Symmetry Reductions for a Class of Discrete-Event Systems

Kurt Rohloff and Stéphane Lafortune

***Abstract*— Issues related to the control and verification of isomorphic modular discrete-event systems are investigated. A finite state automaton system model is considered for the system modules where a set of atomic propositions are defined on the states. A type of symmetry is defined for these modular systems and a restriction of the μ-calculus designed for these systems is introduced. A procedure is shown to reduce the cost in computation time and memory for testing if symmetric modular systems satisfy propositions in this μ-calculus. An example of a symmetric modular UAV platoon leader system is then shown and discussed.**

I. Introduction

This paper discusses how symmetry can be exploited to more efficiently perform verification tasks for modular discrete-event systems. It is well known that many control and verification tasks for modular discrete-event systems are computationally very difficult [5, 10, 12], but few methods have been developed to avoid this computational difficulty. This paper investigates an important special case of modular discrete-event systems verification problems where the various system modules are exact copies of one another except for the renaming of events. Examples of such systems include swarms of Unmanned Aerial Vehicles (henceforth called UAV's), computer networks and resource sharing manufacturing systems. Methods are shown below to alleviate the computational difficulty of verifying a large class of system properties for these symmetric systems.

The modular systems discussed in this paper are modeled as sets of finite state automata. These automata interact through the parallel composition operation which coordinates behavior on the occurrence of common events. Such models are standard in the field of discrete-event systems [1]. However, the models herein are generalized to permit more atomic propositions on the module states than state marking. This model extension is inspired by the work in the formal methods community in computer science. See [2] for an introduction to formal methods. This paper generalizes and extends a number of key results shown in [11].

In this paper, verification tasks for the modular discrete-event systems are performed with respect to μ-calculus specifications. The μ-calculus is a very general specification language developed by the formal methods community that is more general than many other common specification logics such as LTL, CTL and CTL*. Researchers have investigated the use of the temporal logics in the field of discrete-event systems; see e.g., [8, 9, 14]. A specialized μ-calculus is presented that is particularly relevant to the modular systems in this paper containing a type of symmetry defined as *permutation symmetry*. Classes of symmetry equivalent states are defined for the symmetric systems and the version of the μ-calculus presented here is formulated such that μ-calculus propositions hold at a state in a symmetry equivalence class if and only if the propositions hold at all other states in the symmetry equivalence class.

A method is given for reducing the inherent complexity of testing if the specialized μ-calculus propositions hold at a system state by constructing quotient structures that are equivalent with respect to μ-calculus propositions for the original symmetric systems. The use of symmetry for testing μ-calculus propositions has been discussed in [3] with a different system model and assumptions on the μ-calculus propositions used. There are most likely no polynomial time algorithms for the μ-calculus model checking problem, so any method for reducing the difficulty of testing μ-calculus propositions is potentially very useful [2].

Excluding [4, 11], there has been little work in the supervisory control literature that exploits system symmetry when analyzing discrete-event systems. Group-theoretic methods are used in [4] to define classes of states and transitions in a system that are equivalent under defined permutation operations. These classes of equivalent states and transitions of a system are used to define a quotient automaton that has a state space smaller than the original system. This quotient automaton is used to perform various controller synthesis operations for the original system. The quotient automaton presented herein for verifying properties on permutation symmetric systems is also discussed in [11], but for a more restricted class of systems.

Unfortunately, finding the optimal partition of state equivalence classes for the construction of quotient automata as in [4] is computationally rather difficult and at least as difficult as the graph isomorphism problem [7]. This has induced several authors to attempt to design distributed systems with special architectures so that special forms of symmetry are guaranteed to occur *a priori*, therefore avoiding intensive symmetry verification procedures [4, 11]. This paper expands on this work by discussing classes of systems that are assumed to contain permutation symmetry.

The next section presents the model and notational definitions used in our discussions. The definition of permutation symmetry is presented in Section 3. Section 4 introduces

This research was supported in part by NSF grants CCR-0082784 and CCR-0325571. K. Rohloff was also supported by the NSF grant CCR 00-85917 ITR.

K. Rohloff is with the Coordinated Science Laboratory, The University of Illinois, 1308 West Main St., Urbana, IL 61801, USA krohloff@control.csl.uiuc.edu

S. Lafortune is with the Department of Electrical Engineering and Computer Science, The University of Michigan, 1301 Beal Ave., Ann Arbor, MI 48109-2122, USA stephane@eecs.umich.edu, www.eecs.umich.edu/umdes

our restriction of the μ-calculus and in Section 5 a quotient automaton for symmetric systems is presented. In Section 6 it is shown how the quotient automaton can be used to reduce the computational difficulty of testing μ-calculus propositions for systems with permutation symmetry. An example of a permutation symmetric system is given in Section 7 based on swarming UAV's. The paper concludes with a brief discussion of results. Due to the necessary brevity of this work the proofs of lemmas and theorems in this paper can be seen in [13].

II. Modeling and Notational Definitions

The modular systems are composed of sets of isomorphic interacting automata $\{G_1, \ldots, G_n\}$. Each automaton $G_i = (X, x_0, AP, L, \Sigma_i, \delta_i)$ models the behavior of a single module in the system. The set X is a set of system states and x_0 is the initial state. AP is a set of atomic propositions and $L : X \to 2^{AP}$ maps a state to the set of propositions that hold at that state. Σ_i is the set of events relevant to the state evolution of module G_i and $\delta_i : X \times \Sigma_i \to X$ is the state transition function.

