Missouri S&T
Missouri University of Science & Technology
Curtis Laws Wilson Library

ILLIAD Electronic Delivery Cover Sheet

WARNING CONCERNING COPYRIGHT RESTRICTIONS

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted materials. Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "*used for any purpose other than private study scholarship, or research.*" If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use," that user may be liable for copyright infringement.

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Illinois State U 'y

ILLiad TN: 1704

Borrower: UMR

Lending String: *IAI,LUP,I

Patron: McMillin, Bruce

Journal Title: 9th National 'y
Conference Proceedings, 1 986 ;
computer security for today w.

Volume: **Issue:**
Month/Year: September
Pages: 175-183

Article Author: Computer ce (9th ;
1986 September 15-18)

Article Title: D. Sutherland mation

Imprint: [Washington, D.C rea

ILL Number: 48

Date Printed: Wednesd

Searched B

______

______

______

Cop UM______

Cou UM______

Call #: QA76.9.A25 N361986

Location: FL5

ARIEL
Charge
Maxcost: $30IFM

Shipping Address:
Curtis Laws Wilson Library-ILL
University of Science & Technology
400 W. 14th Street
Rolla, MO 65409- 0060

Fax: 573-341-4233
Ariel: 131.151.26.96
Odyssey: 131.151.35.17

Borrowing Notes; We are a free lender. ARTICLES IN ENGLISH ONLY PLEASE. BRI #51-1326; CAI #DD677133; CAS #1089601. Thank you.

9

A MODEL OF INFORMATION

David Sutherland

Odyssey Research Associates, Inc.

1283 Trumansburg Road
Ithaca, New York 14850

The highest level requirements on a secure computing system are usually stated in terms of information, that is, they state that certain information must not be obtained by certain individuals on the system. Formal models of computer security to date have concerned themselves largely with restrictions on the movement of data. While these restrictions capture part of the high level requirements of secure systems, they do not capture all of them, as witnessed by the fact that there is a distinction made between "formal modeling" and "covert channel analysis". True formal verification of a secure system would incorporate the security violations usually covered by "covert channel analysis" into the formal model of the system.

In this paper we present a simple model of information and inference, give a generic instantiation of the model to state machines, apply the instantiation to a simple example, and discuss the relationship between the state machine instantiation and the Goguen-Meseguer non-interference model.

What is information? What does it mean to infer information from other information? We will start with a few naive answers to these questions.

1 Information

In answer to the first question, someone who was used to thinking in terms of formal specifications of systems might answer "the values of some collection of state variables". What's wrong with this answer? One problem with this answer is that it's not general enough. Most instances of information channels involve observing the values of some collection of state variables over the course of time; no one instantaneous value contains the information transmitted. Suppose we amended the above answer to "information is the history of some collection of state variables" where "history" means "history from time 0 to some time t". One problem with this second answer is that, in order to apply it to a system, the system must be expressed as a state machine. Not only does this force us to use a certain representation, but our analysis of information in the system might be unduly sensitive to, say, what state variables we choose. What we wish to do at this point is generalize the second answer so that it's independent of how the system is represented. To do this, we'll look at the second answer in a little more detail.

For the purpose of this discussion we will say an abstract state machine consists of:

1. A set of states

2. A set of possible initial states

3. A set of <u>state transformations</u>, by which we mean a function from states to states.

The set of states is usually defined by giving a set of <u>state variables</u>, each of which has a certain type; a "state" is then an assignment of each state variable to a value in its type.

How do we imagine such an abstract machine actually "running"? We imagine the machine starting out in one of the possible initial states, and then changing state over the course of time as various state transformations are applied. For each possible initial state and each sequence of transformations applied, we get a sequence of states that the machine passes through. Thus, an abstract machine defines a set of possible sequences of states that the machine can pass through. We will call these sequences <u>possible execution sequences</u>. We will regard time as being measured in terms of tne number of state changes the machine has passed through, so "state at time 0" refers to the initial state of the machine, "state at time 1" refers to the state of the machine after one application of a state transformation, and so on.

Given a collection of state variables V and a time T, what do we mean by "the history of the state variables in V from time 0 to time T"? This "history" is actually a function (call it h) whose domain is the set of possible execution sequences. Given a possible execution sequence S, h(S) is a finite sequence of length T + 1 such that:

1. The ith entry of h(S) is an assignment of each state variable in V to a value in its type.

2. For each state variable v in V, the value assigned to v by the ith entry of h(S) is the same value assigned to v by the ith entry of S.

