MEASURE-ADAPTIVE STATE-SPACE CONSTRUCTION*

W. Douglas Obal II
Storage Solutions Center
Hewlett-Packard Company
8000 Foothills Blvd. M/S 5668
Roseville, CA 95747-5668 U.S.A.
doug_obal@hp.com

William H. Sanders
Center for Reliable & High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL, 61801 U.S.A.
whs@crhc.uiuc.edu

ABSTRACT

Measure-adaptive state-space construction is the process of exploiting symmetry in high-level model and performance measure specifications to automatically construct reduced state-space Markov models that support the evaluation of the performance measure. This paper describes a new reward variable specification technique, which, combined with recently developed state-space construction techniques, will allow us to build tools capable of measure-adaptive state-space construction. That is, these tools will automatically adapt the size of the state space to constraints derived from the system model and the user-specified reward variables. The work described in this paper extends previous work in two directions. First, standard reward variable definitions are extended to allow symmetry in the reward variable to be identified and exploited. Then, symmetric reward variables are further extended to include the set of path-based reward variables described in earlier work. In addition to the theory, several examples are introduced to demonstrate these new techniques.

Key words: Measure-Adaptive, State-Space Construction, Performability Modelling

*This material is based upon work supported by DARPA/ITO under Contract No. DABT63-96-C-0069. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA/ITO. This work was completed while Dr. Obal was a Ph.D. student in the Electrical and Computer Engineering Department of the University of Arizona and a Visiting Scholar at the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign.

I Introduction

Model-based evaluation of systems is becoming increasingly valuable as performance and dependability requirements become more stringent and systems become more complex. As a result, much research has been focused on developing the proper mathematical and software tools for effective use of modeling in the product life-cycle.

One line of research has explored the use of Markov process models generated from a higher-level formalism, such as stochastic Petri nets, for the purpose of model-based evaluation. Software tools have been developed for the specification of models in a high-level formalism and for the automatic construction of Markov process models from the high-level specification. The Markov process is typically specified in terms of its initial probability vector and its state transition matrix.

The state space of a model specified in a high-level formalism can be very large ($> 10^7$ states). Research on state-space construction methods has focused on methods for tolerating large state spaces and methods for avoiding large state spaces. Tolerance techniques focus on finding special data structures and efficient algorithms for generating, storing, and computing with the state transition matrix. Examples of this approach include the Kronecker product algorithms of Plateau [1, 2], Buchholz [3], Ciardo et al. [4, 5], Donatelli [6], and Kemper [7], their recent collaboration [8], and the disk-based [9] and "on-the-fly" [10] methods of Deavours and Sanders. Methods for partial exploration of the state space with error bounds for some measures are discussed in [11, 12, 13].

Techniques for avoiding large state spaces typically involve state aggregation, whereby the state space is reduced by partitioning it into equivalence classes and using a single representative from each class in the final state space. Examples of this approach are the papers of Aupperle and Meyer [14, 15], the hierarchical modeling techniques of Buchholz [16], Carrasco's work with stochastic high-level Petri nets [17], stochastic well-formed nets [18], performance evaluation process algebra [19], reduced base model construction [20], and the symmetry exploitation algorithm of Somani [21]. These approaches all use symmetry to reduce the state space by aggregating states that correspond to symmetric configurations. Alternatives to these exact methods are the decomposition method of Ciardo and Trivedi [22], which treats nearly independent submodels as independent and uses fixed-point iteration to solve the model, and the bounds on "quasi-lumpable" models described by Franceschinis and Muntz [23, 24]. The most successful largeness-avoidance techniques are able to construct the reduced state space directly, without first constructing the full detailed state space. Also, most largeness-avoidance techniques exploit symmetry in the model to identify equivalent states.

Another problem, which has long been recognized but left unsolved, is the separation of the "performability measure" from the "system model." Experienced analysts understand that the nature of a model is usually tied directly to the questions one hopes to answer. The performance measure is a formal specification of a question about the system. The system model is a probabilistic specification of the system behavior. Obviously, the choice of modeling methods and the precision with which a model mimics a real system are determined by the accuracy and precision required in the evaluation of the performance measure. This notion is universally accepted. A problem arises when a modeler must artificially add complexity to a model in order to track events or sequences of events that determine the value of the performance measure. We call methods that construct state spaces that are tailored to a particular set of measures "measure-adaptive state-space construction methods."

In previous work, we developed a new technique for detecting and exploiting symmetry in Markov models [25] for the purpose of constructing compact state spaces. In that work we applied group theory to the problem of identifying symmetric states in models composed graphically through shared state variables. Composed models yield a "model composition graph," which is analyzed to find its automorphism group. This group is used to partition the state space into equivalence classes and directly generate a smaller state space. In subsequent work, we developed a new approach to reward variable specification [26] that allows one to define reward measures on sequences of events in the model, as captured in a "path automaton" specification. A "path-based reward variable" can have different reward structures for each state of the path automaton. Another result of this work was a state-space construction procedure for automatically generating a state space from the specification of the system model and the path-based reward variable. This new approach frees the modeler from the need to add additional complexity to a system model in order to support path-based performance measures.

The goal of this paper is to build upon our previous work, presented in [26] and [25], to develop specification and model construction techniques for measure-adaptive state-space construction. In this work, we combine our methods for path-based reward variables with our work on detecting and exploiting model symmetry. This is the final step needed to separate the modeling of system structure and behavior from performance measurement. With this new approach, the model of the system need only reflect real system dependencies. Constraints and dependencies created by the nature of the performance measure are dealt with separately in a new symmetric path-based reward variable formalism.

II Detecting and Exploiting Model Symmetry

In this section we briefly summarize the technique developed in [25] for detecting and exploiting model symmetry. Using a simple abstract modeling formalism, we define models and composed models, and discuss how the information embedded in the composed model is used to detect and exploit symmetries for the purpose of state-space reduction. These concepts are the foundation of measure-adaptive state-space construction, since they determine the structural restrictions on state-space reduction.

A Model Description

We now review the notation we use for describing models.

Definition 1 *A* model *is a five-tuple* $(S, E, \varepsilon, \lambda, \tau)$ *where*

- S *is a set of state variables* $\{s_1, s_2, \ldots, s_n\}$ *that take values in* $\mathbb{N}$*, the set of nonnegative integers. The state of the model is defined as a mapping* $\mu : S \to \mathbb{N}$*, where for all* $s \in S$*,* $\mu(s)$ *is the value of state variable* s*. Let* $M = \{\mu \mid \mu : S \to \mathbb{N}\}$ *be the set of all such mappings.*
- E *is the set of events that may occur.*
- $\varepsilon : E \times M \to \{0, 1\}$ *is the event-enabling function. For each* $e \in E$ *and* $\mu \in M$*,* $\varepsilon(e, \mu) = 1$ *if event* e *may occur when the current state of the model is* μ*, and zero otherwise.*
- $\lambda : E \times M \to (0, \infty)$ *is the transition rate function. For each event* e *and state* μ *such that* $\varepsilon(e, \mu) = 1$*, event* e *occurs with rate* $\lambda(e, \mu)$ *while in state* μ*.*
- $\tau : E \times M \to M$ *is the state transition function. For each* $e \in E$ *and* $\mu \in M$*,* $\tau(e, \mu) = \mu'$*, the new state of the model that is reached when* e *occurs in* μ*.*

Models are connected together through shared state variables to form "composed models." Figure 1 shows an example in which two models are composed via specification of the superposition of two state variables. Models A and B each have state variable sets containing two state variables $\{A.1, A.2\}$ and $\{B.1, B.2\}$. In this case, the second state variable for model instance A is joined to the first state variable for model instance B, forming a single composed model state variable named $C1$. The resulting composed model state variable set $S = \{A.1, C1, B.2\}$. As shown in Figure 1, $C1$ is the connection representing the superposition of $A.2$ and $B.1$.

