

Hierarchical Composition and Abstraction In Architecture Models

Pam Binns and Steve Vestal
Honeywell Labs
{pam.binns, steve.vestal}@honeywell.com

Supported by the Air Force Office of Scientific Research



Compositional Modeling

Compositional Modeling:

- Specify models for individual components
- Automatically generate system model from architecture specification and component models

Desired Benefits:

- Improved reuse of component models
- More easily modifiable architecture specifications
- Improved traceability between designs, models and analyses
- Models that can be used to guide and verify implementations



Compositional Modeling Challenges

Define ADL features and semantic composition rules

- Good level of abstraction (architecture vs component)
- Concise and intuitive for architecture developer
- Permit checks for consistency and completeness
- Permit modular and modifiable architecture specification

Support the development process

- Traceability between well-structured specifications, models, analyses
- Provide design decision insight (e.g. traceable parametric analyses)
- Generate models useful during implementation and verification

Tractable model analysis

- Decomposition analysis methods
- Abstractions
- Mixed-fidelity models



Related Work (that we know of)

For discrete state models

- implements relations
- conformance relations
- abstract interpretations
- compositional model-checking

For hybrid models

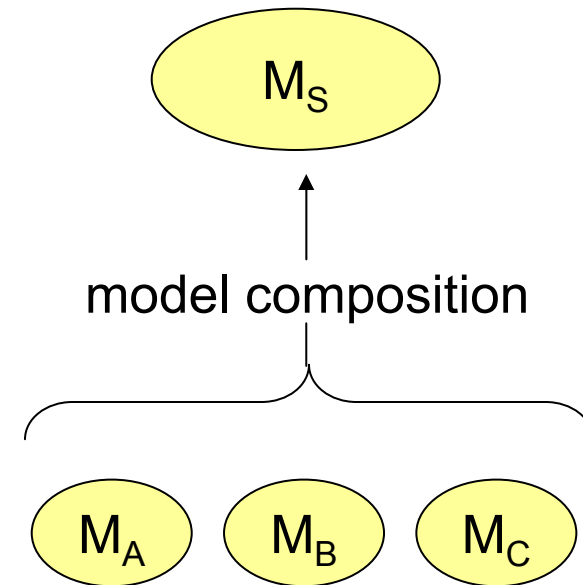
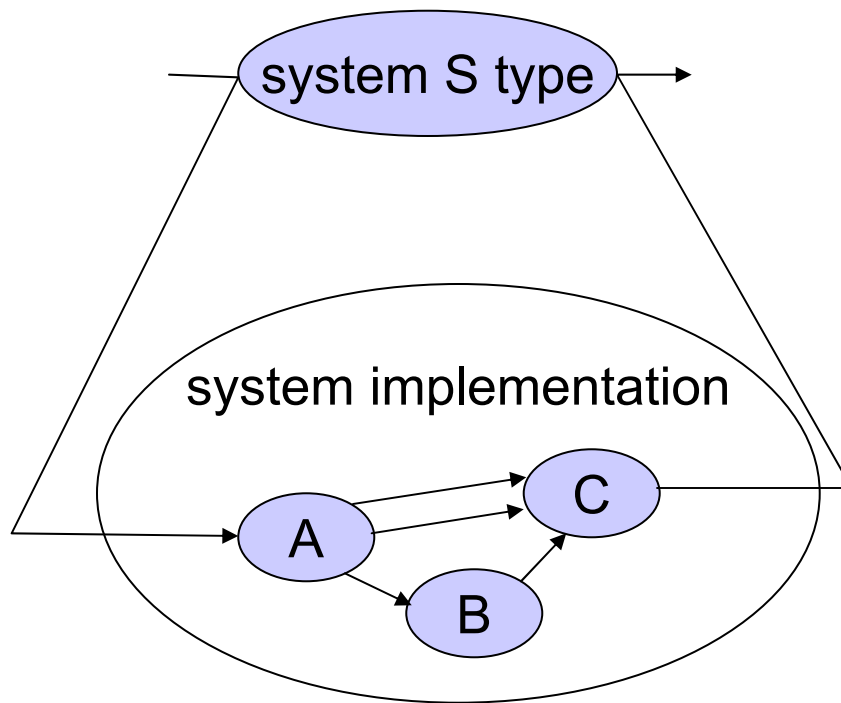
- hierarchical specification languages
- conformance relations

For stochastic models

- stochastic automata
- lumpability theory



Hierarchical Model Composition for Hierarchical Specifications



S may be a component of a larger system, so M_S is composable with other system models.



Hierarchical Model Abstraction



Replace M_S with a simpler abstract model M_S' when composing the higher-level system models.