In other words, H takes the first T + 1 entries of S and extracts out the assignments of the variables in V.

We generalize the above scheme in the following way: we represent a system as a set of <u>possible worlds</u>; this corresponds to the set of possible execution sequences for an abstract state machine. A particular "piece of information" about the system is represented as a function whose domain is the set of possible worlds; we will call such functions <u>information functions</u>. An information function can be thought of as a certain "view" of the system; in a given possible world w, an information function returns what is "seen" of w by someone "looking at" the information function.

2 Inference

We now return to the second question posed at the beginning of the section: what does it mean to infer information from other information? In terms of the formalism developed above, what does it mean to infer something about the value of one information function from the value of another information function? To answer this question, we will imagine a user who:

1. knows what the set of possible worlds W is (this corresponds essentially to knowing how the system is designed);

2. knows what information function he is seeing (call it f1) (this corresponds to the user knowing what his interface to the system is. If, for example, a user could see a terminal but had no idea how the output appearing on the terminal was being generated, he would be able to deduce very little about the system.);

3. knows what information function he wishes to deduce something about (call it f2);

4. is "in" some possible world w, and knows the value of f1(w) (call it x).

What can the user infer about f2(w)? Since he knows that f1(w) = x, he can deduce that w is in the set of all possible worlds y such that f1(y) = x. Call this set S. On the basis of the above knowledge, all the user can deduce about w is that it is in S. From this, he can deduce that f2(w) is in the image of S under f2. Call this image T. If there is some value z in the range of f2 that is achieved in some possible world but which is not in T, then the user has actually gained some information about f2(w), namely, he at least knows that it is not equal to z. If, on the other hand, every value z in the range of f2 that is achieved in some possible world is in T, then the user knows nothing more about f2 than he could have inferred on the basis of knowing W and f2 alone. This leads us to the following definition:

> Given a set of possible worlds W and two functions f1 and f2 with domain W, we say that <u>information flows from f1 to f2</u> if and only if there exists some possible world w and some element z in the range of f2 such that z is achieved by f2 in some possible world but in every possible world w' such that f1(w') = f1(w), f2(w') is not equal to z.

Having given this somewhat complicated but reasonably motivated definition, the first thing we will do is note that it is equivalent to a much simpler statement.

<u>Proposition</u>: Given W, f1 and f2 as above, information does <u>not</u> flow from f1 to f2 if and only if the function f1 x f2 from W to the cross product of the images of f1 and f2 is onto.

<u>Proof</u>: Suppose information does not flow from f1 to f2; we wish to show that f1 x f2 is onto image(f1) x image(f2). Let (x,y) be an element of the cross product. Since x is in the image of f1, there exists a possible world w1 such that x = f1(w1). Likewise, there exists a possible world w2 such that y = f2(w2). If we take the negation of the above definition with w = w1 and z = y, we get that there must exist a possible world w' such that f1(w') = f1(w1) = x and f2(w') = y. In other words, (f1 x f2)(w') = (x,y). Since (x,y) was arbitrary, f1 x f2 is onto.

Conversely, suppose f1 x f2 is onto. Let w be a possible world and z an element of the range of f2 that is acheived by f2 in some possible world (in other words, z is an element of image(f2)). Then (f1(w),z) is an element of image(f1) x image(f2) and so there exists a possible world w' such that (f1 x f2)(w') = (f1(w),z), i.e. f1(w') = f1(w) and f2(w') = z.

<u>Corollary</u>: the information flow relation is symmetric.

The corollary seems somewhat surprising at first glance, since information flow is not usually thought of as being necessarily a two-way street. However, consider the following scenario: a system is designed so that every character which is typed at keyboard K is echoed to screen S. Let f1 be the information function which, given a possible world, returns the sequence of all characters typed on K in that world, and let f2 be the information function which, given a possible world, returns the sequence of characters displayed on S in that world. Information is obviously being transferred from K to S, and so we would expect to find that, according to the definition above, information flows from f1 to f2. This is exactly what we do find. By the corollary, we will also find that information flows from f2 back to f1, i.e., it will find that, knowing the value of f1, one can infer something about f2. But this is in fact true! If one knows the design of the system, and one knows what has been typed at K, one knows something about what shows up on S. A problem arises, however, if we try to assign security levels to information functions and require that information always flow "up." If we assign K a level "lo" (say, because only "lo" individuals are allowed to type at it) and S a level "hi" (say, because only "hi" people are allowed to view it), we will find a flow in both directions, violating the security requirement, despite the fact that the system as described seems secure. The problem is not in the definition of information flow but rather in the choice and labeling of information functions. S may be a "hi" terminal, but the information it gets from K is not "hi" information, and thus should not be labeled as "hi." Actually, the only information which should be labeled as "hi" is information which comes from high sources. We will discuss this further below when we instantiate the information model to state machines.

