Regular Paper

When Can We Safely Reuse Systems, Upgrade Systems, or Use COTS Components?

A. Wayne Wymore and A. Terry Bahill

Systems and Industrial Engineering, University of Arizona, Tucson, AZ 85721-0020

Received March 4, 1999; revised October 24, 1999; accepted March 16, 2000

ABSTRACT

In this article it is proven that two systems are I/O equivalent with respect to an initial state pair if and only if their minimizations are isomorphic images, which means that the minimizations are essentially the same systems with just a renaming of states, inputs, and outputs. The concept of homomorphic images is also defined: One system is a homomorphic image of the other if the one system is simpler than the other, but has essentially the same overall functionality. Two systems that are I/O equivalent with respect to an initial state pair are not necessarily homomorphic images. Hence, a system that implements the one may not implement the other. To assume otherwise can lead to disaster. It is the responsibility of systems engineers assigned to system functional analysis to consider the I/O requirement and the performance requirement (which may include reusability considerations) and to specify functional system designs for implementation. It is the responsibility of systems engineers assigned to system architecture to consider the technology requirement and the cost requirement and to specify buildable system designs to implement those *exact* functional system designs, not the requirements for I/O behavior. This paper shows that the equivalence of two systems cannot be proven by looking only at input and output behavior. © 2000 John Wiley & Sons, Inc. Syst Eng 3: 82–95, 2000

1. INTRODUCTION

Reusability, upgrades, and pressure to use commercial off-the-shelf (COTS) systems force the question "How can we prove that two systems are equivalent?" First, suppose a system called $Z1$ was designed to perform task-1. Next, suppose task-2 needs to be performed. A new system called $Z2$ could be designed, or perhaps $Z1$ could be reused. In many cases, it would be a lot cheaper to reuse $Z1$. For simplicity assume task-2 is a subset of task-1; for example, task-1 could be spelling and grammar checking, and task-2 might be only spelling check-

Systems Engineering, Vol. 3, No. 2, 2000

ing. A necessary, but not sufficient, condition for using *Z*1 for task-2 is that the I/O behavior of *Z*1 and *Z*2 be identical for task-2. For example, the I/O behavior of *Z*1 and *Z*2 would have to be identical when checking spelling.

Second, suppose that we have a large complex system that has been working well for several years, but the hardware is getting old and expensive to maintain. Will our application still work if we upgrade from hardware-1 to hardware-2? Or perhaps our software vendor has come out with a new version. Other sites have upgraded, but we have not, so we are losing compatibility. Will our application still work if we upgrade from software-1 to software-2? A necessary but *not* sufficient condition for success of the upgrade is that the input/output behavior of the application be the same on the old system as it is on the new system.

Third, suppose we used to make custom systems for our customer. They worked very well, but they were expensive. Now our customer wants us to design a new system, but to keep the cost down, he wants us to use commercial off-the-shelf (COTS) components. We have a design, *Z*1, that satisfies the customer's needed functionality. But we want to know if a COTS product, *Z*2, will also satisfy this functionality. For a start, we can create input trajectories, play them into both designs and look to see if the output trajectories are the same.

These three scenarios require answers to the same question as our old systems engineering validation problem. Suppose Engineering designs a system; then Manufacturing builds a physical system. Could an engineer prove that the physical system implements the design? If the states were observable, then the engineer could construct an input trajectory that exercised all state transitions, apply this input trajectory to both systems, and compare the resulting state trajectories. If they were identical, then the systems would be equivalent. However, if the system states were not observable, then the only thing the engineer could work with is the input/output behavior of the systems. In this paper, we show what you can say about systems where only the inputs and outputs can be observed.

Notation: This paper applies to discrete systems of the form $Z = (SZ, IZ, OZ, NZ, RZ)$, where Z is the name of the system, SZ is the set of states, IZ is the set of inputs, OZ is the set of outputs, NZ is the next-state function that describes the next state for all combinations of present state and input values, and RZ is the readout function that specifies the outputs for each state [Chapman, Bahill and Wymore, 1992; Wymore, 1993]. The time scale of Z is denoted TZ and is defined as $TZ = IJS{+}{+}$, meaning zero and the positive integers. The set of all possible input trajectories is denoted ITZ and is defined as $ITZ = FNS(TZ, IZ)$, meaning the set of all functions that are defined on the set TZ with values in the set IZ. Thus, an input trajectory is a listing of input values as a function of time. We typically use the symbol f for a representative input trajectory. For a typical system experiment, we start the system in some initial state, which we symbolize with DSZ, we specify the input trajectory f that will be applied to the system and then we compute the outputs as a function of time, which is called the output trajectory. Thus, the output trajectory is parametrized by the input trajectory and the initial state: the output trajectory for system $Z1$ with input trajectory f and initial state $DSZ1$ is denoted $OTZ1(f, DSZ1)$. The output at a particular time t is denoted $OTZ1(f, DSZ1)(t)$. Similarly, a listing of the states as a function of time, which is called the state trajectory, is parameterized by the input trajectory and the initial state. The state trajectory for system $Z1$ with input trajectory f and initial state $DSZ1$ is denoted $STZ1(f, DSZ1)$. The state at a particular time t is denoted $STZ1(f, DSZ1)(t)$ and $OTZ1(f, DSZ1)(t) = RZ(STZ(f, DSZ1)(t))$. Furthermore, $STZ(f, DSZ)(t + 1) = NZ(STZ(f, DSZ)(t), f(t))$.

2. I/O EQUIVALENCE WITH RESPECT TO AN INITIAL STATE PAIR

Definition: Two systems $Z1$ and $Z2$ are input/output (I/O) equivalent with respect to the initial state pair $(DSZ1, DSZ2)$ if and only if

1. $IZ1 = IZ2$, the inputs are the same,
2. $OZ1 = OZ2$, the outputs are the same,
3. $DSZ1 \in SZ1$, the initial state $DSZ1$ is a state of $Z1$,
4. $DSZ2 \in SZ2$, the initial state $DSZ2$ is a state of $Z2$,
5. for every $f \in ITZ1 = ITZ2$, $OTZ1(f, DSZ1) = OTZ2(f, DSZ2)$, for every possible input trajectory the output trajectories are the same, and
6. every state of $Z1$ is reachable from the initial state $DSZ1$, and every state of $Z2$ is reachable from the initial state $DSZ2$.

Note: When there is no chance of ambiguity, we will abbreviate "input/output equivalent with respect to an initial state pair" with "I/O equivalent."

Nomenclature: Intuitively, systems that have the same functionality, but have different names for the inputs, outputs, and states are isomorphic images. Systems that have essentially the same functionality, but have different names and possibly a different number of inputs, outputs, and states are homomorphic images.

Homophones are words that have the same sound, but differ in spelling, origin and meaning, like night and knight. They sound alike but look different. This paper is about systems that have the same input/output behavior, but differ in purpose, states, and structure: They function alike but look different. Analogously, we say that systems with the same input/output behavior are homophonic echoes.