We assume M_S' is given (e.g. a specification)

We want to show the abstraction is “safe” with respect to the fully detailed compositional model.



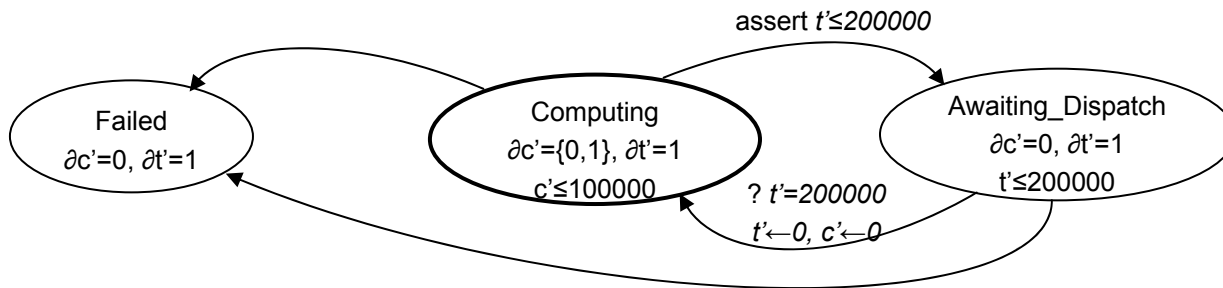
Types of Models

Linear Hybrid Automata for timing/sequencing

Stochastic Automata for safety/reliability



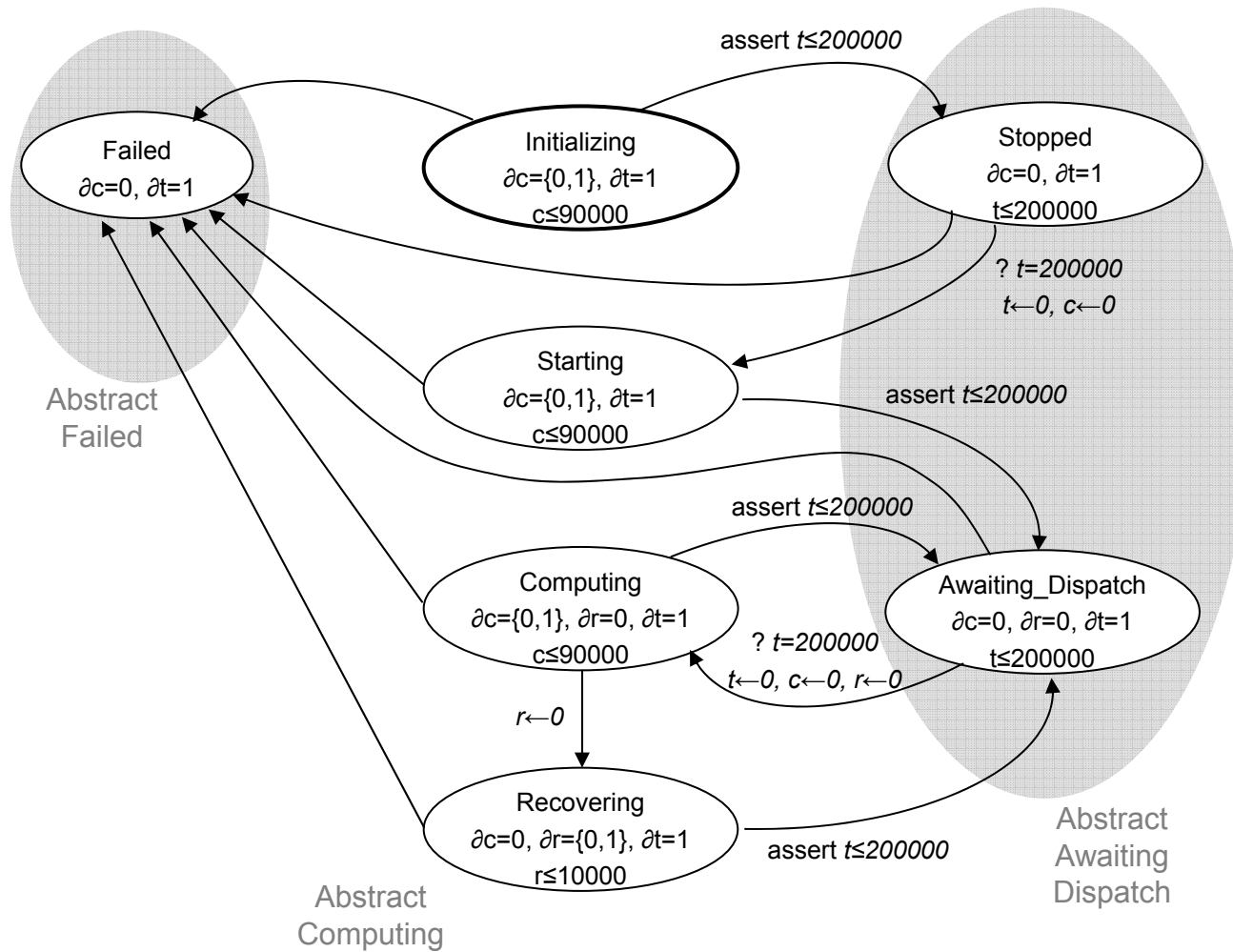
Abstract Linear Hybrid Automata Thread Model



