Unwinding Forward Correctability

Jonathan K. Millen
The MITRE Corporation
Bedford, MA 01730

Abstract

A state-machine formulation is given for forward correctability in event systems, to provide a type of unwinding result for this information flow security property. We show also how regular expression notation provides an easy mechanical tool for verifying forward correctability for small systems, which is necessary for the effective presentation of examples and exercises.

1 Background

1.1 Introduction

The theory of secure information flow in computer systems began in the mid-1970's in response to Lampson's note on the confinement problem, which introduced the concept of "covert channels." Early articles [Den76, Mil76] presented techniques for "flow analysis," to be applied to application programs or operating system kernel specifications, respectively, to determine whether information labelled using a lattice of security levels could be leaked out at lower levels. During that period there were several papers investigating the theory of information flow in programs, its relation to logic, and its application to the design of particular covert channel analysis techniques and software tools.

In the early 1980's came non-interference, originating at SRI as a theoretical foundation for the HDM/Special flow tool [GoMe82]; it was actually a generalization of an earlier SRI model [FLR77]. Based on an abstract state-transition machine model, non-interference was an elegant concept that spawned many interpretations and subsequent advances. The theory is deep enough so that it underlies both access control and covert channel analysis, and does not distinguish between them. It has been applied as a security analysis technique to formal specifications of an operating system kernel [HKMY87]. An important theoretical advance that assisted in such applications was the "unwinding theorem," which permits a noninterference policy to be expressed equivalently as a test on transition specifications rather than traces [GoMe84]. A particularly lucid rephrasing of that result was given by Rushby [Rus85].

Noninterference as originally defined applies only to deterministic systems. Other information flow models were introduced for application to systems that were not adequately represented by the deterministic SRI model. Nondeterministic system models are divided roughly into two types, probabilistic and combinatoric, depending on whether probability distributions are assumed known for the system's nondeterministic choices. A discussion of probabilistic noninterference is given by Gray [Gry92]. The corresponding definitions of information flow can be related to Shannon's classical information theory. Even the combinatoric models and noninterference itself can be seen as special cases of an information-theoretic inter-pretation [Mil87], although they can be expressed more directly using Sutherland's nondeducibility, a functional independence concept [Sut86].

A significant advance in the nondeterministic/combinatoric branch of model development was McCullough's "hookup" security, later called "restrictiveness." Extending noninterference to a nondeterministic event-system context, it was composable. That is, the result of interconnecting two restrictive systems (consistent with input/output labels) is restrictive. As a trace model, it still needed, and lacked, an unwinding theorem. Although an approximately equivalent version of it was given for state machines, restrictiveness did not have a satisfactory unwinding result.

It was argued in [Mil90] that a useful definition of information security should be at least as strong as nondeducibility on inputs (nondeducibility of higher-level inputs from low-level observations), and it also should be composable, i.e., preserved in an interconnection of secure systems. Many of the examples that illustrate the inadequacy of nondeducibility on inputs alone as a satisfactory definition can be recast as examples of compositions of secure systems such that the result is not nondeducibility secure.

The pivotal result in the nondeterministic/combinatoric branch of model development was McCullough's "hookup" security, later called "restrictiveness." Extending noninterference to a nondetermininstic event-system context, it had the sought property of composability. As a trace model, it still needed, and lacked, an unwinding theorem. Although an approximately equivalent version of it was given for state machines, restrictiveness did not have a satisfactory unwinding result.

In 1988, Johnson and Thayer discovered a security condition that was similar to restrictiveness and retained its essential properties, namely that it is at least as strong as nondeducibility and it is composable [JoTh88]. But their security condition, called forward correctability, was an improvement because it was weaker i.e., it was satisfied by more systems. Like the original form of restrictiveness, it was defined for event systems. We show in this paper that any event system can be expressed in state-machine form. The statement of forward correctability for such state machines is presented, and we shall see that it has most, though not all, of the required properties of a useful unwinding result.

Regular expression notation is a traditional way of specifying finite-state machines and supporting proofs of their properties. We show here how this notation can be used to describe event systems and test nondeducibility and forward correctability, at least for small examples.