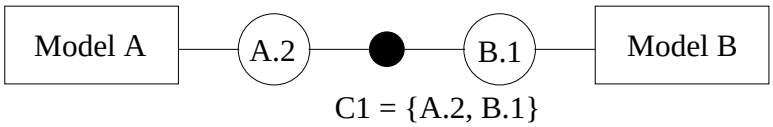


Figure 1: Models are connected through shared state variables.

Definition 2 *A* composed model *is a four-tuple* (Σ, I, κ, C) *where*

- Σ *is a set of models.*
- I *is a set of instances of models in* Σ*. Each instance is a complete copy of a model in* Σ*, and is independent of all other instances, except as explicitly defined through the connection set.*
- $\kappa : I \rightarrow \Sigma$ *is the instance type function.*
- C *is a set of connections. Each connection* $c \in C$ *represents a state variable shared among two or more instances. In this way,* c *represents the superposition of one state variable from each connected instance. The element* c *specifies the set of instance state variables it represents.*

Although the notation in Definition 2 is required for precise definitions of symmetric reward variables, we typically use "model composition graphs" for describing models. As illustrated in Figure 1, boxes represent private state variable fragments, circles denote shared variable fragments, and connection nodes are represented by small solid circles. Lines drawn between the components indicate association.

The following definition of a "model composition graph" was given in [25]:

> A *model composition graph* is an undirected graph, $G = (V, W)$. Elements of the vertex set, V, are private state variable fragments, shared variables or connection nodes. Every instance has exactly one private state variable fragment, but this fragment may be empty. It is possible that all state variables in an instance state variable set are shared. In this case, the empty private state fragment serves as an anchor for the shared variables, as will be understood from the requirements on the edge set. There are two rules that must be satisfied by the edge set, W, of the graph. First, every shared state variable for an instance must be adjacent to the private fragment for that instance. Second, each shared variable is adjacent to exactly one connection node.

The composed model state variable set, event set, and state transition function are defined by the connections made between state variable fragments. The composed model state variable set contains the state variables in each private state fragment, and one state variable for each connection node (recall that connection nodes represent shared state). Likewise, the event set is the union of the event sets for all model instance in the graph.

The state transition function for the composed model is more involved. The global transition from one composed model state to another is defined in terms of the local transition from one model instance state to another. Therefore, the composed model state transition function must be written in terms of the instance state, which is determined by projecting the composed model state onto the state variable set of the instance. As defined in [25], if the next event to occur is in instance A, which has local state transition function τ_A, the composed model state transition function is

$$\tau(e, \mu) = (\mu - \mu_A) \cup \tau_A(e, \mu_A). \tag{1}$$

Equation 1 states that the new composed model state is constructed by first taking the set difference between the global state mapping μ and the local state μ_A, then applying the local state transition function τ_A to the local state, and finally reconstituting the composed model state by taking the union of the new local state and the unchanged portion of the composed model state.

Procedures for generating the state space of a composed model are given in [25].

B Detecting Symmetry

In the composed model formalism of Subsection A, the structure of the model is exposed in the model composition graph. In [25], we developed a method for detecting and exploiting symmetry using the model composition graph. In this new approach, we use the automorphism group of the model composition graph to detect structural symmetry. The main result of that work is a proof that we can construct a Markov process with states that are the partition of the state space induced by the automorphism group of the model composition graph.

A graph automorphism of the model composition graph is a permutation of the names of the instances that does not change the structure of the graph. Such permutations often exist when there are multiple instances of a given model in the composed model. The automorphism group of a model composition graph is the group of all permutations of instance names that leave the graph structure invariant. In [25], we showed that under the rules for constructing model composition graphs, structural symmetry induces behavioral symmetry, so the automorphism group of the model

composition graph can be used as the basis of a procedure for constructing reduced state spaces. A procedure is developed and several examples of state-space reduction are given in [25].

The construction procedure for the reduced state space given in [25] was developed under the assumption that the model structure was required and was designed by the modeler to support whatever performance measures he had in mind. This is the typical state of affairs in modeling. The problem, as pointed out in our work on path-based reward variables [26], is that in many cases it is inconvenient to specify all the structure required for the various interesting performance measures in the system model itself. Thus, the work in [26] focused on performance measures that required some history of the sequence of model states and events. By introducing and utilizing path automata to track the required sequences, and defining reward structures based on the state of the automata as well as the model, we were able to design a procedure for automatically constructing state spaces that support path-based reward variables.

In the next section, we define a reward structure that is compatible with symmetry detection and can be extended to path-based reward variables.

III Symmetric Reward Structures

In this section, we define a symmetric reward structure and give sufficient conditions for constructing reduced state-space Markov processes that support the specified reward structure.

Definition 3 *Given a composed model (Σ, I, κ, C) with state mapping set M and event set E, a* symmetric reward structure $(\mathcal{C}, \mathcal{R}, \Gamma_R)$ *is a pair of functions*

- $\mathcal{C} : E \to \mathbb{R}$*, the impulse reward function;*
- $\mathcal{R} : M \to \mathbb{R}$*, the rate reward function;*

and a group Γ_R defined on the composed model such that for all $\gamma \in \Gamma_R$, $\mathcal{C}(e^\gamma) = \mathcal{C}(e)$ for all $e \in E$, and $\mathcal{R}(\mu^\gamma) = \mathcal{R}(\mu)$ for all $\mu \in M$.

This definition states that for a reward structure to be symmetric, there must exist a group, Γ_R, such that the impulse and rate reward functions are invariant under the permutations of the composed model by the elements of Γ_R.

From Definition 3, it is fairly straightforward to derive a condition sufficient for correct reduced state-space construction.

Proposition 1 *Let $(\mathcal{C}, \mathcal{R}, \Gamma_R)$ be a symmetric reward structure defined on a composed model with automorphism group Γ_S. Then $\Gamma_S \cap \Gamma_R \neq 1$, where 1 indicates the trivial group, is a sufficient condition for constructing a reduced state-space Markov process that supports the reward structure.*

Proof $\Gamma_S \cap \Gamma_R$ is a subgroup of Γ_S, which implies that every element is an automorphism of the composed model. Furthermore, the restrictions on Γ_R in Definition 3 ensure that R is invariant under all automorphisms in the subgroup, so the reward structure is supported. Δ

Proposition 1 shows that we can use the procedure developed for exploiting structural symmetry in the model composition graph [25] to handle reward structure symmetry. The proof follows from the fact that the automorphism group can only be restricted by the symmetry group of a symmetric reward structure. This means that the symmetry group ultimately used to reduce the state space is a subgroup (due to the definition of a group) of the automorphism group of the model composition graph. Therefore, the action of each permutation in the symmetry group on a given composed model state produces another composed model state that is part of the same equivalence class.

Definition 3 concisely describes a symmetric reward structure, but from a practical point of view there remain many issues that must be resolved. The main question is how to specify reward structures so that it is easy to verify the condition on Γ_R. The next section investigates the problems of specifying symmetric reward structures for composed models and deriving Γ_R.

IV Reward Variable Specification

In this section we introduce a reward variable formalism that will allow us to detect and exploit symmetry in the variable definition. Symmetry in the variable definition ultimately is combined with structural symmetry in the model to derive the symmetry group used to reduce the state space.

The basis for this new approach to reward variable specification is the observation that many performance measures may be written as functions that are invariant under permutations of some or all of their arguments.

Definition 4 *A function $f(a_1, a_2, \ldots, a_n)$ has* permutable arguments *if it is invariant under the action of a group, Γ_f, on its argument list. The pair (f, Γ_f) is called a* permutable argument function (PAF). *The function f is said to be* permutable with respect to Γ_f, *which in turn will be called the* argument permutation group of f.

For example, consider a function of three arguments, $f(a_1, a_2, a_3)$, for which we will assume, for now, that the arguments are all elements of the same set, such as $\mathbb{R}$. Many such functions are

invariant under argument permutations. Suppose f computes the average of its three arguments, for example. Then no matter how the arguments are permuted, the value of f remains the same. In this case the argument permutation group is the symmetric group $< (a_1, a_2, a_3), (a_1, a_2) >$, where the angle bracket notation $< p_1, p_2 >$ denotes the group generated from the permutations p_1 and p_2 [27].