Definition of a classical periodic real-time thread.



Concrete Linear Hybrid Automata Thread Model



Generated from a portion of the MetaH middleware code.



When is the Abstract Model Safe?

The abstraction is safe if $t' \leq \text{deadline}$ at thread completion in abstract model implies $t \leq \text{deadline}$ in concrete model.

We show the set of reachable states of the concrete model is a subset of the reachable states of the abstract model according to the abstraction mapping:

failed \rightarrow failed'

initializing, starting, computing, recovering \rightarrow computing'

stopped, awaiting dispatch \rightarrow awaiting dispatch'

$t \rightarrow t'$

$c+r \rightarrow c'$



Proving Containment

Apply a region enumeration algorithm to both models.

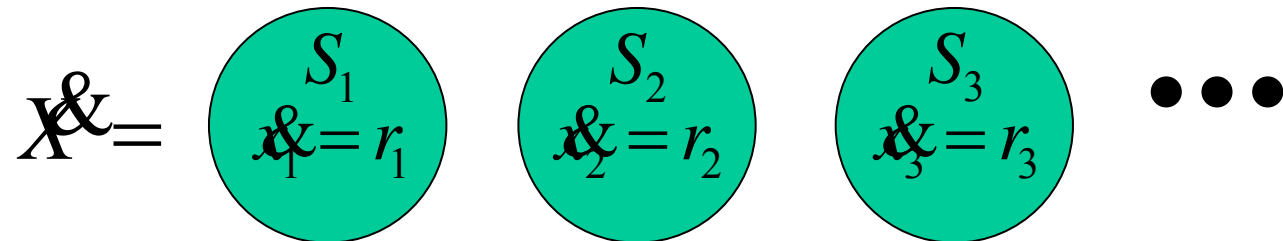
Check for polyhedral containment subject to the abstraction mapping.

This succeeds for any given compute rate, but we need to argue that the abstract model can be substituted for the concrete in a system consisting of composed concrete models.

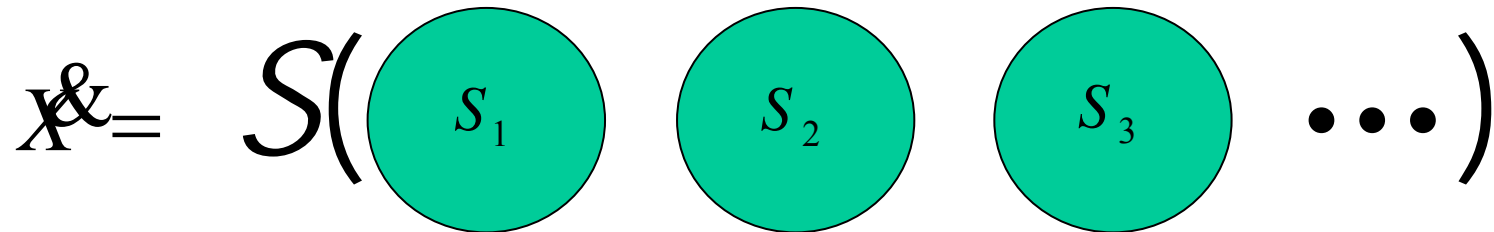


Composing Thread Automata

Instead of this:



Do this:



S is a scheduler function.