1.2 Event Systems

1.2.1 Non-Serializable Systems

Event system or other trace models are usually justified by the need to represent nondeterministic systems. Determinism seems to call for *atomicity*, in the following sense: When a user submits a command to the system, the state change should depend only on the state at the time the input was received, and on the input itself. While there is in fact some time between the system call and the return to the user, it is required for atomicity that no other user's actions (as well as other events such as I/O interrupts) can affect the new state, at least to the extent that it will ever become visible to the user. Also, the transient intermediate states between receiving the input and the return should not be visible to any process; subsequent inputs should not be accepted until a well-defined system state is achieved.

Strict atomicity of transitions is actually not necessary to model a system deterministically; it is sufficient to have serializability. A system is *serializable* if, for every concurrent execution of system calls, there is some possible disjoint ordering of those calls that would produce the same net result. A system with this property is equivalent to one with atomic transitions for modelling purposes.

Figure 1: A non-serializable system

However, not all systems are serializable. Here is a simple example of one that is not. Suppose that there is a system call that sets two entries $T1, T2$ in a global system table, one after the other, to the identity of the processor that executes it. If this system call is executed concurrently on two processors A and B, and the table is not locked as part of the operation, it is possible (due to different clock speeds, interrupts, etc.) that the sequence of execution will be: $T1 := B$, $T1 := A$, $T2 := A$, $T2 := B$. Thus the table will wind up with $T1$ set to "A" and $T2$ set to "B". This is inconsistent with either serialized result, which would leave both entries with the same value, either A or B. The setup is illustrated in Figure 1.

1.2.2 Event Systems

If a system is not serializable, or not representable deterministically for any other reason, it is not possible to model it accurately using the Goguen-Meseguer system model. There are models that handle a limited form of nondeterminism, but are still incapable of representing non-serializable systems. To represent a non-serializable system, we need to both (1) use a finer granularity than a system call, yet (2) retain the sequencing of the components of a system call. These objectives are met by trace models that divide up a system call into input, internal, and output events, and which have a general way of expressing legal sequences of events. One of the earliest and most influential of these models is Hoare's CSP [Hoa85], and some work has been done to express noninterference in the context of CSP and related models [Fol87, Gra93].

We will use the event system model introduced by McCullough [McC88a]. Event systems are expressible in CSP, but they are simpler conceptually, they can be described without the trappings of the CSP algebra, and the results we wish to extend here were presented in an event system context.

1.2.3 Definition: event system

An event system is a quadruple (E, I, O, T) where E is a finite set of *events*, I and O are disjoint subsets of events called *inputs* and *outputs*, respectively, and T is the set of *traces*, some subset of the possible finite event sequences. Events in $E - (I \cup O)$ are called *internal*. There are two restrictions on T:

(1) *event separability*: if $\sigma\tau \in T$ then $\sigma \in T$ (any prefix of a trace is a trace)

(2) *input totality*: if $\sigma \in T$ and $i \in I$ then $\sigma i \in T$ (a trace may be extended by any input).

(Note that concatenation of sequences is represented by juxtaposition, and we do not distinguish between individual events and a sequence consisting of a single event.)

The principal departure from CSP in McCullough's model is input totality. This says that any input must be acceptable at any time. In CSP, inputs are not distinguished, so there is no general rule that all inputs must be accepted after every trace. When an input is "accepted" by an event system, however, all it means is that the input is recorded in the trace; it does not necessarily have any effect on the state of the system (it might be ignored).

1.2.4 Regular Expressions

Although one of the reasons for using event systems rather than CSP is to avoid unfamiliar algebraic notation, there is an old, well-established notation for dealing with sets of sequences. It will come in handy for expressing examples. A term in regular expression notation denotes a set of sequences over a given alphabet; in our case the alphabet is E. A singleton set is represented by its element (a sequence); a single-event sequence is represented by the event. For two terms p and q,

$$pq = \{\sigma\tau | \sigma \in p \wedge \tau \in q\}$$

$$p + q = p \cup q = \{\sigma | \sigma \in p \vee \sigma \in q\}$$

$$p^* = \Lambda + p + pp + ppp + ...$$

where Λ is the empty sequence. Thus, p^* is the set of finite concatenations of elements of p. Since terms represent sets of sequences, the usual set operations like "$\cap$" and "$\cup$" can be used, though we usually prefer "$+$" to "$\cup$" in this context. We use "$-$" for the set difference operator.