3. THEOREM 1 AND ITS IMPLICATIONS

Theorem 1: *I/O equivalence with respect to an initial state pair is reflexive, symmetric, and transitive.*

If $\{Z1, Z2, Z3\}$ are systems, then $Z1$ is I/O equivalent to $Z1$ with respect to $DSZ1$ for any $DSZ1 \in SZ1$ (reflexivity),
if $Z1$ is I/O equivalent to $Z2$ with respect to the initial state pair $(DSZ1, DSZ2)$, then $Z2$ is I/O equivalent to $Z1$ with respect to the pair $(DSZ2, DSZ1)$ (symmetry),
if $Z1$ is I/O equivalent to $Z2$ with respect to the pair $(DSZ1, DSZ2)$, and $Z2$ is I/O equivalent to $Z3$ with respect to $(DSZ2, DSZ3)$, then $Z1$ is I/O equivalent to $Z3$ with respect to $(DSZ1, DSZ3)$, (transitivity).

4. HOMOMORPHISMS

In general, homomorphic images require the definition of three homomorphic functions: *HI*, mapping the input values from one system to the other, *HO*, mapping the output values, and *HS*, mapping the states. However, in this paper we will simplify the mathematics (without loss of generality) and only consider systems that have the same input and output values, meaning both functions *HI* and *HO* are the identity over the sets of inputs and the sets of outputs, respectively. If such systems are homomorphs, then they are called state-homomorphs.

4.1. State-Homomorphisms

Definition: The system $Z1$ is a state-homomorphic image of the system $Z2$ with respect to HS if and only if

1. $IZ1 = IZ2$,
2. $OZ1 = OZ2$,
3. $HS \in FNS\ (SZ2, \text{ONTO}, SZ1)$, a function mapping states of $SZ2$ to states of $SZ1$ must exist and it must satisfy the ONTO property,
4. $HS(NZ2(x, p)) = NZ1(HS(x), p)$ for every $p \in IZ2$ and $x \in SZ2$, the image of the next state of $Z2$, started in the state x, is the next state of $Z1$ with the initial state $HS(x)$ and the same input, (the symbol p is used for a particular value of the input and x is used for a particular value of the state) and
5. $RZ1(HS(x)) = RZ2(x)$ for every $x \in SZ2$, if the states in $RZ1$ are mapped according to HS the result will be equal to $RZ2$.

Notation: A function specifies a mapping of elements of set A (the domain) onto elements of set B (the range). A function must be single-valued, i.e., no two elements of B are images of the same element of A. (Terminology: elements of A are *mapped* onto elements of B, whereas elements of B are *images of* elements of A.) If a function g is 1TO1, denoted $g \in FNS(A, 1TO1, B)$, then no two elements of A are mapped onto the same element of B. If a function g is ONTO, denoted $g \in FNS(A, ONTO, B)$, then each element of B is the image of at least one element of A. The notation $FNS(A, 1TO1, ONTO, B)$ refers to all functions relating A to B that are 1TO1 and ONTO.

Corollaries: (1) State-homomorphism implies identical input sets and identical output sets. $HS(STZ2(f, x)(t) = STZ1(f, HS(x))(t)$ and $OTZ2(f, x)(t) = OTZ1(f, HS(x))(t)$, for every $f \in ITZ$ and time t. (2) Systems $Z1$ and $Z2$ are state-*isomorphic* if and only if $Z1$ and $Z2$ are state-homomorphic with respect to HS, and HS is 1TO1. In other words, isomorphic images have the same number of states and the state relationship can be thought of as a mere renaming.

Definition: $Z1$ is a homophonic echo of $Z2$ with respect to HS if and only if $HS \in FNS(SZ2, \text{ONTO}, SZ1)$ such that $OTZ2(f, x)(t) = OTZ1(f, HS(x)(t)$ for all $\in IJS{+}{+}$ and $RZ2(x) = I2Z1(HS(x))$ for all $x \in SZ2$.

4.2. Homomorphism Examples

Notation: To define a system (Z) we must specify its set of states (SZ), its set of inputs (IZ), its set of outputs (OZ), its next-state function (NZ), and its readout function (RZ). For example, let us define system ZxA.

Specification of example system *ZxA*: Let $ZxA = (SZxA, IZxA, OZxA, NZxA, RZxA)$, where

$SZxA = \{1, 2, 3, 4\}$ are the names of the system's four states,

$IZxA = \{5, 6, 7\}$ are the legal values for the system inputs,

$OZxA = \{8, 9, 10\}$ are the output values produced by the system,

$NZxA = \{((1, 5), 2), ((1, 6), 3), ((1, 7), 1),$
$(2, 5), 3), ((2, 6), 3), ((2, 7), 1),$
$((3, 5), 2), ((3, 6), 4). ((3, 7), 1),$
$((4, 5), 3), ((4, 6), 3), ((4, 7), 1)\}.$

The next-state function shows the transitions between states; each entry contains ((a present state, an input

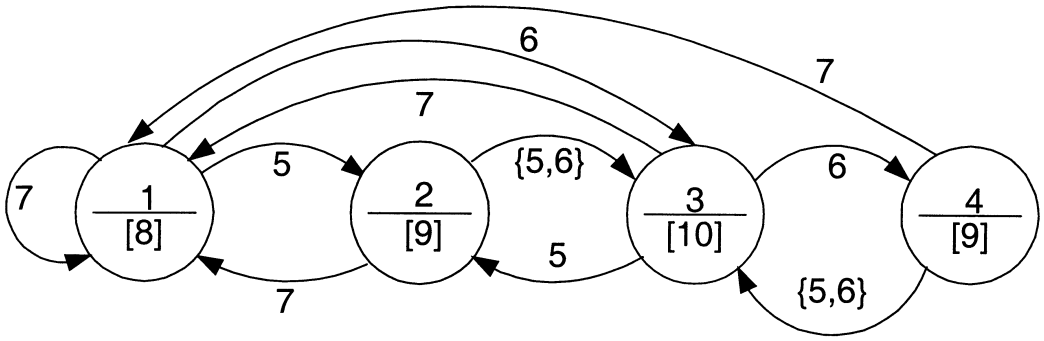

Figure 1. *ZxA*.

value), and a next state); e.g., the first entry above, ((1, 5), 2), says that if the system is in state 1 and it gets an input of 5, then it will go to state 2.

$$RZxA = \{(1, 8), (2, 9), (3, 10), (4, 9)\}.$$

The readout function lists the states and their associated outputs; e.g., if this system is in state 1, then its output is 8.

Figure 1 shows the relationships in the definition of *ZxA*.

Specification of *ZxC*: Let $ZxC = (SZxC, IZxC, OZxC, NZxC, RZxC)$, where

$$\begin{aligned}
SZxC &= \{a, b, c\},\\
IZxC &= \{5, 6, 7\},\\
OZXC &= \{8, 9, 10\},\\
NZxC &= \{((a, 5), b), ((a, 6), c), ((a, 7), a),\\
&\quad ((b, 5), c), ((b, 6), c), ((b, 7), a),\\
&\quad ((c, 5), b), ((c, 6), b), ((c, 7), a)\},\\
RZxC &= \{(a, 8), (b, 9), (c, 10)\}.
\end{aligned}$$

See Figure 2.

Relationship Between *ZxA* and *ZxC*: Systems *ZxA* and *ZxC* look similar. It seems like there has been a renaming and a collapse of two states into one, states 2 and 4 have collapsed into state *b*. This fits the definition of homomorphic images. We can transform *ZxA* into *ZxC* by renaming the states. In *ZxA*, wherever the state is 1 replace it with *a*, where the state is either 2 or 4 replace it with *b*, and where the state is 3 replace it with *c*. Stated formally *ZxC* is a homomorphic image of *ZxA* with the homomorphic state function $HSCA = \{(1, a), (2, b), (3, c), (4, b)\}$. *ZxC* is a minimization of *ZxA*. Digital systems textbooks such as Katz [1994] present many techniques for minimizing systems.

