Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Analysis of Arbitrarily Large Networks of Discrete-Event Systems

J.G. Thistle and S. Nazari

Abstract— **Many engineering systems can be usefully modelled as networks of interacting, isomorphic, finite-state discrete-event systems. Examples include communication and transportation networks. For practical purposes, the number of subsystems is often arbitrary. In such cases, key problems of analysis are generally undecidable; however, inductive semidecision procedures can be formulated for checking whether networks of arbitrary size are equivalent to networks of bounded size. The appropriate notion of equivalence may vary, depending on the properties being analyzed. We examine a range of possible equivalences, identify system properties that they preserve, and show that semidecision procedures exist for checking these equivalences. On the other hand, we show that equivalence of networks to networks of bounded size is undecidable for a broad range of process equivalences, even for the simple network topologies of rings and line segments.**

I. INTRODUCTION

Many practical systems can be reasonably modelled as collections of interacting, isomorphic, finite-state systems. Examples include communication networks, such as telephone or "sensor networks," and transportation networks, made up of automated subway trains or other guided vehicles. The number of subsystems may, for all practical purposes, be arbitrarily large. In this paper, we examine analysis problems posed by such systems.

These constitute a class of *parameterized systems*. A parameterized system is actually a family of finite-state systems indexed by the respective values of one or more system parameters: in the present instance, the parameter in question is the number of interacting subsystems; in a model of a manufacturing system, parameters might include buffer capacities. For many realistic examples, it seems reasonable to expect that the logic used to design or analyze systems should be, in its essence, independent of specific parameter values. This paper represents a step in the direction of identifying and studying such examples.

Specific motivation is provided by a study of the development of call-processing services for telephone networks [1]. Typically developed in modular fashion, perhaps even by different vendors, such services or their component features sometimes interact in unforeseen and undesirable ways. A framework for formulating and addressing this problem within a DES control context was presented in [1], where the telephone network was viewed as the "plant," and individual services as distributed controllers. Feature interactions were formally characterized as the blocking of one system component from its marked subset by the rest of the system. To allow practical modelling and analysis, the network was assumed to be made up of only a few subscribers; indeed, in industrial practice, new services are also tested and evaluated on testbench networks of small size. The implicit assumption is that problems that may occur in real networks are reproducible, and therefore detectable, in small networks. This paper attempts to formulate ways of testing such assumptions formally.

A more general problem, that of model-checking any parameterized computer program, was shown to be undecidable in [2]. Other studies have considered the model-checking of parameterized networks of similar subsystems – like the systems examined here. But our work is inspired by modular control synthesis, which entails the testing of specific properties of blocking or deadlock.

Our approach was initially outlined in [3], where it was shown that the problem of checking for blocking of one component subsystem by the rest of the network was undecidable, even when all subsystems are isomorphic. However, if every network is, from the perspective of a given subsystem, weakly bisimilar to a network of bounded size, then blocking can be checked for arbitrary networks. Such bisimilarity is semidecidable.

In this paper we propose the use of other kinds of process equivalences, such as weak possible-futures equivalence for the analysis of blocking, and weak failures equivalence for the testing of deadlock. These equivalences also admit semidecision procedures. We then prove undecidability results that apply to all equivalences that refine weak trace equivalence and are coarser than weak bisimulation – in other words, that apply to the weak versions of all of the equivalences in the so-called "linear time – branching time spectrum" [4]. We conclude with a discussion of related work and future research.

II. PRELIMINARIES

A. Networks of isomorphic subsystems

We shall assume that all component subsystems are isomorphic to a "template" generator

$$G = (\Sigma, Q, \delta, q_0, Q_m) \ ,$$

where, as usual, Σ is a finite alphabet, Q is a finite state set, $\delta : \Sigma \times Q \longrightarrow 2^Q$ is a transition function (2^Q denoting the power set of Q), $q_0 \in Q$ is an initial state, and $Q_m \subseteq Q$ is a subset of marked states. (The transition function extends to strings in the standard fashion.)