Regular expressions over a finite alphabet such as E are capable of expressing only regular languages or sets, which are recognizable by finite-state machines. However, we may use the regular expression operators on variables representing arbitrary sets of sequences. We mention this as a caution so that the reader will not assume that an expression such as p^* necessarily represents a regular set, since p might not be a regular set itself.

We introduce here two operations on event sequences that play an important role in definitions of security. They are not native to regular expressions; in fact, they are closely related to two operations in CSP, and we will use the CSP notation for them.

The first is restriction. Suppose σ is an event sequence and A is a set of events. Then $\sigma|A$, the *restriction* of σ to A, is the subsequence of σ consisting of elements of A. Thus, if $A = \{a, c\}$ and $\sigma = abc$, then $\sigma|A = ac$. Restriction may be applied to sets of sequences elementwise. That is, if $p \subseteq E^*$ then $p|A = \{(\sigma|A)|\sigma \in p\}$.

The *purge* of A from p, written $p \setminus A$, is the restriction of p to non-A events. That is,

$$p \setminus A = p|(E - A).$$

1.3 Multilevel Security

Since we are concerned here with multilevel security, we begin by adding a sensitivity level assignment to events in an event system. This gives us a labelled event system.

1.3.1 Definition: labelled event system

A labelled event system is a triple (S, L, λ) where:

$S = (E, I, O, T)$ is an event system,

L is a lattice (of sensitivity levels)

$\lambda : E \to L$ (a level assignment function).

Sometimes we will refer to S itself as a labelled event system, with the existence of a level lattice and assignment function implied.

1.3.2 Notation

Given any sensitivity level $s \in L$, one can partition E into two sets

$$\underline{E} = \{e \in E|\lambda(e) \leq s\}$$

called the "low" events, and

$$\overline{E} = E - \underline{E},$$

called the "high" events. This notation is ambiguous because it does not indicate the level s used to define it, but that level will ordinarily be clear from context.

Also, for any set of sequences $p \subseteq E^*$, we define $\underline{p} = p|\underline{E}$ and $\overline{p} = p|\overline{E}$.

1.3.3 Forward Correctability

The first definition of information flow security that was shown to be composable is McCullough's restrictiveness (originally called "hook-up security"). Like noninterference, restrictiveness is defined to apply in a broader context than multilevel security, but we are concerned here only with its specialization to multilevel security. We will not present restrictiveness here because it has been superseded by a property that is better, in the sense that, while it is composable and stronger than nondeducibility on inputs, it is weaker than restrictiveness. This property is forward correctability, due to Johnson and Thayer [JoTh88].

To define forward correctability, we must first define simple perturbations and corrections. These definitions are in the context of some given event system and some level s to differentiate "high" and "low."

1.3.4 Definition: simple perturbation

δ is a *simple perturbation* of α before γ if there exists a high input $x \in \overline{I}$ such that for some β:

(1) $\alpha = \beta\gamma$ and $\delta = \beta x \gamma$ (x is inserted into α before γ) or

(2) $\alpha = \beta x \gamma$ and $\delta = \beta\gamma$ (x is deleted from α before γ).

1.3.5 Definition: correction

α' is a correction of δ in γ if $\delta = \phi\gamma$ and $\alpha' = \phi\gamma'$ such that: $\underline{\gamma'} = \underline{\gamma}$ and $\gamma|I = \gamma'|I$.

Thus, a correction in γ is a modification of high non-inputs in γ.

1.3.6 Definition: forwardly correctable

A labelled event system is *forwardly correctable* if, for all levels $s \in L$, for all traces $\alpha \in T$, and for all low inputs $a \in \underline{I}$,

if (δ is a simple perturbation of α before γ or δ is a simple perturbation of α before $a\gamma$)

and $\gamma|\overline{I} = L$ (γ contains no high inputs),

then there exists a correction α' of δ in γ such that $\alpha' \in T$.