Proof that *ZxC* is a homomorphic image of *ZxA*: *HSCA* is a function: No state of *ZxA* maps to more than one state of *ZxC*. *HSCA* is ONTO: all states of *ZxC* are imaged to at least one state in *ZxA*. Table I shows that parts (4) and (5) of the definition at 4.1 are true. Columns 1 and 2 list all possible combinations of present states (called *x*) and inputs (called *p*). Column 3 shows the next state for *ZxA* for each state-input pair. Column 5 shows the next states for *ZxC*. Now, if our homomorphic state function $HSCA = \{(1, a), (2, b), (3, c), (4, b)\}$ is applied to column 3, we get column 4, which is

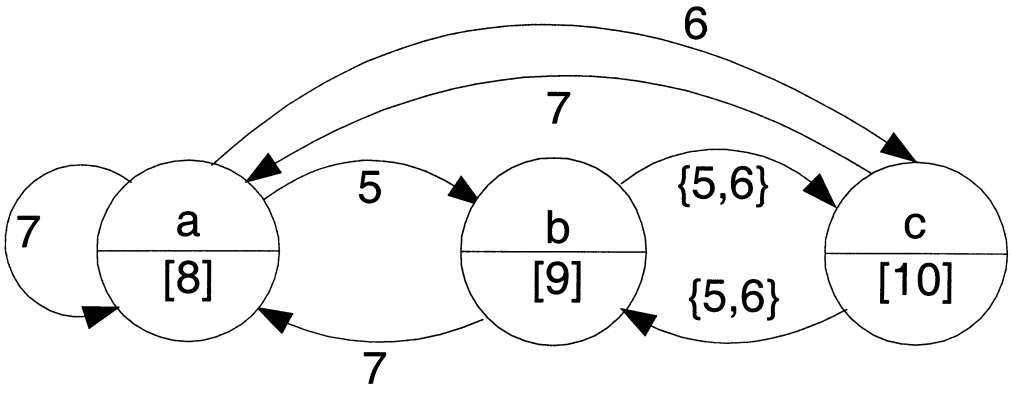

Figure 2. *ZxC*.

Table I. Proof that Z*x*C Is a Homomorphic Image of Z*x*A

1	2	3	4	5	6	7
State, x	Input, p	NZxA(x, p)	HSCA(NZxA(x, p))	NZxC(HSCA(x), p)	RZxA(x)	RZxC(HSCA(x))
1	5	2	b	b	8	8
	6	3	c	c		
	7	1	a	a		
2	5	3	c	c	9	9
	6	3	c	c		
	7	1	a	a		
3	5	2	b	b	10	10
	6	4	b	b		
	7	1	a	a		
4	5	3	c	c	9	9
	6	3	c	c		
	7	1	a	a		

identical to column 5. Furthermore, columns 6 and 7 are identical. Column 6 is derived by applying the system readout function, *RZxA*, to column 1. Column 7 is derived by applying the homomorphic state function, *HSCA*, to column 1 followed by the readout function *RZxC*. Therefore, *ZxC* is a homomorphic image of *ZxA*.

5. THEOREM 2 AND ITS IMPLICATIONS

5.1. Theorem 2

Theorem 2: *State-homomorphism implies I/O equivalence with respect to any pair of initial states of which the one is the homomorphic image of the other and from which the respective systems are completely reachable.*

If

$Z1$ is a state-homomorph of $Z2$ with respect to HS,
every state of $Z1$ is reachable from the initial state $DSZ1$,
every state of $Z2$ is reachable from the initial state $DSZ2$, and
$HS(DSZ2) = DSZ1$,

then $Z1$ and $Z2$ are I/O equivalent with respect to the initial state pair $(DSZ1, DSZ2)$.

Proof: All conditions required by the definition of I/O equivalence with respect to an initial state pair are fulfilled by the hypotheses except condition (5), which is proved as follows:

If $f \in ITZ1 = ITZ2$, then

$$OTZ2(f, DSZ2)(t) = OTZ1(f, HS(DSZ2))(t),$$

by the definition of homomorphism and the corollaries to that definition,

$$= OTZ1(f, DSZ1)(t),$$

since $HS(DSZ2) = DSZ1$.

However, *I/O equivalence with respect to an initial state pair (a homophonic relationship) does* ***not*** *imply a homomorphic relationship between the systems.*

5.2. A Conjectured Theorem

If $Z1$ and $Z2$ are I/O equivalent with respect to the initial state pair $(DSZ1, DSZ2)$, then $Z1$ is a state-homomorph of $Z2$ or $Z2$ is a state-homomorph of $Z1$. This Conjectured Theorem is not true as will be shown with the following counterexample.

Counterexample: Systems *ZxA* and *ZxC* have already been defined. A third system *ZxB* will now be defined with the same state set as *ZxA*: {1, 2, 3, 4}. It will be shown that neither *ZxA* nor *ZxB* is a state-homomorphic image of the other. Then it will be shown that *ZxC* is a state-homomorphic image of both *ZxA* and *ZxB*, with respect to mappings *HSCA* and *HSCB*, respectively. It will be seen that $HSCA(1) = HSCB(1) = a$, and hence, by Theorem 2 proven above, *ZxA* is I/O equivalent to *ZxC* with respect to the initial state pair $(1, a)$, and *ZxB* is I/O equivalent to *ZxC* with respect to the initial state pair $(1, a)$ and hence, by the symmetry and transitivity of I/O equivalence asserted in Theorem 1, *ZxA* and *ZxB* are I/O equivalent with respect to the initial state pair (1, 1).

Hence, it is *not* the case that I/O equivalence with respect to an initial state pair implies a state-homomorphism between the systems.

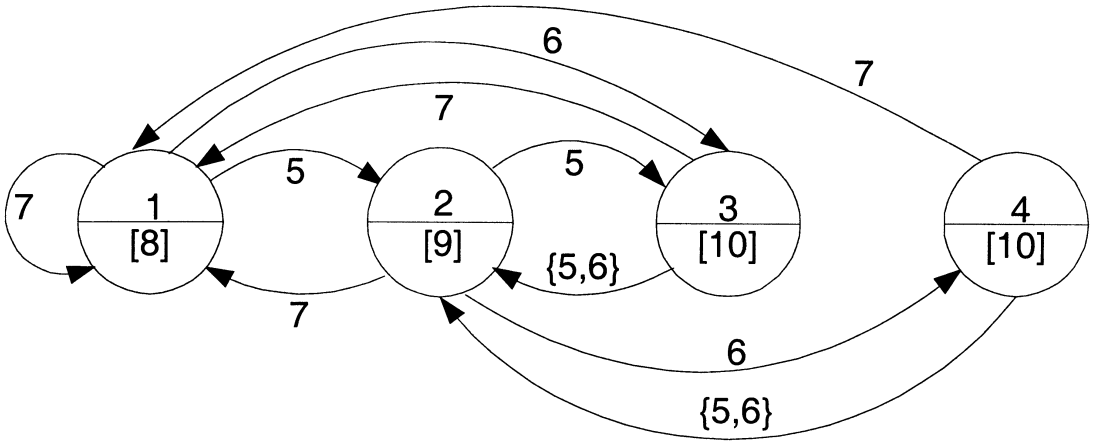

Figure 3. *ZxB*.

