Towards Refinement and Generalization of Reliability Models Based on Component States

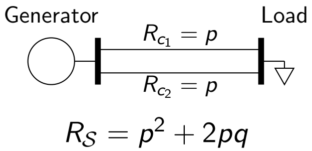
Natasha Jarus, Sahra Sedigh Sarvestani, and Ali Hurson

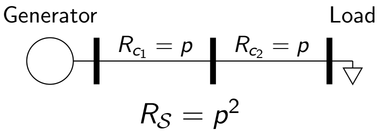
Department of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, USA 65409
Email: {jarus, sedighs, hurson}@mst.edu

November 6, 2019

Introduction

- Critical complex systems must be *resilient*.
- To achieve resilience, designers need a variety of data and models.
- Integrating these data and models requires *metamodeling* — understanding how the implications of various results overlap and interrelate.
- As a demonstration of our metamodeling approach, we use *reliability*: the probability that a system remains functional up to time t.
- Connecting the same components differently can yield different reliability at the system level.
  - For example, a parallel system is more reliable than a series system using the same components.





Metamodeling for Model-Based Design

- Complex systems are often designed iteratively:
  - requirements are gathered,
  - an initial design is modeled,
  - more detail is added to the design or the design is improved to meet requirements,
  - the improved design is modeled, and the process repeats.
- *Metamodeling* approaches can assist designers in creating and updating models throughout this process by
  - reducing the labor involved,
  - preventing modeling mistakes,
  - helping designers explore the design space, and
  - giving models context by relating them to other models.

We seek to metamodel the creation and update of models.

Modifying Reliability Models

Two main goals motivate modification of a model:

- *Refinement* — adding more detail — such as:
  - adding new components,
  - dividing a component into sub-components, or
  - strengthening a constraint on how a component behaves.
- *Generalization* — removing detail — such as:
  - simplifying an unnecessarily complex representation of the system or
  - relaxing a constraint that is unrealistic or renders the design infeasible.

Ultimate Research Objective

A Verifiable Method for Model Refinement and Generalization

Refinement and *Generalization* ultimately modify the assumptions a model makes about the system it represents. Therefore, we

1. analytically describe the system properties captured by a family of related models,
2. formally define refinement and generalization in terms of these properties.

Our goal is for the approach to be verifiable and automatable.

As proof of concept, we apply our method to a commonly used family of reliability models — Markov Imbedded Structure (MIS).

Assumptions of MIS Reliability Models

- System state is determined by the state of its *n* components.
- Reliability (unlike availability) does not account for repair; a component cannot become functional after failure.
- Components can be interdependent: the failure of one can lead to the failure of others.

These assumptions admit a Markov model:

- Initially, all components are functional.
- Component interdependencies define transitions between system states.
- Component reliabilities define the probability of specific transitions.
- A trajectory through the Markov chain corresponds to a sequence of component failures.
- System reliability is determined by the probability of trajectories where the system remains functional.

System Properties of Reliability Models

An MIS reliability model captures the following system properties:

- the components in the system,
- the reliability of each component, and
- which components depend on others to remain functional.

We represent these as

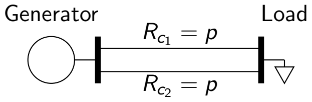
- A finite set of *components*, $\mathbf{C} \subset \mathbf{Comps}$;
- A function giving a *lower bound* on their *reliability*, $\mathrm{R} : \mathbf{C} \to [0, 1]$; and
- A finite set of *dependencies*, $\mathbf{D} \subset \mathbf{Deps}$.

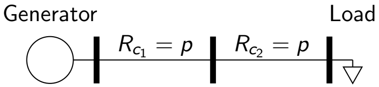
Thus, a collection of system properties $p \in \mathbb{P}\mathbf{rop}$ is a triple $p = (\mathbf{C}, \mathrm{R}, \mathbf{D})$.

Dependencies

Every element of **Deps** is a relation $\langle \mathbf{c} \rightsquigarrow \mathbf{e} \rangle : \mathcal{P}(\mathbf{C}) \to \mathcal{P}(\mathbf{C} \cup \mathcal{S})$.

- **c** are the *causes* of failure.
- **e** are the *effects*.
- The failure of the causes immediately leads to the failure of their effects.
- If $\mathcal{S}$ (the system as a whole) appears in **e**, the causes in **c** also bring down the system.



