lattice of possible transformations.
  - Projects: Rosetta (University of Kansas, 2012)

# Open Problems

Our work is necessary because:

- Existing approaches, such as graph or class inheritance transformation, are difficult to apply or inapplicable to certain model types.
  - In particular, relating discrete- and continuous-time models is a challenge.
  - This challenge is critical to address for CPSs.

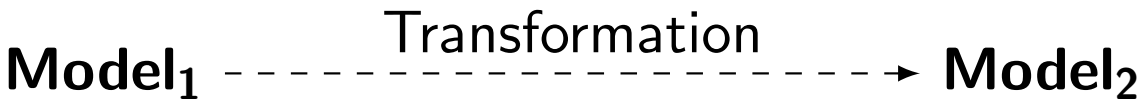
# Open Problems

Our work is necessary because:

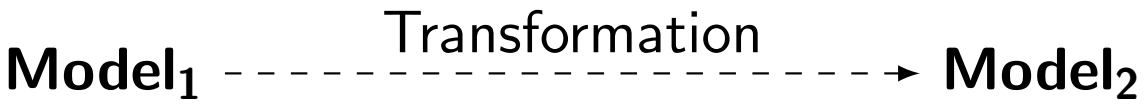
- Existing approaches, such as graph or class inheritance transformation, are difficult to apply or inapplicable to certain model types.
  - In particular, relating discrete- and continuous-time models is a challenge.
  - This challenge is critical to address for CPSs.
- Model transformation techniques must exhibit two attributes:
  - *Correctness:* the inferred model describes the same system as the source model
  - *Specificity:* the inferred model contains at least as much information as possible from the source model

  Of these attributes, correctness is only addressed by Rosetta, and to our knowledge no approaches address specificity.

# Transformation



# Transformation



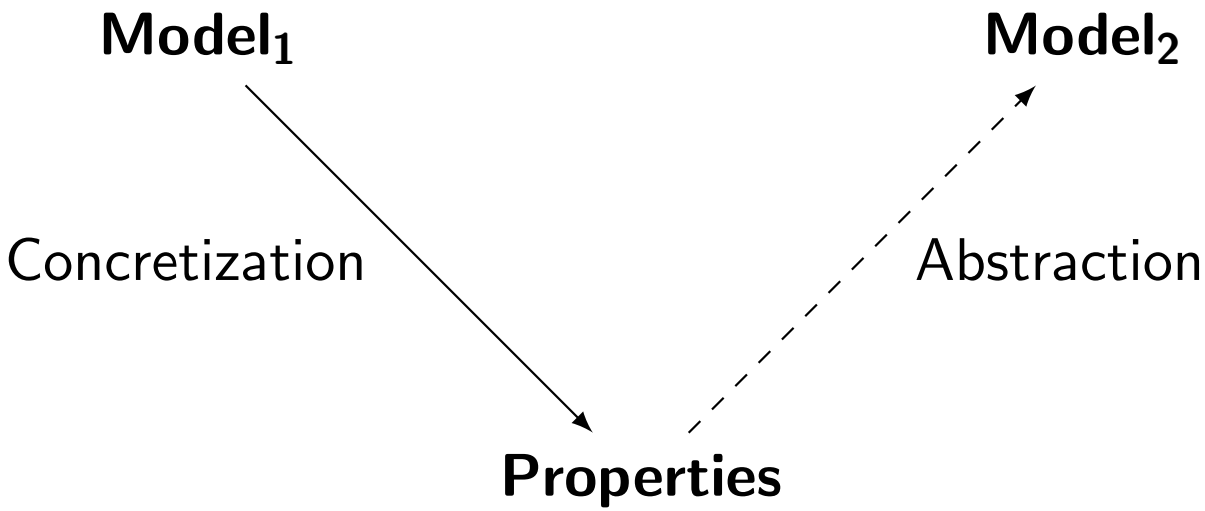

- How do we define transformations between very different models?
  - For example, transforming a model of a nonfunctional attribute into a performance model
  - Can we always make a well-defined mapping from one model of type 1 to exactly one model of type 2?
- The task is complicated by a potential lack of transitivity:
  - Transforming $\textbf{Model}_1 \rightarrow \textbf{Model}_2$, then $\textbf{Model}_2 \rightarrow \textbf{Model}_3$ may not be equivalent to directly transforming $\textbf{Model}_1 \rightarrow \textbf{Model}_3$.