This definition would be turned into the equivalent of restrictiveness if the low input a could be replaced in the definition by an arbitrarily long sequence of low inputs. Johnson and Thayer showed that forward correctability is strictly weaker than restrictiveness, by exhibiting an event system that is forwardly correctable but not restrictive. They also proved that forward correctability is composable.

The remaining goal for forward correctability is an wnwinding theorem. We will present a result that has the form of an unwinding result, though it is not idea from a practical point of view. The first step is to show how event systems can be represented as state-transition machines.

2 The State-Machine Approach

2.1 An Unwinding Result

Unwinding addresses the need for a practical way to verify security of a system, working from the code or a higher-level formal specification of it. Both noninterference and nondeducibility are stated in terms of traces. While some systems may be specified directly in terms of their possible traces (see, e.g., [McL92]), program code and many other formal specification approaches assume a state-transition model, and specify individual state transitions. It is helpful, therefore, to express a security property as a condition on individual state transitions. A theorem stating the equivalence of a trace-based security condition with a transition-based security condition is called an unwinding theorem. An unwinding result for noninterference has been available since 1984 [GoMe84].

2.1.1 Event System Acceptors

Given an event system, there is a state-transition machine (or just "state machine") that acts as an acceptor for traces.

2.1.2 Definition: acceptor

An acceptor is a tuple $M = (Q, I, A, \delta, q_0)$ where Q is a set of states, I is a set of inputs, $A \subseteq Q$ is the set of accept states, $\delta : Q \times I \to Q$ is the next-state function, and $q_0 \in Q$ is the initial state.

If δ is extended as usual to $Q \times I^*$, we will say that an input sequence σ is *accepted* if $\delta(q_0, \sigma) \in A$. Note that we have not required an acceptor to have only a finite number of states.

To construct an acceptor for an event system, we begin by introducing an operator on event sequences that is familiar both from a regular expression context and from CSP. We will use the CSP notation for it.

2.1.3 Definition: /

If $q \subseteq E^*$ is a set of event sequences, and $\sigma \in E^*$,

$$q/\sigma = \{\tau | \sigma\tau \in q\}.$$

In CSP, q/σ is called "q after σ." For regular expressions, this is the Brzozowski derivative [Brz64]. It strips σ off all elements of q that had σ as a prefix, leaving the tails. The derivative operation has some obvious elementary properties which we will use without proof, such as $(q/\sigma)/\tau = q/(\sigma\tau)$.

2.1.4 Definition: event system acceptor

M is an acceptor for an event system $S = (E, I, O, T)$ if

$$M = (Q, E, Q - \{\emptyset\}, \delta, T)$$

where $\delta(q, e) = q/e$, and $Q = \{T/\sigma | \sigma \in E^*\}$.

We are begging the question by calling M the acceptor for S. It is obvious that M is an acceptor, but we must prove that it accepts exactly the traces of S.

2.1.5 Theorem

If M is the acceptor for S, M accepts σ iff $\sigma \in T$.

Proof Simply note that $T/\sigma \neq \emptyset$ if and only if σ is a prefix of some trace in T.$\blacksquare$

2.1.6 s-Equivalence

The unwinding-like result for forward correctability will be similar in that we begin by identifying an equivalence relation on the states of the acceptor. Two states will be considered s-equivalent if they have the same projection under π, defined as the low view of high-input-free continuations:

2.1.7 Definition: projection

$\pi(q) = (q \cap (E - \overline{I})^*)|\underline{E}$.

Although π depends on the level s, we are continuing our practice of leaving s implicit.

2.1.8 Definition: s-equivalence

q is *s-equivalent* to q', written $q \sim q'$, if $\pi(q) = \pi(q')$.

It will come in handy later to observe that π is a regular expression homomorphism, that is:

2.1.9 Proposition

$\pi(p + q) = \pi(p) + \pi(q)$ and $\pi(pq) = \pi(p)\pi(q)$.

This implies also that $\pi(p^*) = (\pi(p))^*, \pi(\Lambda) = \Lambda$, and $\pi(\emptyset) = \emptyset$.

2.1.10 The Unwinding Theorem

The definition for s-equivalence looks peculiar, but it is justified by the following unwinding-like theorem. What keeps it from being a truly satisfactory unwinding theorem is the fact that it includes a condition on two successive transitions, rather than one. But this is still far better than dealing with event sequences of arbitrary length. Another drawback is that the state equivalence used is much less intuitive than that for noninterference, though we will show how to deal with it for some small examples.