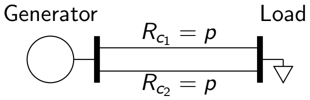


$$\begin{aligned} \mathbf{C} &= \{c_1, c_2\} \\ \mathrm{R}(c_1) &= \mathrm{R}(c_2) = p \\ \mathbf{D} &= \{\langle c_1 \rightsquigarrow \emptyset \rangle, \\ &\quad \langle c_2 \rightsquigarrow \emptyset \rangle, \\ &\quad \langle c_1, c_2 \rightsquigarrow \mathcal{S} \rangle\} \end{aligned}$$

$$\begin{aligned} \mathbf{C} &= \{c_1, c_2\} \\ \mathrm{R}(c_1) &= \mathrm{R}(c_2) = p \\ \mathbf{D} &= \{\langle c_1 \rightsquigarrow \mathcal{S} \rangle, \langle c_2 \rightsquigarrow \mathcal{S} \rangle\} \end{aligned}$$

Examples

The same parallel system can be used to provide either redundancy or extra capacity:



Redundancy:

$$\begin{aligned}\mathbf{D} = \{&\langle c_1 \rightsquigarrow \emptyset\rangle,\\ &\langle c_2 \rightsquigarrow \emptyset\rangle,\\ &\langle c_1, c_2 \rightsquigarrow \mathcal{S}\rangle\}\end{aligned}$$

Capacity:

$$\begin{aligned}\mathbf{D} = \{&\langle c_1 \rightsquigarrow c_2, \mathcal{S}\rangle,\\ &\langle c_2 \rightsquigarrow c_1, \mathcal{S}\rangle\}\end{aligned}$$

In the second system, failure of one component will be catastrophic: the system can no longer deliver the required capacity.

Generalization and Refinement of Reliability Constraints

A straightforward generalization is to loosen the reliability constraint on a component to $r < \mathsf{R}(c)$:

$$\begin{aligned}
&\mathsf{relax\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_,\_] : \mathbf{C} \to [0,1] \to \mathbb{P}\mathbf{rop} \\
&\mathsf{relax\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[c, r] \triangleq (\mathbf{C}, \mathsf{R}', \mathbf{D})
\end{aligned}$$

Conversely, we can refine properties by tightening a constraint to $r > \mathsf{R}(c)$:

$$\begin{aligned}
&\mathsf{tighten\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_,\_] : \mathbf{C} \to [0,1] \to \mathbb{P}\mathbf{rop} \\
&\mathsf{tighten\_rel}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[c, r] \triangleq (\mathbf{C}, \mathsf{R}', \mathbf{D})
\end{aligned}$$

where

$$\mathsf{R}'(c') \triangleq \begin{cases} r & \text{if } c = c' \\ \mathsf{R}(c') & \text{otherwise.} \end{cases}$$

Generalization: Merging Components

Two distinct components c_1 and c_2 can be merged into a single component c_m by replacing every instance of c_1 or c_2 with c_m:

$$\mathrm{merge}_{(\mathbf{C},\mathrm{R},\mathbf{D})}[\_, \_ \to \_] : \mathbf{C} \to \mathbf{C} \to \mathbf{Comps} \to \mathbb{P}\mathbf{rop}$$

$$\mathrm{merge}_{(\mathbf{C},\mathrm{R},\mathbf{D})}[c_1, c_2 \to c_m] \triangleq (\mathbf{C}', \mathrm{R}', \mathbf{D}')$$

where

- $\mathbf{C}'$: Remove c_1, c_2 and add c_m to $\mathbf{C}$.
- R': Set the reliability bound of c_m to the minimum of $\mathrm{R}(c_1)$ and $\mathrm{R}(c_2)$.
- $\mathbf{D}'$: Replace every instance of c_1 or c_2 in $\mathbf{D}$ with c_m.

Refinement: Splitting Components

One component c_m can be split into two fully interdependent components c_1 and c_2. Full interdependence adds the fewest possible assumptions about the system.

$$\mathrm{split}_{(\mathbf{C},\mathrm{R},\mathbf{D})}[\_ \to \_, \_] : \mathbf{C} \to \mathbf{Comps} \to \mathbf{Comps} \to \mathbb{P}\mathbf{rop}$$
$$\mathrm{split}_{(\mathbf{C},\mathrm{R},\mathbf{D})}[c_m \to c_1, c_2] \triangleq (\mathbf{C}', \mathrm{R}', \mathbf{D}')$$

where

- $\mathbf{C}'$: Remove c_m and add c_1, c_2 to $\mathbf{C}$.
- R': Set the reliability bound of c_1, c_2 to $\mathrm{R}(c_m)$.
- $\mathbf{D}'$: Replace each dependency containing c_m with several dependencies involving c_1 and c_2.