As stated, the system modules, $\{G_1, \ldots, G_n\}$, are isomorphic to one another, but state transition labels are modified. The module G_j is a copy of G_i except that local transition labels Σ_i are replaced with the respective events from Σ_j according to a predefined $\Psi_{ij} : \Sigma_i \to \Sigma_j$ translation mapping. The function $\Psi_{ij}(\cdot)$ translates events relevant to module i to events relevant to module j. To formalize, for $x \in X, \gamma \in \Sigma_i$, $\Psi_{ij}(\cdot)$ is defined such that $\delta_i(x, \gamma) = \delta_j(x, \Psi_{ij}(\gamma))$. The function $\Psi_{ij}(\cdot)$ is also extended in the usual manner to be defined over strings and languages.

The inverse function $\Psi_{ij}^{-1} : \Sigma_j \to \Sigma_i$ is defined such that $\Psi_{ij}^{-1}(\cdot) = \Psi_{ji}(\cdot)$. Note that it is possible that $\sigma \in \Sigma_i \cap \Sigma_j$ but $\Psi_{ij}(\sigma) \neq \sigma$. For an event $\sigma_i \in \Sigma_i$, the notation σ_j is used to represent $\Psi_{ij}(\sigma_i)$ when it can be done without ambiguity.

Note that the set of system states, X, is not indexed nor is the set of atomic propositions P or the state labeling function $L(\cdot)$. Therefore, when two modules are at the same system state, the same atomic propositions hold at both states. The replication of state labels is used later in this paper. If the system state labels are restricted to a binary state marking (i.e., to $AP = \{m\}$), then this model is equivalent to the commonly used model in supervisory control theory [1].

An extended parallel composition operation, denoted by $\|$, is used to model the interaction between modules. For a set of isomorphic modules $\{G_1, \ldots, G_n\}$, the interaction of these modules is the system $G^{\|} = G_1 \| \cdots \| G_n = (X^{\|}, x_0^{\|}, AP, L^{\|}, \Sigma, \delta^{\|})$. The $X^{\|}$, $x_0^{\|}$ and $\delta^{\|}$ components are defined in the usual manner for the parallel composition operation. Let $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_n$. The states of the composed system $G_1 \| \cdots \| G_n$ (i.e., $x^{\|} \in X^{\|}$) are called *composed states* and are n-tuples of the module states. The individual states of a module (i.e., $x \in X$) are called *module states*. For a composed state n-tuple $x^{\|}$, the ith module state is represented as $x^{\|i}$. A transposition operator $\phi_{ij} : X^{\|} \to X^{\|}$ is defined where $\phi_{ij}(x^{\|})$ is $x^{\|}$ with the ith and jth module states swapped. The composed state labeling function $L^{\|} : X^{\|} \to 2^{AP}$ is not predefined except to require that for $x_a^{\|}, x_b^{\|} \in X^{\|}$ and $i, j \in \{1, \ldots, n\}$, if $\phi_{ij}(x_a^{\|}) = x_b^{\|}$ then $L^{\|}(x_a^{\|}) = L^{\|}(x_b^{\|})$.

A state permutation operator is constructed as follows from a set of transposition operators for a given input $x^{\|} \in X^{\|}$:

$$\Phi_{[(i_1 j_1)(i_2 j_2)\cdots(i_m j_m)]}(x^{\|}) = \phi_{i_1 j_1}\left(\phi_{i_2 j_2}\left(\cdots \phi_{i_m j_m}\left(x^{\|}\right)\right)\right)$$

Given a state $x^{\|}$, the set of all possible permutation operators $\Phi_{[\cdots]}(\cdot)$ can be used to define the set of composed states that are permutations of the components in the n-tuple $x^{\|}$. These states are called the permutation equivalent states of $x^{\|}$. Given a state space $X = \{x_1, \ldots, x_k\}$ for an isomorphic module system $G_1, \ldots, G_n$, an arbitrary ordering can be placed on the states in X such that $x_1 < x_2 < \cdots < x_k$. Therefore, for any set of permutation equivalent states from $X^{\|}$ there is always one state such that the module states of the composed states have the correct relative order with respect to the ordering $x^{\|1} \leq x^{\|2} \leq \cdots \leq x^{\|n}$. This state is called the *standard permutation* of all states in its equivalence class. A function $SP : X^{\|} \to X^{\|}$ is defined such that when given an n-tuple composed state $z^{\|}$, $SP(z^{\|}) = y^{\|}$ is another composed state with the module states in the correct order, i.e. $y^{\|1} \leq y^{\|2} \leq \cdots \leq y^{\|n}$.

Let $\vec{X} = \{x^{\|} \in X^{\|} | x^{\|} = SP(x^{\|})\}$ be the set of standard permutations. The inverse function $SP^{-1} : \vec{X} \to 2^{X^{\|}}$ returns the set of states that has the input as its standard permutation. Note that the initial state $x_0^{\|}$ is its own standard permutation because $x_0^{\|} = (x_0, \ldots, x_0)$. Also note that using the notation just defined, the above mentioned requirement on the composed state labeling function that for $x_a^{\|}, x_b^{\|} \in X^{\|}$, $i, j \in \{1, \ldots, n\}$ if $\phi_{ij}(x_a^{\|}) = x_b^{\|}$, then $L^{\|}(x_a^{\|}) = L^{\|}(x_b^{\|})$ can be rephrased as follows for all $\vec{x} \in \vec{X}$ and $x^{\|} \in X^{\|}$

$$\left(x^{\|} \in SP^{-1}(\vec{x})\right) \Rightarrow \left(L^{\|}(x^{\|}) = L^{\|}(\vec{x})\right).$$

For all $x^{\|} \in X^{\|}$ there is a (non-unique) string of index pairs $\left[(i_1^{x^{\|}} j_1^{x^{\|}})(i_2^{x^{\|}} j_2^{x^{\|}}) \cdots (i_m^{x^{\|}} j_m^{x^{\|}})\right]$ such that $\Phi_{\left[(i_1^{x^{\|}} j_1^{x^{\|}})(i_2^{x^{\|}} j_2^{x^{\|}})\cdots(i_m^{x^{\|}} j_m^{x^{\|}})\right]}(x^{\|}) = SP(x^{\|})$. With this string of index pairs, a permutation operator $\Phi_{x^{\|}}(\cdot) : X^{\|} \to X^{\|}$ is defined such that $\Phi_{x^{\|}}(\cdot) = \Phi_{\left[(i_1^{x^{\|}} j_1^{x^{\|}})(i_2^{x^{\|}} j_2^{x^{\|}})\cdots(i_m^{x^{\|}} j_m^{x^{\|}})\right]}(\cdot)$. Therefore, $\Phi_{x^{\|}}(x^{\|}) = SP(x^{\|})$.