# Original Research Contribution

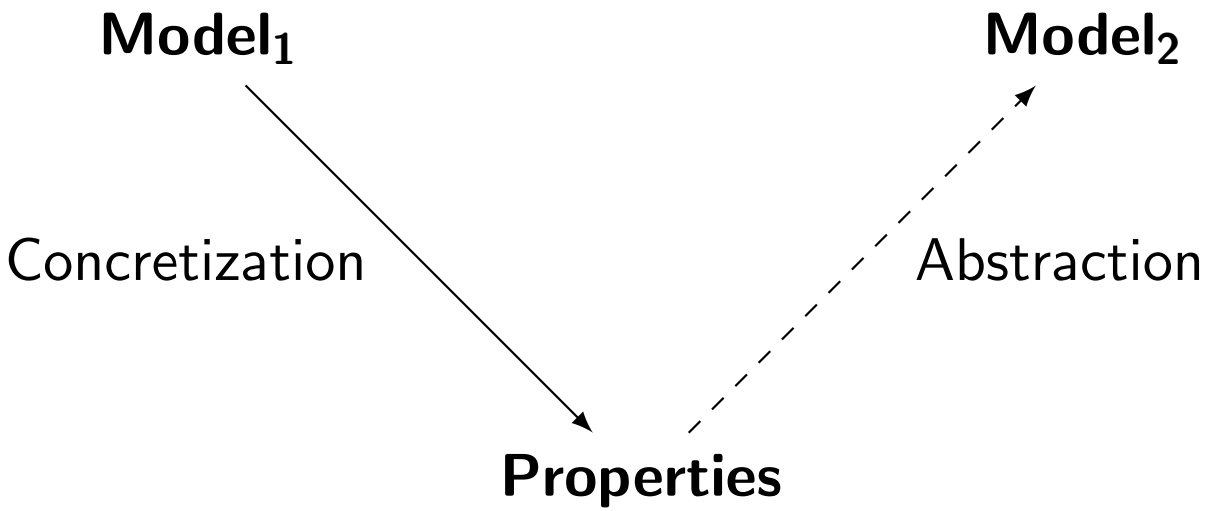
The intended original research contribution of this work is the creation of a model transformation method based on *abstract interpretation*.

- Each system has *concrete semantics* which we capture via *properties*.
- Models' semantics *approximate* the semantics of the system they model.
- We *concretize* system properties from models of that system.
- We *abstract* semantically consistent models from a set of properties.
- We demonstrate that this technique is both correct and specific.

# Concretization



# Concretization




- It is always feasible to *concretize* system properties from a given model.
- However, the information present in the concretized properties may be insufficient to construct a *single model* of a different type.

$$\begin{bmatrix} p & 0 & q & 0 \\ 0 & p & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