3 Security Conditions

We now have a model of information and a definition of information flow. What we need to complete the model is a definition of "secure". Informally, a system is secure if nobody can get information he's not entitled to. How can we express this in the above formalism?

First of all, each "piece of information" which has restrictions on who may "get" it is represented by an information function. Second, each "entity" on the system which has restrictions on what information it can "get" is represented by an information function corresponding to the entity's "view" of the system. We will denote the set of information functions corresponding to pieces of information and entities by IF. We represent the restrictions on which entities are entitled to "get" which pieces of information by a binary relation "legal_to_get" on IF.

How can we formally express the notion of an entity E "getting" a piece of information I? If f1 is the information function corresponding to E and f2 is the information function corresponding to I, we interpret "E 'gets' I" to mean "information flows from f2 to f1".

Under these interpretations, the informal statement of security given at the beginning of the section is formalized as

> For all f1 and f2 in IF, if information flows from f2 to f1 then legal_to_get(f1,f2)

Readers familiar with formal computer security may at this point be asking "Where are the security levels in this model?" The answer is that security levels are a particular instance of the model. We could assign security levels to the functions in IF and define legal_to_get(f1,f2) if and only if the level of f1 is less than or equal to the level of f2. This is just one possible way of defining legal_to_get; the model allows for others, e.g. discretionary access restrictions.

4 Summary of the Model

In this section we summarize the information model briefly. An instance of the model consists of:

1. A set W of possible worlds
2. A set IF of functions with domain W
3. A binary relation on IF, legal_to_get

Such an instance is secure if and only if for every f1 and f2 in IF, if information flows from f2 to f1 then legal_to_get(f1,f2) (where information flow between functions with domain W is defined as above).

5 State Machine Instantiation

In this section, we instantiate the information model given above to a state machine. We begin by defining what we will mean by a state machine.

5.1 State Machines

"State machine" will mean a non-deterministic finite automaton with null moves except that the state space of the automaton is not required to be finite. In other words, a state machine consists of a set of states, a non-empty set of initial states, an alphabet, and a set of "arrows." Each "arrow" starts at one state and points to another; an arrow may be labeled with a single element of the alphabet or it may be unlabeled. The "operation" of the machine is to start at an initial state and change state in steps, with each state change accompanied by one or no letters of the alphabet.

We now add a few extra structures and some additional axioms. First of all, at this point we will stop using the word "alphabet" and refer to the set we formerly called the alphabet as the signals of the state machine. The signals of the machine are partitioned into the input signals and the output signals. There is also a set of security levels, partially ordered by a relation <=, and a function from signals to levels.

We require that for every state of a state machine and every input signal, there is an arrow which starts at the given state and is labeled with the given signal. In other words, it is always possible for a state machine to receive a given signal (even if its only response is to remain in its former state). We also require that for every state of the machine there is an arrow which starts at the given state which is not labeled with an input signal. In other words, it is always possible for the state machine to "go ahead", even in the absence of an input.

5.2 Instantiating the Information Model

We wish to interpret the information model in terms of state machines. In other words, we want to give a procedure which takes a state machine and returns the corresponding information model. Security for the state machine is then defined simply as security for the corresponding information model. Thus, we need to give a procedure which, given a state machine, gives a corresponding set of possible worlds, a set of information functions for that set of worlds, and a binary relation on those information functions defining what information flows between them are legal. We now describe this procedure.

Suppose we have a state machine described by:

1. A set of states S
2. A set of initial states I
3. A set of input signals X
4. A set of output signals Y
5. A set of "arrows" A (we won't give a completely formal definition of what an "arrow" is, as it would be more obfuscatory than anything else).
6. A set of levels L with partial ordering <=
7. A function from X U Y to L

The set of possible worlds we associate with this machine is the set of finite sequences H = { H[i] | 0 <= i <= length(H) - 1 } such that:

1. Each H[i] is either a state or a signal
2. No two consecutive entries in the sequence are both signals
3. H[0] is an initial state
4. If H[i] and H[i+1] are states, there is an unlabeled arrow from H[i] to H[i+1]
5. If H[i] is a state and H[i+1] is a signal, there is an arrow starting at H[i] labeled with H[i+1]
6. If H[i] is a state, H[i+1] is a signal and H[i+2] is a state, there is an arrow from H[i] to H[i+2] labeled with H[i+1]

There is actually an important reason why we have chosen finite sequences rather than infinite sequences as our possible worlds. Under the above instantiation, a possible world is literally a possible run of the system up to a given point in time. Thus, any inference made in such a world can only take into account the information about the world which has manifested itself up to the point in time being considered. We can think of such finite worlds as being initial segments of some real, complete possible world (i.e. an infinite sequence), but any inference must be made at some finite point in it. This choice literally affects whether some systems are formally secure or not, and the choice we have made seems to make the "right" systems formally secure.