III. Modular Symmetry

Some properties of a special class of isomorphic module systems are now discussed. Consider the following simple example.

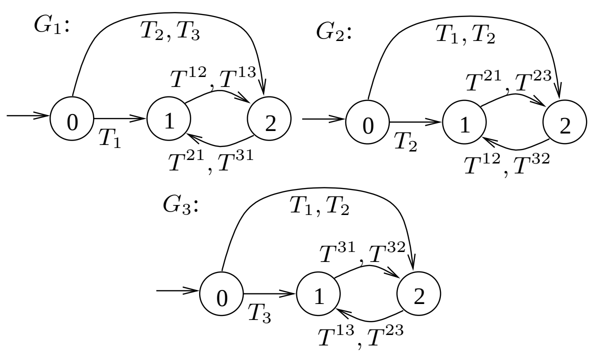

Fig. 1. Modules G_1, G_2, G_3 for Example 1.

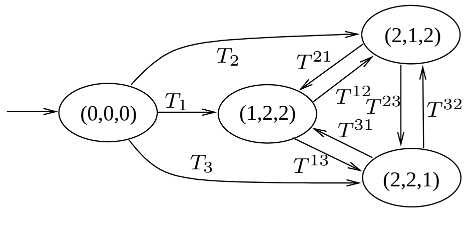

Fig. 2. The composed $G_1\|G_2\|G_3$ for Example 1.

Example 1: Consider the set of isomorphic token passing modules $\{G_1, G_2, G_3\}$ shown in Figure 1.

This example can be thought of as a controller for resource sharing where module i would have exclusive access to a resource if and only if it possesses the token. At initialization, all of the modules are at state 0 and no modules possess a token. On the occurrence of event T_i, a token is given to module i. Module i is then in state 1 can be thought of as possessing the token, and the other modules are in state 2 and do not possess a token. The token is passed from module i to module j on the occurrence of event T^{ij} so that module i enters state 2 and j enters state 1. The composition $G_1\|G_2\|G_3$ can be seen in Figure 2.

For $G_1\|G_2\|G_3$ there are two classes of states that could be considered equivalent with respect to permutations of the component states. The two sets of equivalent state classes are $\{(0,0,0)\}$ and $\{(1,2,2),(2,1,2),(2,2,1)\}$. A valid state labeling for the $G_1\|G_2\|G_3$ automaton might be that states $\{(1,2,2),(2,1,2),(2,2,1)\}$ have proposition labeling T to signify there is exactly one token in the system and the state $(0,0,0)$ has labeling F to signify that the modules possess no tokens.

Note that there is a transition $\delta^{\|}(x_a,\sigma) = x_b$ if and only if for all state permutation operators $\Phi(\cdot)$, there exists an event σ' such that $\delta^{\|}(\Phi(x_a),\sigma') = \Phi(x_b)$. More specifically, consider the state transitions $\delta^{\|}((0,0,0),T_1) = (1,2,2)$ and $\delta^{\|}((0,0,0),T_2) = (2,1,2)$. The pairs of the starting and ending states are permutation equivalent. Also, the events that drive these transitions are the same event with different indices. Now consider the state transitions $\delta^{\|}((1,2,2),T^{12}) = (2,1,2)$ and $\delta^{\|}((2,2,1),T^{31}) = (1,2,2)$. Note that the initial and final composed states in both of these transitions are also permutations of each other and the events that drive these transitions are the same event with different indices.

The intuition gained in Example 1 is that for some special isomorphic module systems, there is a transition $\delta^{\|}(x_a,\sigma) = x_b$, if and only if for all state permutation operators $\Phi(\cdot)$, there exists an event translation operator $\Pi_\Phi(\cdot)$ such that $\delta^{\|}(\Phi(x_a),\Pi_\Phi(\sigma)) = \Phi(x_b)$. That is, for some systems, for any event that drives a transition between two states in that system, then for any permutation of those states, another event can be computed such that there is a transition between the permuted states driven by the computed event. This intuition can be used to define a property below called *permutation symmetry* that forces a large class of global properties to hold at states independent of the composed state orderings.

Suppose a set of permutation operators $\{\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)\}$ are given for $i_1, j_1, \ldots, i_m, j_m \in \{1,\ldots,n\}$. Suppose there is a class of doubly indexed functions $\{\pi_{ij}(\cdot) : \Sigma \to \Sigma | i,j \in \{1,\ldots,n\}\}$ and define a composition of these functions

$$\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\sigma) = \pi_{i_1j_1}\left(\pi_{i_2j_2}\left(\cdots\pi_{i_mj_m}(\sigma)\right)\right).$$

Definition 1: Suppose a set of functions $\{\pi_{ij}\}$ are given such that for all strings of module index pairs $(i_1j_1)(i_2j_2)\cdots(i_mj_m)$ and $(i'_1j'_1)(i'_2j'_2)\cdots(i'_{m'}j'_{m'})$ such that

$$\left(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot) = \Phi_{[(i'_1j'_1)(i'_2j'_2)\cdots(i'_{m'}j'_{m'})]}(\cdot)\right)$$

then

$$\left(\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot) = \Pi_{[(i'_1j'_1)(i'_2j'_2)\cdots(i'_{m'}j'_{m'})]}(\cdot)\right).$$