For the sake of formal simplicity, we restrict attention to networks with the topology of a ring; this is sufficient for

Research partially supported by grant number RGPIN-155594-01 of the Natural Sciences and Engineering Research Council of Canada.

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 {jthistle|snazari}@kingcong.uwaterloo.ca

0-7803-9568-9/05/$20.00 ©2005 IEEE

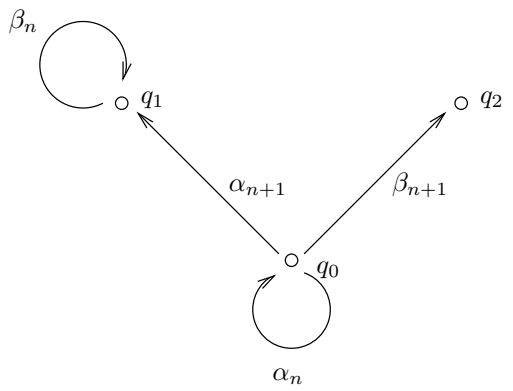

Fig. 1. Example template generator.

the establishment of negative, undecidability results, while the positive results developed within this framework can be extended to more general topologies.

It is therefore convenient to define a notation for sequences of "instances" of the template G, where each instance shares event symbols only with its immediate neighbors in the sequence. We suppose that each symbol in Σ carries a subscript which is an expression – either n or $n+1$ – in an indeterminate n. (See figure 1.) Given any $\sigma \in \Sigma$ and $i \in \mathbb{N}$, $\sigma|_{n=i}$ is the symbol σ with its subscript evaluated to a numerical value by setting $n = i$; For any $i \in \mathbb{N}$, define the alphabet $\Sigma_i := \{\sigma|_{n=i} : \sigma \in \Sigma\}$. Individual subsystems are modelled as generators G_i over the respective alphabets Σ_i:

$$G_i = (\Sigma_i, Q, \delta_i, q_0, Q_m) \ ,$$

where $\delta_i : \Sigma_i \times Q \longrightarrow 2^Q$ is defined so that $\delta_i(\sigma|_{n=i}, q) = \delta(\sigma, q)$, for all $\sigma \in \Sigma$ and $q \in Q$.

We can now create a linear network by simply taking the standard synchronous product of an initial subsequence:

$$P_N := \Pi_{i=0}^{N-1} G_i \ .$$

To "close the loop" and create a ring network, we define $G_i(\mathrm{mod}\ N)$, for $i, N \in \mathbb{N}$, exactly like G_i, but with subscripts of event symbols evaluated using mod N arithmetic (so that $n+1$ evaluates to 0 when $n = N-1$). Then we can create a ring of N template instances by taking the following synchronous product:

$$\begin{aligned} R_N &:= \Pi_{i=0}^{N-1}(G_i(\mathrm{mod}\ N)) \\ &= \Pi_{i=0}^{N-2} G_i \times G_{N-1}(\mathrm{mod}\ N) \end{aligned}$$

Denote the alphabet of $G_i(\mathrm{mod}\ N)$ by $\Sigma_i(\mathrm{mod}\ N)$.

B. Process-algebra notation

It will be useful to have a means of *hiding* events, in the style of process algebra. Let $H = (\Delta, X, \xi, x_0, X_m)$ be a generator and let $\Delta' \subseteq \Delta$. Then

$$H \downarrow \Delta' := (\Delta' \cup \{\tau\}, X, \xi', x_0, X_m) \ ,$$

where

$$\begin{aligned} &\xi' : (\sigma, x) \mapsto \xi(\sigma, x) && \forall \sigma \in \Delta', x \in X \\ &\xi' : (\tau, x) \mapsto \textstyle\bigcup_{\sigma \in \Delta \setminus \Delta'} \xi(\sigma, x) && \forall x \in X \end{aligned}$$

Thus, the hiding of all event symbols not belonging to the subalphabet Δ' relabels the corresponding transitions with the special symbol τ.