Specification of ZxB:

$$ZxB = (SZxB, IZxB, OZxB, NZxB, RZxB),$$

where

$$SZxB = \{1, 2, 3, 4\},$$
$$IZxB = \{5, 6, 7\}$$
$$OZxB = \{8, 9, 10\},$$
$$NZxB = \{((1, 5), 2), ((1, 6), 3), ((1, 7), 1), ((2, 5), 3), ((2, 6), 4), ((2, 7), 1), ((3, 5), 2), ((3, 6), 2), ((3, 7), 1), ((4, 5), 2), ((4, 6), 2), ((4, 7), 1)\},$$
$$RZxB = \{(1, 8), (2, 9), (3, 10), (4, 10)\}.$$

See Figure 3.

I/O Equivalence of *ZxA* and *ZxB*: Table II is a system experiment illustrating that *ZxA* and *ZxB* are I/O equivalent with respect to the initial state pair (1, 1). If both systems are started in state 1 and the indicated input trajectory is applied, then the output trajectories are identical.

Table II shows that *ZxA* and *ZxB* are I/O equivalent with respect to initial states. If this experiment were performed on physical systems, the state trajectories might not be seen, perhaps only the input and output trajectories could be seen. But to prove I/O equivalence with respect to an initial state pair only "power-up" or "start" states need be observable. However, Table II is merely an illustration of I/O equivalence with respect to an initial state pair; it is not a proof, because condition (5) of the definition of I/O equivalence with respect to an initial state pair requires that the output trajectories be the same *for every possible input trajectory* and it is not likely that we can exercise every possible input trajectory. However, the input trajectory of Table II actually is sufficient, because it was constructed using the next-state function and it exercises all state transitions in both systems. The concept of I/O equivalence with respect to an initial state pair is a model-based concept. The proofs and theorems in this paper apply to models of the physical systems. The conclusions drawn are only as good as the models. Finally, we note that proving that two systems are I/O equivalent with respect to initial states does not prove that the systems themselves are equivalent.

There is no state-homomorphism between *ZxA* and *ZxB*. First, let us try to find a function *HSBA* that would make *ZxB* a state-homomorphic image of *ZxA*. The necessity to satisfy condition (5) of the definition at 4.1, namely $RZxA(x) = RZxB(HS(x))$, implies that $HSBA = \{(1, 1), (2, 2), (3, 3), (4, 2)\}$, which is single-valued but not ONTO (state 4 of *ZxB* is missing), or else $HSBA = \{(1, 1), (2, 2), (3, 3), (4, 2), (3, 4)\}$, which is ONTO but not single-valued (state 3 of *ZxA* is mapped to two values of *ZxB*). Therefore, it must be concluded that *ZxB* is not a state-homomorphic image of *ZxA*. The same problems arise in trying to define *HSAB* in order for *ZxA* to be a state-homomorphic image of *ZxB*: Either $HSAB = \{(1, 1), (2, 2), (3, 3), (4, 3)\}$, which is single-valued but not ONTO, or $HSAB = \{(1, 1), (2, 2), (3, 3), (4, 3), (2, 4)\}$, which is ONTO but not single-valued. Therefore, *ZxA* is not a state-homomorphic image of *ZxB*.

ZxC is a state-homomorphic image of both *ZxA* and *ZxB*. We have already shown that *ZxC* is a state-homomorphic image of *ZxA*. Let us now show that *ZxC* is also a state-homomorphic image of *ZxB*. Let $HSCB = \{(1, a), (2, b), (3, c), (4, c)\}$. Table III shows the results of computations proving that $HSCB(NZxB(x, p)) = NZxC(HSCB(x), p)$ (columns 4 and 5 are identical) and $RZxB(x) = RZxC(HSCB(x))$ (columns 6 and 7 are identical) for every $x \in SZxB$ and $p \in IZxB$. See Table III.

Table II. A System Experiment on Z*x*A and Z*x*B Illustrating I/O Equivalence

Time trajectory	Input trajectory, f	State trajectory, STZxA(f, 1)	Output trajectory, OTZxA(f, 1)	State trajectory, STZxB(f, 1)	Output trajectory, OTZxB(f, 1)
0	5	1	8	1	8
1	5	2	9	2	9
2	6	3	10	3	10
3	5	4	9	2	9
4	5	3	10	3	10
5	7	2	9	2	9
6	5	1	8	1	8
7	6	2	9	2	9
8	6	3	10	4	10
9	6	4	9	2	9
10	7	3	10	4	10
11	6	1	8	1	8
12	6	3	10	3	10
13	7	4	9	2	9
14	7	1	8	1	8
15	5	1	8	1	8
16	6	2	9	2	9
17	5	3	10	4	10
18	5	2	9	2	9
19	7	3	10	3	10
20		1	8	1	8

Hence, *ZxC* is a state-homomorphic image of both *ZxA* and *ZxB* with respect to *HSCA* and *HSCB*, respectively. Every state of *ZxC* is reachable from the state $a \in SZxC$ and every state of both *ZxA* and *ZxB* is reachable from the state $1 \in SZxA = SZxB$. Furthermore, $HSCA(1) = HSCB(1) = a$. So, according to Theorem 2, *ZxA* and *ZxB* are both I/O equivalent to *ZxC* with respect to the initial state pair (1, *a*) in both cases. By the symmetry and transitivity of I/O equivalence with respect to an initial state pair as asserted in the corollary to the definition of I/O equivalence with respect to an initial state pair (Theorem 1), *ZxA* and *ZxB* are I/O equivalent with respect to the initial state pair (1, 1). But neither *ZxA* nor *ZxB* is a state-homomorphic image of the other.

6. THE ASSUMPTION OF REUSABILITY MAY BE DANGEROUS

The practical implication of such a possibility can be seen from the following scenario: System functional analysts in Systems Engineering want the functionality of *ZxC*, which is not commercially available. But they know that *ZxA* easily implements *ZxC*. Therefore, they specify that *ZxA* be bought. However, system architects in Systems Engineering buy *ZxB*, because it is cheaper. That's OK, isn't it? After all, *ZxA* and *ZxB* are I/O equivalent with respect to an initial state pair and, furthermore, if either *ZxA* or *ZxB* is implemented, then *ZxC* is automatically implemented. So what difference does it make which of *ZxA* or *ZxB* is implemented?

Later, in the same or another project, Systems Engineering discovers that a certain function can be performed by *ZxA* started in state 4. Wanting to reuse what they already have bought, Systems Engineering specifies *ZxB*, and the system fails because *ZxA* and *ZxB* are not I/O equivalent with respect to the initial state pair (4, 4). As shown in Table II, *ZxA* and *ZxB* have the same input–output behavior, if they are started in the initial state pairs (1, 1), (2, 2), or (3, 3), but not (4, 4).

This substitution of *ZxB* for *ZxA* would work if *ZxB* could be configured and documented to prevent state 4 from ever being an initial state. But, for complex systems, it would be difficult to detect all forbidden initial states, and it would be even harder to make the docu-

Table III. Proof that Z*x*C Is a State Homomorphic Image of Z*x*B

1	2	3	4	5	6	7
State, x	Input, p	NZxB(x, p)	HSCB(NZxB(x, p))	NZxC(HSCB(x), p)	RZxB(x)	RZxC(HSCB(x))
1	5	2	b	b	8	8
	6	3	c	c		
	7	1	a	a		
2	5	3	c	c	9	9
	6	4	c	c		
	7	1	a	a		
3	5	2	b	b	10	10
	6	2	b	b		
	7	1	a	a		
4	5	2	b	b	10	10
	6	2	b	b		
	7	1	a	a		

mentation describing this restriction stay with the system for its entire life cycle.