2.1.11 Theorem

If M is the acceptor for the labelled event system S, S is forwardly correctable iff for each level s, high input $x \in \overline{I}$, low input $a \in \underline{I}$, and state $q \in Q$,

$$q/x \sim q^{and}(q/x)/a \sim q/a.$$

Proof This result is actually quite mechanically straightforward. It is simply a translation of the definition of forward correctability into the acceptor context. The proof has four parts, corresponding to the two directions of implication, and either the two state equivalences above, or whether or not the low input is present before γ in the definition of forward correctability. We will prove one part below in detail, and omit the rest.

Suppose S is forwardly correctable. Assume $x \in \overline{I}$ and $q \in Q$. We will show that $q/x \sim q$. This is trivial if $q = \emptyset$, so assume $q \neq \emptyset$.

We must show that $\pi(q/x) = \pi(q)$, i.e., that $(q/x \cap (E - \overline{I})^*)|\underline{E} = (q \cap (E - \overline{I})^*)|\underline{E}$. We show first that the left side is a subset of the right side. If the left side is empty, we are done; otherwise, produce $\gamma \in q/x \cap (E - \overline{I})^*$. We must find $\gamma' \in q \cap (E - \overline{I})^*$ with $\underline{\gamma'} = \underline{\gamma}$.

For some $\sigma \in T$, we have $q = T/\sigma$. Then $q/x = T/\sigma/x = T/(\sigma x)$, so $\sigma x \gamma \in T$, and $\sigma\gamma$ is a simple perturbation of $\sigma x \gamma$ before γ. Furthermore, since $\gamma \in (E - \overline{I})^*$, γ has no high inputs. By forward correctability, there exists γ' such that $\sigma\gamma' \in T, \underline{\gamma'} = \underline{\gamma}$, and $\gamma'|I = \gamma|I$. This means that $\gamma' \in q$ and γ' also has no high inputs. But then $\gamma' \in q \cap (E - \overline{I})^*$, and we are done.

To show that that $(q \cap (E - \overline{I})^*)|\underline{E} \subseteq (q/x \cap (E - \overline{I})^*)|\underline{E}$, the argument is essentially the same because deleting x is also a simple perturbation.$\blacksquare$

2.2 Testing Forward Correctability

The motivation for unwinding results is to provide a practical way of testing a security condition, in this case forward correctability. It turns out that if an event system acceptor is small, it is very easy to test s-equivalence. Thus, this unwinding result can be useful at least as a way of checking small illustrative examples.

Acceptors for small examples are usually representable without difficulty either in the form of a regular expression for the set of traces, or as a state graph. In either case it is straightforward to express each state in regular expression form. We will show how to find the projection $\pi(q)$ of a state q given as a regular expression.

2.2.1 Calculating Projections of Regular Expressions

First, we observe that if $A \subseteq E$, then $q \cap (E - A)^*$ may be obtained from the regular expression for q by *replace each occurrence of an event in A with* $\emptyset$. This has the effect of eliminating those sequences in q having any occurrence of an event in A. To find $\pi(q)$, we will first replace high inputs in q with $\emptyset$.

For example, (a + bc + c)* is the set of all sequences of a's, ab's and c's, such as a, ab, c, aa, cac, abc, etc. If E = {a, b, c}, the intersection (a + ab + c)* $\cap$ (E - {a})* may be found easily if we replace a by $\emptyset$ in (a + ab + c)*; the result is $(\emptyset + \emptyset + \mathrm{c})^* = \mathrm{c}^*$.

Second, we observe that $q|A$ may be obtained from the regular expression for q by *replacing each occurrence of an event in E - A with* Λ. This deletes elements of A from each sequence in q. Note, in particular, that $\underline{q} = q|\underline{E} = q(E - \overline{E})$, so $\underline{q}$ may be found by replacing high events in q with Λ.