For each level l in L, we define two information functions over the possible worlds defined above:

1. view(l) is the function which, given a possible world H as above, returns the subsequence of H consisting of those signals s whose levels are <= l
2. hidden_from(l) is the function which, given a possible world H, returns the subsequence of H consisting of those input signals s whose levels are not <= l

Finally, we specify that it is illegal for information to flow from hidden_from(l) to view(l) for any l.

This choice of information functions and illegal flows is based on the following picture: there is a collection of "entities" external to the machine which interact with it, and each of these entities has a level. It is assumed that each entity only "knows" some signals going into and coming out of the machine, and that an entity of level l is allowed (by procedural safeguards or whatever) to see at most all of the signals that go into or out of the machine whose levels are <= l. The choice of illegal flows simply reflects the policy that an entity of level l should not be able to deduce from what it is legal that he see anything about what it is not legal that he see. Notice that, according to the instantiation, there is nothing wrong per se in a low level entity being able to deduce a high level output signal. If a high level output signal is unconnected to any high level input signals, then it is not a violation of security for a low level entity to see it. If, on the other hand, such a high level output has some connection to a high level input, then this connection will presumably be reflected as an information flow and thus an information flow will be found from the high inputs to the low entity, violating the policy as stated.

6 An Example

In this section we give a simple example of the use of the state machine instantiation of the information model. We will first describe the machine informally.

The machine is a simple message-passing system. It consists of a collection of "ports" and a queue of "message entries."

Each port is labeled with a security level indicating what level entities external to the machine can access the port. Ports can input messages to the machine, which get put on the queue in a message entry. The message entry also contains the information of which port the message came from. The intended destination of the message is contained in the message. When a message entry comes to the head of the queue, one of two actions is taken: (1) if the level of the destination port is greater than or equal to the level of the source port, the message is output to the destination port and the message entry is removed from the queue; (2) otherwise, the message entry is removed from the queue and no output occurs.

We now describe the above machine formally. We denote the set of ports by P, the set of messages by M, the set of security levels by L, the partial ordering on L by <=, the function which takes a port and returns its level by lvl (a function from P to L), and the function which takes a message and returns its destination by dest (a function from M to P).

The state of the machine is a sequence of pairs (m,p) where m is in M and p is in P. The initial state of the machine is the empty sequence.

A signal of the machine is a triple (m,p,x) where m is in M, p is in P and x is in the set {input,output}. Such a signal is an input signal if x is "input" and an output signal if x is "output". If x is "input", p is the port from which the signal came. If x is "output", p is the port to which the signal goes. The level of a signal (m,p,x) is lvl(p).

The arrows of the machine are as follows:

- For each state s, each m in M and each p in P, let s' be the sequence s with the pair (m,p) prepended; there is an arrow from s to s' labeled with (m,p,input).

- For each state s, each m in M and each p in P, let s'' be the sequence s with the pair (m,p) appended:

  * If lvl(p) <= lvl(dest(m)), there is an arrow from s'' to s labeled with (m,dest(m),output).

  * If lvl(p) is not <= lvl(dest(m)), there is an unlabeled arrow from s'' to s.

- There is an unlabeled arrow from the empty sequence to itself.

We will now prove that this machine meets the security condition of the state machine instantiation of the information model.

Fix a level l in L. We wish to prove that no information flows from hidden_from(l) to view(l). First, we will define a function R from states to states as follows: if s is a state (i.e. a sequence of message-port pairs), R(s) is the subsequence of s consisting of the entries (m,p) such that lvl(p) <= l.

We want to examine what happens to a possible world of the machine (i.e. a finite sequence of states and signals meeting the conditions described in the previous section) when we apply R to every state in it. To do this, we must examine the effect of R on the initial state of the machine and the pairs of states at the ends of the various arrows.

The initial state of the machine is the empty sequence, and R of the empty sequence is the empty sequence. Thus, R of the initial state is the initial state.