A system composed of $\{G_1,\ldots,G_n\}$ is said to have modular state permutation symmetry with respect to $\{\pi_{ij}\}$ *or* permutation symmetry *for short if* $\forall i,j \in \{1,\ldots,n\}$ *and* $x_a^{\|}, x_b^{\|} \in X^{\|}$,

$$\left(\delta^{\|}(x_a^{\|},\sigma) = x_b^{\|}\right) \iff \left(\delta^{\|}(\phi_{ij}(x_a^{\|}),\pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})\right).$$

A fundamental result of group theory is that any permutation operator can be constructed from the composition of transposition operators [6]. Therefore, for any state permutation operator $\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$, corresponding event permutation operator $\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$, and permutation symmetric system $G^{\|}$, then $\forall x_a^{\|}, x_b^{\|} \in X^{\|}, \sigma \in \Sigma^{\|}$,

$$\left(\delta^{\|}(x_a^{\|},\sigma) = x_b^{\|}\right) \iff \left(\begin{array}{c}\delta^{\|}\left(\Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_a^{\|}),\Pi_{[(i_1j_1)\cdots(i_mj_m)]}(\sigma)\right) = \\ \Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_b^{\|})\end{array}\right).$$

The intuition behind the definition of permutation symmetry for $G^{\|}$ is that for any state transition in $G^{\|}$, if the composed state ordering in the transition is permuted, then there is a corresponding event such that there is a transition between the permuted states. More formally, if there exists a transition $\delta^{\|}(x_a^{\|}, \sigma) = x_b^{\|}$, then for any state transposition operator $\phi_{ij}(\cdot)$, there is a language translation operator $\pi_{ij}(\cdot)$ such that $\delta^{\|}(\phi_{ij}(x_a^{\|}), \pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})$.

Note that with the restriction in Definition 1 on $\{\pi_{ij}\}$, $\Pi^{-1}_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot) = \Pi_{[(i_mj_m)\cdots(i_sj_2)(i_1j_1)]}(\cdot)$. The $[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]$ subscripts are sometimes dropped on $\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$ and $\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$ when it can be done without ambiguity. If $\Phi_{x^{\|}}(\cdot) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$ then define $\Pi_{x^{\|}}(\cdot) = \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$.

Using the notation of [4], a language $\mathcal{L}(G^{\|})$ has a *symmetric group* of operators $S_\Sigma = \{\pi : \Sigma \to \Sigma\}$ if $\forall \pi \in S_\sigma$, then $\mathcal{L}(G^{\|}) = \pi(\mathcal{L}(G^{\|}))$. This prompts the following lemma and theorem.

Lemma 1: Suppose $\{G_1, \ldots, G_n\}$ *has permutation symmetry with respect to* $\{\pi_{ij}\}$. *Then*

$$s \in \mathcal{L}(G^{\|}) \Rightarrow \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s) \in \mathcal{L}(G^{\|}).$$

Theorem 1: The set of operators $\{\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}\}$ *constructed from* $\{\pi_{ij}\}$ *forms a symmetric group for the language* $\mathcal{L}(G^{\|})$ *if* $\{G_1, \ldots, G_n\}$ *has permutation symmetry with respect to* $\{\pi_{ij}\}$.

Theorem 1 shows that the language generated by an isomorphic module system with state permutation also contains a type of symmetry such that the generated language is identical no matter how the modules are permuted.

IV. Permutation Symmetric μ-Calculus

In the most commonly accepted version of the μ-calculus as discussed in [2], a transition system $M = (S, T, AP, L)$ is given where S is a set of states, T is a set of transition classes $T \subseteq 2^{S\times S}$, AP is a set of atomic propositions and $L : S \to 2^{AP}$ is a state labeling function. A transition class $T_{\sigma_i} \in T, T_{\sigma_i} \subseteq S \times S$ can be thought of as the set of all transitions of the same "type", similar to how transitions might be labeled by events in discrete-event systems. The μ-calculus system also uses a set of relational variables $VAR : \{Q_1, Q_2, \ldots\}$ where each relational variable $Q_i \in VAR$ can be assigned a subset of S. Alternatively, a relational variable can be thought of as a variable set of states $Q_i \subseteq S$. Following the notation used in [2], $e : VAR \to 2^S$ denotes an *environment* where states are assigned to the relational variables. Let Q be an arbitrary element of $\{Q_1, Q_2, \ldots\}$. The expression $e_{[Q\leftarrow W]}$ is used to denote a new environment that is the same as e except $e_{[Q\leftarrow W]}(Q) = W$. That is, with $e_{[Q\leftarrow W]}$, the states in W are assigned to Q.

The μ-calculus can be used to express a set of formulas and a μ-calculus formula f can be said to hold at some states in S, but not in others. The notation $M, s \models f$ is used if the formula f holds at state s in M. The expression $[\![f]\!]_M e$ also denotes the set of states in M where f holds with environment e. The sets of formulas that can be expressed in the μ-calculus are now recursively defined with some notational definitions.

- If $p \in AP$, then p is a formula. An atomic proposition holds at a state according to the state labeling function $L(\cdot)$.

$$[\![p]\!]_M e = \{s \in S | p \in L(s)\}$$

- A relational variable $Q_i \in VAR$ is a formula. A relational variable holds at a state if that state is assigned to the relational variable.

$$[\![Q]\!]_M e = e(Q)$$

- If f and g are formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are formulas.

$$[\![\neg f]\!]_M e = S \setminus [\![f]\!]_M e$$

$$[\![f \wedge g]\!]_M e = [\![f]\!]_M e \cap [\![g]\!]_M e$$

$$[\![f \vee g]\!]_M e = [\![f]\!]_M e \cup [\![g]\!]_M e$$

- If f is a formula and $T_{\sigma_i} \in T$ then $[T_{\sigma_i}]f$ and $\langle T_{\sigma_i}\rangle$ are formulas. The formula $[T_{\sigma_i}]f$ holds at a state $s_1 \in S$ if for all $s_2 \in S$ such that $(s_1, s_2) \in T_{\sigma_i}$, f holds at s_2. Similarly, $\langle T_{\sigma_i}\rangle f$ holds at a state $s_1 \in S$ if there exists some $s_2 \in S$ such that $(s_1, s_2) \in T_{\sigma_i}$ and f holds at s_2.

