Formal Semantics for Visual Specification of Security

Mark W. Maimone, J.D. Tygar, and Jeannette M. Wing
December 1988
CMU-CS-88-173r

This paper appears in the Proceedings of the IEEE 1988 Workshop on Visual Languages, Pittsburgh, Pennsylvania, October 10-12, 1988.

This research was sponsored in part by a contract from the National Computer Security Center, a Presidential Young Investigator grant by the National Science Foundation, and by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 4976(Amendment 20), under contract F33615-87-C-1499 monitored by the Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB and in part by the National Science Foundation under grant CCR-8620027. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Formal Semantics for Visual Specification of Security

Mark W. Maimone J. D. Tygar Jeannette M. Wing

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

The Miró Project at Carnegie Mellon University is designing and implementing a visual language for specifying properties of large software systems. We are designing the language in tandem with giving it a formal semantics. We present the semantics of our language as applied to the security domain.

I Introduction

Visual languages, like all languages, need a formal semantics. This paper presents an outline of a visual language and gives a formal definition of its meaning.

Pictures from a language that has ambiguous (informal) interpretations for graphical constructs only serve to frustrate the user of the visual language, and confuse the reader ("But what does it *mean*?"). Some languages at least come equipped with rules that determine when a "picture"[1] is well-formed. A formal semantics, however, would describe not only the syntactically valid pictures, but more importantly, their mathematical interpretation. That is, it does not suffice to give only a BNF for pictures; one must additionally map each well-formed picture onto some underlying mathematical entity.

The Miró Project at Carnegie Mellon University is designing and implementing a visual language for specifying properties of large software systems. The first class of properties to which we are applying our visual notation is security, e.g., secrecy and integrity of files, as first described in [1]. Unlike the development of many other visual languages, we are designing Miró in tandem with giving it a formal semantics. We therefore benefit from not only the concision of visual notation, but the precision of a rigorous semantics.

II Informal Description

Security is an especially compelling area in which to apply our techniques. Besides the inherent importance of the subdiscipline, it is an area based on a clearly defined underlying model (e.g., Lampson's access matrix representation) and has motivated much research in formal specification techniques. The goal of the Miró Project is to present a formal specification model that is straightforward to understand and still has a mathematically precise meaning. Previous work in protection systems tended to concentrate on operating system mechanisms for specifying file system access on a file-by-file or process-by-process basis; these mechanisms usually involve adding auxiliary information to the information on the file, where the

[1] We use the term "picture" here generically to mean some ensemble of graphical objects drawn using some visual language; our pictures are diagrams, not raster images.

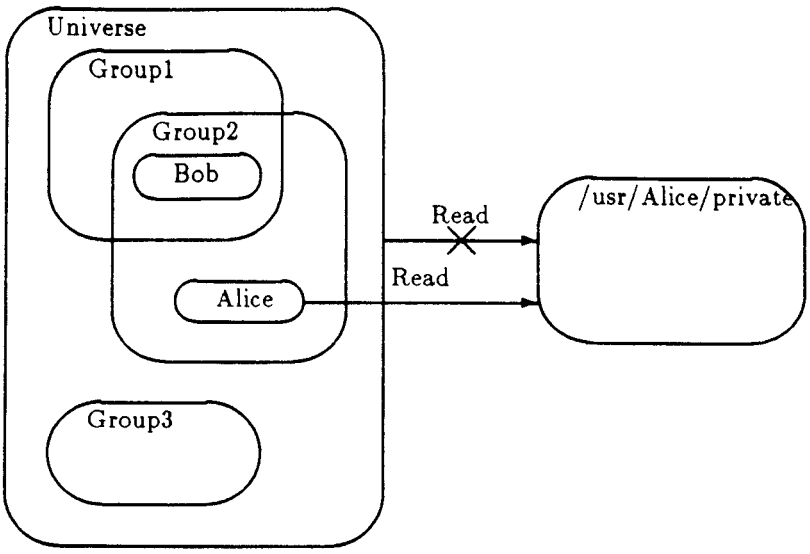

Figure 1: A sample Miró security specification.

auxiliary information is encoded in some fixed format. However, heterogeneous systems typically have vastly different representations of protection information, and even on a single system it is frequently difficult or impossible to trace completely patterns of access without inspecting the encoded information stored with each file or process. In UNIX,[2] for example, each file has associated with it an *owner* and a *group*. The *owner* represents a single user and the *group* represents a set of users. Three access types are defined: *read*, *write*, and *execute*. The meaning of these access types varies with the type of file. (e.g., *execute* privileges not only control the right to run executable programs, they also govern the right to access any files inside a directory) These rights are stored as bits, and the interpretation of these bits must be understood by a user. As a result, many naive users make frequent errors by allowing unauthorized individuals to access their files. These problems are further complicated when we include other UNIX features such as "set user ID" bits and "sticky" bits.