What *can* be said about two systems *Z*1 and *Z*2 that are I/O equivalent with respect to an initial state pair is that for each system there is another system *Z*1* and *Z*2*, respectively, which is a state-homomorphic image of *Z*1 and *Z*2 respectively, and *Z*1* and *Z*2* are state-isomorphic, or essentially the same system. This will be elaborated in the next three sections.

7. I/O EQUIVALENCE AND MINIMALITY

Definition: If $\{x1, x2\} \subset SZ$, then *states* $x1$ and $x2$ are I/O equivalent if and only if $OTZ(f, x1) = OTZ(f, x2)$ for every $f \in ITZ$. The system Z is minimal if and only if Z has no I/O equivalent states. The set of states of Z equivalent to the state $x1$ is denoted $EQZ(x1)$. The following theorem is well known from the literature (see, e.g., section 9.2 of Katz [1994]).

7.1. Theorem 3 and Examples

Theorem 3: *For every system Z there exists a minimal system denoted MINSY(Z) such that MINSY(Z) is a state-homomorphic image of Z. Furthermore, if Z$ is a minimal system and a state-homomorphic image of Z with respect to EQZ, then Z$ and MINSY(Z) are state-isomorphic.*

If $Z^* = MINSY(Z)$, then Z^* can be defined as follows:

$$SZ^* = \{EQZ(x): x \in SZ\};$$

the states of SZ^* are the equivalence classes made from the states of SZ,

$$IZ^* = IZ,$$
$$OZ_{Z^*} = OZ,$$
$$NZ^* = \{((EQZ(x), p), EQZ(NZ(x, p))): x \in SZ; p \in IZ\},$$
$$RZ^* = \{EQZ(x), RZ(x)): x \in SZ\}.$$

Examples of minimal systems: The system ZxC is minimal since $RZxC(x1) <> RZxC(x2)$ for every $\{x1, x2\} \subset SZxC, x1 <> x2$. So $SZxC$ has no equivalent states, and ZxC is minimal.

8. TIME INVARIANCE AND CAUSALITY CONDITIONS

Every system must satisfy the **time invariance condition**: If

$$f \in ITZ, \quad x \in SZ, \quad t \in TZ, \quad \text{and} \quad r \in TZ,$$

then

$$STZ(f \to r, STZ(f, x)(r))(t) = STZ(f, x)(r + t),$$

where $(f \to r)(t) = f(r + t)$ for every t. The time invariance condition can be interpreted in terms of the results of three system experiments on the system model Z.

Experiment 1: Start the system in the state x, introduce the input trajectory f, and run the system to time r: record the state of the system at that time, $STZ(f, x)(r)$.

Experiment 2: Start the system in the state $STZ(f, x)(r)$, introduce the input trajectory $f \to r$ (this notation means that input trajectory f is time shifted r units to the left) whose first two values are $f \to r(0) = f(r), f \to r(1) = f(r + 1)$, ($f \to r$ takes up where f left off at the end of

Experiment 1), and run the system to time t: record the state of the system at that time, $STZ(f \rightarrow r, STZ(f, x)(r))(t)$.

Experiment 3: Start the system in the state x, introduce the input trajectory f, and run the system to time $r + t$: record the state of the system at that time, $STZ(f, x)(r + t)$.

The state of the system at the end of Experiments 2 and 3 must be the same:

$$STZ(f \rightarrow r, STZ(f, x)(r))(t) = STZ(f, x)(r + t).$$

The time invariance condition is satisfied by all systems that do not change with time, as well as some other systems.

Every system must satisfy the **causality (or nonanticipatory) condition:** If $\{f1, f2\} \in ITZ$, $x \in SZ$, $t \in TZ$, and $f1$ and $f2$ agree at all time values from 0 up to but not necessarily including time t (or any values beyond t), then $STZ(f1, x)(t) = STZ(f2, x)(t)$. The causality condition can be interpreted in terms of the results of two system experiments on the system model Z.

Experiment 1: Start the system in the state x, introduce the input trajectory $f1$, and run the system to time t: record the state of the system at that time, $STZ(f1, x)(t)$.

Experiment 2: Start the system in the state x, introduce the input trajectory $f2$, and run the system to time t: record the state of the system at that time, $STZ(f2, x)(t)$.

If $f1$ and $f2$ have exactly the same values from time 0 up to, but not including time t (and maybe even differ wildly at and after time t), then the states of Z at the end of both experiments must still be the same: $STZ(f1, x)(t) = STZ(f2, x)(t)$.

The causality condition says that a system cannot look into the future.

9. THEOREM 4 AND ITS IMPLICATIONS

Theorem 4: *If two systems have an initial state pair from which each is, respectively, completely reachable, then the two systems are I/O equivalent with respect to that initial state pair if and only if they have state-isomorphic minimizations.*

If $Z1$ and $Z2$ are systems, for $i \in \{1, 2\}$: $DSZi$ is an initial state of Zi such that every state of Zi is reachable from $DSZi$ and $Zi^* = MINSY(Zi)$, such that $DSZi^* = EQZi(DSZi)$, then $Z1$ and $Z2$ are I/O equivalent with respect to the initial state pair $(DSZ1, DSZ2)$, if and only if $Z1^*$ is state-isomorphic to $Z2^*$.

Proof: First, assume that $Z1$ and $Z2$ are I/O equivalent with respect to the initial state pair $(DSZ1, DSZ2)$. Then we must show that $Z1^*$ and $Z2^*$ are state-isomorphic. By Theorems 2 and 3, Zi and Zi^* are I/O equivalent with respect to the initial state pair $(DSZi, DSZi^*)$. By hypothesis, $Z1$ and $Z2$ are I/O equivalent with respect to the initial state pair $(DSZ1, DSZ2)$. By the symmetry and transitivity of I/O equivalence, $Z1^*$ and $Z2^*$ are I/O equivalent with respect to the initial state pair $(DSZ1^*, DSZ2^*)$.

It must be shown that there exists $HS \in FNS(SZ1^*, 1TO1, ONTO, SZ2^*)$ such that for $(x, p) \in SZ1^* IZ1^*$, $HS(NZ1^*(x, p)) = NZ2^*(HS(x), p)$ and $RZ1^*(x) = RZ2^*(HS(x))$.