We will use PAFs to define reward structures on models. The arguments of the function will be state mappings of model instances and information on the most recent event. In general, the arguments of a reward structure function will not be homogeneous. However, the case of a heterogeneous argument set is easily handled by partitioning the set into cells of instances of the same model, and forming the direct product of the permutation groups defined on each cell of the partition.

Before presenting the details of the symmetry theory, we first define the new reward variable specification technique. The new technique is designed to allow relatively compact descriptions of reward structures for large composed models and to facilitate exploitation of symmetry in the reward structure. The first step in doing this is to define an "iterated reward structure."

Definition 5 *An* iterated reward function *is written* **Apply**(f, L) *where* (f, Γ_f) *is a permutable argument function and* L *is a list of argument lists for* f*. The result of* **Apply**(f, L) *is a list,* R*, with the same number of elements as* L*. Each element* r_i *of* R *is* $f(l_i)$*, the result of* f *applied to the corresponding element of* L*.*

The iterated reward function provides a compact description of a reward function defined on a composed model. The list component, L, is a list of lists of instance state mappings. The motivation for defining reward functions this way is that composed models will often have repeated subgraphs. Rather than specify a reward function for each copy of a subgraph, the iterated reward function allows us to specify the reward function once and then specify the subgraphs to which it should be applied. Each list in L is a list of instances that comprise one subgraph. Fortunately, the list L often can be described compactly in terms of the structure of the composed model. To do this, we use the notion of the orbit of a set of instances within the automorphism group of the model composition graph.

In some cases, the performance measure implies dependencies that are not reflected in the structure of the composed model. An iterated reward function can be used to express these dependencies and, if necessary, restrict the symmetry group that is used to generate the reduced state space. In these cases, the list L is fully described, and is analyzed to generate the proper symmetry group.

(f, Γ_f), (g, Γ_g) : permutable argument functions
L : list of argument lists that are compatible with f
n : number of elements in L
Γ : symmetry group

1. Construct a generating set S such that $< S >= \prod_{i=1}^{n} \Gamma_f$
2. For $\gamma \in \Gamma_g$
3. $\quad S = S \cup L^{\gamma}$
4. $\Gamma =< S >$

Figure 2: Procedure for constructing the symmetry group of a compound reward function

The next step is to consider functions of iterated reward functions. The motivation for this additional level of complexity is that the performance measure of interest will likely be a function of the iterated reward function. For example, the performance measure might be the sum of the rewards generated from each subgraph. Other possibilities include the minimum or maximum, the number above or below a given threshold, and so forth. We will call a function of an iterated reward function a "compound reward function," since it is a combination of the reward functions produced by the iterated reward function.

If (g, Γ_g) is a PAF of an iterated reward function **Apply**(f, L), then Γ_g will act on the elements of L as blocks. For example, suppose f is a function of m arguments, and L has n m-element argument lists. Writing out g yields

$$g(\textbf{Apply}(f, L)) \quad = \quad g(f(l_{1,1}, l_{1,2}, \ldots, l_{1,m}), f(l_{2,1}, l_{2,2}, \ldots, l_{2,m}), \ldots, f(l_{n,1}, l_{n,2}, \ldots, l_{n,m})).$$

If f is a PAF with argument permutation group Γ_f, then we construct the symmetry group Γ of $g(\textbf{Apply}(f, L))$ as follows. First we construct a set of generators for the direct product of n instances of Γ_f. Then, we add generators for the block moves of the $l_{i,j}$ according to the action of Γ_g on the arguments of g. For example, if there is a permutation in Γ_g that transposes the i-th and j-th arguments of g, then we add $(l_{i,1}, l_{j,1})(l_{i,2}, l_{j,2}) \cdots (l_{i,m}, l_{j,m})$ to the generating set for Γ.

Figure 2 shows the procedure for constructing the symmetry group of a compound reward function. The group $< S >$ formed from the direct product of n copies of Γ_f is a group defined on the set $l_1 \cup l_2 \cup \cdots \cup l_n$. The subsets l_i are systems of imprimitivity in $< S >$, since the elements of these subsets are only permuted among themselves. In Line 3, the notation L^{γ} denotes the permutation on $l_1 \cup l_2 \cup \cdots \cup l_n$ that corresponds to permuting the subsets l_i as blocks, such that if l_i is mapped to l_j by the permutation γ, then $l_{i,k} \rightarrow l_{j,k}$ for all k.

To construct a symmetric reward structure, we use compound reward functions to define the impulse and rate reward functions. The symmetry group of the symmetric reward structure is the intersection of the symmetry groups of the two compound reward functions. The next proposition states that a symmetric reward structure can be specified in terms of compound reward functions.

Proposition 2 *The symmetry group constructed from a compound reward function using the procedure in Figure 2 satisfies the requirements placed on the symmetry group of a symmetric reward structure in Definition 3.*

Proof Follows from the definitions of PAFs and the direct product of groups. Δ

V Example Reward Structure Specifications

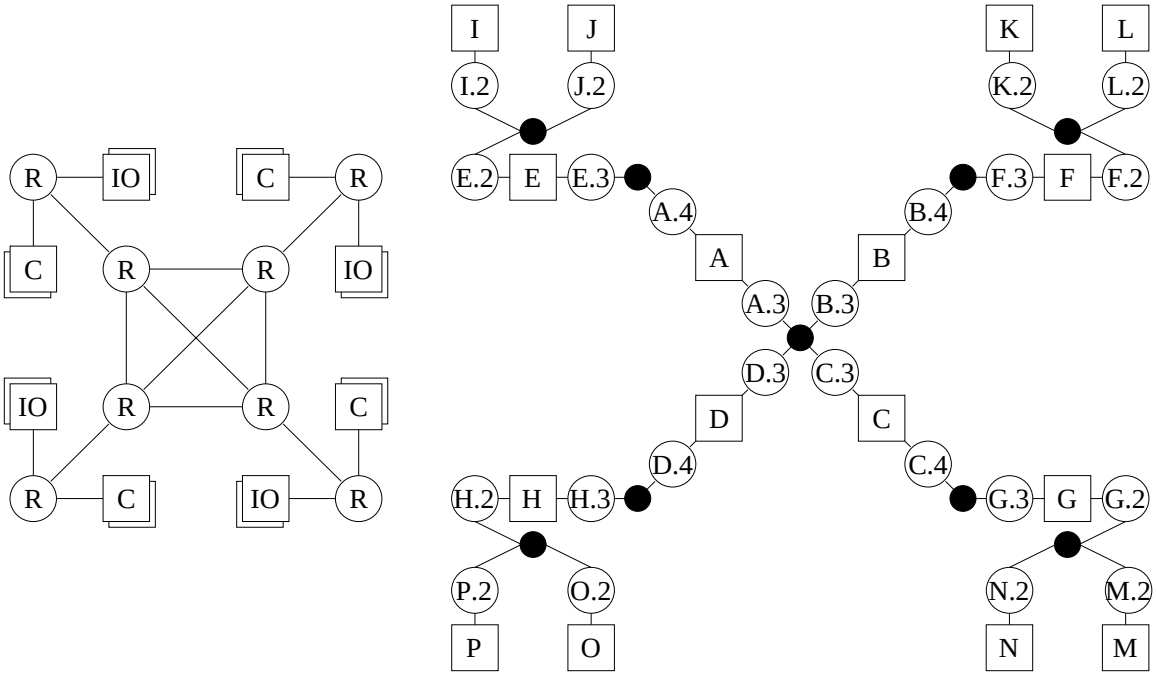
We use the composed model in Figure 3 to demonstrate the specification of compound reward structures. This example was used in [25] to demonstrate state-space reduction. For our purpose here, the details of the model are not important. We only need the structure and information on which instances are from the same model. In Figure 3-a, the boxes marked "C" correspond to instances of a computation node, those marked "R" correspond to routing nodes in the interconnection network, and those marked "IO" correspond to I/O processors.