The constructs of Miró are simple: *boxes* and *arrows*, each optionally labeled.[3] In giving such constructs a semantics, it is essential to provide an interpretation for individual boxes and arrows and their compositions when forming *pictures*. Depending on the class of properties of interest, the interpretation of the boxes and arrows will change. In this paper, we present a complete interpretation for the Miró constructs in the domain of security properties. The underlying semantic model for security is simple: an *access-rights* matrix, where each *process* and each *file* (e.g., users and programs) has a (possibly empty) set of *rights* governing access (e.g., Read, Write, Execute). Informally, boxes represent individual processes and files or collections of processes and files; arrows represent rights. Since we have *negative* arrows as well, we can express the absence of rights. The presence of negative arrows introduces some non-triviality to our semantics, since in particular, we wish to disallow ambiguous pictures.

For example, Figure 1 shows a Miró security specification that reflects some aspects of the UNIX file protection scheme. The outermost lefthand box depicts a *universe*, Universe, of users, three (out of possibly many not explicitly shown) *groups*, Group1, Group2, and Group3, and two (out of many not explicitly shown) *users*, Alice and Bob. The containment and overlap relationships between the universe, groups, and users indicate that all users are in the universe, and users can be members of more than one group.[4] The

[2] UNIX is a trademark of AT&T.

[3] Boxes are drawn as rectangles with rounded corners, inspired by Harel's Statechart notation [2].

[4] We do not address the property that a user must belong to at least one group here, though Miró does provide for this expressibility.

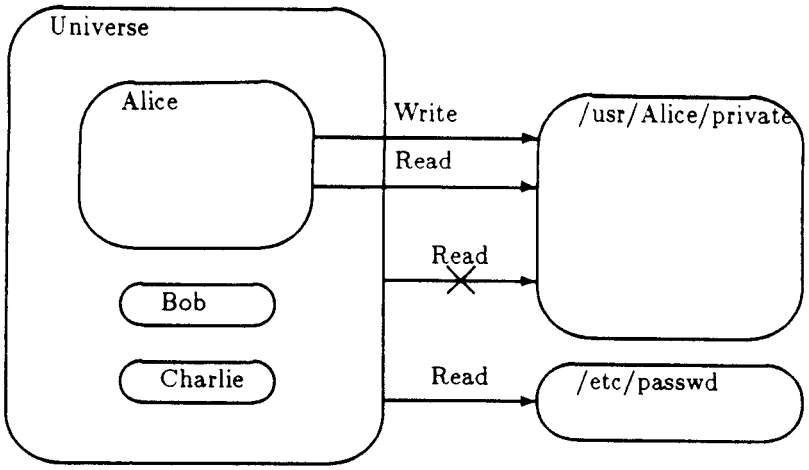

Figure 2: Another security specification.

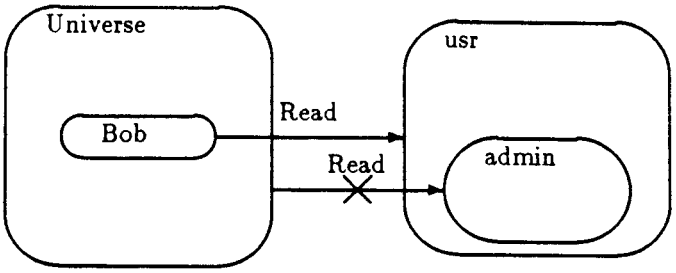

Figure 3: An ambiguous security specification.

righthand box denotes the set of files in Alice's private directory. The arrows indicate that Alice, and no other user, has Read access to her private files. That is, the direct positive arrow from Alice overrides the negative arrow from Universe.

Figure 2 illustrates some more features of the language. Here only Alice has *Read* and *Write* rights to `/usr/Alice/private`. What of the rest of the users — do they or do they not have *Write* access to Alice's pirvate directory? The answer is no, they do not. We define the absence of an appropriate arrow to mean no access. Both Figure 1 and Figure 2 also illustrate the distinction between user and file boxes: the former lie at the tails of all arrows, the latter at the heads.

Finally, what does the picture in Figure 3 mean? Is Bob a special user who has access to all programs in **usr**, including **admin**? Or are no users (including Bob) allowed access to the **admin** directory? Either interpretation seems valid; therefore this picture is *ambiguous*. One result of this paper is a closed form expression for determining when an entry in the access-rights matrix is ambiguous.