Let $HS = \{(STZ1^*(f, DSZ1^*)(t), STZ2^*(f, DSZ2^*)(t)): (f, t) \in ITZ1^* >< TSZ1^*\}$, as the most "natural" candidate for the definition of the state-isomorphism because as (f, t) runs over all of $ITZ1 >< TZ1$, $(STZi^*(f, DSZi^*))(t)$ runs over all of SZi^* because of the reachability assumption. Assuming $HS \in FNS(SZ1^*, 1TO1, ONTO, SZ2^*)$: for $(x, p) \in SZ1^* >< IZ1^*$,

$$HS(NZ1^*(x, p)) = HS(NZ1^*(STZ1^*(f, DSZ1^*)(t), p)),$$

where $x = STZ1^*(f, DSZ1^*)(t)$ for some $(f, t) \in ITZ1^* >< TSZ1^*$, where it can be assumed that $f(t) = p$,

$$\begin{aligned}
&= HS(STZ1^*(f, DSZ1^*)(t + 1)) \\
&= STZ2^*(f, DSZ2^*)(t + 1)), \quad \text{by the definition of } HS, \\
&= NZ2^*(STZ2^*(f, DSZ2^*)(t), f(t)) \\
&= NZ2^*(HS(x), p)
\end{aligned}$$

and

$$\begin{aligned}
&RZ1^*(x) \\
&= RZ1^*(STZ1^*(f, DSZ1^*)(t)), \quad \text{where } x = STZ1^*(f, DSZ1^*)(t) \\
&= OTZ1^*(f, DSZ1^*)(t) \\
&= OTZ2^*(f, DSZ2^*)(t)
\end{aligned}$$

since $Z1^*$ and $Z2^*$ are I/O equivalent with respect to the initial state pair $(DSZ1^*, DSZ2^*)$,

$$\begin{aligned}
&= RZ2^*(STZ2^*(f, DSZ2^*)(t) \\
&= RZ2^*(HS(x)).
\end{aligned}$$

Now all that must be shown is that $HS \in FNS(SZ1^*, 1TO1, ONTO, SZ2^*)$. That $HS \in FNS(SZ1^*, SZ2^*)$ is a consequence of the following three observations/deductions corresponding to the conditions required by the definition of a function:

1. $HS \subset SZ1^* >< SZ2^*$.
2. If $x \in SZ1^*$, then, by the reachability hypothesis, there exists $(f, t) \in ITZ1^* >< TSZ1^*$ such that $x = STZ1^*(f, DSZ1^*)(t)$. Then $(x, STZ2^*(f, DSZ2^*)(t)) \in HS$.
3. Now it must be shown that HS is single-valued; let

$$STZ1^*(f1, DSZ1^*)(t1) = x = STZ1^*(f2, DSZ1^*)(t2).$$

It must be shown that, if $y1 = STZ2^*(f1, DSZ2^*)(t1)$ and $y2 = STZ2^*(f2, DSZ2^*)(t2)$, then $y1 = y2$.

$Z2^*$ is minimal, by hypothesis, and therefore, $Z2^*$ has no two I/O equivalent states, by definition. If it can be shown, therefore, that $y1$ and $y2$ are I/O equivalent, then it can be concluded that $y1 = y2$. To this end, it must be shown that for every $f \in ITZ2^*$, $OTZ2^*(f, y1) = OTZ2^*(f, y2)$.

Let $f \in ITZ2^*$ and $t \in TZ2^*$ be arbitrary. It will be shown that $OTZ2^*(f, y1)(t) = OTZ2^*(f, y2)(t)$. Let $f3$ be the concatenation of the input trajectories $f1$ and f at time $t1$, that is,

$$f3 = f1(t), \qquad \text{if } 0 \le t \le t1,$$
$$f3 = f(t - t1), \qquad \text{if } t1 \le t.$$

Let $f4$ be the concatenation of $f2$ and f at time $t2$. Then $f3 \to t1 = f4 \to t2 = f$, that is, the values of both $f3$ and $f4$ at and after times $t1$ and $t2$, respectively, are both given by the function f. This follows from the definition of concatenation. Note that

$$x = STZ1^*(f3, DSZ1^*)(t1)$$
$$= STZ1^*(f4, DSZ1^*)(t2)$$

and

$$y1 = STZ2^*(f3, DSZ2)(t1) \quad \text{and}$$
$$y2 = STZ2^*(f4, DSZ2^*)(t2),$$

by the causality condition because $f3$ is identical to $f1$ to time $t1$ and $f4$ is identical to $f2$ up to time $t2$.

Then

$$OTZ2^*(f, y1)(t) = RZ2^*(STZ2^*(f, STZ2^*(f1, DSZ2^*)(t1))(t)),$$

by the definition of $y1$ and of $OTZ2$,

$$= RZ2^*(STZ2^*(f3, DSZ2^*)(t1 + t)), \text{ by the time invariance condition,}$$
$$= OTZ2^*(f3, DSZ2^*)(t1 + t)$$
$$= OTZ1^*(f3, DSZ1^*)(t1 + t),$$

since $Z2^*$ and $Z1^*$ are I/O equivalent with respect to $DSZ2^*$ and $DSZ1^*$, as proved in the first paragraph of this proof,

$$= RZ1^*(STZ1^*(f3, DSZ1^*)(t1 + t)),$$
by the definition of $OTZ1^*$,
$$= RZ1^*(STZ1^*(f3 \to t1, STZ1^*(f3, DSZ1^*)(t1))(t),$$
by the time invariance condition,
$$= RZ1^*(STZ1^*(f, STZ1^*(f1, DSZ1^*)(t1))(t)),$$

since $f3 \to t1 = f$ and $f3$ prior to time $t1$ is $f1$,

$$= RZ1^*(STZ1^*(f, x)(t)),$$

because $x = STZ1^*(f1, DSZ1^*)(t1)$, by hypothesis,

$$= RZ1^*(STZ1^*(f4 \to t2, STZ1^*(f2, DSZ1^*)(t2))(t)$$

$$= RZ1^*(STZ1^*(f4\ DSZ1^*)(t + t2)),$$

by the definition of $f4$ and the time invariance condition,

$$= OTZ1^*(f4, DSZ1^*)(t2 + t)$$

$$= OTZ2^*(f4, DSZ2^*)(t2 + t),$$

because $Z1^*$ and $Z2^*$ are I/O equivalent for $DSZ1^*$ and $DSZ2^*$,

$$= RZ2^*(STZ2^*(f4 \to t2, STZ2(f4, DSZ2^*)(t2))(t)),$$
by the time invariance condition,
$$= RZ2^*(STZ2^*(f, y2)(t)),$$

by the causality condition since the values of $f4 \to t2$ for time $t2$ and beyond are given by f,

$$OTZ2^*(f, y2)(t).$$

Hence, $OTZ2^*(f, y1)(t) = OTZ2^*(f, y2)(t)$, $y1$ and $y2$ are I/O equivalent and, since by hypothesis $Z2^*$ is minimal, $y1 = y2$, and HS is single-valued. Hence $HS \in FNS(SZ1^*, SZ2^*)$.

HS is ONTO by the reachability hypothesis. Exactly the same argument used to prove that HS is single-valued can be used to prove that HS is 1TO1 by exchanging the roles of $Z1$ and $Z2$. Hence, $Z1^*$ and $Z2^*$ are state-isomorphic, $Z1^*$ is a homomorphic image of $Z1$, $Z2^*$ is a homomorphic image of $Z2$ and Zi^* is completely reachable for $DSZi^*$.

Assume now that $Z1^*$ is state-isomorphic to $Z2^*$ with respect to HS, and $HS(DSZ1^*) = DSZ2^*$. Then $Z1^*$ is completely reachable from $DSZ1^*$ and $Z2^*$ is completely reachable from $DSZ2^*$.