With (a + ab + c)* again, suppose that b is low and a and c are high. Note that

$$\underline{(\mathrm{a} + \mathrm{ab} + \mathrm{c})^*} = (\mathrm{a} + \mathrm{ab} + \mathrm{c})^*|\{\mathrm{b}\}.$$

The result is found by replacing occurrences of a and c with Λ. This gives $(\Lambda + \Lambda\mathrm{b} + \Lambda)^* = (\Lambda + \mathrm{b})^* = \mathrm{b}^*$.

Putting the two examples together, suppose that a and c are high and b is low, a is an input, and b and c are outputs. Thus, $\underline{E}$ = {b} and $\overline{I}$ = {a}. Let q = (a + ab + c)* again. Then

$$\pi(q) = ((\mathrm{a} + \mathrm{ab} + \mathrm{c})^* \cap (E \text{ - } \{\mathrm{a}\})^*)|\{\mathrm{b}\} = \mathrm{c}^*|\{\mathrm{b}\} = \Lambda^* = \Lambda.$$

2.2.2 A Small Example

Two small example systems appeared in [McC88b], called A and B, to illustrate the non-composability of a strawman generalization of noninterference. Those systems have some interesting properties with respect to the definitions we have given:

(1) System A is forwardly correctable.

(2) System B is nondeducibility secure on inputs, but not forwardly correctable.

(3) The composition of systems A and B is not nondeducibly secure on inputs.

In order to some of those properties, we will specify systems A and B with finite-state acceptors and show how to check forward correctability for them. Checking nondeducibility on inputs, and constructing an acceptor for the composition, are each possible but relatively laborious, and those activities will not be undertaken here.

System A has two high inputs, a low input, a high output, and three low outputs. One low output is a "stop-count" output, and the other two represent an even or odd parity for the number of high events that preceded the stop-count output. System B is the same except that it has only one high input, and the stop-count event is an input instead of an output. The figure below shows a diagram like McCullough's for these systems. The figure actually shows a sample vertical timeline for each system, with inputs and outputs as horizontal arrows. The thick shaded arrows are the high-level events. We prefer single-character identifiers for events, so the parity outputs are 0 and 1 and the stop-count event is c. The figure also suggests how the two systems are composed.

Both A and B have even (0) and odd (1) parity outputs, but, for example, 0 in A is distinct from 0 in B. Hence the parity outputs are subscripted to identify which system they belong to. Thus, for system A, we have $E = \{x, a, b, c, 0_A, 1_A\}$, with $I = \{x, b\}, O = \{a, c, 0_A, 1_A\}$, and $\underline{E} = \{c, 0_A, 1_A\}$.

One can construct an acceptor for a small system by specifying a regular expression for the traces and then calculating derivatives, but it is often easier, as in this case, to write down the state graph for the acceptor directly. The acceptor for system A is shown in Fig. 3.

Figure 2: McCullough's systems A and B

Figure 3: Acceptor for system A

The initial state is state 0, indicated by the short arrow at the upper left. The $\emptyset$ state is not shown. Transitions not shown, such as the transition caused by event c from state 2, go to the $\emptyset$ state. Note that for all inputs, namely x and b, there is always a transition to a non-$\emptyset$ state. This is necessary and sufficient for the graph to be the acceptor for some event system. Recall that an event sequence is a trace if it takes the acceptor from its initial state to any non-$\emptyset$ (visible) state.

The graph shown for system A shuttles back and forth between states 0 and 1 to keep track of the parity of events x, b, and a. When output c occurs, the next output will be determined by the last recorded parity. After output c has occurred, the inputs x and b have no effect.

There may very well be other event systems that satisfy McCullough's informal description of system A, and which could generate the sample timelines shown. However, this acceptor graph is a formal specification of the particular interpretation of system A that we will work with.