Already, the reader might wonder the following: What do "containment" and "overlap" mean when some boxes look like they denote atomic objects (e.g., Alice and Bob) and some denote sets of atomic and non-atomic objects (e.g., Group2)? (What do "atomic" and "non-atomic" mean?) What is the interpretation of the absence of a box or arrow? What is the interpretation of seemingly conflicting negative and positive arrows (i.e., what are the rules for overriding arrows)? How are ambiguous pictures dealt with? These are the sorts of questions a formal semantics can answer precisely.

In what follows, we capture these visual notions in a logical setting: the set theoretic notions hinted at in Figure 1 are made explicit. We formally present the syntactic domains in Section III, and the semantic domains (access-rights matrix) and interpretation in Section IV. We present a careful treatment of ambiguity in Section V, and close with some final remarks about current and future work in Section VI.

III Syntax

Although the Miró language itself is primarily visual, we will give its semantics in a well-known denotational language; the language of mathematics. We will build propositions out of set theoretic constructs and use first-order logic to reason about them. Table 1 lists the definitions we will need to build these propositions. We will motivate each of these definitions in the following section.

But before beginning the presentation, we should clarify some aspects of our notation. Throughout the paper, indentation is used to reduce the number of parentheses, negation symbols ($\neg$) will bind more closely than conjunction ($\wedge$), and conjunction will bind more closely than disjunction ($\vee$). Implication ($\Rightarrow$) is less restrictive than these, but will bind more closely than the quantifiers ($\forall$, $\exists$). The symbol ($\uplus$) will be used to denote the union of two disjoint sets, and ($\triangleq$) will be used to define new constructs.

III.1 Motivation For the Syntax Table

Consider the kinds of things that make up a picture. There are two types of boxes, User and File boxes. Each kind of box has an identifier associated with it. Let F_{id} be the set of file identifiers given for a particular picture, and P_{id} the set of user (or Process) identifiers.

Now that we have names for both kinds of boxes, let us look carefully at just what these boxes stand for. We will ignore the fact that there are two kinds of boxes for now, and consider just one type (i.e., we'll leave the identifiers uninstantiated).

Just what does a box represent? The simplest kind of box, the atomic box, represents exactly one entry in the access matrix; that entry has the name of the box. The box's size and location do not affect the matrix. So we can fully characterize atomic boxes with just their labels.

Now what about more complicated boxes? Consider boxes that contain only atomic boxes. These boxes each have a name and a list of (atomic) boxes contained within them. Going one step further, consider boxes containing these (that is, boxes whose subboxes contain only atomic boxes). Again, the meaning of these boxes is wholly contained in the boxes' identifiers and lists of subboxes.

Thus, every box can be uniquely characterized by its identifier and the set of boxes contained within it. Notice that this definition also takes care of overlapping boxes; we place no constraints on the list of subboxes, so one box may be contained in many others. We should also point out that when two boxes overlap in a picture, but have no other boxes in the overlapping section, then the overlapping is ignored; it has no bearing on the access matrix. This is an intended, though controversial, aspect of the semantics.

We can express this box encapsulation more formally, and do so in Table 1. We define $B_{\mathcal{I}}^{i}$ inductively, in terms of larger enclosing sets. $B_{\mathcal{I}}^{0}$ is the set of atomic boxes, and $B_{\mathcal{I}}^{1}$ is the set of atomic boxes *plus* all boxes which contain only atomic boxes. We continue in the same fashion, adding at each level boxes which contain all of the boxes listed at the previous level.

There is a quirk in the formalism. In order to make the mathematical expression for *contains all boxes at lower levels* more concise, we replicate each earlier level in the later ones. So every earlier set of boxes is included in the later ones. Formally, $\forall_i \forall_{j \text{ where } 0 \leq j < i} B_{\mathcal{I}}^{j} \subset B_{\mathcal{I}}^{i}$. This allows us to use the power set notation $2^{B_{\mathcal{I}}^{i-1}}$ (see Table 1) quite conveniently. We remove the empty set from the power set for convenience only; boxes with no subboxes are added to the set by the union with $B_{\mathcal{I}}^{0}$, the set of atomic boxes. It could have just as easily (and more importantly, just as *correctly*) been left in.

Finally, we can talk about the set of all possible boxes. We could say that $B_{\mathcal{I}}^{\infty}$ is the set of all possible boxes, but to avoid trying to prove that the limit exists (since there is none), we instead define the set of all boxes to be the union of all levels. Since each earlier level is contained in all later ones, this union seems intuitively superfluous.