This special symbol will play the same role as in process algebra: in synchronous products of generators, it is not treated as a shared event, even if it belongs to the alphabets of both component generators. Given the above generator H, a string $s \in \Delta^*$ and states $x, x' \in X$, we write

$$x \xrightarrow{s} x'$$

to mean that $x' \in \xi(s, x)$; namely, there is a path from x to x' labelled by s. If ϵ is the empty string over Δ, $\sigma \in \Delta \setminus \{\tau\}$, and, again, $s \in (\Delta \setminus \{\tau\})^*$, we write

$$\begin{aligned} &x \stackrel{\epsilon}{\Rightarrow} x' \quad \text{if } x' \in \xi(\tau^*, x) \ ; \\ &x \stackrel{\sigma}{\Rightarrow} x' \quad \text{if } (\exists x_1, x_2 \in X)[x \stackrel{\epsilon}{\Rightarrow} x_1, x_1 \xrightarrow{\sigma} x_2 \ \&\ x_2 \stackrel{\epsilon}{\Rightarrow} x'] \ ; \\ &x \stackrel{s\sigma}{\Rightarrow} x' \quad \text{if } (\exists x_1 \in X)[x \stackrel{s}{\Rightarrow} x_1 \ \&\ x_1 \stackrel{\sigma}{\Rightarrow} x'] \end{aligned}$$

That is, $x \stackrel{s}{\Rightarrow} x'$ if there exists a path from x to x' labelled by some interleaving of the symbols of s with τ symbols. We write $x \xrightarrow{s}$ (resp. $x \stackrel{s}{\Rightarrow}$ if there exists $x' \in X$ such that $x \xrightarrow{s} x'$ (resp. $x \stackrel{s}{\Rightarrow} x'$).

C. Blocking in networks

Inspired by the example of the telephone networks of [1], we say that, given two generators $H_1 = (\Delta_1, X_1, \xi_1, x_{01}, X_{m1}), H_2 = (\Delta_2, X_2, \xi_2, x_{02}, X_{m2})$, the synchronous product $H_1 \times H_2$ *blocks* the component generator H_1 (resp. H_2) if there exist a string $s \in (\Delta_1 \cup \Delta_2)^*$, and some state (x_{11}, x_{12}) such that in the product $(x_{01}, x_{02}) \xrightarrow{s} (x_{11}, x_{12})$, and for all $s' \in (\Delta_1 \cup \Delta_2)^*$ and states (x_{21}, x_{22}) such that $(x_{11}, x_{12}) \xrightarrow{s'} (x_{21}, x_{22})$, the H_1 (resp. H_2) state component x_{21} (resp. x_{22}) is unmarked. In other words, the product blocks H_1 if H_1 may become unable to reach its marked state subset.

An approach to the analysis of blocking in ring networks was outlined in [3]. The next section presents an improved procedure.

III. ANALYSIS VIA PROCESS CONGRUENCES

The blocking of one component subsystem in a ring network can be studied by examining the behaviour of the rest of the network from the perspective of that subsystem. We shall therefore analyze blocking in R_N by considering the linear segment made up of all subsystems other than G_0, with events outside of Σ_0 hidden:

$$S_N := (\Pi_{i=1}^{N-2} G_i \times G_{N-1}(\mathrm{mod} N)) \downarrow \Sigma_0$$

Proposition 1 ([3]): The ring network R_N blocks G_0 if and only if $G_0 \times S_N = R_N \downarrow \Sigma_0$ blocks G_0.

Proof: (The hiding of events outside of Σ_0 is irrelevant to the blocking of G_0.) ■

Thus, checking blocking of G_0 by R_N is equivalent to checking blocking of G_0 by $G_0 \times S_N = R_N \downarrow \Sigma_0$. The

latter could be done for arbitrary network size N if it could be established that every network segment S_N is equivalent, as far as blocking of G_0 is concerned, to some S_M, where the value of M is uniformly bounded. This raises the question of the definition of an adequate notion of equivalence. Such a property is furnished by *possible futures* equivalence.

Definition 1: Let $H = (\Delta, X, \xi, x_0, X_m)$ be a generator. The set of *weak possible futures* of H is given by

$$PF(H) := \{(s, L) \in \Delta^* \times 2^{\Delta^*} : (\exists x_1 \in X)[x_0 \stackrel{s}{\Rightarrow} x_1 \ \& \ \{s' : x_1 \stackrel{s'}{\Rightarrow}\} = L]\}$$

Two generators H_1 and H_2 are *weakly possible-futures equivalent* if $PF(H_1) = PF(H_2)$.