The next step is to determine the equivalence set of each state. State expressions are generated by noting that, since $\Lambda \in q$ for each state q,

$$q = \Lambda + \Sigma\{e(q/e) | e \in E\}$$

and reading q/e off the state graph. This is a standard technique found in, e.g., [Gin68]. Let q_i be the state numbered i in the graph. We have:

$$q_0 = \Lambda + (x + a + b)q_1 + cq_2$$

$$q_1 = \Lambda + (x + a + b)q_0 + cq_3$$

$$q_2 = \Lambda + (x + b)q_2 + 0_A q_4$$

$$q_3 = \Lambda + (x + b)q_3 + 1_A q_4$$

$$q_4 = \Lambda + (x + b)q_4$$

We could "solve" these equations to obtain explict expressions for each q_i, but that will not be necessary. We can use these equations to find the projections. Recall that the projection is homomorphic, yields $\emptyset$ on high inputs, and Λ on other high events.

$$\pi(q_0) = \Lambda + (\emptyset + \Lambda + \emptyset)\pi(q_1) + c\pi(q_2) = \Lambda + \pi(q_1) + c\pi(q_2)$$

$$\pi(q_1) = \Lambda + (\emptyset + \Lambda + \emptyset)\pi(q_0) + c\pi(q_3) = \Lambda + \pi(q_0) + c\pi(q_3)$$

$$\pi(q_2) = \Lambda + \emptyset + 0_A\pi(q_4) = \Lambda + 0_A\pi(q_4)$$

$$\pi(q_3) = \Lambda + \emptyset + 1_A\pi(q_4) = \Lambda + 1_A\pi(q_4)$$

$$\pi(q_4) = \Lambda + \emptyset = \Lambda$$

At this point we will have to do some solving, using Gaussian elimination. Beginning with q_4 and substituting as we go, we get:

$\pi(q_4) = \Lambda$

$\pi(q_3) = \Lambda + 1_A$

$\pi(q_2) = \Lambda + 0_A$

$\pi(q_1) = \Lambda + \pi(q_0) + c(\Lambda + 1_A)$

$\pi(q_0) = \Lambda + \Lambda + \pi(q_0) + c(\Lambda + 1_A) + c(\Lambda + 0_A) = \Lambda + c(\Lambda + 1_A + 0_A) + \pi(q_0)$

Recursive equations are solved using Arden's rule, which says that the equation $p = r + sp$ has the solution $p = s^*r$. Solving the last two,

$\pi(q_0) = \Lambda + c(\Lambda + 1_A + 0_A)$

$\pi(q_1) = \Lambda + \Lambda + c(\Lambda + 1_A + 0_A) + c(\Lambda + 1_A) = \Lambda + c(\Lambda + 1_A + 0_A) = \pi(q_0)$.

We are now ready to check the conditions for forward correctability. This system has no low inputs, so the only condition to check is that $q/x \sim q$ for high inputs x (namely, x and b). By symmetry, we only need to check using x. We have:

$\pi(q_0/x) = \pi(q_1)$

$\pi(q_1/x) = \pi(q_0)$

$\pi(q_2/x) = \pi(q_2)$

$\pi(q_3/x) = \pi(q_3)$

$\pi(q_4/x) = \pi(q_4)$

Thus, system A is forwardly correctable.

McCullough's system B has the same state graph as system A, except that (1) there is no event x, (2) the parity outputs have "B" subscripts, (3) c has become an input, and (3) a and b are exchanged, since a becomes an input and b an output.

The reader can check that, for system B,

$\pi(q_0) = \Lambda + \pi(q_1) + c\pi(q_2)$

$\pi(q_1) = \Lambda + \pi(q_0) + c\pi(q_3)$

$\pi(q_2) = \Lambda + c\pi(q_2) + 0_B\pi(q_4)$

$\pi(q_3) = \Lambda + c\pi(q_3) + 1_B\pi(q_4)$

Figure 4: Acceptor for system B

$\pi(q_4) = \Lambda + c\pi(q_4)$

This yields:

$\pi(q_4) = c^*$

$\pi(q_3) = \Lambda + c^*(\Lambda + 1_B c^*) = c^*(\Lambda + 1_B c^*)$

$\pi(q_2) = c^*(\Lambda + 0_B c^*)$

$\pi(q_1) = \Lambda + \pi(q_0) + cc^*(\Lambda + 1_B c^*)$

$\pi(q_0) = \Lambda + \Lambda + \pi(q_0) + cc^*(\Lambda + 1_B c^*) + cc^*(\Lambda + 0_B c^*) = \Lambda + \pi(q_0) + cc^*(\Lambda + 0_B c^* + 1_B c^*)$

To test forward correctability for system B, we must check, for all states q, that both $q/a \sim q$ and $q/a/c \sim q/c$, since c is a low input. In particular,

$$\pi(q_0/a/c) = \pi(q_3) \text{ and } \pi(q_0/c) = \pi(q_2)$$

And this shows that system B is *not* forwardly correctable, since $\pi(q_3) \neq \pi(q_2)$. This result is somewhat stronger than McCullough's observation that system B is not restrictive, since forward correctability is a weaker property.