Restricting the Scheduler Function

S is a scheduling function for a system of concrete models

S' is a scheduling function when some are replaced by abstract models

Impose the restriction

$S(..a..) \approx S'(..a'..)$ when $a \rightarrow a'$ (\approx accounts for abstraction mapping)

Then

- each thread is scheduled the same regardless of whether other threads are modeled concretely or abstractly
- the abstract variables accumulate at the same rate as their corresponding abstraction mapping, e.g.
 - $t = t'$
 - $c+r = c'$

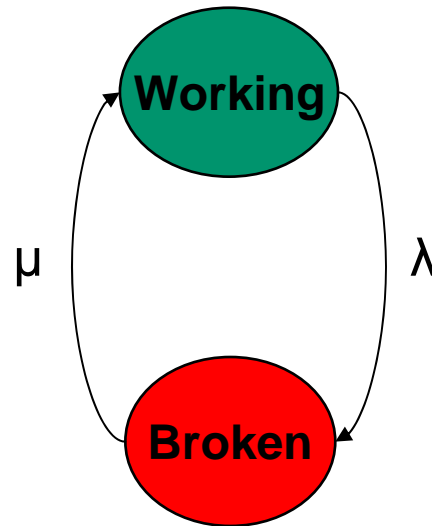
are invariant as the two systems run (assuming this is initially true)

Model-checking needs to cover a class of $S \approx S'$

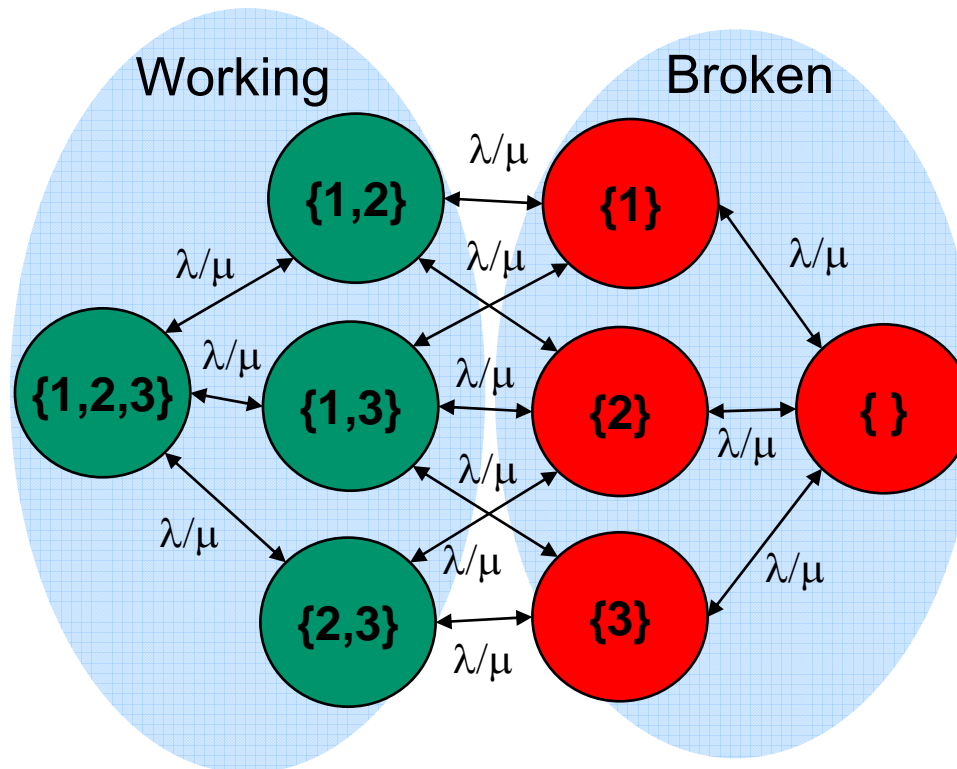
(e.g. we looked at proportional share)



An Abstract Safety Model



A Concrete Safety Model



Abstraction Mappings

- A many-to-one mapping of concrete to abstract states
- An edge between a pair of abstract states exists only if an edge between a pair of concrete states exists where the two concrete states map to the two abstract states. (Corresponding abstract and concrete edges do not necessarily have the same rates.)



Safe Steady State Abstractions

Let π_s be the steady state probability of being in state s .

Let C_a be the concrete states that map to abstract state a .

In general, user may want to specify

$$\pi_a \geq \sum_{C_a} \pi_c \quad \text{for some given } C_a$$
$$\pi_a \leq \sum_{C_a} \pi_c \quad \text{for other given } C_a$$