Entity	Symbol	Example
Set of File Identifiers	F_{id}	{ *passwd, vmunix, tty* }
Set of Process Identifiers	P_{id}	{ *alice, bob, charlie* }
Box Constructor:	$B^0_{\mathcal{I}} = \{\langle \emptyset, id\rangle \mid id \in \mathcal{I}\}$	$B^0_{\mathcal{I}}$ is the set of atoms
Set of Boxes with identifiers in $\mathcal{I}$,	$B^1_{\mathcal{I}} = B^0_{\mathcal{I}} \cup \{\langle x, id\rangle \mid x \in (2^{B^0_{\mathcal{I}}} - \{\emptyset\}) \wedge id \in \mathcal{I}\}$	$B^1_{\mathcal{I}}$ is the set of boxes containing atoms
where $\mathcal{I} = F_{id}$ or P_{id}	$B^i_{\mathcal{I}} = \bigcup_{j=0}^{i-1} B^j_{\mathcal{I}} \cup \{\langle x, id\rangle \mid x \in (2^{B^{i-1}_{\mathcal{I}}} - \{\emptyset\}) \wedge id \in \mathcal{I}\}$	$\langle \emptyset, alice\rangle \simeq \boxed{alice}$ $\langle \{\langle \emptyset, alice\rangle, \langle \emptyset, bob\rangle\}, students\rangle$
Set of File Boxes	$F \subseteq \bigcup_{i=0}^{\infty} B^i_{F_{id}}$	
Set of Process Boxes	$P \subseteq \bigcup_{i=0}^{\infty} B^i_{P_{id}}$	
Set of All Boxes	$BOXES = F \cup P$	{ $\boxed{alice}$, $\boxed{\begin{array}{c}students\\ \boxed{alice}\ \boxed{bob}\end{array}}$ }
Subbox Operators	$\sigma_F : F \rightarrow 2^F$ $\sigma_P : P \rightarrow 2^P$ $\sigma_{\mathcal{I}}(\langle X, id\rangle) = X$ $\sigma = \sigma_F \cup \sigma_P$	
	$\tau : 2^{BOXES} \rightarrow 2^{BOXES}$ $\tau(X) = \bigcup_{x \in X} \sigma(x)$ $\tau^*(X) = \bigcup_{j=0}^{\infty} \tau^j(X)$	
Atomic Boxes	$ATOMS = \{x \mid x \in BOXES \wedge \sigma(x) = \emptyset\}$	
Set of Relation Types	$TYPES$	{ *read, write, execute* }
Arrows	$ARROWS \subseteq P \times F \times TYPES \times \{ pos, neg \}$	

Table 1: Miró syntactic entities

Now that we have a mechanism for describing the meanings of boxes, we can apply it to the security domain by instantiating the box identifiers to file identifiers and process identifiers in turn, giving us sets F and P of boxes. We will call the set of all boxes in the security domain $BOXES$ (this is just the union of F and P).

Now we know how to construct boxes for this domain. We define a few operators on these box objects which will be useful later on: subbox (sigma, or σ) and all-subboxes (tau-star, or τ^*, defined below). Sigma functions are defined with respect to the set of box identifiers; here, we have σ_F and σ_P. If we consider mappings to be equivalent to the (possibly infinite) set of pairs of (*input*, *function value*), we can take the union of σ_F and σ_P and call it sigma. Sigma applied to some box returns the set of boxes contained within it. So now we have a mechanism for getting at the subboxes of one box.

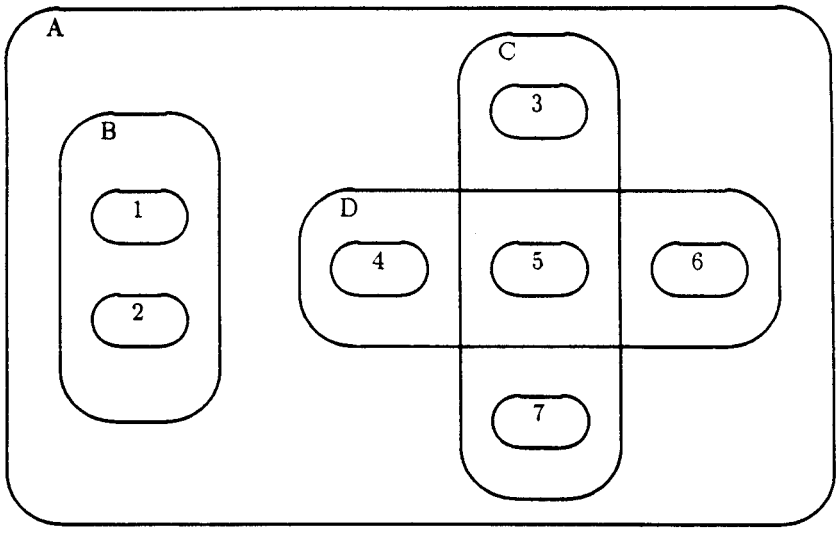

Figure 4: Illustration for the auxiliary definitions. Atomic boxes are labeled with numbers, non–atomic boxes with letters.