Proof that Zi^* is completely reachable from $DSZi^*$, for $i \in \{1, 2\}$: Since, by hypothesis, Zi is reachable from $DSZi$, then Zi^* is completely reachable from $DSZi^* = EQZ(DSZi)$. Let $y^* \in Zi^*$, $y^* = EQZi(y)$ for $y \in SZi$. Then

$$y = STZi(f, DSZi)(t) \quad \text{for some } f \in ITZi \text{ and } t \in TZi,$$

and

$$y^* = EQZi(y) = EQZi(STZi(f, DSZi)(t)),$$

$$y^* = STZi^*(f, EQZi(DSZi))(t),$$

$$y^* = STZi^*(f, DSZi^*)(t) \qquad \text{QED.}$$

Then $Z1^*$ and $Z2^*$ are I/O equivalent with respect to the pair ($DSZ1^*$, $DSZ2^*$) of initials states by Theorem 2. Hence, $Z1$ is I/O equivalent to $Z1^*$ which is I/O equivalent to $Z2^*$ which is I/O equivalent to $Z2$. Hence, $Z1$ is I/O equivalent to $Z2$.

10. IMPLICATIONS

Suppose an engineer tried to implement the design ZxA in, say, TTL circuitry. How could he or she test this hardware to prove that it did indeed implement the design? If the states were observable, then the engineer could construct an input trajectory (or a set of input trajectories) that exercised all state transitions, apply this input trajectory to ZxA and the TTL circuit, and compare the resulting state trajectories. If they were identical, then the systems would be equivalent. If there were m states, n input possibilities, and k output values, then the lower limit on the length of the input trajectory would be mn, 12 for ZxA. The actual length required of the input trajectory is undecidable: In Table II, it took 20 transitions to exercise all of the state transitions. The upper limit on the length required of the input trajectory is mmn. However, what if the states were not observable? Well, then the engineer could set each state as an initial state and then prove I/O equivalence with respect to an initial state pair as in Table II. This would require mn state transitions. Finally, what if you cannot set the initial states? Then you cannot prove system equivalence!

"The cost of testing a chip using conventional methods will, by 2012, eclipse the cost of fabricating that chip" [Runyon, 1999], and even with this cost, we cannot guarantee that the chip will do what it is supposed to do! The most obvious method of proving conformance of a real system to its design is to show the equivalence of their state trajectories. However, this could be very costly if there were millions of states, and it would be impossible if the states were not observable. Such difficulties have pushed computer designers to abandon input/output testing to prove equivalence. Instead, they now design built-in self-tests for most integrated circuits [Runyon, 1999].

We have mentioned four reasons why engineers might want to prove that two systems are equivalent: desire to reuse existing systems, plans to upgrade systems, pressure to use commercial off the shelf (COTS) systems, and the need to verify that a physical system conforms to its design. Of course, there are many more reasons. One of the best ways to prove system equivalence is to design an input trajectory (or perhaps a set of input trajectories) that exercises every possible state transition, apply it to both systems, and ensure that the two state trajectories are identical. However, this might be impossible, because the states are not observable: or it might be too expensive, due to the large number of states. Here are a few techniques that have been used in lieu of proving system equivalence: (1) Create multiple reset states and prove I/O equivalence with respect to an initial state pair for all of them. (2) Build an observer to estimate the system states. (3) Design built-in self-tests. (4) Add extra outputs so that the states can be identified by examining the outputs.

A very common technique for describing the desired behavior of a system is to describe acceptable input/output trajectories for the system. Such descriptions of input and output behavior as functions of time are variously called trajectories, behavioral scenarios, use cases, threads, operational scenarios, logistics, functionality, test vectors, sequence diagrams, or interaction diagrams. When using such techniques the following question often arises, "How do you know when you have enough scenarios?" The answer seems to be *never*. Because merely looking at input/output behavior can never guarantee correct system behavior: we must be able to see the states.

We have shown that systems should be designed so that they have the needed functionality rather than the desired input/output behavior. In this paper, we used only one method for doing the functional design: finite state machines. There are of course, many other methods. See Bahill et al. [1998] for an overview.

Computer engineers have struggled to minimize circuits in order to reduce the number of flip-flops or even diodes. But currently, with these components being so inexpensive, it does not seem to be worth the effort to minimize state machines anymore. In this paper, we have shown a new reason to minimize state machines: It will make it easier to prove equivalence and therefore facilitate reuse.

Nowadays, all systems are supposed to be designed so that they are better, faster, and cheaper. The mantra for doing this seems to be reuse. Databases are being put on the Internet for widespread use. Integrated Product Development teams continually try to reuse parts of previous teams' efforts. Object libraries are created to encourage reuse. However, inadequate testing of reused software caused the failure of the Ariane 5 missile [Kunzig, 1997]. And reuse in object-oriented software is cautioned by the following infamous Internet myth. The Australian military's virtual reality simulators for helicopter combat training have detailed landscapes of the Australian Outback. To increase realism, they were asked to add mobs of kangaroos. Being efficient programmers, they reused some code originally designed to model the behavior of infantry detachments under the same stimuli. They changed the mapped icon from a soldier to a kangaroo, and increased the figures' speed of movement. In an early demonstration, a helicopter in

low-level flight buzzed the virtual kangaroos. The kangaroos scattered, as predicted, and the visitors nodded appreciatively. Then they did a double take as the kangaroos reappeared from behind a hill and launched a barrage of Stinger missiles at the hapless helicopter. Apparently, the programmers did not remove the launch weapons functionality from the objects. The lesson? The programmers got the behavior they wanted: retreat when a helicopter approaches, but that behavior was not a complete description of the functionality of that object. Reference: http://www.sie.arizona.edu/sysengr/OOSE/ kangaroo.html.

11. SOMETIMES IT WORKS

Why is it so enticing to use input/output behavior to prove system equivalence? Because *sometimes* it works!

In the field of Digital Design (e.g., computer design), the two basic types of systems are called combinational and sequential [Katz, 1994]. In the field of cybernetics, von Foerster [1982] called them, respectively, trivial and nontrivial systems. In combinational systems, the output depends only on the present inputs, whereas, in sequential systems, the output depends on the sequence of previous inputs. Combinational problems can be modeled and implemented as sequential systems. But not vice versa. Note: sequential system, state machine, and nontrivial system are usually considered to be synonymous phrases.

Consider a household three-way light system with one light and two switches, one at each end of a hallway. Define the inputs to be the position of the switches {up, down}. We can create the following sequential model for this system:

$$Z = (SZ, IZ, OZ, NZ, RZ),$$

where

SZ = {($S1$up – $S2$up), ($S1$up – $S2$down), ($S1$down – $S2$up), ($S1$down – $S2$down)},

$I1Z$ = {up, down},

$I2Z$ = {up, down},

OZ = {on, off},

NZ = {((($S1$up – $S2$up), (up, down)), ($S1$up – $S2$down)), ((($S1$up – $S2$up), (down, up)), ($S1$down – $S2$up)), ((($S1$up – $S2$down), (up, up)), ($S1$up – $S2$up)), ((($S1$up – $S2$down), (down, down)), ($S1$down – $S2$down)), ((($S1$down – $S2$down), (up, down)), ($S1$up – $S2$down)), ((($S1$down – $S2$down), (down, up)), ($S1$down – $S2$up)), ((($S1$down – $S2$up), (down, down)), ($S1$down – $S2$down)), ((($S1$down – $S2$up), (up, up)), ($S1$up – $S2$up))}.

For simplicity, input combinations that do not produce a change of state are not shown; also, simultaneous changes of both switches are not shown.

RZ = {(($S1$up – $S2$up), on), (($S1$up – $S2$down), off), (($S1$down – $S2$up), off), (($S1$down – $S2$down), on)}.

Given the present state and an input trajectory, we can compute the state trajectory. Therefore, the three-way light system can be modeled as a sequential system, and it can be implemented with flip-flops.