3 Summary

Following the nondeterministic/combinatoric thread of information flow security modelling, we made these points:

The event system model is needed to represent non-serializable systems.

The ideal information flow model is (1) at least as strong as nondeducibility on inputs, (2) composable, and (3) the subject of an unwinding result.

For any event system, there is a state-machine acceptor that recognizes its traces.

There is an unwinding result for forward correctability, involving s-equivalence of certain states after at most one high and one low input.

Event system acceptors give rise to regular expressions for their states, which can be used to test s-equivalence easily.

References

[Brz64] J. A. Brzozowski, "Derivatives of Regular Expressions," *J. ACM,* Vol. 11, No. 4 (Oct. 1964), pp. 481-494

[Den76] D. E. Denning , "A Lattice Model of Secure Information Flow," *Comm. ACM*, Vol. 19, No. 5 (May 1976), pp. 236-242

[FLR77] R. J. Feiertag, K. N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design ," *Proc. 6th ACM Symp. Operating System Principles*, pp. 57-65

[Fol87] S. N. Foley, "A Universal Theory of Information Flow ," *Proc. 1987 IEEE Symp. on Security and Privacy*, pp. 116-122

[Gin68] A. Ginzburg, "Algebraic Theory of Automata," Academic Press, 1968

[GoMe82] J. A. Goguen and J. Meseguer, "Security Policies and Security Models," *Proc. 1982 IEEE Symp. on Security and Privacy*, pp. 11-22

[GoMe84] J. A. Goguen and J. Meseguer, "Unwinding and Inference Control," *Proc. 1984 IEEE Symp. on Security and Privacy*, pp. 75-85

[Gra93] J. Graham-Cumming, "Laws of Non-Interference in CSP," *J. Computer Security*, Vol. 2, No. 1, 1993, pp. 37-52

[Gry92] J. W, Gray, III, "Toward a Mathematical Foundation of Information Flow Security," *J. Computer Security*, Vol. 1, No. 3-4, 1992, pp. 255-294

[HKMY87] J. T. Haigh, R. A. Kemmerer, J. McHugh, and W. D. Young, "An Experience Using Two Covert Channel Analysis Techniques on a Real System Design," *IEEE Trans. on Software Engineering*, Vol. SE-13, No. 2 (Feb. 1987), pp. 157-168

[Hoa85] C. A. R. Hoare, "Communicating Sequential Processes," Prentice-Hall, 1985

[JoTh88] D. M. Johnson and F. J. Thayer, "Security and the Composition of Machines," *Proc. of The Computer Security Foundations Workshop*, The MITRE Corporation, M88-37, 1988, pp. 72-89

[McC88a] D. McCullough, "The Theory of Security in Ulysses," Odyssey Research Associates, Ithaca, N.Y., 1988

[McC88b] D. McCullough, "Noninterference and the Composability of Security Properties," *Proc. 1988 IEEE Symp. on Security and Privacy*, pp. 177-186

[McL92] J. McLean, "Proving Noninterference and Functional Correctness Using Traces," *J. Computer Security*, Vol. 1, No. 1, 1992, pp. 37-58

[Mil76] J. K. Millen, "Security Kernel Validation in Practice," *Comm. ACM*, Vol. 19, No. 5 (May 1976), pp. 243-250

[Mil87] J. K. Millen, "Covert Channel Capacity," *Proc. 1987 IEEE Symp. on Security and Privacy*, pp. 60-66

[Mil90] J. K. Millen, "Hookup Security for Synchronous Machines," *Proc. of The Computer Security Foundations Workshop III*, IEEE Computer Society, 1990, pp. 84-90

[Rus85] J. Rushby, "The SRI Security Model," Computer Science Laboratory, SRI International, 1985

[Sut86] David Sutherland, "A Model of Information," *Proc. National Computer Security Conference*, 1986, pp. 175-183