Suppose there is an arrow from s to s' labeled with (m,p,input); s' must therefore be s with (m,p) prepended. What do R(s) and R(s') look like? If lvl(p) <= l, R(s') is R(s) with (m,p) prepended, and so there is an arrow from R(s) to R(s') labeled with (m,p,input). If lvl(p) is not <= l, R(s) equals R(s'). In other words, if the arrow corresponds to an input signal of level <= l, the states at the ends of the arrow are mapped to states at the ends of an arrow corresponding to the same input signal; in this case we will say that R "preserves" the arrow. If the arrow corresponds to an input signal whose level is not <= l, the states at the end of the arrow are mapped to the same state; in this case we will say that R "masks" the arrow.

Suppose there is an arrow from s'' to s labeled with (m,dest(m),output). s'' must therefore be s with (m,p) appended for some p such that lvl(p) <= lvl(dest(m)). In this case, if lvl(p) <= l then R(s'') is R(s) with (m,p) appended, and there is an arrow from R(s'') to R(s) labeled with (m,dest(m),output). Again, we say that R preserves the arrow. If lvl(p) is not <= l, R(s'') equals R(s), and we say that R masks the arrow.

Suppose there is an unlabeled arrow from s'' to s. There are two possibilities for s'' and s. First, s'' and s can both be the empty sequence, in which case R(s'') = R(s) = the empty sequence and we say that R preserves the arrow. Second, s'' can be s with (m,p) appended for some p such that lvl(p) is not <= lvl(dest(m)). In this case, if lvl(p) <= l then R(s'') is R(s) with (m,p) appended, and there is an unlabeled arrow from R(s'') to R(s). Again, we say R preserves the arrow. If lvl(p) is not <= l, R(s'') equals R(s) and we say that R masks the arrow.

What happens when we take a possible world H and apply R to every state in it? We can think of H informally as consisting of a sequence of arrows, with the first arrow starting at the initial state and with the ending and starting states of consecutive arrows matching. Each such arrow will either be preserved by R or masked by R. If we "throw away" the arrows that are masked by R, we get a new possible world of the machine. Call this world RH. What is the relationship between RH and H? RH is essentially H with all input signals whose levels are not <= l removed. In addition, all outputs and state changes resulting from such signals (i.e. being queued, being dequeued) are also removed. On the other hand, all of the input and output signals of level <=1 are the same (the proof of this relies on the fact that the machine only allows signals from a given port to be output to ports of equal or greater level). In other words, view(l)(RH) = view(l)(H) while hidden_from(l)(RH)= the null sequence. Given any sequence s of input signals of level not <=l, we can add them on to the end of RH to get a possible world H' such that view(l)(H')= view(l)(RH) and hidden_from(l)(H')=S. Since H was arbitrary, this shows that for any value v1 in image(view(l)), any any value v2 in image(hidden_from(l)), there exists a possible world H' such that view(l)(H')= v1 and hidden_from(l)(H')= v2. In other words, view(l) x hidden_from(l) is onto image(view(l)) x image(hidden_from(l)) so there is no information flow between them. Since l was arbitrary, this proves the security condition.

7 Connection with the Goguen-Meseguer Model

We can think of the above proof as taking place in two stages. We start with an arbitrary possible world H, and we show that:

1. We can replace H by RH so that there are no signals of level not <= l, without changing any signal of level <= l.

2. We can replace RH by H' to make the signals of level not <= l anything we want, without changing any signal of level <= l.

The first step looks something like a proof of a non-interference condition as in the Goguen-Meseguer model, i.e. it shows that the inputs of entities of level not <=l can be deleted without effecting what is seen by entities of level <=l. What is the relationship between the state machine instantiation of the information model and the Goguen-Meseguer non-interference model?

First of all, by "The Goguen-Meseguer model" we will hereinafter mean the model as set forth in [1]. We will only consider what are referred to as "static systems" in [1]. The first problem we encounter in comparing our state machine instantiation with the Goguen-Meseguer Model is that the Goguen-Meseguer model uses a different kind of automaton than the state machine instantiation.

The second problem we encounter is that the Goguen-Meseguer model allows us to express a much broader class of security policies that can be expressed in the state machine instantiation. In our reformulation of the state machine instantiation, we will broaden the policies expressible to include arbitrary non-interference assertions as in [1].