We investigated the case

$$\pi_a = \sum_{C_a} \pi_c$$



Issues

Abstractions of Markovian concrete models are not necessarily Markovian

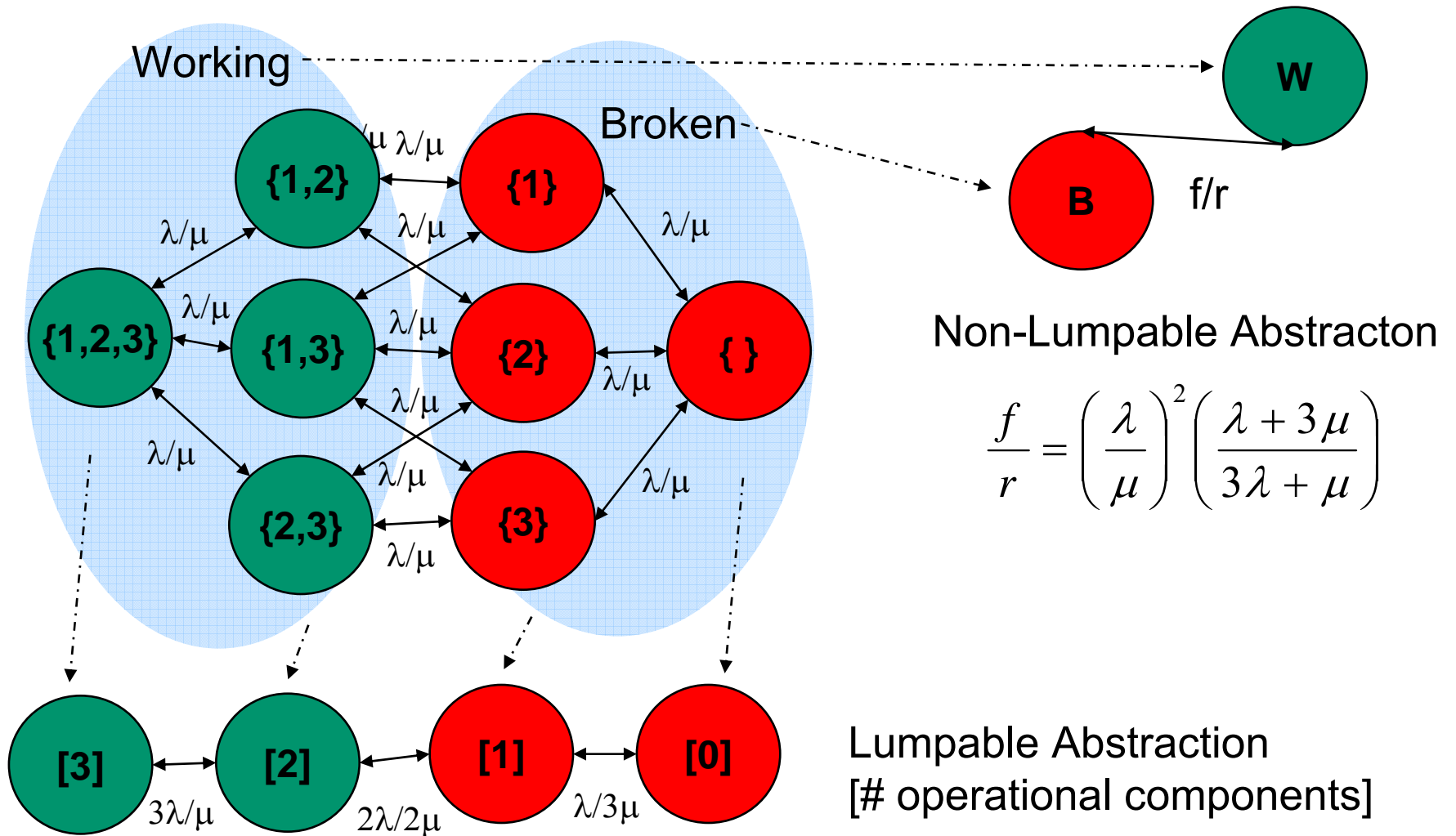
Abstract rates are not necessarily uniquely determined by concrete rates

A concrete model is lumpable if there exists a partitioning of its states (an abstraction) that is Markovian.

Thm: If the abstract model is a lumping of the concrete model, then there is a unique mapping between abstract and concrete rates.



Two Abstractions



$$\frac{f}{r} = \left(\frac{\lambda}{\mu}\right)^2 \left(\frac{\lambda + 3\mu}{3\lambda + \mu}\right)$$

transition rates uniquely defined by concrete rates



Transient Analysis

- Lumpable abstractions have precisely defined transition rates, so transient analyses for safe probability of being in an abstract state appears to hold (using time-varying equality notion of safety)
- For non-lumpable abstractions, might approach this with a looser definition for safe abstraction.



Remarks

Need more general and powerful methods for proving that a given abstract model is a safe abstraction of a given concrete model.

What abstraction mappings and safety relations are adequate in practice?

Does putting the human in the loop help with the tractability problem?

What methodology helps define good abstractions (e.g. is a good specification a good abstraction)?

What tool support would help?