$$\begin{aligned}&[\![\langle T_{\sigma_i}\rangle f]\!]_M e = \\ &\quad \{s_1 | \exists s_2 \, [((s_1, s_2) \in T_{\sigma_i}) \wedge (s_2 \in [\![f]\!]_M e)]\}\end{aligned}$$

$$\begin{aligned}&[\![[T_{\sigma_i}]\, f]\!]_M e = \\ &\quad \{s_1 | \forall s_2 \, [((s_1, s_2) \in T_{\sigma_i}) \wedge (s_2 \in [\![f]\!]_M e)]\}\end{aligned}$$

- If $Q \in VAR$ and f is a formula that is a function of Q, then $\mu Q.f$ and $\nu Q.f$ are formulas, provided that f is syntactically monotone with respect to Q. The least fixpoint of states $\mu Q.f$ is the set of states such that if Q holds in those states, then f also holds in those states. The greatest fixpoint $\nu Q.f$ is similarly defined. A formula f is said to be *syntactically monotone* with respect to a relational variable Q if all occurrences of Q fall under an even number of negations in f. Define $\tau(W)$ to be $[\![f]\!]_M e_{[Q\leftarrow W]}$

$$[\![\mu Q.f]\!]_M e \text{ is the least fixpoint of } \tau(W)$$

$$[\![\nu Q.f]\!]_M e \text{ is the greatest fixpoint of } \tau(W)$$

A function τ is monotonic if $S \subseteq S' \Rightarrow \tau(S) \subseteq \tau(S')$. Because of the monotonicity of all possible functions $\tau(W) = [\![f]\!]_M e_{[Q\leftarrow W]}$ there are simple methods for finding the least and greatest fixpoints. To find the least fixpoint, assign $W_0 := \emptyset, W_1 := \tau(W_0), \ldots$ and so on until $W_{i+1} = W_i$. Then W_i is the least fixpoint. For a k state system the least fixed point will be found in less than k iterative compositions of the $\tau(\cdot)$ operation. Similarly, the greatest fixpoint is found using a similar method by setting $W_0 = S$.

Several important properties of the transition system M can be easily expressed using the μ-calculus. For instance, for a transition system M, $M, s \models \vee_i \langle T_{\sigma_i} \rangle True$ could be used to express that a state $s \in S$ does not deadlock. Also, $M, s \models \nu Q. \left(\vee_i \langle T_{\sigma_i} \rangle True\right) \wedge \left(\wedge_i \left[T_{\sigma_i}\right] Q\right)$ denotes that all states reachable from s are deadlock free. Furthermore, $M, s_0 \models \nu Q_1. \left(\mu Q_2. (m) \vee \left(\vee_i \langle T_{\sigma_i} \rangle Q_2\right)\right) \wedge \left(\wedge_i \left[T_{\sigma_i}\right] Q_1\right)$ can be used to express that all states reachable from s_0 can eventually lead to a state where an atomic proposition m holds.

Given a μ-calculus formula f, the *depth* of f (denoted by $depth(f)$) is defined recursively as follows. Let f_1 and f_2 be two μ-calculus formulas and let T_{σ_i} be a transition class.

- If $f \in AP$ or $f \in VAR$, then $depth(f) = 0$.
- If $f = \neg f_1$, $f = [T_{\sigma_i}] f_1$, $f = \langle T_{\sigma_i} \rangle f_1$, $f = \mu Q. f_1$ or $f = \nu Q. f_1$, then $depth(f) = depth(f_1) + 1$.
- If $f = f_1 \wedge f_2$ or $f = f_1 \vee f_2$, then $depth(f) = \max\{f_1, f_2\} + 1$.

Now that the standard μ-calculus and its properties have been introduced, it is shown how the μ-calculus can be restricted to express behavior in permutation symmetric systems and still take advantage of the permutation symmetry such that two permutation equivalent states satisfy versions of the same μ-calculus formulas. This restriction of the μ-calculus for permutation symmetric systems and permutation symmetric properties is called the *permutation symmetric μ-calculus*.

As a small example how permutation symmetric formulas could be useful, consider a system with permutation symmetry such that module i enters always enters a failure state on the occurrence of local event σ_i when module i is in state x_1 and all other modules are in state x_2. In an n module system there are n permutations of the modular state $(x_1, x_2, \ldots, x_2)$ composed of one x_1 state and $n-1$ x_2 states. If were desired to verify that in this system that no σ_i event could occur in all permutations of the symmetric system structure, n different permutations would need to be checked, but because of underlying system symmetry it is only needed to verify that $\neg \langle T_{\sigma_1} \rangle$ from state $(x_1, x_2, \ldots, x_2)$ and the verification of all of the other redundant symmetric propositions could be avoided.

With this small example as motivation, transition classes and relational variables are defined for $G^{\|}$ so that a μ-calculus formula f can be written for two states $x_a^{\|}, x_b^{\|} \in X^{\|}$ such that $\left(\phi_{ij}(x_a^{\|}) = x_b^{\|}\right) \Rightarrow \left(G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f\right)$.

The μ-calculus is restricted as follows.