It is interesting to derive the symmetry groups for the examples, since this exercise demonstrates the power and flexibility of the representation. After each example reward structure specification, we discuss the nature of the associated symmetry group.

Example 1 Suppose we wish to evaluate the number of computers that are functioning correctly. The computers in the system are represented by instances I, K, M, and O. To specify this measure we would write $\mathcal{R} = \sum \textbf{Apply}(f, \textbf{Orbit}(I))$ to get the compound reward function $f(I) + f(K) + f(M) + f(O)$, where

$$f(x) = \begin{cases} 1 & \text{instance } x \text{ is working} \\ 0 & \text{otherwise.} \end{cases}$$

In Example 1, the PAF in the iterated reward function has only one argument, so it does not restrict the symmetry in any way. Furthermore, the compound reward function is constructed using addition, which is commutative, so the symmetry group is $< (I, K, M, O), (I, K) >$, which consists of all 24 permutations of the four instances. This compound reward function does not depend on any other instances in the composed model, so to form the full symmetry group of this



(a) (b)

Figure 3: (a) Network with fully connected core and (b) Model composition graph

reward function relative to the composed model, we take the direct product of this group with the symmetric group of the other instances in the model. The final symmetry group used to reduce the state space is the intersection of the full symmetry group of the compound reward function with the automorphism group of the model composition graph. In this example, the final symmetry group is simply the structural symmetry group, since the compound reward function does not place any new restrictions on that group.

Example 2 Now suppose we wanted to count the number of clusters that satisfy some property $p(a, b, c)$, where a is a computer, b is an I/O device, and c is a router on the network. (Note that p is a PAF but its argument permutation group is trivial.) Assuming p is one when the property holds and zero otherwise, we would write $\sum$ **Apply**$(p,$ **Orbit**$(I, J, E))$ to get the compound reward function $p(I, J, E) + p(K, L, F) + p(M, N, G) + p(O, P, H)$. It is easy to see that this compound reward function is invariant under automorphisms of the graph, since the list of argument lists for p was generated using the automorphism group.

In Example 2, the compound reward function is more complicated, since the function operates on a subgraph comprising a router, a computer and an I/O system. These are instances of different

submodels, so it makes sense that f is restricted to the identity permutation. Once again, the outer function in the compound reward function is summation, which is invariant under all permutations. In this case, the symmetry group of the reward function is the symmetric group acting on the blocks $[I, J, E]$, $[K, L, F]$, $[M, N, G]$, and $[O, P, H]$, which is $< (I, K, M, O)(J, L, N, P)(E, F, G, H)$, $(I, K)(J, L)(E, F) >$. This group is compatible with the automorphism group of the model composition graph, so the compound reward function does not restrict the symmetry group.

Example 3 Finally, consider the application of p to some subset of the clusters in the last example. For example, suppose we wish to compute

$$\sum \textbf{Apply}(p, [(I, J, E), (M, N, G)]).$$

By restricting the list to two of the four possible clusters, we have distinguished these clusters and can no longer assume they are permutable with (K, L, F) and (O, P, H). Therefore, the symmetry group used to reduce the state space must be prevented from permuting between $\{(I, J, E), (M, N, G)\}$ and $\{(K, L, F), (O, P, H)\}$, although permutations within each set are still supported.

Example 3 demonstrates how the definition of the compound reward function can impact the symmetry group. In this case, the PAF of the iterated reward function is the same as that in Example 2, but the list of argument lists for the PAF is different. The summation function is totally symmetric, so the symmetry group of the compound reward function is the symmetric group acting on the blocks $[I, J, E]$ and $[M, N, G]$, which is the group $< (I, M)(J, N)(E, G) >$. Forming the direct product of this group and the symmetric group over the rest of the instances, and intersecting the result with the automorphism group of the model composition graph, results in a subgroup of the automorphism group. The subgroup has only four permutations, versus the twenty-four permutations in the full automorphism group.

VI Symmetric Path-Based Reward Structures

In this section we extend symmetric reward structures to paths in composed models. Building on our work in [26], we define path automata for composed models and then introduce "symmetric path-based reward structures." Composed models require an extension of the theory in [26] to multiple interconnected models and automorphisms.

When a path automaton is defined on a composed model that has a nontrivial automorphism group, care must be taken in defining the state transition function. Unless the path automaton state

transition function is written carefully to be invariant under all the automorphisms of the model composition graph, the constructed state space will not, in general, represent a Markov process. For example, suppose that we define $\delta(\phi, e, \mu) = \phi'$. Now suppose that the canonical label for μ is μ^γ. Unless $\delta(\phi, e^\gamma, \mu^\gamma) = \phi'$, it is an error to place (e^γ, μ^γ) in the same equivalence class as (e, μ), even though both of these events are in the same Γ_S equivalence class. To solve this problem, we introduce the "symmetric path automaton," which is compatible with the symmetry theory developed for composed models. The main difference between this new definition and that of the plain path automaton is the definition of the state transition function.

Definition 6 *A* symmetric path automaton *defined on a composed model* (Σ, I, κ, C) *with state mapping set* M *and event set* E *is a five-tuple* $(\Phi, F, X, \delta, \Gamma_P)$*, where*

- Φ *is the set of internal states;*
- F *is the set of final states;*
- $X = E \times M$ *is the set of inputs;*
- $\delta : \Phi \times X \to \Phi \cup F$ *is the state transition function; and*
- Γ_P *is a group defined on the composed model such that for all* $x \in X$ *and all* $\gamma \in \Gamma_P$, $\delta(\phi, x^\gamma) = \delta(\phi, x)$.

Definition 6 defines the state transition function, δ, to be invariant under the action of Γ_P on the model composition graph. Therefore, δ can be defined in terms the Γ_P-induced equivalence classes of X, and the symmetric path automaton can place additional restrictions on the symmetry that may be exploited to reduce the state space.

We will define the state transition function in terms of PAFs, which will in turn define the symmetry group, Γ_P.

Example 4 Consider the network model in Figure 3, and suppose we want to make a path automaton that transitions from ϕ_1 to ϕ_2 at the first instant that two computers are unreachable. To define this correctly, we begin by defining a PAF $(f(c, r_1, r_2), \Gamma_f)$ where

$$f(c, r_1, r_2) = \begin{cases} 1 & \text{instance } c \text{ is down} \\ 1 & \text{instance } r_1 \text{ or } r_2 \text{ is down} \\ 0 & \text{otherwise,} \end{cases}$$

and $\Gamma_f =< (r_1, r_2) >$. It is easy to see that $\sum$ **Apply**$(f,$ **Orbit**$([I, A, E]))$ indicates the number of computers that are unreachable. (Note that the orbit of the instance list $[I, A, E]$ is $[I, A, E]$, $[K, B, F]$, $[M, C, G]$, $[O, D, H]$.) To catch the condition that should trigger the transition from ϕ_1 to ϕ_2, we also need a function that identifies the right failure event. Therefore, we define another PAF $(g(c, r_1, r_2), \Gamma_g)$ where

$$g(c, r_1, r_2) = \begin{cases} 1 & \text{instances } c, r_1, r_2 \text{ are up and} \\ & \text{event } e \text{ corresponds to failure} \\ & \text{of } c, r_1, \text{ or } r_2 \\ 0 & \text{otherwise,} \end{cases}$$

and $\Gamma_g =< (r_1, r_2) >$. As defined, $\sum$ **Apply**$(g,$ **Orbit**$([I, A, E])) > 0$ indicates that a computer is about to become unreachable. Finally, we can state the condition on the transition from ϕ_1 to ϕ_2 as

$$\begin{aligned} \sum \textbf{Apply}(g, \textbf{Orbit}([I, A, E])) > 0 \quad \bigcap \\ \sum \textbf{Apply}(f, \textbf{Orbit}([I, A, E])) \quad = \quad 1. \end{aligned} \tag{2}$$