Before giving our reformulated instantiation, we will note a few assumptions about what users can "see" that seem to be implicit in the Goguen-Meseguer model. First, it seems implicit a user u cannot "see" the state machine making a transition from state s1 to state s2 if out(s1,u)=out(s2,u). In other words, a user cannot tell the difference between seeing a given output once and seeing it "twice in a row". If this were <u>not</u> the case, then whenever any user issued any command to the state machine, it would be seen by every user, either as a change in output or a "repeat" of the same output.

Second, it seems implicit that users cannot "see" time passing. In other words, a user cannot tell the difference between a sequence of state changes carried out over a "long" time and the same sequence of state changes carried out over a "short" time. Indeed, the Goguen-Meseguer model has no way of expressing this difference.

We now give a reformulation of our state machine instantiation in terms of Goguen-Meseguer-type state machines. Fix a state machine M consisting of a set of users, U, a set of states, S, a set of commands, C, a set of outputs, OUT, a function "out" from S x U into OUT, a function "do" from S x U x C into S and an initial state s. In addition, fix a set of commands A and sets of users G1 and G2. We wish to give an instantion of the information model to M whose information functions and information flow restrictions express the non-interference assertion A, G1 :| G2.

The set of possible worlds associated with M is the set of all finite sequences of elements of UxC. Since Goguen-Meseguer state machines are deterministic and have a unique starting state, the behavior of M during a given sequence of commands from users is completely determined by the sequence of users and commands issued. We choose finite sequences for the same reasons explained in subsection 5.2

We will now define a few functions we will need later. Given a possible world H=<(u[o],c[o]),...,(u[n],c[n])>, we can define a sequence of alternating states and elements of Ux6 ST(H)=<s[o], (u[o], c[o]), s[1], (u[1], c[1]), ..., s[n], (u[n], c[n]), s[n+1]> where s[o]= s (the initial state of M) and s[i+1]= do (s[i], u[i], c[i]) for i= , ..., u. In other words, ST(H) is just H with the states that M passes through M the course of H "interpolated".

Given a state s, we can define a functions V(s) from G2 into OUT by V(s)[g]=out(s,g) for all g in G2. V(s) is thus the "G2-tuple" of outputs "seen" by the member of G2 in state s.

We will now complete the instantiation. We associate two information functions with the non-interference assertion A,G1 :| G2 :

1. <u>INPUT</u> is the function which, given a possible world H, returns the subsequence of H consisting of the entries (u,c) where u is in G1 and c is in A.

2. VIEW is the function which, given a possible world H, returns a sequence obtained as follows:

   - Start with ST(H). Delete from ST(H) all entries (u,c) such that u is not in G2. Call the result X. (X will be a sequence of states and user-command pairs, not necessarily alternating).

   - Replace every entry s of X by V(s). Call the result Y. (Y will be a sequence of user-command pairs and functions from G2 into OUT).

   - Remove all consecutive repetitions of functions from G2 into OUT from Y. the result is VIEW(H).

INPUT simply extracts from H the history of inputs from users in G1 which are in the command set A. VIEW is slightly more complicated. Given H, it returns the history of commands from users in G2 and outputs to users in G2, with repeated outputs ignored. These functions are similar to the hidden-from and view functions of the first instantiation.

Finally, we specify that it is illegal for information to flow from INPUT to VIEW.

What is the relationship between the two definitions of security for M? First of all, a bit of pathology arises if A is non-empty and G1 and G2 overlap. Since users in G2 "know" what commands they're given, if a user u who is both G1 and G2 issues a command from A, then the users of G2 can deduce something about commands in A issued by users in G1. On the other hand, it is possible for A, G1 :| G2 to hold. For example, suppose G1=G2={u} and A={c} where do (s,u,c)=x for all states S. Then A,G1 :| G2 holds, i.e. u literally cannot interfere with himself by issuing c because c never causes any state change and so never causes any change in the output seen by u. The state machine instantiation takes into account the fact that u "knows" more than just what he "sees"; u also "knows" what he "does".

Thus, in the degenerate case where A is non-empty and G1 and G2 overlap, non-interference does not imply no flow from INPUT to VIEW. However, we do have

Proposition 1: If G1 and G2 are disjoint and A,G1 :| G2, then there is no flow from INPUT to VIEW.

Let p be the function which, given a possible world H, returns H with all entries in G1xA deleted. Proposition 1 will follow from the following

Lemma 1: If G1 is disjoint from G2 and A,G1 :| G2, then for all possible worlds H, VIEW(H)=VIEW (p(H)).