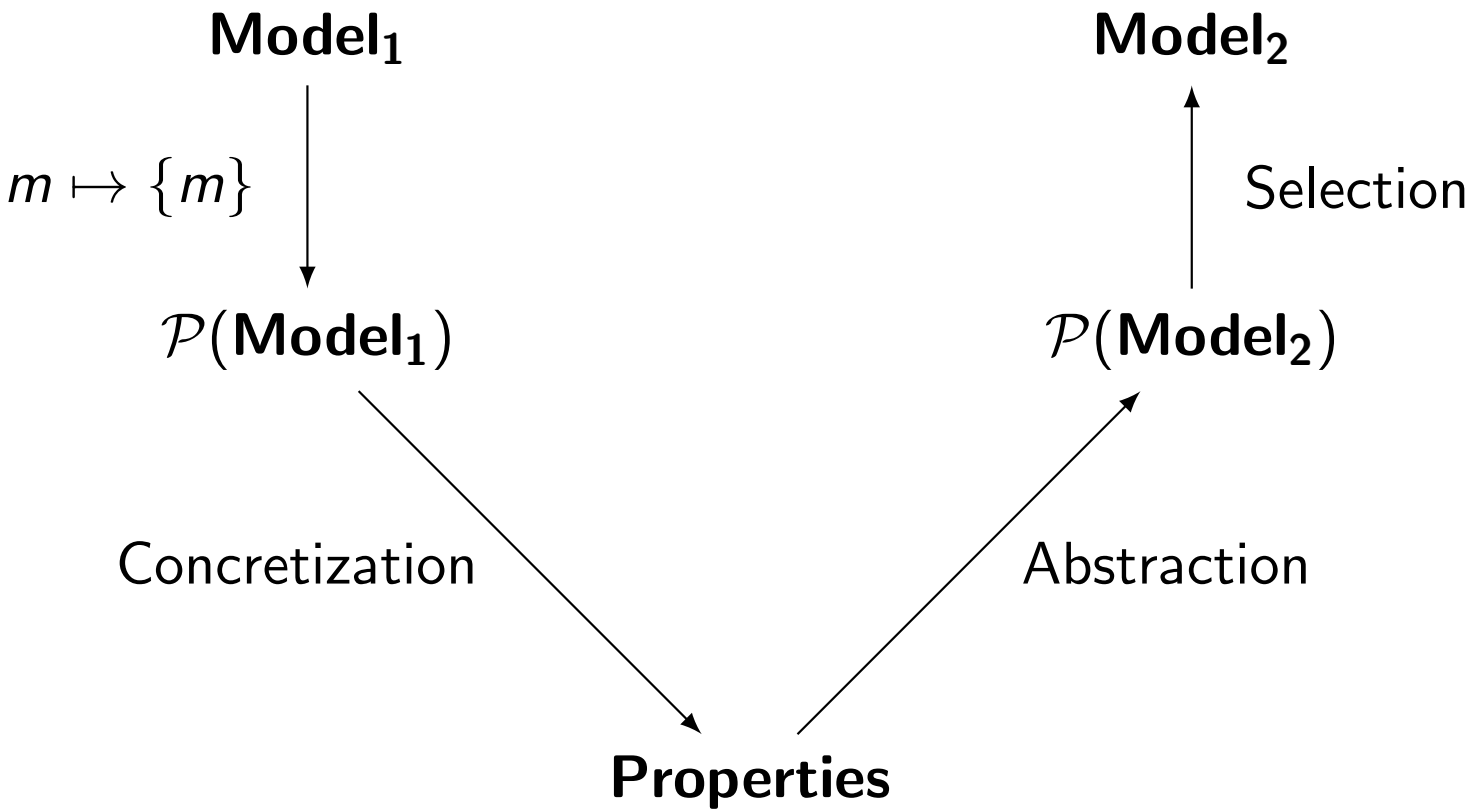
↓ Concretization

**Dependencies**

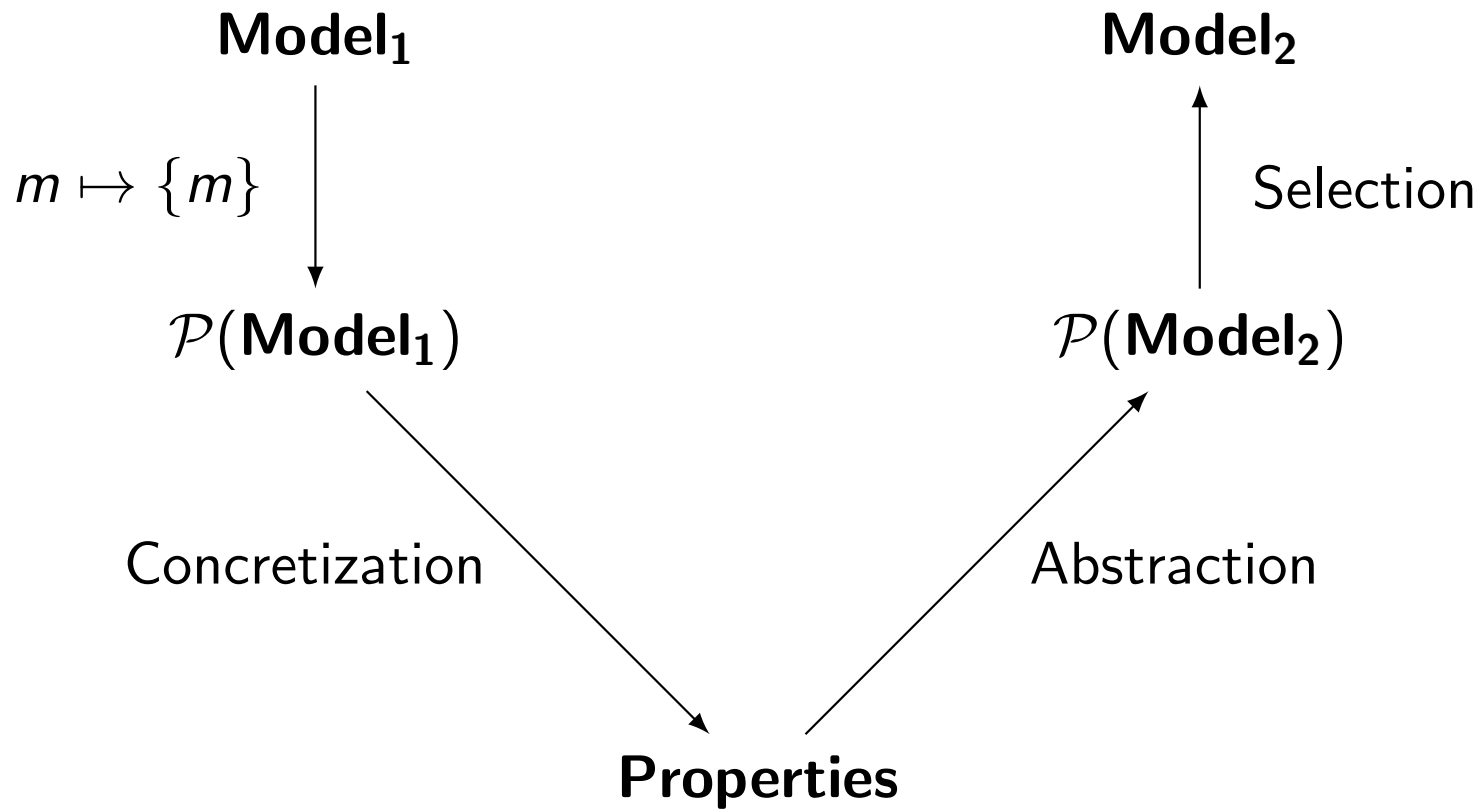
↓ ?