- For all $x_a^{\|}, x_b^{\|}$ such that $SP(x_a^{\|}) = SP(x_b^{\|})$, it is required that $L(x_a^{\|}) = L(x_b^{\|})$.
- For all $x_a^{\|}, x_b^{\|}, Q$ such that $SP(x_a^{\|}) = SP(x_b^{\|})$, it is required that $x_a^{\|} \in Q \iff x_b^{\|} \in Q$.
- For $x_a^{\|}, x_b^{\|} \in X^{\|}$, $\sigma_1 \in \Sigma$, if $\delta^{\|}\left(SP(x_a^{\|}), \sigma_1\right) = \Phi_{x_a^{\|}}(x_b^{\|}))$, then $(x_a^{\|}, x_b^{\|})$ is assigned to T_{σ_1}.

For the first bullet the state labeling function was previously restricted to be permutation independent in Section 2. For the second bullet the assignments to the relational variables must be permutation independent as with the state labeling function. The definition of the transition classes in the third bullet ensures that all permutation equivalent transitions (according to $\Phi(\cdot)$ and $\Pi(\cdot)$ mappings) must be in the same permutation class. Excepting these restrictions, formulas in the permutation symmetric μ-calculus can then be constructed in the usual manner. Permutation equivalent states in permutation symmetric systems therefore satisfy the same permutation symmetric μ-calculus formulas.

Theorem 2: Suppose a permutation symmetric system $G^{\|}$ is given with two states $x_a^{\|}, x_b^{\|} \in X^{\|}$ and a permutation symmetric μ-calculus formula f. Then,

$$\left(\phi_{ij}(x_a^{\|}) = x_b^{\|}\right) \Rightarrow \left(G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f\right)$$

V. Permutation Symmetry Quotient Automata

Given a set of automata $\{G_1, \ldots, G_n\}$ such that the size of their respective state spaces is bounded by k, the composed automaton $G_1 \| \cdots \| G_n$ has k^n reachable states in the worst case. As n grows, this state space can become unbearably large and it becomes progressively more difficult to perform any procedure that requires enumerations over all reachable states of $G_1 \| \cdots \| G_n$ in a time-efficient manner. Therefore, it would be impossible to efficiently perform many verification procedures on the composed system $G_1 \| \cdots \| G_n$ with respect to a global specification K using currently known methods. To potentially avoid these restrictions, a quotient automaton $\vec{G}$, as seen in [4, 11], is constructed from the modules $\{G_1, \ldots, G_n\}$ that avoids this problem. The automaton $\vec{G}$ has a predefined quotient structure that does not need to be precomputed.

The automaton $\vec{G}$ is a 6-tuple $\vec{G} = \left(\vec{X}, \vec{x_0}, AP, L^{\|}, \Sigma^{\|}, \vec{\delta}\right)$ such that $\vec{G}$ uses the set of the standard permutations $\vec{X}$ as its state space. A state $\vec{x}$ in $\vec{G}$ is also in $G^{\|}$, so the same proposition labeling function and proposition variable assignments can be used for states in $\vec{G}$ and $G^{\|}$. Let $\vec{x_o}$ be the initial state of $G_1 \| \cdots \| G_n$. Assume that the transition structures of $\{G_1, \cdots, G_n\}$ are generalized such that if $\sigma \notin \Sigma_i$, then $\forall x \in X, \delta_i(x, \sigma) = x$. The state transition function $\vec{\delta} : \vec{X} \rightarrow \vec{X}$ is defined as follows:

$$\vec{\delta}(\vec{x}, \sigma) = \left\{ \begin{array}{l} SP\left(\left(\delta_1(\vec{x}^1, \sigma), \ldots, \delta_n(\vec{x}^n, \sigma)\right)\right) \\ \qquad \text{if } \delta_1(\vec{x}^1, \sigma)! \wedge \cdots \wedge \delta_n(\vec{x}^n, \sigma)! \\ \text{undefined otherwise.} \end{array} \right\}$$

The definition of $\vec{\delta}(\cdot, \cdot)$ can be generalized to be defined over strings of arbitrary length in the usual manner. The isomorphic module system introduced in Example 1 can be used to demonstrate the construction of a reduced state space composed automaton $\vec{G}$.

Example 2: Consider the isomorphic module system G_1, G_2, G_3 introduced in Example 1 above. The set of standard representations of the sets of equivalent states is $\{(0,0,0), (1,2,2)\}$. The reduced state space composed automaton $\vec{G}$ can be seen in Figure 3.

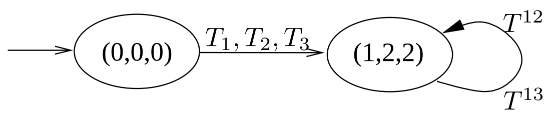

Fig. 3. The automaton $\vec{G}$ constructed from G_1, G_2, G_3.

VI. Verifying Symmetric μ-Calculus Formulas

It is now shown how the $\vec{G}$ system can be used to test permutation symmetric μ-calculus propositions in the original $G^{\|}$ system. Suppose a set of automata $\{G_1, \ldots, G_n\}$ and a formula f in the restricted μ-calculus are given and it is desired to test for a state $x^{\|}$ of $G^{\|}$ if $G^{\|}, x^{\|} \models f$.

Suppose $\{T_{\sigma_1}, \ldots\}$ is the set of transition classes of $G^{\|}$ for the restricted μ-calculus defined above. First, construct the automaton $\vec{G}$ from $\{G_1, \ldots, G_n\}$.

A set of transition classes $\{\vec{T}_{\sigma_1}, \ldots\}$ is constructed from $\{T_{\sigma_1}, \ldots\}$ as follows. For all $(\vec{x}_1, x_2^{\|}) \in T_{\sigma_i}$, $(\vec{x}_1, SP(x_2^{\|}))$ is assigned to $\vec{T}_{\sigma_i}$.

Finally, a μ-calculus formula $\vec{f}$ is constructed from f, $\{T_{\sigma_1}, \ldots\}$ and $\{\vec{T}_{\sigma_1}, \ldots\}$ such that $\vec{f}$ is a copy of f with any occurrence of T_{σ_i} in f replaced with the corresponding $\vec{T}_{\sigma_i}$. This formula construction can be used to test if $G^{\|}, \vec{x} \models f$ as indicated in Theorem 3.