Like other process equivalences, weak possible-futures equivalence has a stronger version obtained by replacing the double arrows in the definition with single arrows. A possible future of a generator thus consists of a string that labels a path from the initial state to some successor state, and of a language that labels all paths from the successor state. Generators are possible-futures equivalent if they share the same possible futures. As do other process equivalences, possible-futures equivalence reduces to trace equivalence (see below) for deterministic generators.

Proposition 2: Weak possible-futures equivalence is a congruence for synchronous product and for event hiding: let H_{11}, H_{12}, H_{21} and H_{22} be generators; then if $PF(H_{11}) = PF(H_{21})$ and $PF(H_{12}) = PF(H_{22})$, we have

$$PF(H_{11} \times H_{12}) = PF(H_{21} \times H_{22}) \ ;$$

and for any subalphabet Δ, we have

$$PF(H_{11} \downarrow \Delta) = PF(H_{21} \downarrow \Delta)$$

Moreover, it is also a congruence for event renaming. ■

Proposition 3: Let H_1, H_2 and H_3 be generators. If $PF(H_1) = PF(H_2)$, then $H_1 \times H_3$ blocks H_3 if and only if $H_2 \times H_3$ blocks H_3. ■

Proposition 2 implies the existence of a semidecision procedure for the existence of a bound such that every linear segments S_N is weakly possible-futures equivalent to some linear segments with bounded S_M where M is bounded by a constant. By Proposition 3, the existence of a bound allows one to check for blocking in networks of arbitrary size.

The semidecision procedure is as follows. Construct linear segments S_N, and after appropriate renaming of events, check each against the previous ones for weak possible-futures equivalence. If S_L is equivalent to S_K, for $K < L$, then, by Proposition 2, S_{K+1} is equivalent to S_{L+1}, and so forth. Hence, every segment S_N is weakly possible-futures equivalent to some S_M, where $M < L$; in particular, if $L = K + 1$, S_N is equivalent to S_K for all $N \geq K$. For instance, this happens for $K = 3$ in the case of the "toy" example of Figure 1.

Suppose that this semidecision procedure establishes such a bound. Then by Propositions 1 and 3, blocking in networks of arbitrary size can be checked, by considering only ring networks of bounded size.

As a means of checking blocking, this improves on the procedure of [3], which was based on *weak bisimilarity*.

Definition 2: Two generators H_1 and H_2 with initial states x_{01} and x_{02}, respectively, are *weakly bisimilar* if there exists a binary relation $B \subseteq X_1 \times X_2$ such that for all $\sigma \in (\Delta \setminus \tau)^*$,

1) $(x_{01}, x_{02}) \in B$,
2) $(x_{11}, x_{12}) \in B$ and $x_{11} \stackrel{\sigma}{\Rightarrow} x_{21}$ together imply $(\exists x_{22} \in X_2)[x_{12} \stackrel{\sigma}{\Rightarrow} x_{22}] \ \& \ B(x_{21}, x_{22})]$, and
3) $(x_{11}, x_{12}) \in B$ and $x_{12} \stackrel{\sigma}{\Rightarrow} x_{22}$ together imply $(\exists x_{21} \in X_1)[x_{11} \stackrel{\sigma}{\Rightarrow} x_{21}] \ \& \ B(x_{21}, x_{22})]$.

Intuitively, so long as two generators are in weakly bisimilar states, each can duplicate any external actions of the other in such a way as to preserve the weak bisimilarity of their states. Weak bisimilarity shares the congruence properties of possible-futures equivalence, and is a finer equivalence relation. It therefore leads to a semidecision procedure which, if it terminates, allows checking for blocking. But because it is a finer equivalence relation, the semidecision procedure need not terminate whenever that based on weak possible futures does; and when it does terminate, it need not do so as quickly.