We can do something similar for sets of boxes. We define tau of a set of boxes ($\tau(X)$) to be the union of all subboxes. Tau-star returns a set containing all possible subboxes (those at every level in each input box). Technically, $\tau^*(X)$ returns the transitive closure of boxes in X with respect to the subbox operator σ. So σ and τ only return the subboxes at the next level down, but τ^* returns *all* possible subboxes.

Just a few more definitions before we continue. *ATOMS* is the set of security domain boxes that have no subboxes, i.e., the set of atomic boxes. Our pictures will contain not only boxes, but also arrows. *TYPES* will be the set of identifiers allowed on these arrows. Each arrow must join two boxes (a User box and a File box), have some identifier in *TYPES*, and can have positive or negative parity. The set of arrows used in a particular picture will be called *ARROWS*.

III.2 Auxiliary Definitions

Now that we have a formal representation for the objects in our pictures, we can construct predicates that talk about object interaction. In particular, we want to say how different boxes are related, and which arrows join which boxes. We present these set constructors and predicates below. They are illustrated in Figure 4 and Table 2. Free variables in the definitions below (x, y, P, P', N, N') range over elements in *BOXES*.

Recall that the shapes of boxes do not determine the final access matrix. Instead, the atoms contained in the boxes are important. The set constructor *members* gives us the set of all atoms contained within a box. The other set constructors use *members* in their definitions.

members(x): The set of atoms contained in (or equal to) x.

$$members(x) \triangleq \{\, a \mid a \in ATOMS \wedge a \in \tau^*(\{\, x \,\}) \,\}$$

Many pictures have boxes that nest in a hierarchical fashion. We use *inside* and *contains* to give us the descendants and ancestors of a particular box.

X	*members(X)*	*inside(X)*	*contains(X)*	*crisscrosses(X)*
1	{ 1 }	∅	{ 1, A, B }	∅
2	{ 2 }	∅	{ 2, A, B }	∅
3	{ 3 }	∅	{ 3, A, C }	∅
4	{ 4 }	∅	{ 4, A, D }	∅
5	{ 5 }	∅	{ 5, A, C, D }	∅
6	{ 6 }	∅	{ 6, A, D }	∅
7	{ 7 }	∅	{ 7, A, C }	∅
A	{ 1, 2, 3, 4, 5, 6, 7 }	{ 1, 2, 3, 4, 5, 6, 7, B, C, D }	{ A }	∅
B	{ 1, 2 }	{ 1, 2 }	{ A, B }	∅
C	{ 3, 5, 7 }	{ 3, 5, 7 }	{ A, C }	{ D }
D	{ 4, 5, 6 }	{ 4, 5, 6 }	{ A, D }	{ C }

Table 2: Some properties of the picture in Figure 4.

inside(x): The set of boxes whose atoms form a proper subset of those in x. In particular, $x \notin inside\ (x)$. The reader may find it helpful to read $y \in inside\ (x)$ as "box y is strictly inside box x."

$$inside(x) \triangleq \{\ b \mid b \in BOXES \wedge members(b) \subset members(x)\}$$

contains(x): The set of boxes whose atoms are at least those of x. In particular, $x \in contains\ (x)$. Note that for a given x, *inside* (x) and *contains* (x) are disjoint sets. Read $y \in contains\ (x)$ as "box y is or contains box x."

$$contains(x) \triangleq \{\ b \mid b \in BOXES \wedge members(x) \subseteq members(b)\}$$

We do not *require* strictly hierarchical pictures, however. So there will be some pictures that contain overlapping boxes. We define the set *crisscrosses(x)* to be all boxes that share some, but not all, of the atoms in x, and that are not *inside(x)*. We also have the relation $\bowtie$ which we define as *crisscrosses or equals*. These will be useful in the Closure Lemma below.

crisscrosses(x): The set of boxes which share some, but not all of their atoms with x. Note that this is symmetric, in that $X \in crisscrosses(Y) \Rightarrow Y \in crisscrosses(X)$. Also note that no box can be both inside and crisscrossing another box (e.g., in Figure 4, box B does not crisscross box A). Read $y \in crisscrosses\ (x)$ as "box y crisscrosses box x (and x crisscrosses y)."

$$crisscrosses(x) \triangleq \{\ b \mid\ b \in BOXES \wedge\ (members(x) \cap members(b) \neq \emptyset) \wedge (b \notin (inside(x) \uplus contains(x)))\}$$

$x \bowtie y$: x is equal to, or crisscrosses y. Note that this is not a transitive relation. Read $x \bowtie y$ as "box x is equal to or crisscrosses box y."

$$x \bowtie y \triangleq (members(x) = members(y) \vee x \in crisscrosses\ (y))$$

There are two final definitions. POS(P,P') and NEG(N,N'). These are true when a positive (negative) arrow connects boxes P and P' (N and N').