Theorem 3: Let $\vec{x}$ be a state of a permutation symmetric system $G^{\|}$ and let f be a permutation symmetric μ-calculus formula. From this system construct $\vec{G}$, $\{\vec{T}_{\sigma_1}, \ldots\}$ and $\vec{f}$ as described above. Then,

$$\left(G^{\|}\vec{x} \models f\right) \iff \left(\vec{G}\vec{x} \models \vec{f}\right).$$

Due to the permutation symmetry of $G^{\|}$ and f with Theorem 2 and Theorem 3, if f holds at a state $\vec{x}$, then f holds at all states in the same permutation equivalence class.

Corollary 1: Let $x^{\|}$ be a state of a permutation symmetric system $G^{\|}$ and let f be a permutation symmetric μ-calculus formula. From this system construct $\vec{G}$, $\{\vec{T}_{\sigma_1}, \ldots\}$ and $\vec{f}$ as described above and let $\vec{x}$ be $SP(x^{\|})$. Then,

$$\left(G^{\|}x^{\|} \models f\right) \iff \left(\vec{G}\vec{x} \models \vec{f}\right).$$

This corollary is important because μ-calculus propositions are very powerful and are expressive enough to be more general than many common logics such as CTL*, CTL and LTL. The $\vec{G}$ automaton also has a much smaller state space than the $G^{\|}$ automaton, so the state explosion problem inherent to many modular systems is not as problematic; this is demonstrated below in an example.

A version of the $\vec{G}$ automaton is presented in [11] for a restricted class of permutation symmetric isomorphic systems for testing nonblockingness and state reachability in these systems. This paper demonstrates that the $\vec{G}$ of [11] can be adapted to a larger class of symmetric systems with more general μ-calculus specifications.

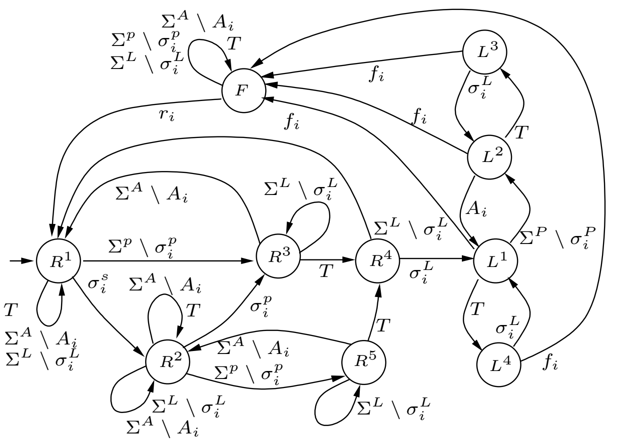
F
$\Sigma^A \setminus A_i$
$\Sigma^p \setminus \sigma_i^p$
$\Sigma^L \setminus \sigma_i^L$
T
L^3
f_i
σ_i^L
r_i
f_i
f_i
T
L^2
$\Sigma^A \setminus A_i$
$\Sigma^L \setminus \sigma_i^L$
A_i
$\Sigma^L \setminus \sigma_i^L$
$\Sigma^P \setminus \sigma_i^P$
R^1
$\Sigma^p \setminus \sigma_i^p$
R^3
T
R^4
σ_i^L
L^1
σ_i^s
$\Sigma^A \setminus A_i$
T
σ_i^p
T
σ_i^L
T
T
$\Sigma^A \setminus A_i$
$\Sigma^L \setminus \sigma_i^L$
R^2
$\Sigma^A \setminus A_i$
$\Sigma^p \setminus \sigma_i^p$
R^5
L^4
f_i
$\Sigma^L \setminus \sigma_i^L$
$\Sigma^A \setminus A_i$
$\Sigma^L \setminus \sigma_i^L$

Fig. 4. High Level UAV Swarm Communication State Automaton.

It should be apparent that the $\vec{G}$ constructed from the isomorphic module system $\{G_1, \ldots, G_n\}$ has a smaller state space than the composed system $G^{\|}$. Suppose that $k = |X|$, the size of the state space of each individual module. Following the approach in [11], it can be shown that $\vec{G}$ has $\begin{pmatrix} k+n-1 \\ n \end{pmatrix}$ classes of states in the worst case. Although this is still a large number of classes of states that need to be verified, it is certainly smaller than k^n, which is the size of $G^{\|}$ in the worst case. This allows for a large real-world reduction in the computation time required to verify propositions in the restricted μ-calculus.

VII. Example

An example of a permutation symmetric distributed leader selection protocol for a swarm of identical UAV's is now discussed. When in a normal operating mode the UAV's have one "leader" that relays communications between the platoon and the home base. The leader may enter a failure state where it may not be able to communicate and then the other platoon members should select a new leader.

A discrete-event system representation of a protocol for an individual UAV can be seen in Figure 4. A UAV is in a regular mode og operation if it is in an R^i state, leader mode if it is in an L^i state and in failure mode if it is in state F. There should be at most one leader that communicates with the platoon's home base. Once in a failure state, a UAV can reset into the initial state.

Starting from the initial state, when a private event σ_i^S occurs in a UAV, the UAV is prompted to broadcast σ_i^p to the other members of the platoon. It is assumed that transmissions between platoon members are never lost. When a leader j observes the broadcast event σ_i^p, it should be acknowledged with the A_j event. If a σ_i^p event is not acknowledged within a certain time, the global broadcast timer event T occurs to signal to the platoon members that a new leader may need to be selected as the previous leader j may have entered a failure state due to the occurrence of a private failure f_j. If the previous leader j has not entered the failure state, it broadcasts σ_j^L to signify that it is still the leader. On the absence of an event in $\Sigma^L \setminus \sigma_i^L$, i then broadcasts σ_i^L to signify that it is declaring itself the new platoon leader, and on the observation of this event, all other regular platoon members return to the initial state. If UAV i enters a failure state, it is reset on the occurrence of r_i to the initial state. Note that at initialization no platoon members are declared leader.