Indeed, weak bisimilarity is a relatively strong equivalence relation. It preserves the satisfaction of all formulas of observable modal logic [5], and in an extensive catalogue of strong process equivalences [4], bisimilarity is the most refined. There also exists a least refined equivalence in this so-called "linear time – branching time spectrum" – namely, weak trace equivalence.

Definition 3: Two generators H_1 and H_2, with initial states x_{01} and x_{02}, respectively, are *weakly trace equivalent* if, for all $s \in (\Delta_1 \cup \Delta_2 \setminus \tau)^*$, $x_{01} \stackrel{s}{\Rightarrow}$ in H_1 if and only if $x_{02} \stackrel{s}{\Rightarrow}$ in H_2.

Thus, two generators are weakly trace-equivalent if they generate the same substrings of non-τ events. It is not hard to see that, owing to potential preemption by the "internal event" τ, trace equivalence is in fact too coarse a relation to preserve blocking properties.

Intermediate in coarseness between weak trace equivalence and weak bisimilarity are many equivalences which may in principle be of use in the analysis of networks [4]. For example, *weak failures equivalence* preserves deadlock properties. In the next section, we prove, for all such equivalences, that equivalence of networks to networks of bounded size is undecidable. These results serve not merely to establish the fact of undecidability, but to provide an indication of the kinds of structure that would have to be introduced to yield decidability – any suitable sufficient conditions would have to exclude the constructions used in the proofs.

IV. UNDECIDABILITY RESULTS

We begin by extending a result that was proved for weak bisimilarity in [3]:

Theorem 1: Consider any equivalence that refines weak trace equivalence and is coarser than weak bisimilarity. There is no algorithm that determines whether all generators $R_N \downarrow \Sigma_0, N \in \mathbb{N}$ fall into a finite number of equivalence classes,

and if so, computes a bound on the size of the smallest members of all equivalence classes. In other words, it is impossible to decide whether all ring networks are equivalent to rings of bounded size, and, if this is so, to compute a corresponding bound.

Proof: In [3], a construction was given that produced, for an arbitrary Turing machine, a finite-state "template" with the property that, if the Turing machine halted on the empty input, all $R_N \downarrow \Sigma_0$ were weakly bisimilar, for sufficiently large N.

On the other hand, if all $R_N \downarrow \Sigma_0, N \in \mathbb{N}$ are trace-equivalent to rings of size smaller than a known bound, then by the construction, it can be determined whether or not the Turing machine halts on the empty input. It follows that if all $R_N \downarrow \Sigma_0$ are equivalent to rings of size smaller than a known bound, for any given one of our equivalences, halting can be checked.

Suppose then that there exists an algorithm of the sort described in the statement of the result. Apply it. If the networks are not all equivalent to networks of bounded size, the Turing machine does not halt. On the other hand, if networks are all equivalent to networks smaller than a given bound, halting can be checked. It follows that the halting problem is decidable, a contradiction. ■

The above result thus shows that no algorithm tests for equivalence of networks of arbitrary size to networks of bounded size, and computes a suitable bound if one exists. But our semidecision procedures work not with the $R_N \downarrow \Sigma_0$ themselves, but with the "linear segments" S_N. We now show that equivalence of linear segments of arbitrary size to segments of bounded size is undecidable, as is equivalence of all sufficiently long linear segments. This holds for any equivalence that refines weak trace equivalence or is coarser than weak bisimulation.

The proof is by reduction from the Turing machine *mortality problem* – or, equivalently, from the problem of deciding whether a Turing machine is *uniformly mortal*. Consider a Turing machine with a two-way infinite tape. Let the (finite) tape alphabet be Σ and let the possible states of the control unit be labelled with elements of a finite alphabet Q. The configuration, or full state, of the Turing machine can be encoded as a two-way infinite string $lqr \in \Sigma^\omega Q \Sigma^\omega$, where $l, r \in \Sigma^\omega$ are infinite strings of tape symbols and $q \in Q$ is a control-state symbol, supposing, for example, that the tape symbol under the read/write head is the leftmost symbol in r. Such a Turing machine is said to be *mortal* if all of its computations, regardless of the configuration in which they begin, eventually halt. Call the problem of deciding whether an arbitrary Turing machine is mortal the *mortality problem*.