The symmetry group for Condition 2 can be constructed using the procedure in Figure 2. The symmetry groups of the two components in the intersection are the same, so we only derive the one for the sum over f applied to the orbit of $[I, A, E]$. In the first step, the generating set, S, of the direct product of four copies of Γ_f must be constructed. Since $\Gamma_f =< (r_1, r_2) >$,

$$\begin{aligned} \left(\prod_{\textbf{Orbit}([I,A,E])} \Gamma_f \right) \quad = \quad & < (A, E), (B, F), \\ & (C, G), (D, H) > . \end{aligned}$$

The next step is to add generators for the block permutations of the arguments of f according to the argument permutation group of the outer function, $\sum$. As discussed before, $\sum$ is commutative, so its argument permutation group is the symmetric group over the arguments. The permutations corresponding to these block moves are generated by $(A, B, C, D)(E, F, G, H)(I, K, M, O)$ and $(A, B)(E, F)(I, K)$. Therefore, the group returned by the procedure is

$$\begin{aligned} \Gamma_P \quad = \quad & < (A, B, C, D)(E, F, G, H)(I, K, M, O), \\ & (A, B)(E, F)(I, K), \\ & (A, E), (B, F), (C, G), (D, H) > . \end{aligned}$$

Finally, note that Γ_P does not contain any permutations the refer to the I/O instances $\{J, L, N, P\}$. In order to make Γ_P compatible with Γ_S, so that their intersection has meaning, we need to augment Γ_P by forming the direct product with the symmetric group over $\{J, L, N, P\}$.

$$\Gamma_P = \Gamma_P \times < (J, L, N, P), (J, L) > .$$

Now Γ_P is a group defined over the same set of instances as Γ_S. Taking the intersection $\Gamma = \Gamma_P \cap \Gamma_S$, we obtain the final symmetry group. In this example, since we used the **Orbit** operator to form the sets we end up with $\Gamma = \Gamma_S$. However, it is important to note that $\Gamma = \Gamma_S$ is a fact that was *derived*, using an unambiguous procedure, rather than assumed.

The final step in the extension of path-based reward variables to composed models is the extension of the reward structures associated with each symmetric path automaton state to symmetric reward structures.

Definition 7 *A* symmetric path-based reward structure, *defined on a symmetric path automaton* $(\Phi, F, X, \delta, \Gamma_P)$ *is a pair of functions*

- $\mathcal{C} : \Phi \times X \rightarrow I\!R$;
- $\mathcal{R} : \Phi \times M \rightarrow I\!R$;

and a group Γ_R *such that for all* $\gamma \in \Gamma_R$, $\mathcal{C}(\phi, x^\gamma) = \mathcal{C}(\phi, x)$ *and* $\mathcal{R}(\phi, \mu^\gamma) = \mathcal{R}(\phi, \mu)$.

The specification of a symmetric path-based reward structure follows naturally from the specifications of symmetric reward structures and symmetric path automata. We use PAFs to construct the state transition function for the symmetric path automaton, and we use PAFs to specify the reward structures associated with each state of the automaton. Then we augment each of these groups so that they are defined over the set of all instances and intersect them with the automorphism group of the model composition graph. The resulting subgroup is used in the state generation procedure. Details for these procedures are given in [25].

VII Example State Spaces for Symmetric Reward Variables

In this section, we introduce an example system and three symmetric reward variables that demonstrate the use of this new technique. We show how the size of the state space changes according to the specified reward variables. The toroidal mesh shown in Figure 4 serves as the basis for the examples in this section. We will consider dependability measures for this system, and begin with

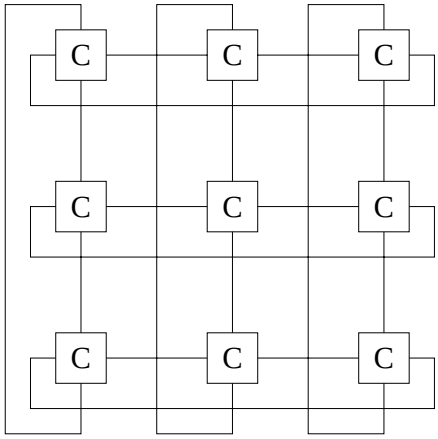


Figure 4: Toroidal mesh system

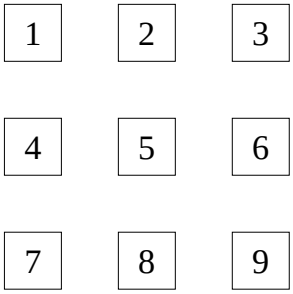


Figure 5: Model composition graph for toroidal mesh with independent failures

a very simple model composition graph, shown in Figure 5. In Figure 5, the CPUs are independent, so the model composition graph has no arcs. The structural symmetry group is the symmetric group defined on the nine instances of the CPU model. The CPU model has only one state variable and two states: working or failed.

The first dependability measure we consider is the number of CPUs that are working at a given instant of time. We construct this measure in three steps. First, we define a function

$$g(x) = \begin{cases} 1 & \text{if } \mu_x = \text{working} \\ 0 & \text{otherwise} \end{cases}$$

on the state of a CPU model. Since we only care about the number of instances that are in the working state, we can use the iterated reward function $\textbf{Apply}(g, \textbf{Orbit}(1))$, which creates a list of g applied to each of the nine instances. The second step is to define the compound reward function $f(a_1, a_2, \ldots, a_9) = \sum_{i=1}^{9} g(a_i)$. Note that f is commutative, meaning that Γ_f is the symmetric group defined on the nine arguments. Now, for the third step, we use (f, Γ_f) as the rate reward component of a symmetric reward structure representing the desired dependability measure:

$$\begin{aligned} \mathcal{C}(x) &= 0 \\ \mathcal{R}(\mu) &= f(\mu_1, \mu_2, \ldots, \mu_9) \end{aligned}$$

Since Γ_f is already defined on all nine instances in the composed model, it need not be augmented, and Γ_R is the symmetric group over the nine instances. Therefore, the intersection $\Gamma_R \bigcap \Gamma_S = \Gamma_S$, so the full symmetric group (362,880 permutations) can be used to reduce the state space. As shown in Table 1, the result is a very large reduction in the number of states that must be generated. The reduced state-space is only 2% of the detailed state space.

For the second example, we consider measuring the number of columns with at least one failed CPU. In this case, although the CPUs remain independent so that the structural group is the same as in the first example, the dependability measure in this example requires that the CPU instances be grouped into columns, as shown in Figure 6. As in the first example, we construct this measure in three steps. First, we define a function

$$f(a, b, c) = \begin{cases} 1 & \text{if } \mu_a = \text{failed or } \mu_b = \text{failed} \\ & \text{or } \mu_c = \text{failed} \\ 0 & \text{otherwise,} \end{cases}$$

which takes the states of three CPU instances as arguments. Then we define the iterated reward

Table 1: State-space size versus reward structure

Reward Structure	States	Relative Size
Number of working CPUs	10	2%
Number of columns with at least one failed CPU	20	4%
Number of rows and number of columns with at least one failed CPU	36	7%
Status of each CPU	512	100%

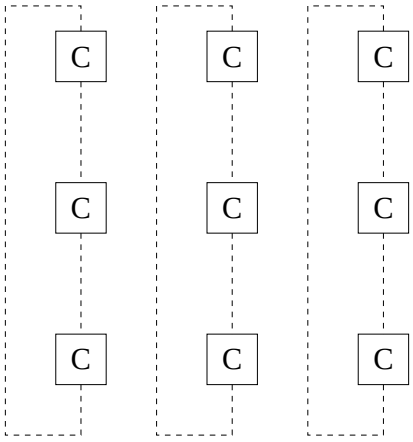


Figure 6: Logical grouping of CPU instances into columns

function

$$\textbf{Apply}(f, [[1,4,7],[2,5,8],[3,6,9]]),$$

which creates a list of f applied to each of the three columns. The second step is to define the compound reward function $g = \sum \textbf{Apply}(f, [[1,4,7],[2,5,8],[3,6,9]])$. Now, for the third step, we use (g, Γ_g) as the rate reward component of a symmetric reward structure representing the desired dependability measure:

$$\begin{aligned} \mathcal{C}(x) &= 0 \\ \mathcal{R}(\mu) &= \sum \textbf{Apply}(f, [[1,4,7],[2,5,8],[3,6,9]]) \end{aligned}$$

To derive the group Γ_R for this reward structure, we note that f and g are both PAFs that are commutative. Therefore,

$$\begin{aligned} \Gamma_{\text{cols}} &= < (1,4,7),(1,4),(2,5,8),(2,5),(3,6,9),(3,6), \\ &\quad (1,2,3)(4,5,6)(7,8,9),(1,2)(4,5)(7,8) > . \end{aligned}$$

Since this dependability measure identifies each instance with a column, more detail is needed in the state space to support the measure. In this case

$$\Gamma_{\text{cols}} \bigcap \Gamma_S = \Gamma_{\text{cols}},$$

which, with only 1296 elements, is much smaller than the symmetric group over the nine instances. However, Table 1 shows that we still obtain a large reduction in the number of states that must be generated, compared to the 2^9 states that are required if all CPUs are distinguished.