Generalization: Adding a Dependency

Independence, as compared to dependence, of two components is a stronger constraint with significant consequences. Adding a dependency from **c** to a component *e* means that **c** brings down *e*.

$$\mathrm{add\_dep}_{(\mathbf{C},\mathrm{R},\mathbf{D})}[\_ \rightsquigarrow \_] : \mathcal{P}(\mathbf{C}) \to \mathbf{C} \to \mathbb{P}\mathbf{rop}$$
$$\mathrm{add\_dep}_{(\mathbf{C},\mathrm{R},\mathbf{D})}[\mathbf{c} \rightsquigarrow e] \triangleq (\mathbf{C}, \mathrm{R}, \mathbf{D}')$$

where

- $\mathbf{D}'$: Add *e* to the direct effects of the dependency **c**.
- $\mathbf{D}'$: Add *e* to the indirect effects of **c** — every dependency that **c** causes.

Refinement: Removing a Dependency

Removing a dependency $\langle \mathbf{c} \rightsquigarrow e \rangle$ implies that e is *independent* of all components in $\mathbf{c}$.

$$\begin{aligned} &\mathsf{remove\_dep}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\_ \rightsquigarrow \_] : \mathcal{P}(\mathbf{C}) \to \mathbf{C} \to \mathbb{P}\mathbf{rop} \\ &\mathsf{remove\_dep}_{(\mathbf{C},\mathsf{R},\mathbf{D})}[\mathbf{c} \rightsquigarrow e] \triangleq (\mathbf{C}, \mathsf{R}, \mathbf{D}') \end{aligned}$$

where

- $\mathbf{D}'$: Remove e from the effects of every dependency whose cause includes a component in $\mathbf{c}$.

Note that

- add_dep affects any dependency that captures the failure of *all* of $\mathbf{c}$.
- rem_dep affects any dependencies that captures the failure of *any* of $\mathbf{c}$.

Example

Consider the dependencies of a system with three independent components:

$$\begin{aligned}\mathbf{D} = \{&\langle c_1 \rightsquigarrow \emptyset\rangle, \langle c_2 \rightsquigarrow \emptyset\rangle, \langle c_3 \rightsquigarrow \emptyset\rangle,\\ &\langle c_1, c_2, c_3 \rightsquigarrow \mathcal{S}\rangle\}\end{aligned}$$

Introducing the dependency $\langle c_1, c_2 \rightsquigarrow c_3\rangle$ results in the following dependencies:

$$\begin{aligned}\mathbf{D}' = \{&\langle c_1 \rightsquigarrow \emptyset\rangle, \langle c_2 \rightsquigarrow \emptyset\rangle, \langle c_3 \rightsquigarrow \emptyset\rangle,\\ &\left.\begin{aligned}&\langle c_1, c_2 \rightsquigarrow c_3\rangle^{\dagger},\\ &\langle c_1, c_2 \rightsquigarrow c_3, \mathcal{S}\rangle^{\ddagger}\end{aligned}\right\} \equiv \langle c_1, c_2 \rightsquigarrow c_3, \mathcal{S}\rangle\\ &\}\end{aligned}$$

- † is the new dependency added by add_dep.
- ‡ results from modifying $\langle c_1, c_2, c_3 \rightsquigarrow \mathcal{S}\rangle$.
- Both rules reduce to one as they share the same cause.

Properties of Generalization

- Apply a list of generalisations, $g = (g_1, g_2, \dots)$, to $p \in \mathbb{P}\textbf{rop}$ by first applying g_1 to p, then applying g_2 to the result of g_1, etc.
- Written: $[\![g]\!](p)$.

- Properties p_g *generalize* properties p_r if there exists g such that $p_g = [\![g]\!](p_r)$.
- We can show that generalization forms a *partial order*: $p_r \sqsubseteq p_g \iff \exists g, p_g = [\![g]\!](p_r)$.

- No "loops": impossible to have $p \sqsubseteq p' \sqsubseteq \dots \sqsubseteq p$.
- Lays the groundwork for proofs of soundness.

Properties of Refinement

- It is not the case that every refinement undoes its corresponding generalization.
  - For example, merging two independent components, then splitting the resulting component results in two fully interdependent components.

- Apply a list of refinements, $r = (r_1, r_2, \dots)$, to $p \in \mathbb{P}\textbf{rop}$ by $[\![r]\!](p)$.
- Properties p_r *refine* properties p_g if there exists r such that $p_r = [\![r]\!](p_g)$.
- We can show that refinement forms a *dual order* to generalization: $p_g \sqsupseteq p_r \iff \exists r, p_r = [\![r]\!](p_g)$.

- These notions of refinement and generalization are compatible: in this sense, each “undoes” the other.