Proof of Proposition 1 from Lemma 1: We need to show that for any A in image(INPUT) and any B in image(VIEW), there exists a possible world H such that INPUT(H)=A and VIEW(H)=B. B in image(VIEW)=> there exists HO such that B=VIEW(HO). A in image(INPUT)=> A is a sequence of elements of G1xA. Let H= p(HO)^A. H is a possible world. Clearly, INPUT(H)=A. By the Lemma, VIEW(H)= VIEW(p(H))= VIEW(p(p(HO)^A))= VIEW(p(p(HO)))= VIEW(p(HO))= VIEW(HO)=B.

///

Before giving the proof of Lemma 1, we will note a few relevant facts.

Suppose H is a possible world, u is a user and c is a command. What is VIEW(H^<(u,c)>)? To compute VIEW(H^<(u,c)>), we must first compute ST(H^<(u,c)>). If s is the last element of ST(H) (i.e., the state of the machine after "doing" it), then ST(H^<(u,c)>) is ST(H)^<(u,c), do(s,u,c)>. Next, we delete all user-command pairs with user not in G2. Let X be the result of performing the operation on ST(H^<(u,c)>) is :

X^<(u,c), do(s,u,c)> if u is in G2

X^<do(s,u,c)> if u is not in G2

Next, we apply V to all states in the sequence. Let Y be the result of performing this operation on X; Then the result of performing the operation on the sequence above is:

Y^<(u,c), V(do(s,u,c))> if u is in G2

Y^<V(do(s,u,c))> if u is not in G2

Note that the last entry of Y is V(s). The last step in constructing VIEW(H^<(u,c)>) is to eliminate consecutive repetitions of values of V. The result of doing this to Y is VIEW(H). Therefore, we have

Fact 1: VIEW(H^<(u ,c)>)=

VIEW(H)^<(u,c), V(do(s,u,c))>, if u is in G2

VIEW((H)^<V(do(s,u,c))>, if u is not in G2 and V(5) is not = V(do(s,u,c))

VIEW(H), otherwise.

As mentioned above, for any possible world H, the last element of ST(H) is the state of M after "doing" H. It is easily seen that the last element of VIEW(H) is therefore the function form G2 to OUT which maps each user in G2 to the output "seen" by that user after H is "done". We denote the last element of a sequence Q by last(Q). A straightforward translation of the definition of non-interference in [GM] yields

Fact 2: A,G1 :| G2 if and only if for all possible worlds H, last(VIEW(H))= last(VIEW(p(H))).

Proof of Lemma 1: The proof is by induction on the length of H.

The base case is H=<> (the empty sequence). Then p(H) = <> = H, so VIEW(H) = VIEW(p(H)).

We now do the inductive step. Assume H=HO^<(u,c)> and VIEW(HO)=VIEW(p(HO))

Case1: (u,c) is in G xA. Then p(H)=p(HO^<(u,c)>)=p(HO), so by Fact 2, last(VIEW(H))= last(VIEW(p(H)))= last(VIEW(p(HO)))= last(VIEW(HO)).

Since G1 and G2 are disjoint, u is not G2, so by Fact 1, VIEW(H)=VIEW(HO^<(u,c,)>)=

VIEW(HO)^<V(do(s,u,c))>, if V(s) is not = V(do(s,u,c))

VIEW(HO), otherwise.

But last(VIEW(HO)) is V(s), so the only way that last(VIEW(H)) can = last(VIEW(HO)) is if the second case above holds, so VIEW(H)= VIEW(HO). By the inductive hypothesis, VIEW(HO)= VIEW(p(HO)), and p(HO)= p(H), so VIEW(H)= VIEW(p(H)).

Case 2: (u,c) is not in G xA and u is not in G2. Then p(H)= p(HO^<(u,c)>)=p(HO)^<(u,c)>. By Fact 2, VIEW(H)=

VIEW(HO)^<V(do(s',u,c))>, if V(s') is not=V(do(s',u,c))

VIEW(HO), otherwise.

Where s=last(ST(HO)). Again by Fact 2, VIEW(p(H))=VIEW(p(HO)^ <(u,c)>)=

VIEW(p(HO))^<V(do(s',u,c))>, if V(s') is not= V(do(s',u,c))

VIEW(p(HO)), otherwise.

Where s'=last(ST(p(HO))). By the inductive hypothesis, VIEW(HO)= VIEW(p(HO)). Therefore, V(s)= last(VIEW(HO))= last(VIEW(p(HO)))= V(s').