Theorem 2 (Hooper [6]): The mortality problem is undecidable.

Note the differences between the mortality problem and the halting problem: in the former case, the "input" to the Turing machine is a two-way infinite string; and the initial state of the finite-state control is arbitrary. This second feature is important for our application. We also need the following result. Call a Turing machine *uniformly mortal* if, whatever its initial configuration, it halts after a number of computation steps that is bounded by a constant. It turns out that this condition is equivalent to mortality:

Theorem 3 (Hillebrand, Kannelakis et al [7]): A Turing machine is mortal if and only if it is uniformly mortal.

Theorems 2 and 3 show that the problem of determining whether an arbitrary Turing machine is uniformly mortal is undecidable. We shall use a recursive reduction from this problem to show that it is undecidable whether, for an arbitrary finite-state template, linear network segments fall into a finite number of equivalence classes, or whether all sufficiently long segments are equivalent, and this, for any equivalence that is stronger than weak trace equivalence and weaker than weak bisimulation.

Before stating the result, we describe the construction used in the reduction. Suppose that an arbitrary Turing machine $\mathcal{M}$ is given. It is convenient to assume that the tape alphabet Σ and the set of labels Q of states of the control unit are disjoint.

It is also convenient to modify the scheme proposed above for encoding Turing machine configurations. Strings of the form

$$\ldots l_3 l_2 l_1 q r_1 r_2 r_3 \ldots$$

will instead be written as one-way infinite strings

$$q r_1 l_1 r_2 l_2 r_3 l_3 \ldots$$

The idea is to design a finite state machine into which a string of the latter form can be fed as a sequence of "inputs," encoding a configuration of the Turing machine, so that the state machine can "output" a string of the same form that encodes the successor configuration. Naturally, the finite state machine will have to perform this function using only a bounded amount of storage. We shall describe the machine using notation of the following sort:

q	r_1	l_1	r_2	l_2	r_3	l_3		$\ldots$
	q'	r_1'	l_1'	r_2'	l_2'	r_3'	l_3'	$\ldots$

The string above the line represents a sequence of inputs to the machine, while that below the line denotes a corresponding output sequence, both temporally ordered from left to right. (The machine will only accept input strings in $Q\Sigma^*$.) The alignment of the two strings describes the temporal interleaving of input and output events: an event in the output sequence occurs immediately after the input event that is written directly above it. The above example therefore means that after the first two input events, q and r_1, occur, the output event q' takes place; after the next input event, l_1, the output r_1' is generated; and so forth.

We shall construct the machine so as to simulate the transition relation of the Turing machine. Thus, if the input string encodes a configuration, the output string is to encode that configuration's successor. This can be achieved with a finite-state template: the state machine's logic need only incorporate copies of the Turing machine's state transition function, and the table governing head motion and the writing of tape symbols, together with sufficient memory to store a

bounded number of past input symbols. The machine will be designed so that it admits only input strings that do encode Turing machine configurations: the initial input event must be a state symbol, and subsequent input events must be tape symbols.

It is convenient to assume, without loss of generality, that the Turing machine has a single halting state. We can then prescribe an input-output function according to the following rules:

1) If q represents a nonhalting-state symbol and r_1 a tape symbol such that, when it reads r_1 while in state q the Turing machine moves to state q', writes a symbol r_1' on the tape, and does not move its read/write head, then

q	r_1	l_1	r_2	l_2	r_3	l_3	$\ldots$
		q'	r_1'	l_1	r_2	l_2	$\ldots$

2) If q represents a nonhalting-state symbol and r_1 a tape symbol such that, when it reads r_1 while in state q the Turing machine goes to state q', writes a symbol r_1' on the tape, and moves its read/write head to the right, then

q	r_1	l_1	r_2	l_2	r_3	l_3	r_4	$l_4 \ldots$
		q'	r_2	r_1'	r_3	l_1	$\ldots$	

3) If q represents a nonhalting-state symbol and r_1 a tape symbol such that, when it reads r_1 while in state q the Turing machine moves its read/write head to the left, goes to state q' and writes a symbol r_1' on the tape, then

q	r_1	l_1	r_2	l_2	r_3	l_3	r_4	$l_4 \ldots$
		q'	l_1	l_2	r_1'	l_3	r_2	$l_4 \ldots$