The set Σ^A denotes the set of acknowledgment events that could be sent by the various modules when there are in leader mode. The set Σ^p denotes the events the modules broadcast when in regular mode to communicate with a leader and Σ^L is the set of events the modules broadcast to declare that they are in leader mode. Note that Σ^A, Σ^p and Σ^L all denote broadcast-type events that occur in all of the modules, but on different transitions on each module. For instance, the occurrence of σ_i^L in state R^4 of module i means that module i transitions to leader mode and state L^1. However, the occurrence of σ_i^L in state R^4 of module j means that module j remains in regular mode and returns to state R^1. Using the $\Psi_{ij}(\cdot)$ notation introduced above, $\Psi_{ij}(\sigma_i^L) = \sigma_j^L$ and $\Psi_{ij}(\sigma_j^L) = \sigma_i^L$. This relationship with the translation mapping also holds for events in Σ^A and Σ^p.

The set of failure events and reset events are denoted by Σ^f and Σ^r respectively. The set Σ^s denotes the set of private events for each module to internally signal it would like to broadcast a Σ^p event to the platoon. The events in Σ^f, Σ^r and Σ^s are private to their respective modules and occur in exactly one module each. Using the $\Psi_{ij}(\cdot)$ notation introduced above, $\Psi_{ij}(f_i) = f_j$ and $\Psi_{ij}(f_j)$ is not defined. This relationship with the translation mapping also holds for events in Σ^r and Σ^s.

The timer alarm event T is a global unindexed event that occurs in all modules simultaneously and for identical state transitions. Therefore, $\Psi_{ij}(T) = T$.

For this example, a permutation transition function can be defined as follows for $\sigma \in \Sigma$:

$$\pi_{ij}(\sigma) = \begin{cases} \Psi_{ij}(\sigma) & \text{if} \quad \sigma \in \Sigma_i \\ \Psi_{ji}(\sigma) & \text{if} \quad \sigma \in \Sigma_j \\ \sigma & \text{if} \quad \text{otherwise} \end{cases}$$

Using the π_{ij} definition above, the UAV leader selection protocol in Figure 4 is permutation symmetric. Therefore, a quotient automaton can be constructed to test system properties such as "It is possible for two modules to enter leader mode at the same time?" For even relatively small systems like the one in this example where the modules have 10 local states, there is a significant reduction in the computational difficulty of testing system properties due to use of the quotient automaton. For a small platoon of 4 UAV's, the normal $G^{\|}$ automaton constructed from the composition of the automaton in Figure 4 has 2707 states while the reduced $\vec{G}$ automaton constructed for this example has only 246 states. (These results were found using the UMDES toolbox.) This is over a factor of 10 reduction in the size of the automata and this reduction increases as more automata are added. Testing a relatively simple property like state reachability is linear in the size of the state space, but there are most likely no polynomial time methods for testing more complicated μ-calculus propositions. Therefore, the reductions is state space size with the $\vec{G}$ are potentially very important for "large scale" systems.

VIII. Discussion

A class of isomorphic module systems and a state permutation symmetry definition for these systems has been introduced. A permutation symmetric μ-calculus was introduced that was tailored for these permutation symmetric systems. It was shown how a predefined quotient structure can be used to more efficiently test if permutation symmetric μ-calculus propositions hold at states in permutation symmetric systems. These constructions allow for a large reduction in the real-world difficulty of performing many verification tasks for discrete-event systems. It would be interesting to extend this work to cases where systems are not perfectly symmetric, but allow for some minor variation in local behavior.

References

[1] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, Boston, MA, 1999.

[2] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* The MIT Press, Cambridge, MA, 2002.

[3] E.A. Emmerson and A.P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, August 1996.

[4] J.M. Eyzell and J.E.R. Cury. Exploiting symmetry in the synthesis of supervisors for discrete event systems. *IEEE Trans. Auto. Contr.*, 46(9):1500–1505, September 2001.

[5] P. Gohari and W.M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):643–652, 2000.

[6] I.N. Herstein. *Topics in Algebra.* John Wiley & Sons, Inc., 1982.

[7] C. Hoffman. *Graph Isomorphism and Permutation Groups, LNCS 132.* Springer-Verlag, 1982.

[8] J.F. Knight and K.M. Passino. Decidability of a temporal logic used in discrete-event systems analysis. *International Journal of Control*, 52(6):1489–1506, 1990.

[9] F. Lin. Analysis and synthesis of discrete event systems using temporal logic. *Control Theory and Advanced Technologies*, 9:341–350, 1993.

[10] K. Rohloff and S. Lafortune. On the computational complexity of the verification of modular discrete-event systems. In *Proc. 41st IEEE Conf. on Decision and Control*, Las Vegas, Nevada, December 2002.

[11] K. Rohloff and S. Lafortune. The control and verification of similar agents operating in a broadcast network. In *Proc. 42nd IEEE Conf. on Decision and Control*, Maui, Hawaii, December 2003.

[12] K. Rohloff and S. Lafortune. Supervisor existence for modular discrete-event systems. In *Proc. 2nd IFAC Conf. on Control Systems Design*, Bratislava, Slovakia, September 2003.

[13] K. Rohloff and S. Lafortune. Symmetry reductions for a class of discrete-event systems. Technical Report CGR04-02, Department of Electrical Engineering and Computer Science, University of Michigan, 2004. Available at `http://www.eecs.umich.edu/umdes/sym_tKrep.ps`.

[14] J.G. Thistle and W.M. Wonham. Synthesizing processes and schedulers from temporal specifications. *International Journal of Control*, 44(4):943–976, 1986.