Markov Imbedded Structure Modeling

- System states are defined by the components that are functional.
  - $\textcircled{1101}$ corresponds to the state of a 4-component system where components 1, 2, and 4 are functional and component 3 has failed.
- Transitions between different states occur when a component fails.
- The system starts with all components functional.
- Every state except the last is considered functional.

$$\begin{aligned}
T_i &\triangleq \text{Transition matrix for component } i \\
\Pi_0 &\triangleq [1, 0, \dots] \text{ Initial state probability vector} \\
u &\triangleq [1, \dots, 0] \text{ Functional state vector} \\
\mathrm{R}(\mathcal{S}) &\triangleq \Pi_0^T * T_1 * T_2 * \cdots * T_n * u
\end{aligned}$$

MIS Refinement Example

Goal: Refine a 2-of-3 system from properties p_1.

$$p_1 = (\{c_1\}, \mathrm{R}(c_1) = p, \{\langle c_1 \rightsquigarrow \mathcal{S}\rangle\}).$$

p_1 produces this Markov chain:

$$\begin{tikzcd} \boxed{1} \arrow[loop left, "c_1 : p"] \arrow[rr, "c_1 : q"] & & \boxed{0} \arrow[loop right, "c_1 : 1"] \end{tikzcd}$$

And this MIS model:

$$T_1 = \begin{pmatrix} p & q \\ 0 & 1 \end{pmatrix}$$
$$R(\mathcal{S}) = \Pi_0^T * T_1 * u = p$$

Example (continued)

Create another component via $p_2 = \mathsf{split}_{p_1}[c_1 \to c_1, c_2]$:

$$\begin{aligned} p_2 = (\{c_1, c_2\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = p, \{ & \\ & \langle c_1 \rightsquigarrow c_2, \mathcal{S}\rangle, \langle c_2 \rightsquigarrow c_1, \mathcal{S}\rangle \\ & \langle c_1, c_2 \rightsquigarrow \mathcal{S}\rangle \\ \}) & \end{aligned}$$

Which produces the Markov chain:

$$\begin{tikzcd} 11 \arrow[loop left, "{c_1, c_2 : p}"] \arrow[rr, "{c_1, c_2 : q}"] & & 00 \arrow[loop right, "{c_1, c_2 : 1}"] \end{tikzcd}$$

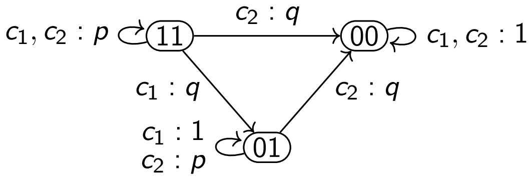
Now $\mathsf{R}(\mathcal{S}) = p^2$ as we now take two steps through the Markov chain.

Example (continued)

We can avoid adding excessive dependencies later by making c_2 independent of c_1: $p_3 = \text{remove\_dep}_{p_2}[c_1 \rightsquigarrow c_2, \mathcal{S}]$.

$$\begin{aligned} p_3 = (\{c_1, c_2\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = p, \{ \\ &\langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow c_1, \mathcal{S} \rangle \\ &\langle c_1, c_2 \rightsquigarrow \mathcal{S} \rangle \\ \}) \end{aligned}$$

This adds a new state to the Markov chain:



Either both components remain functional, or c_1 fails and c_2 remains functional, so $\mathsf{R}(\mathcal{S}) = p^2 + pq$.

Example (continued)

Introduce the third component c_3 by $p_4 = \mathsf{split}_{p_3}[c_2 \to c_2, c_3]$.

$$
\begin{aligned}
p_4 = (\{c_1, c_2, c_3\}, \mathsf{R}(c_1) = \mathsf{R}(c_2) = \mathsf{R}(c_3) = p, \{ & \\
& \langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow c_1, c_3, \mathcal{S} \rangle, \langle c_3 \rightsquigarrow c_1, c_2, \mathcal{S} \rangle \\
& \langle c_1, c_2 \rightsquigarrow \mathcal{S} \rangle, \langle c_1, c_3 \rightsquigarrow c_2, \mathcal{S} \rangle, \langle c_2, c_3 \rightsquigarrow c_1, \mathcal{S} \rangle \\
\}) &
\end{aligned}
$$

The Markov chain is similar to the one derived from p_3, but c_3 adds its own transition probabilities.

$$\begin{tikzcd}
\boxed{111} \arrow[loop left, "{c_1, c_2, c_3 : p}"] \arrow[rr, "{c_2, c_3 : q}"] \arrow[dr, "{c_1 : q}"'] & & \boxed{000} \arrow[loop right, "{c_1, c_2, c_3 : 1}"] \\
& \boxed{011} \arrow[loop left, "{\begin{array}{c} c_1 : 1 \\ c_2, c_3 : p \end{array}}"] \arrow[ur, "{c_2, c_3 : q}"'] &
\end{tikzcd}$$

Thus, $\mathsf{R}(\mathcal{S}) = p^3 + p^2 q$.