POS $^t(P, P')$: A positive arrow of type t exists between P and P'.

$$POS^t(P,\ P') \triangleq \langle P,\ P',\ t,\ pos\rangle \in ARROWS$$

NEG $^t(N,\ N')$: A negative arrow of type t exists between N and N'.

$$NEG^t(N,\ N') \triangleq \langle N,\ N',\ t,\ neg \rangle \in ARROWS$$

To make the interactions of these definitions clearer, we introduce the concept of *box level* and the Closure Lemma. Box level refers to the hierarchy imposed on boxes through containment. Two boxes are said to be *at the same level* if and only if $X \bowtie Y$.[5] If $X \in inside\ (Y)$, Y is said to have a *higher level* than box X, and X a *lower level* than box Y. In Figure 5, A and B have the same level, neither C nor D is related by level to any other box, F has a lower level than E, E has a higher level than F, and F has the same level as itself. It should be noted that box level does *not* provide a partial ordering of boxes, however.

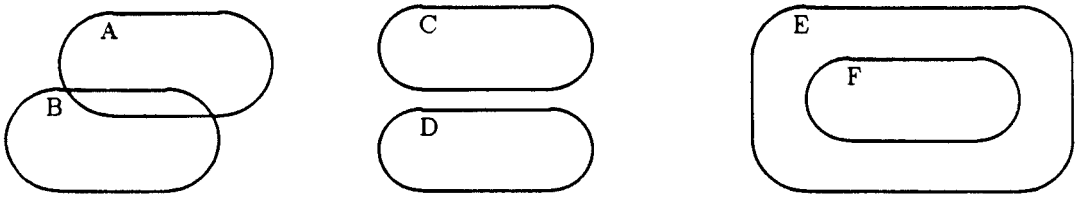

Figure 5: Illustration of *box level.*

The following lemma illustrates some relationships among these definitions.

Lemma 1 (Closure) *If two boxes B, B' both contain the same atomic box, then exactly one of $B \bowtie B'$, $B \in$ inside (B'), or $B' \in$ inside (B) is true.*

Proof We will show first that *at least one* of the conditions holds. Let B, B' be given such that $members(B) \cap members(B') \neq \emptyset$. We will show that the result holds for each of four independent cases, one of which must always hold. First suppose $members(B) \subset members(B')$; then by definition of *inside*, $B \in inside\ (B')$. Otherwise, suppose $members(B') \subset members(B)$; then $B' \in inside\ (B)$, also by definition of *inside*. If neither of these holds, suppose $members(B) = members(B')$; then $B \bowtie B'$ by definition of $\bowtie$. Finally, if none of these holds, we know $B \notin contains\ (B')$ since $members(B) \neq members(B')$ and $members(B) \not\subset members(B')$. From that we infer that $B \in crisscrosses\ (B')$ and thus $B \bowtie B'$. Now to see that *at most one* condition will hold, refer to the definitions of $\bowtie$, *inside*, and *crisscrosses*. ♠

IV Semantics

IV.1 Access Rights Matrix

The ultimate interpretation of a Miró picture in the security domain is an access-rights matrix. An access-rights matrix is a standard security entity that represents binary access relations between entities, such as the right for one entity to modify another.

The access-rights matrix Z is three-dimensional, taking a *Process*, a *File*, and a type of *Relation*, with possible values *pos*, *neg*, and *ambig*. The expressions below determine the value of a particular matrix element. Let t be the type of the relation, p an atomic box representing the process, and f an atomic box representing the file. The interpretation is that if $Z\ (p,\ f,\ t)$ is *pos* then process p can access file f according to relationship type t. If $Z\ (p,\ f,\ t)$ is *neg*, then p cannot access f according to t. If $Z\ (p,\ f,\ t)$ is *ambig*, the access cannot be determined. We want to detect and eliminate all such ambiguity in the matrix.

[5] Just as $\bowtie$ is not transitive, neither is *at the same level.*

Before going through the formal procedure for computing values in the matrix, consider an example. The access-rights matrix for the picture in Figure 2 is given in Table 3. Features of the elements in the matrix include the property that any relation not explicitly specified is given the value *neg*. This is a consequence of the clause labeled C in formula 2 below. So the negative arrow in the picture is not strictly necessary, but it is good "visual programming style" to make the absence of *read* rights explicit.