For our third example, we further distinguish the CPUs by identifying each CPU with a row in the mesh as well as with a column. The dependability measure we are interested in is the number of rows, as well as columns, in which there is at least one failed CPU. The development of the reward structure is similar to the last measure. We define the iterated reward function as $g = \textbf{Apply}(f, [[1,2,3],[4,5,6],[7,8,9]])$. We use (g, Γ_g) as the rate reward component of a symmetric reward structure representing the dependability measure:

$$\begin{aligned} \mathcal{C}(x) &= 0 \\ \mathcal{R}(\mu) &= \sum \textbf{Apply}(f, [[1,2,3],[4,5,6],[7,8,9]]) \end{aligned}$$

To derive the group Γ_{rows} for this reward structure, we note that g and f are both PAFs that are commutative. Therefore,

$$\begin{aligned}\Gamma_{\text{rows}} &= < (1,2,3),(1,2),(4,5,6),(4,5),(7,8,9),(7,8),\\ &\quad (1,4,7)(2,5,8)(3,6,9),(1,4)(2,5)(3,6) > .\end{aligned}$$

Finally, since we will use this reward structure on the rows, and the similar structure defined for columns, we derive the final reward symmetry group by intersecting the two groups:

$$\begin{aligned}\Gamma_{\text{rows}} \bigcap \Gamma_{\text{cols}} &= < (1,2,3)(4,5,6)(7,8,9),\\ &\quad (1,2)(4,5)(7,8),\\ &\quad (1,4,7)(2,5,8)(3,6,9),\\ &\quad (1,4)(2,5)(3,6) > .\end{aligned}$$

The resulting group has only 36 permutations, but as shown in Table 1, the reduced state-space size is still only 7% of the detailed state-space size.

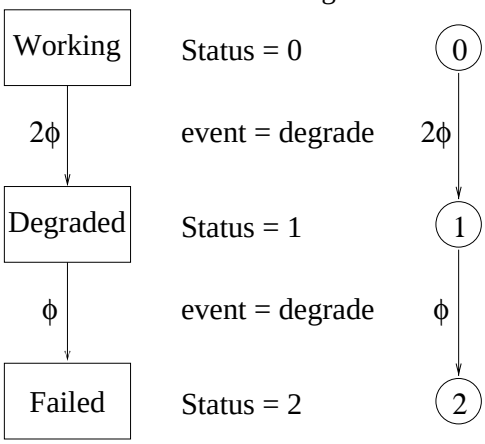
VIII Example State Space for Symmetric Path-Based Reward Variable

In this section we describe an example system and a symmetric path-based reward variable, and demonstrate the construction of the state-space that supports the variable. To make the presentation of the example clear, complete, and understandable, we use a small example.

Consider a cluster of two servers where each server may be in one of three states. The first state is perfect working order, the second state is partially degraded, and the last state is failed. Now suppose that this cluster is supported by an aggressive maintenance policy designed to assure a high level of availability, and consider the system available as long as at least one server is operating (even in degraded mode). Each server has its own repair facility, but repair is not completely independent. Each repair facility is aware of the state of the other server. If the system degrades to a dangerous state, where a single additional fault will bring the system to a halt, repair activity is accelerated on the server that is down. If the system does fail, repair activity is accelerated on both servers. The model for an individual server is given in Figure 7. This model handles the failure transitions of an individual server. Figure 8 shows the model of a repair facility's behavior. The repair facility maintains a single server, but its rate of work is sensitive to the state of a second server.

The model composition graph for the clustered servers is shown in Figure 9. Server 1 and

State transition diagram



Transition function and rate

State	Event	Next State	Rate
0	e_1	1	2ϕ
1	e_1	2	ϕ

State variables

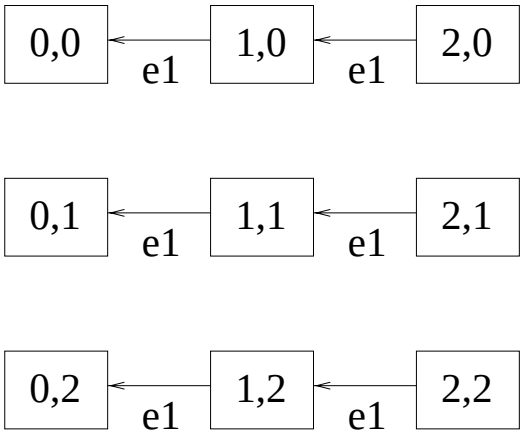
Identifier	Description
s_1	server status

Event set

Identifier	Description	Enabling Condition
e_1	degrade	$s_1 < 2$

Figure 7: Server model

State transition diagram



Transition function and rate

State	Event	Next State	Rate
1,0	e_1	0,0	ρ
2,0	e_1	1,0	ρ
1,1	e_1	0,1	ρ
2,1	e_1	1,1	2ρ
1,2	e_1	0,2	ρ
2,2	e_1	1,2	2ρ

State variables

Identifier	Description
s_1	server status
s_2	status of other server

Event set

Identifier	Description	Enabling Condition
e_1	repair	$s_1 > 0$

Figure 8: Repair model

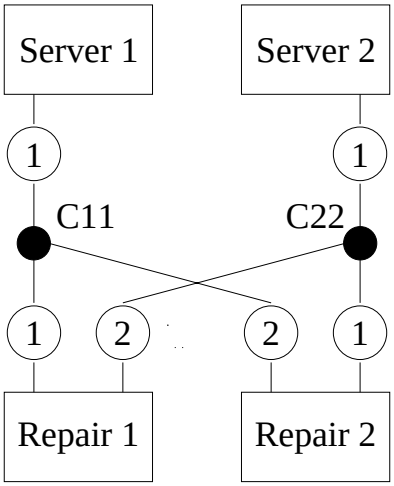


Figure 9: Model composition graph for clustered server

Server 2 are instances of the server submodel, and Repair 1 and Repair 2 are instances of the repair submodel. The repair submodel has two state variables that are external. The server submodel has only one state variable (the status of the server) and it is external. Figure 10-a shows the detailed state-space for the model, which contains 9 states. The accelerated repairs are reflected in the transition rates between states 12, 21, 11 and 22.

The automorphism group of the model composition graph in Figure 9 is obviously order two, the one permutation executing the flip of the server and repair person states. Thus for symmetric reward variables with symmetry groups that contain this permutation, the state-space can be reduced to 6 states. As shown in Figure 10-b, the compact state-space has 6 states, versus 9 in the detailed state-space.

As an example of a symmetric path-based reward variable, consider the expected number of times within some interval of time that starting with both servers in perfect working order, one server fails before the other degrades at all. Figure 11-a shows the path automaton for this path. The automaton has three states. It sits in the first state, A, until one of the servers degrades, at which point the automaton transitions to state B. If the next event is the failure of the degraded server, the automaton transitions to state C, and an impulse reward of 1 is earned. The next event returns the automaton to state A, where it stays until the system is brought back to normal working order.