Example (completed)

Remove the excess interdependencies for c_2 and c_3: $p_6 = \llbracket \text{remove\_dep\_}[c_2 \rightsquigarrow c_1, c_3, \mathcal{S}], \text{remove\_dep\_}[c_3 \rightsquigarrow c_1, c_2, \mathcal{S}] \rrbracket (p_4)$.

$$\begin{aligned} p_6 = (\{c_1, c_2, c_3\}, \mathrm{R}(c_1) = \mathrm{R}(c_2) = \mathrm{R}(c_3) = p, \{ \\ &\langle c_1 \rightsquigarrow \emptyset \rangle, \langle c_2 \rightsquigarrow \emptyset \rangle, \langle c_3 \rightsquigarrow \emptyset \rangle \\ &\langle c_1, c_2 \rightsquigarrow c_3, \mathcal{S} \rangle, \langle c_1, c_3 \rightsquigarrow c_2, \mathcal{S} \rangle, \langle c_2, c_3 \rightsquigarrow c_1, \mathcal{S} \rangle \}) \end{aligned}$$

The abstracted Markov chain has two new states:

$$\begin{tikzcd}
111 \arrow[loop left, "{c_1, c_2, c_3 : p}"] \arrow[rr, "c_3 : q"] \arrow[dr, "c_2 : q"] \arrow[dd, bend right, "c_1 : q"'] & & 110 \arrow[loop right, "{c_1, c_2 : p \quad c_3 : 1}"] \arrow[dd, bend left, "{c_1, c_2 : q}"] \\
& 101 \arrow[loop above, "{c_1, c_3 : p \quad c_2 : 1}"] \arrow[dr, "{c_1, c_3 : q}"'] & \\
011 \arrow[loop left, "{c_1 : 1 \quad c_2, c_3 : p}"] \arrow[rr, "{c_2, c_3 : q}"'] & & 000 \arrow[loop right, "{c_1, c_2, c_3 : 1}"]
\end{tikzcd}$$

This is the desired system with $\mathrm{R}(\mathcal{S}) = p^3 + 3p^2q$.

Conclusions

Our goal is enabling metamodeling for resilience. To that end:

- We have created an approach to refining and generalizing MIS reliability models.
- Our approach makes explicit the system properties manipulated by such operations.
- Operations on these properties can alter reliability constraints, add or remove components, and add or remove dependencies.
- We apply these operations to MIS reliability models.
- This research enables iterative design of systems.

Future work will take several directions:

- We will prove that generalizations are sound.
- We will extend the approach to higher-level modeling operations, such as model composition.
- We plan to connect these system properties to other model formalisms, allowing exchange of information across models.

Dependency Constraints

Not every subset of **Deps** is a coherent dependency relation; we constrain them by the following equivalences:

- Components do not cause their own failure:

$$\langle c \cdots_1 \rightsquigarrow c \cdots_2 \rangle \equiv \langle c \cdots_1 \rightsquigarrow \cdots_2 \rangle,$$

- Effects are deterministic:

$$\left\{ \begin{array}{l} \langle \cdots_1 \rightsquigarrow \cdots_2 \rangle \\ \langle \cdots_1 \rightsquigarrow \cdots_3 \rangle \end{array} \right\} \equiv \{ \langle \cdots_1 \rightsquigarrow \cdots_2 \cdots_3 \rangle \} ,$$

- Every dependency has a cause:

$$\{ \langle \emptyset \rightsquigarrow \cdots \rangle \} \equiv \emptyset.$$

Dependency Constraints

…and these properties:

- Every component is the sole cause of some effect:

$$\forall c \in \mathbf{C}, \exists \langle c \rightsquigarrow \cdots \rangle \in \mathbf{D},$$

- The system can fail:

$$\exists \langle \cdots_1 \rightsquigarrow \mathcal{S} \cdots_2 \rangle \in \mathbf{D},$$

- Components cannot recover from failure:

$$\begin{gathered} \forall \langle \cdots_1 \rightsquigarrow \cdots_2 \rangle \in \mathbf{D}, \\ \forall \langle \cdots_1 \cdots_3 \rightsquigarrow \cdots_4 \rangle \in \mathbf{D}, \\ \cdots_2 \subseteq \cdots_3 \cup \cdots_4 \,. \end{gathered}$$