	/etc/passwd			/usr/alice/private		
	read	write	execute	read	write	execute
alice	pos	neg	neg	pos	pos	neg
bob	pos	neg	neg	neg	neg	neg
charlie	pos	neg	neg	neg	neg	neg

Table 3: Example of an access matrix; the matrix for Figure 2

In what follows, P and P' will identify the boxes at the tail and head, respectively, of a positive arrow, and N and N' will identify those at the tail and head of a negative arrow. If a positive and negative arrow both emanate from the same box, both P and N might label the same process box. Similarly, P' and N' might label the same file box. Boxed symbols (e.g., x) are used in the formulas below to name clauses for later reference, and have no semantic or logical interpretation.

$$Z(p,f,t) \text{ is } pos \text{ iff} \tag{1}$$

$$\boxed{A}$$

$$\exists_{P,P'}\ p \in members(P) \wedge f \in members(P') \wedge POS^t(P,\ P') \wedge$$
$$\forall_{N,N'}\ (p \in members(N) \wedge f \in members(N') \wedge NEG^t(N,\ N'))$$
$$\Rightarrow \neg \left[\begin{array}{ll} \boxed{1} & (P \trianglerighteq N \wedge P' \trianglerighteq N') \vee \\ \boxed{2} & N' \in inside\ (P') \vee \\ \boxed{3} & N \in inside\ (P) \end{array} \right]$$

Z is positive when the smallest enclosing boxes have only positive arrows; call these boxes P and P'. We require that no negative arrow join the following pairs of boxes: boxes at the same level as P and P' (case 1 above); one box at a lower level than P or P', and the other box at any level (cases 2 and 3 above).

$$Z(p,f,t) \text{ is } neg \text{ iff} \tag{2}$$

$$\boxed{B}$$

$$\exists_{N,N'}\ p \in members(N) \wedge f \in members(N') \wedge NEG^t(N,\ N') \wedge$$
$$\forall_{P,P'}\ (p \in members(P) \wedge f \in members(P') \wedge POS^t(P,\ P'))$$
$$\Rightarrow \neg \left[\begin{array}{ll} \boxed{1} & (P \trianglerighteq N \wedge P' \trianglerighteq N') \vee \\ \boxed{2} & P' \in inside\ (N') \vee \\ \boxed{3} & P \in inside\ (N) \end{array} \right]$$
$$\vee \boxed{C}$$
$$\forall_{B,B'}\ B \in contains\ (p) \wedge B' \in contains\ (f) \Rightarrow$$
$$\neg POS^t(B,\ B') \wedge \neg NEG^t(B,\ B')$$

Z is negative when the smallest enclosing boxes have only negative arrows (call these boxes N and N'), or when no surrounding boxes are connected by arrows. In the former case, we require

that no positive arrow join the following pairs of boxes: boxes at the same level as N and N' (case 1 above); one box at a lower level than N or N', and the other box at any level (cases 2 and 3 above).

$$Z(p, f, t) \text{ is } \textit{ambig} \text{ otherwise.} \tag{3}$$

The value of an element of Z is ambiguous when neither a positive nor a negative relationship holds. An explicit derivation of those pictures that are ambiguous follows.

IV.2 Uniqueness

Before we derive the explicit conditions for ambiguity, let us first ensure that the other matrix elements are unique. That is, we intend to show that no two atoms can have both a *pos* and *neg* relationship with the same type.

Lemma 2 *A relation between two atomic boxes may not be both* pos *and* neg.

Proof Let atomic boxes p and f, and relation type t be given. We will prove the lemma by contradiction, using formulas (1) and (2) above. Suppose *Z(p, f, t)* is both *pos* and *neg*. Then A (in formula (1)) is true, and from A we know there are boxes P and P' containing p and f with a positive arrow connecting them; let us choose such boxes and call them P and P'. We can infer that C (in formula (2)) is false because P and P' exist. Thus B must be true since we assumed that formula (2) was true. Now from B we may choose N and N' containing p and f with a negative arrow connecting them. Using this we can determine that clauses A1 through A3 must be false, in order for A to be true. Likewise, clauses B1 through B3 must be false because P and P' exist and have a positive arrow. Returning to the clauses in A, we have the following results: by ¬ A1 we know that either $N \not\trianglelefteq P$ or $N' \not\trianglelefteq P'$; by ¬ A2 we know that $N' \notin \textit{inside}\ (P')$; by ¬ A3 we know that $N \notin \textit{inside}\ (P)$. Suppose $N \not\trianglelefteq P$. Then by Lemma 1 and ¬ A3, $P \in \textit{inside}\ (N)$. But $P \notin \textit{inside}\ (N)$ by ¬ B3, a contradiction. Suppose instead that $N' \not\trianglelefteq P'$. Then by Lemma 1 and ¬ A2, $P' \in \textit{inside}\ (N')$. But $P' \notin \textit{inside}\ (N')$ by ¬ B2, a contradiction. ♠