Now, Suppose V(s)= V(d0(s,u,v)) but V(s') is not= V(do(s',u,c)). Then VIEW(H)= VIEW(HO) and VIEW(p(H))= VIEW(P(HO))^<V(do(s',u,c))>. By Fact 2, last (VIEW(H))= last(VIEW(p(H))). But then

V(s')=V(s)= last(VIEW(HO))= last(VIEW(H))= last(VIEW(p(H)))= V(do(s',u,c)), a contradiction. Therefore, if V(s)= V(do(s,u,c)), then V(s')= V(do(s',u,c)), and so VIEW(H)= VIEW(HO)= View(p(HO))= VIEW(p(H)).

Next, suppose V(s) is not= V(do(s,u,c)) but V(s')= V(do(s',u,c)). By an argument completely analogous to that of the previous paragraph, this lead to a contradiction, so if V(s) is not= V(do(s,u,c)) then V(s') is not= V(do(s',u,c)); in this case,

VIEW(H)=VIEW(HO)^<V(do(s,u,c))>

VIEW(p(H))=VIEW(p(HO))^<V(do(s',u,c))>

VIEW(HO)= VIEW(p(HO)) by the inductive hypothesis, and by Fact 2, V(do(s,u,c))=last(VIEW(H))= last(VIEW(p(H)))= V(do(s',u,c)), so VIEW(H)= VIEW(p(H)).

Case 3: (u,c) is not in G xA and u is in G2. Then p(H)= p(HO)^<(u,c)>. By Fact 1,

VIEW(H)=VIEW(HO)^<(u,c), V(do(s,u,c))>

VIEW(p(H))=VIEW(p(HO))^<(u,c), V(do(s',u,c))>

VIEW(HO)= VIEW(p(HO)) by the inductive hypothesis, and the last elements of the above sequences are= by Fact 2, so again, VIEW(H)= VIEW(p(H))

///

The converse of Proposition1 fails in a non-pathological case however.

Proposition 2: No flow from INPUT to VIEW does not imply A,G :| G .

Proof: Consider the following state machine:

```
U={u1,u2,u3}

S={0,1}x{0,1}

C={flip1,flip2}

OUT={0,1}

out((b1,b2),u)=

    b1 if u=u1
    0    otherwise

do ((b1,b2),u,c)=

    (b1,b2)      if u=u1
    (b1,b2)      if u=u2 and b2=0
    (b1,b2)      if u=us and b2=1 and c=flip2
    (1-b1,b2)    if u=u2 and b2=1 and c=flip1
    (1-b1,b2)    if u=u3 and c=flip1
    (b1,1-b2)    if u=u3 and c=flip2

so=(1,1)
```

Briefly, the state consists of 2 flags. u1 can see the first flag, while u2 and u3 can't see anything. There are 2 commands, one to flip the first flag and one to flip the second flag. Commands from u1 are always ignored. Commands from u2 to flip the second flag are always ignored, whereas commands form u2 to flip the first flag are carried out if the second flag is 1, and are ignored otherwise. Commands from u3 to flip either flag are always carried out.

Let A={flip1}, G1={u2} and G2={u1}.

Claim 1: A,G1 :| G2 does not hold. Consider the possible world H=<(u2,flip1)>. After H, u1 is "seeing" a p(H)=<>/ After p(H), u1 is "seeing" a 1. By definition of non-interference, the above assertion does not hold.

Claim 2: There is no flow from INPUT to VIEW. Rather than give a completely rigorous proof, we will simply indicate why claim 2 is true. We wish to show that u1 cannot possibly deduce anything about u2's inputs by observing his own inputs and his outputs. The reason this is true is because anything that u1 sees could be the result of u3 issuing flip2, u3 issuing flip1 a certain number of times, and u2 issuing any sequence of commands. In other words, no matter how many times u1 sees the first flag flip, it could always be the result of u3 issuing flip 1, and if u3 in addition issues flip 2 immediately, all of u2's inputs are "masked out".

The essential difference in this example between the Goguen-Meseguer model and the information model instantiation, is that Goguen-Meseguer requires that there be no change in what u1 sees when u2's inputs are deleted _while holding u3's inputs fixed_. The Goguen-Meseguer model requires u3's inputs to be held fixed even though u1 has no way of knowing what they are.

In conclusion, the state machine instantiation of the information model given above seems (if certain pathological situations are ruled out) to be a generalization of the Goguen-Meseguer model. It is a proper generalization in the sense that it is implied by Goguen-Meseguer but does not always imply Goguen-Meseguer.

REFERENCES

[1] Goguen, J.A. and Meseguer, J., _Security Policies and Security Models_, Proceedings of the 1982 Symposium on Security and Privace, April 1982.