However, in actuality, the three-way light system is only a combinational problem: it does not need a sequential solution. The present state depends only on the present positions of the switches. The system can be modeled with the truth table of Table IV.

From this Table IV, we can derive the following equation:

$$Light = \overline{S1} \bullet \overline{S2} + S1 \bullet S2.$$

This equation (or model) can be implemented using only AND and OR gates. This combinational system implementation is simpler than the sequential system implementation presented above. Furthermore, the equivalence of this combinational model with the physical system can be proven using only input/output behavior.

Contrast this three-way light system with a three-way lamp that can be off, on with 50 watts, on with 100 watts, or on with 150 watts. Turn the switch 90°, and the lamp is on dimly. Turn it again, and the lamp is on with medium brightness. Turn it a third time, and the lamp is on brightly. A final turn, turns the lamp off. The behavior of this system clearly depends on its previous state: It is a sequential problem and requires a sequential solution. The equivalence of the sequential solution (model) with the physical system cannot be proven using only input/output behavior.

Therefore, in the real world, there are combinational problems that should have combinational solutions; however, they could also have sequential solutions. An engineer *can* prove the equivalence of these combinational problems using only input/output behavior. However, real-world sequential problems must have

Table IV. A Truth Table

S2	S1	Light
down	down	on
down	up	off
up	down	off
up	up	on

sequential solutions. And the equivalence of these solutions cannot be proven using only input/output behavior. These same observations have been made previously by von Foerster [1982].

In this paper, we have shown that an engineer cannot prove equivalence of two sequential systems using only input/output behavior. But there are many engineers who say that they have proven the equivalence two systems using only input/output behavior. We suggest that their systems were merely combinatorial problems. And for combinatorial problems, equivalence can be proven using only input/output behavior.

12. CONCLUSIONS

If $Z1$ implements $Z2$, and $Z3$ is a homomorphic image of $Z2$, then $Z1$ also implements $Z3$. The minimization, say $Z2^*$, of a system $Z2$ is a state-homomorphic image of $Z2$, so, if $Z1$ implements $Z2$, then $Z1$ also implements $Z2^*$.

Two systems are I/O equivalent with respect to an initial state pair if and only if they have state-isomorphic minimizations. However, two systems that are I/O equivalent with respect to an initial state pair are not necessarily homomorphic images. Hence, a system that implements the one may not implement the other. That is, the fact that $Z1$ is an implementation of $Z2^*$ does not imply that $Z1$ is also an implementation of $Z2$. To assume otherwise can lead to disaster. It is the responsibility of systems engineers assigned to system functional analysis to consider the I/O requirement and the performance requirement and to specify functional system designs for implementation. It is the responsibility of systems engineers assigned to system architecture to consider the technology requirement and the cost requirement and to specify buildable system designs to implement those *exact* functional system designs.

In this paper we have shown that (1) proving that two systems are I/O equivalent with respect to an initial state pair does not prove that the systems are equivalent (homophonic images are not homomorphic images), (2) the assumption of reusability may be dangerous, and (3) if two systems are input/output equivalent with respect to an initial state pair, then either they are isomorphs or at least one of them is not minimal and their minimizations are isomorphic images.

These conclusions should be important to senior systems engineers and program managers, because these people influence the selection the system architecture. If the system architecture prescribes a system that has the desired input/output behavior, rather than the needed functionality, then the system may fail.

APPENDIX

Notation: References (i.e., A1.*) are to paragraphs in Appendix-1 of Wymore [1993].

- $+$ the addition operator (A1.114, A1.115, A1.201)
- $\rightarrow$ the translation or time shift operator (A1.284)
- $\in$ the "is an element of" relation (A1.2, A1.126)
- $<>$ the "not equals" relation (A1.16, A1.128)
- $=$ the "equals" relation (A1.2, A1.16, A1.128)
- $\cap$ the intersection of two sets (A1.74 to A1.91)
- $\circ$ the function composition operator (A1.269)
- $\subset$ the subset relation (A1.3, A1.133)
- $FNS(SZ1, SZ2)$ the set of all functions that are defined on the sets $SZ1$ and $SZ2$
- $FNS(A, 1TO1, B)$ functions that map one value in set A to one value in set B (A1.222)
- $FNS(A, ONTO, B)$ functions where each element of B is imaged by at least one element of A (A1.224 to A1.226)
- $FNS(A, 1TO1, ONTO, B)$ functions relating A to B that are 1TO1 and ONTO
- ID the identity function (A1.165 to A1.168)
- f a symbol for some particular input trajectory
- p a symbol for some particular value of an input
- x a symbol for some particular value of a state
- y a symbol for some particular value of a next-state
- t a symbol for some particular value of time
- r a symbol for another particular value of time

REFERENCES

A.T. Bahill, M. Alford, K. Bharathan, J. Clymer, D.L. Dean, J. Duke, G. Hill, E. LaBudde, E. Taipale, and A. W. Wymore, The Design-Methods Comparison Project, IEEE Trans Syst, Man, Cybernet Part C: Appl Rev, 28 (1998), 80–103; also at http://www.sie.arizona.edu/sysengr/methods2.

W.L. Chapman, A.T. Bahill, and A.W. Wymore, Engineering modeling and design, CRC Press, Boca Raton, FL, 1992.

R.H. Katz, Contemporary logic design, Benjamin/Cummings, Redwood City, CA, 1994.

R. Kunzig, Europe's dream, Discover, 18 (May 1997), 96–103.

S. Runyon, Testing big chips becomes an internal affair, IEEE Spectrum 36(4) (1999), 49–55.

H. von Foerster, Observing systems, Intersystems, Seaside, 1982.

A.W. Wymore, Model-based systems engineering, CRC Press, Boca Raton, FL, 1993.

Wayne Wymore earned B.S. and M.S. degrees at Iowa State University, and the Ph.D. at the University of Wisconsin, Madison, all in mathematics. He is Professor Emeritus of Systems and Industrial Engineering at the University of Arizona, where he was the first Chairman of the Department of Systems Engineering and first Director of the Computing Center. While managing, teaching, researching, and consulting internationally (50 organizations in 13 countries in 21 fields of application), he authored *A Mathematical Theory of Systems Engineering: The Elements,* 1967, *Systems Engineering Methodology for Interdisciplinary Teams,* 1976, and *Model-Based Systems Engineering,* 1993. *System Functional Analysis and System Design, Phase 2 of Model-Based Systems Engineering* is forthcoming. He is a Fellow of INCOSE.

Terry Bahill is a Professor of Systems Engineering at the University of Arizona in Tucson. He received his Ph.D. in electrical engineering and computer science from the University of California, Berkeley, in 1975. Bahill has worked with Hughes Missile Systems in Tucson, Sandia Laboratories in Albuquerque, Lockheed Martin Tactical Defense Systems in Eagan, Minnesota, Boeing Information, Space and Defense Systems in Kent, Washington, and Idaho National Engineering and Environmental Laboratory in Idaho Falls. For these companies he presented seminars on Systems Engineering, worked on product development teams, and helped them describe their Systems Engineering Process. He holds a U.S. Patent for the Bat Chooser, a system that computes the Ideal Bat Weight for individual baseball and softball batters. He is Editor of the CRC Press Series on Systems Engineering. He is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE) and of the International Council on Systems Engineering (INCOSE). He is Chair of the INCOSE Fellows Selection Committee.