The example variable is easily expressed in the formalism we have developed for symmetric path-based reward variables. The state transition function of the path automaton is expressed in

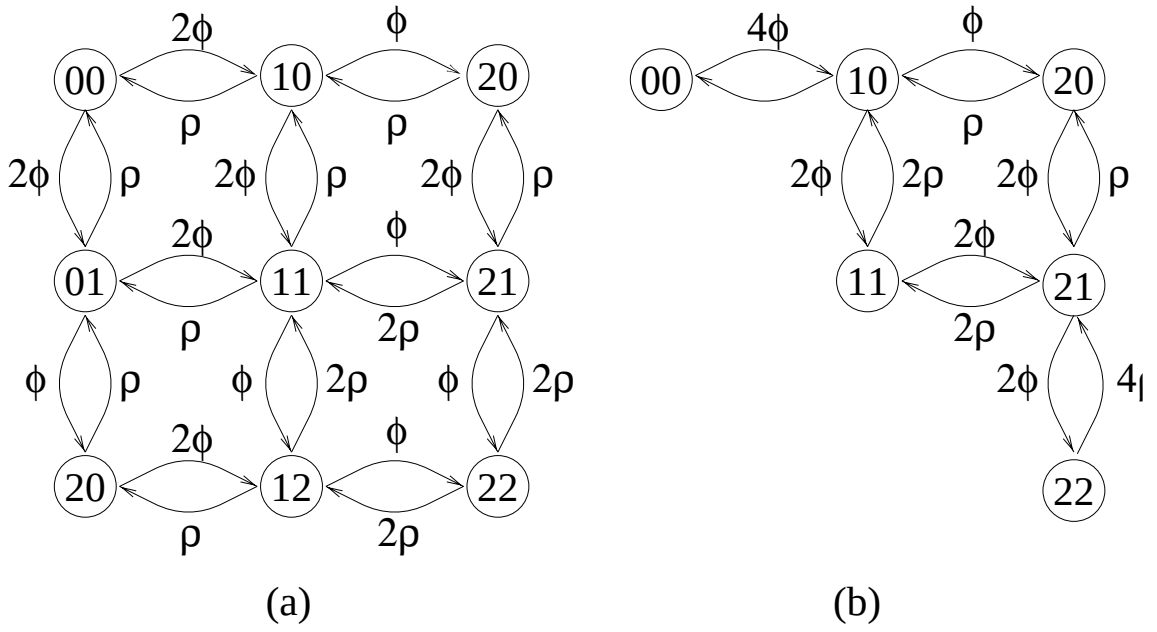


Figure 10: Detailed and reduced state-spaces for the example model

terms of PAFs. The first PAF captures the event that one of the servers degrades.

$$p_{AB}(a,b) = \begin{cases} 1 & \text{if } \mu_a = 0 \text{ and } \mu_b = 0 \text{ and} \\ & e = \text{degradation of server 1 or server 2} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Since the AND function is commutative, the arguments of $p_{AB}(a,b)$ may be interchanged without changing the result. When applied to the cluster model, this leads to the symmetry group consisting of the interchange of the two server instances.

In state B, the transition function is based on a similar predicate, which this time matches a server status state variable mapped to 1 (degraded) with an event for the same server that will move that server to a status of 2 (failed).

$$p_{BC}(a,b) = \begin{cases} 1 & \text{if } \mu_a = 1 \text{ and } \mu_b = 0 \text{ and } e = \text{failure of } a \text{ OR} \\ & \mu_a = 0 \text{ and } \mu_b = 1 \text{ and } e = \text{failure of } b \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

As written in Equation 4, p_{BC} is a simple logical OR between the condition tested on each server's status state variable. Since logical OR is commutative, the function has a symmetry group of order 2 that corresponds to the interchange of the two server instances.

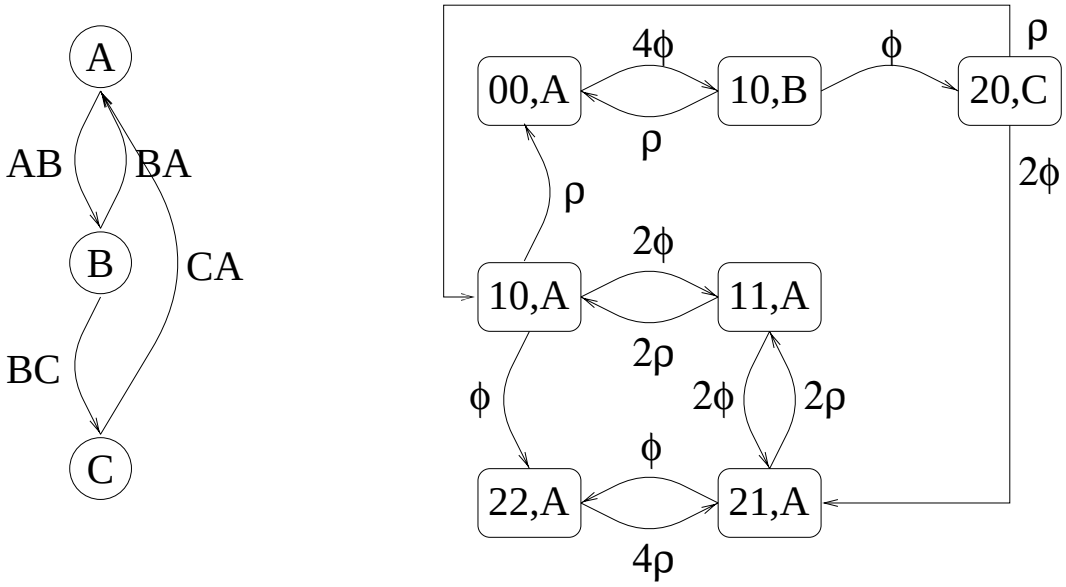


Figure 11: Path automaton and extended state-space

By using the symmetric path-based reward variable, we are able to exploit the symmetry in the model so that the path-based variable does not expand the state-space nearly as much as it would otherwise. Figure 11-b shows the reduced state space that supports the path-based reward variable. The reduced state-space is one state smaller than the original detailed state-space. By exploiting the symmetry in this example, we have removed the cost associated with retaining the history needed to support the path-based reward variable. This example serves to demonstrate the advantages of exploiting symmetry whenever possible.

IX Conclusion

We have presented new techniques for specifying performance, dependability, and performability measures, and automatically constructing state spaces tailored to model and measure symmetry. First, we introduced symmetric reward structures, then, we extended that concept to symmetric path-based reward structures. We demonstrated the application of these new reward structures through several examples.

By developing these new techniques, we have separated the specification of the system model from the specification of the performance, dependability, or performability measure. With these techniques, system modeling is simplified, because interdependencies and symmetry constraints required by the measure are part of the reward variable representing the measure, rather than part

of the system model. This new theory of measure-adaptive state-space construction will allow us to construct tools that are easier to use and that produce smaller state spaces that can be solved in less time than the state spaces produced by the current state-of-the-art tools.

X Appendix

In this appendix we define basic group theory concepts and explain the notation. The interested reader is referred to the books by Hall [27] and Mathewson [28] for a full introduction to the theory and application of groups and group theory.

A *group* is a collection of elements and a product operation that satisfies the following properties:

Closure For every pair of elements in the group, the product exists and is a unique element of the group.

Associative Law If a, b, and c are all elements of the same group, then $(ab)c = a(bc)$.

Identity There is an element I in the group such that $Ia = aI = a$.

Inverse For every element a in the group, there is another element a^{-1} such that $aa^{-1} = a^{-1}a = I$

Note that the product operation on a group is not required to satisfy the commutative property. For our purposes, we are using *permutation groups*, where the elements of the group are permutations of the elements of some set.

A *permuation* is a one-to-one mapping of a set onto itself. Suppose we have a set $\{1, 2, 3\}$. One way of representing the permutation that swaps the positions of 1 and 2 is

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}.$$

This notation is clear, but bulky. For efficient representation of permutations, we use the *cycle notation*. Using cycle notation, the permutation the swaps 1 and 2 is represented by (12). This should be read as a circular list, so we see that (12) maps 1 to 2 and 2 is mapped to 1.