**Topology**

# Abstraction



# Abstraction

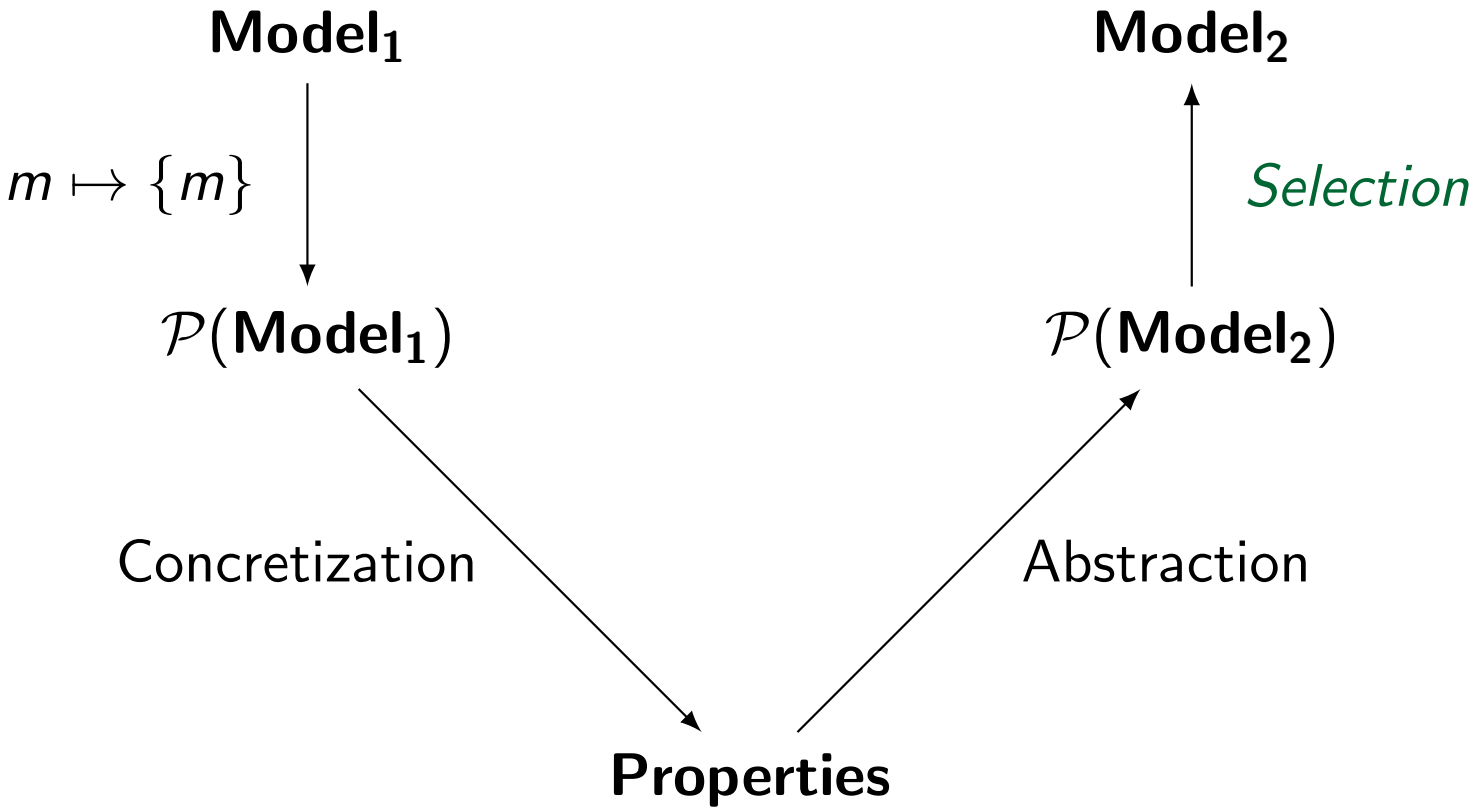



To generate a model of type 2 from a given model of type 1:

- We first *concretize* the properties of the type 1 model.
- Then, we *abstract* a set of type 2 models from these properties.
- A single type 2 model is *selected* from the resulting set.

# Specificity

- *More comprehensive* sets of properties yield *greater specificity*.
- *Greater specificity* leads to *fewer possible* inferred models.
- The selection process requires incorporation of information not present in the initial model.



# Correctness

The inferred models are *consistent* with the properties underlying the initial model.

- Abstraction(Concretization($m$)) $= m$
  - Start with a set of models $m$.
  - Concretize properties from that set.
  - Then, abstract a set of models of the same type: the result must be the initial set $m$!
- Concretization(Abstraction($p$)) $\sqsupseteq p$
  - Start with a set of properties $p$.
  - Abstract a set of models from that set.
  - Concretize properties from those models: the properties must be consistent with $p$!

# Example: MIS Reliability Model

- The *Markov Imbedded Structure* (MIS) technique can be used to derive system reliability from individual component reliability.
- Each state in the Markov chain ($S_0 - S_3$) corresponds to a combination of functional and failed system components.
- State transitions resulting from the behavior of $c_1$ are captured by $P(c_1)$.