4) If q_h represents the halting state, then

q_h	$\ldots$
q_h	

The first three rules require the state machine to simulate nonhalting moves of the Turing machine, in the sense that the input string represents an encoding of a nonhalting Turing machine configuration, and the output string encodes that configuration's successor. Note that implementation of these rules entails only the storage of a bounded number of past symbols. The fourth rule covers the case of a halting input configuration: note that this rule allows the halting state to appear at the state-machine output as soon as it has appeared at the input.

Consider a cascade of machines satisfying these rules. Strings can be "inputted" to the first machine, and a resulting output string will be generated by the last. If the input string encodes the initial configuration of the Turing machine, then the input to the second machine will encode the Turing machine's second configuration, that of the third machine the third configuration, and so forth. If ever a string that encodes a halting configuration is generated, by any of the machines, it immediately becomes possible for the halting-state symbol to be passed to the output of the last machine.

Theorem 4: It is undecidable whether there exists a finite number of equivalence classes of linear network segments S_N for an arbitrary finite state template G and any process equivalence relation that is stronger than weak trace equivalence and weaker than weak bisimulation.

Proof: Suppose that an arbitrary Turing machine is given. A finite-state template can be constructed that suitably simulates the finite-state machine constructed above.

We shall first show that, if there exist linear network segments based on this template that are weakly trace-equivalent, then the mortality of the Turing machine can be established or disproved. First note that the existence of weakly trace-equivalent segments implies that all segments are weakly trace-equivalent to segments of bounded size, and that a suitable bound can be therefore be computed. But suppose that it is known that all segments are weakly trace-equivalent to segments of length l or smaller. Then by our construction, the Turing machine can pass through at most $l+1$ distinct configurations on any computation, regardless of its initial configuration. It can therefore read at most $l+1$ distinct tape cells, so it in effect has a tape of length $2l+1$. Hence, it can be determined whether it is mortal or not.

On the other hand, if the Turing machine is uniformly mortal, then sufficiently long segments of instances of our template are weakly bisimilar. Indeed, suppose that, regardless of its initial configuration, the Turing machine halts after at most c computation steps. For any two segments of length greater than c one can define a binary relation R on their state sets that holds precisely when the first c instances of one segment are in the same states as their respective counterparts in the other instance, and when the last instances of both segments are in the same state. This is a weak bisimulation between the two segments, for given the same inputs, the initial segments of c instances can make exactly the same transitions; and the last machines in the two segments can only do nothing or else output one halting-state symbol, and they can do the latter if and only if one of the first c instances has outputted the halting-state symbol.

Now suppose that the theorem does not hold for one of the equivalences that it covers. Then there exists a decision procedure for that equivalence relation. For an arbitrary Turing machine, apply the decision procedure to the template produced by our construction. If the number of equivalence classes is finite, there must exist two segments that are weakly trace-equivalent. But then we can determine whether or not the Turing machine is mortal, as shown above. On the other hand, if the number of equivalence classes is infinite, then so is the number of weak-bisimulation equivalence classes. It follows that the Turing machine must not be mortal. Thus, the mortality problem is decidable, which is a contradiction. ■

V. CONCLUSION

We have considered the analysis of a class of parameterized discrete-event systems – namely, networks of arbitrary size, composed of isomorphic subsystems. The analysis of blocking and other properties is generally undecidable, but

process equivalences such as weak possible futures admit semidecision procedures for the equivalence of arbitrary networks to networks of bounded size. Because such equivalences preserve important system properties, they allow their effective checking when semidecision procedures terminate. Our approach was first outlined in [3], for the particular case of weak bisimulation equivalence.