The product operation for permutation groups is composition of the permutation maps. For example,

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}.$$

In cycle notation, $(12)(132) = (23)$. Alternatively, we could use *one-line* notation, which would represent the result as 132, using just the bottom line of the two-line result.

An example of a permutation group is the group containing all the permutations of $\{1, 2, 3\}$. This group is the *symmetric group* of degree three. The *degree* of a permutation group is the number of elements in the given set that are permuted by elements in the group. The elements of this group, expressed in one-line notation, are $123, 132, 213, 231, 312, 321$.

A special form of permutation group used in this article is the *automorphism group of a graph.* This group is defined on the set of vertices, V, of the graph. Each permutation in the automorphism group is required to maintain adjacency among the vertices. That is, if two vertices have an edge between them before the permutation is applied, then they will have an edge between them after the permutation has been applied.

Since groups can have a very large number of elements, *generating sets* are often used to represent the group. A generating set for a permutation group is a set of permutations that when expanded to meet the criteria for a group, generates all elements of the group it is meant to represent. For example, consider the group of all permutations of $\{1, 2, 3, 4\}$. This is called the symmetric group of degree four, and is generated by (1234) and (12). In this way, two elements represent a group containing sixteen permutations. To indicate a generating set, we enclose it in angle brackets. For example, for a group Γ generated by a set of permutations P, we use the notation $\Gamma =< P >$.

The direct product of two groups, A and B, is denoted $A \times B$, and is constructed using the product rule

$$(a_1, b_1)(a_2, b_2) = (a_1 a_2, b_1 b_2).$$

A permutation group is called *imprimitive* if the elements of the set can be divided into disjoint subsets, $S_1, S_2, \ldots, S_n$ such that every permutation in the group maps the elements of S_i onto themselves or onto the elements of another set S_j. If a group is imprimitive, the sets S_i are called *systems of imprimitivity.*

We use two notations that relate the elements of the set E upon which the permutation group is defined. For example, consider the set $\{1, 2, 3, 4\}$ and the permutation group $\Gamma =< (1, 2, 3, 4), (1, 2)$. The permutation group generated by those two permutations is the symmetric group of degree four. This group will map each element in the set $1, 2, 3, 4$ to every other element in the set. Given a specific element e_i, the set of elements that e_i is mapped into by the permutations in the associated group is called the *orbit* of e_i. We use the notation **Orbit**(e_i). For example, the orbit of 1 under the permutations in Γ is $\{1, 2, 3, 4\}$.

The second notation we use allows us to refer to the *action* of a permutation on an element of the underlying set. In this case, the notation e_i^γ indicates the element in the orbit of e_i that γ maps e_i to. In notation: $\gamma : e_i \rightarrow e_i^\gamma$. We use this notation in Definition 3 to refer to the action of the reward variable group on the instance event sets and states.

REFERENCES

[1] B. Plateau, K. Atif, Stochastic automata networks for modeling parallel systems, IEEE Transactions on Software Engineering 17 (10) (1991) 1093–1108.

[2] B. Plateau, J.-M. Fourneau, A methodology for solving Markov models of parallel systems, Journal of Parallel and Distributed Computing 12 (4) (1991) 370–387.

[3] P. Buchholz, P. Kemper, Numerical analysis of stochastic marked graph nets, in: Proceedings of the Sixth International Workshop on Petri Nets and Performance Models, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 32–41, cat. No.95TB100003.

[4] G. Ciardo, M. Tilgner, On the use of Kronecker operators for the solution of generalized stochastic Petri nets, ICASE Report #96-35 (NASA CR-198336), NASA Langley Research Center (May 1996).

[5] G. Ciardo, M. Tilgner, Parametric state space structuring, ICASE Report #97-67 (NASA/CR-97-206267), NASA Langley Research Center (December 1997).

[6] S. Donatelli, Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space, Performance Evaluation 18 (1) (1993) 21–36.

[7] P. Kemper, Numerical analysis of superposed GSPNs, IEEE Transactions on Software Engineering 22 (9) (1996) 615–628.

[8] P. Buchholz, G. Ciardo, S. Donatelli, P. Kemper, Complexity of Kronecker operations on sparse matrices with applications to solution of Markov models, ICASE Report #97-66 (NASA/CR-97-206274), NASA Langley Research Center (December 1997).

[9] D. Deavours, W. H. Sanders, An efficient disk-based tool for solving very large Markov models, in: Computer Performance Evaluation: Proceedings of the 9th International Conference on Modelling Techniques and Tools (TOOLS '97), St. Malo, France, 1997, pp. 58–71, lecture Notes in Computer Science, no. 1245.

[10] D. Deavours, W. H. Sanders, "On-the-Fly" solution techniques for stochastic Petri nets and extensions, in: Proceedings of the 7th International Workshop on Petri Nets and Performance Models (PNPM '97), St. Malo, France, 1997, pp. 132–141.

[11] J. A. Carrasco, J. Escriba, A. Calderon, Efficient exploration of availability models guided by failure distances, Performance Evaluation Review 24 (1) (1996) 242–251.

[12] E. de Souza e Silva, P. M. Ochoa, State space exploration in Markov models, Performance Evaluation Review 20 (1) (1992) 152–166.

[13] B. R. Haverkort, In search of probability mass: Probabilistic evaluation of high-level specified Markov models, The Computer Journal 38 (7) (1995) 521–529.

[14] B. E. Aupperle, J. F. Meyer, Fault-tolerant BIBD networks, in: 18th International Symposium on Fault-Tolerant Computing (FTCS-18), Tokyo, Japan, 1988, pp. 306–311.

[15] B. E. Aupperle, J. F. Meyer, State space generation for degradable multiprocessor systems, in: 21st International Symposium on Fault-Tolerant Computing (FTCS-21), Montréal, Canada, 1991, pp. 308–315.

[16] P. Buchholz, Hierarchical Markovian models: Symmetries and reduction, Performance Evaluation 22 (1) (1995) 93–110.

[17] J. A. Carrasco, Automated construction of compound Markov chains from generalized stochastic high-level Petri nets, in: Proceedings of the Third International Workshop on Petri Nets and Performance Models, Kyoto, Japan, 1989, pp. 93–102.

[18] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, Stochastic well-formed colored nets and symmetric modeling applications, IEEE Transactions on Computers 42 (11) (1993) 1343–1360.

[19] J. Hillston, A Compositional Approach to Performance Modelling, Distinguished Dissertations in Computer Science, Cambridge University Press, Cambridge, Great Britain, 1996.

[20] W. H. Sanders, J. F. Meyer, Reduced base model construction methods for stochastic activity networks, IEEE Journal on Selected Areas in Communications, special issue on Computer-Aided Modeling Analysis, and Design of Communication Networks 9 (1) (1991) 25–36.

[21] A. K. Somani, Reliability modeling of structured systems: Exploring symmetry in state-space generation, in: 1997 Pacific Rim International Symposium on Fault-Tolerant Systems, Taipei, Taiwan, 1997.

[22] G. Ciardo, K. S. Trivedi, A decomposition approach for stochastic reward net models, Performance Evaluation 18 (1993) 37–59.

[23] G. Franceschinis, R. R. Muntz, Bounds for quasi-lumpable Markov chains, Performance Evaluation 20 (1–3) (1994) 223–243.

[24] G. Franceschinis, R. Muntz, Computing bounds for the performance indices of quasi-lumpable stochastic well-formed nets, IEEE Transactions on Software Engineering 20 (7) (1994) 516–525.

[25] W. D. Obal II, Measure-adaptive state-space construction methods, Tech. Rep. UILU-ENG-99-2212, University of Illinois at Urbana-Champaign (July 1999).

[26] W. D. Obal II, W. H. Sanders, State-space support for path-based reward variables, Performance Evaluation 35 (1) (1999) 233–251.

[27] M. Hall Jr., The Theory of Groups, 2nd Edition, Chelsea Publishing Company, New York, 1976.

[28] L. C. Mathewson, Elementary Theory of Finite Groups, Houghton Mifflin Company, Boston, 1930.