- $c_1$ and $c_2$ have the same reliability $p = 1 - q$.

| | Components | | |
|---|---|---|---|
| States | $c_1$ | $c_2$ | System |
| $S_0$ | 1 | 1 | Works |
| $S_1$ | 1 | 0 | Works |
| $S_2$ | 0 | 1 | Works |
| $S_3$ | 0 | 0 | Fails |

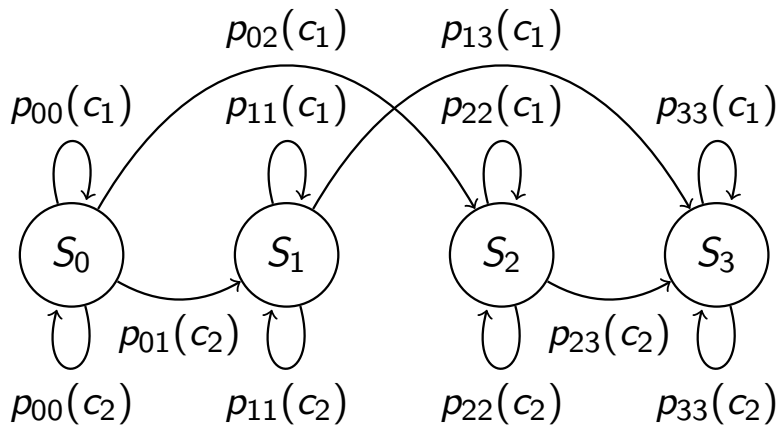
$$P(c_1) = \begin{bmatrix} p & 0 & q & 0 \\ 0 & p & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P(c_2) = \begin{bmatrix} p & q & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & p & q \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Example: MIS Reliability Model

- $\Pi_0$ is the initial state probability vector.
- The vector $u$ identifies the states that are considered functional; here, both lines must fail for the system to fail.
- System reliability is then given by $R$.
  - We assume the behavior of each component is independent.

$$\Pi_0 = [1, 0, 0, 0]$$
$$u = [1, 1, 1, 0]$$
$$R = \Pi_0^T * P(c_1) * P(c_2) * u$$
$$= p^2 + 2pq$$



# Example: Concretized Properties of the MIS Model

- We can extract per-component reliabilities and overall system reliability.
- We can also extract dependencies among sets of components and the conditions under which the system fails.
  - We represent this as sets of *causes* (original component failures) and *effects* (subsequent component or system failures).

| Cause | Effect |
|---|---|
| $\{c_1\}$ | $\emptyset$ |
| $\{c_2\}$ | $\emptyset$ |
| $\{c_1, c_2\}$ | $\{S\}$ |

($S$ denotes the system as a whole.)