A considerable amount of related work is reported in the model-checking literature. Among the earliest is a study that proposes a special bisimulation relation, which must be constructed by *ad hoc* means, and an associated logic Indexed CTL* [8], [9]. The modification of the definition of bisimulation and the specialized logic are necessitated by an interest in checking properties of the behavior of the network as a whole, rather than viewing it from the perspective of one node, as we have here. Without the modifications, the bisimulation relation and logic would be capable of distinguishing networks of different sizes. Unfortunately, the modification of bisimulation destroys its congruence property, which in turn requires the *ad hoc* construction of a *process closure*. To address the drawbacks of these approaches, induction schemes were proposed in [10], [11] to show that networks "implemented" a specification, or were related to it by language containment, or some other process-algebra preorder. These approaches sometimes require the replacement of the specification with a "network invariant," constructed in *ad hoc* fashion. Because we are concerned not with equivalences for entire networks but for the segments S_N obtained by hiding events outside Σ_0, we are able to use standard process equivalences.

Future work will seek to identify simpler sufficient conditions reducing analysis to the case of bounded networks. These will of course have to rule out the constructions of our undecidability proofs. Some such conditions have been identified in the literature, but they represent severe restrictions on communication among subsystems. Some require that subsystems be not only isomorphic, but identical, so that no two are distinguishable from the perspective of a third [12], [13]; others limit communication to the passing of unary tokens [14].

REFERENCES

[1] J. G. Thistle, R. P. Malhamé, H.-H. Hoang, and S. L. Lafortune, "Feature interaction modelling, detection and resolution: A supervisory control approach," in *Feature Interactions in Telecommunication Networks*, P. Dini, R. Boutaba, and L. Logrippo, Eds. IOS Press, 1997, pp. 13–22.

[2] K. R. Apt and D. C. Kozen, "Limits for automatic verification of finite-state concurrent systems," *Information Processing Letters*, vol. 22, pp. 307–309, 1986.

[3] J. G. Thistle and S. Nazari, "Model reduction in distributed supervision," in *16th International Symposium on Mathematical Theory of Networks and Systems*, 2004.

[4] R. J. van Glabbeek, "The linear time – branching time spectrum II; the semantics of sequential processes with silent moves," in *Proceedings CONCUR '93* LNCIS No. 715. Springer-Verlag, 1993, pp. 66–81.

[5] C. Stirling, *Modal and Temporal Properties of Processes*, ser. Texts in Computer Science. Springer-Verlag, 2001.

[6] P. K. Hooper, "The undecidability of the Turing machine immortality problem," *Journal of Symbolic Logic*, vol. 31, no. 2, pp. 219–234, June 1966.

[7] G. G. Hillebrand, P. C. Kanellakis, H. G. Mairson, and M. Y. Vardi, "Undecidable boundedness problems for datalog programs," *Journal of Logic Programming*, vol. 25, no. 2, pp. 163–190, 1995.

[8] E. M. Clarke, O. Grümberg, and M. C. Browne, "Reasoning about networks with many identical finite-state processes," Carnegie-Mellon University, Department of Computer Science CMU-CS-86-155, Oct. 1986.

[9] E. M. Clarke and O. Grümberg, "Avoiding the state explosion problem in temporal logic model-checking algorithms," in *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, 1987, pp. 294–303.

[10] P. Wolper and V. Lovinfosse, "Verifying properties of large sets of processes with network invariants," in *Proceedings of the international workshop on Automatic verification methods for finite state systems*. Springer-Verlag New York, Inc., 1990, pp. 68–80.

[11] R. Kurshan and K. McMillan, "A structural induction theorem for processes," *Information and Computation*, vol. 117, pp. 1–11, 1995.

[12] S. M. German and A. P. Sistla, "Reasoning about systems with many processes," *Journal of the Association for Computing Machinery*, vol. 39, no. 3, pp. 675–735, July 1992.

[13] E. A. Emerson and V. Kahlon, "Reducing model checking of the many to the few," in *Conference on Automated Deduction*, 2000, pp. 236–254. [Online]. Available: citeseer.nj.nec.com/emerson00reducing.html

[14] E. A. Emerson and K. S. Namjoshi, "Reasoning about rings," in *Conference Record of POPL '95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1995, pp. 85–94.