Intuitively:

- The system is considered functional if either or both of the components are functional.
- The failure of each component is independent of the other.

# Example: Abstraction of a Topological Model

- The set of topology models abstracted from these properties is infinite.
- We can abstract topological semantics from our system properties and use those to guide construction of a topological model.

# Example: Abstraction of a Topological Model

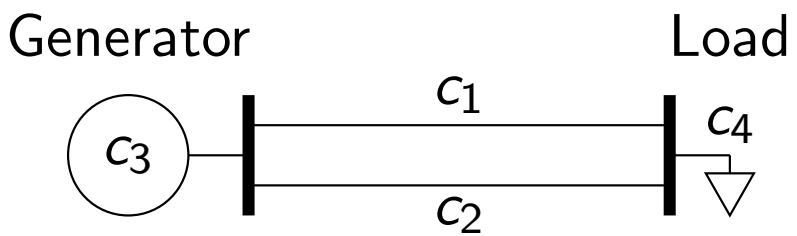
- The set of topology models abstracted from these properties is infinite.
- We can abstract topological semantics from our system properties and use those to guide construction of a topological model.

- First, we stipulate that $c_1$ and $c_2$ are lines in a power grid.
- These components can carry a maximum current represented by $\mathit{capacity}(c_1)$ and $\mathit{capacity}(c_2)$.
- There are two additional components: a generator, $c_3$, and a load, $c_4$.
- The generator has a maximum supply $\mathit{supply}(c_3)$ and the load a demand, $\mathit{load}(c_4)$.

# Example: Abstraction of a Topological Model

- We can constrain $supply(c_3) \geq load(c_4)$, otherwise the system would never be functional.
- Placing $c_1$ and $c_2$ in series would result in a system failure if either fail.
- Therefore, $c_1$ and $c_2$ must be placed in parallel.
- Furthermore, $capacity(c_1) \geq load(c_4)$ and $capacity(c_2) \geq load(c_4)$.

The resulting system topology is shown below:



# Conclusions

- We presented a model transformation approach that has been proven to be correct and specific.
- It can be used to transform models across domains.
- It can also facilitate transformation between models of nonfunctional and functional attributes.
- We have demonstrated an example of both the correctness and specificity of our approach: we abstract a specific model from the properties of another model.
- This research can accelerate advances in design and analysis of complex systems by enabling cross-domain transfer of knowledge.

This work is done in collaboration with Jaxson Johnston.

This work is supported by the Intelligent Systems Center at Missouri University of Science and Technology

# Graph Transformation

- Formulate models as graphs and model transformation as rewriting of the graphs.
- Each model type has a meta-model that describes how its graph can be transformed to graphs of other model types.
- Applies to many formalisms, including Petri nets and Markov chains, but not all.
- Projects: AToM$^3$, CHESS, CONCERTO
  - CHESS and CONCERTO are more focused on modeling multi-core computer systems.

back

# Class Inheritance Transformation

- Each model type corresponds to a class in an object-oriented class hierarchy.
- Models are instances of their type's class.
- Transformation occurs by using inheritance principles to convert a model from one type to another.
- Easy to travel 'up' the class hierarchy; hard to travel back 'down'.
- Projects: OsMoSys, SIMTHESys

back

# Coalgebraic Transformation

- Each modeling formalism is described as a *coalgebra* – a mathematical system useful for describing transitions among states.
- The coalgebras are placed in a lattice to provide a structure for determining which transformations can be performed.
- Can relate different types of models of the same system, such as a model of system functionality and a model of system power consumption.
- Projects: Rosetta

back

# Related Work: Model Composition

- A hierarchical approach is most commonly taken.
- Build subsystem-level models and link them together into system-level models.
  - Subsystem models may be of different types
  - Example: composing a discrete-time model of control software and a continuous-time model of a water valve
- Projects: Ptolemy (Berkeley, 2018), Möbius (University of Illinois, 2015)
- Both projects typically involve models of functional attributes and are created to facilitate system simulation.