V Ambiguity

We described above explicit conditions for pictures whose contents are known to have a positive or negative relationship. In this section we show that ambiguity can be intuitively defined, derive explicit conditions for pictures whose contents have an ambiguous relationship, and demonstrate these conditions by drawing the corresponding pictures.

V.1 Intuition

We now use the box level concept to state the intuitive definition of ambiguity.

A relation between p and f is ambiguous when no single arrow has the smallest scope of all (similarly typed) arrows surrounding p and f. These cases are illustrated in Figure 6 and Figure 7.

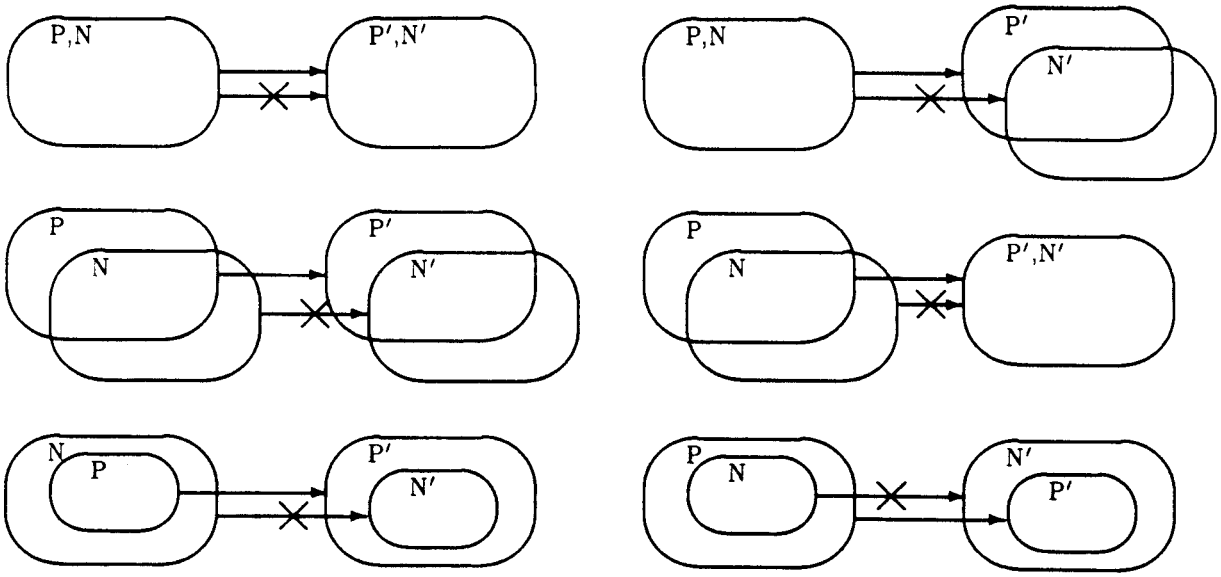

Figure 6: Ambiguous Pictures

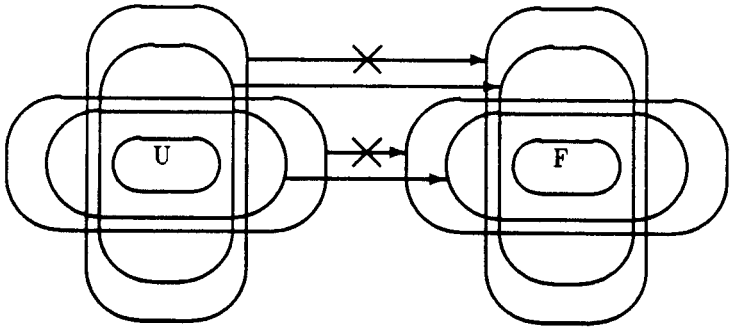

Figure 7: Another kind of ambiguous picture.

Figure 6 shows the kinds of ambiguity most likely to arise in typical usage. Since crisscrossing boxes have the same level, these pictures all share the property that no single arrow has both ends attached to the smallest enclosing boxes. Each case has both a positive and negative arrow attached to the boxes at the lowest level.

In Figure 7, we see another kind of ambiguity. If we consider just the vertical boxes, we see that there is a single positive arrow connecting the smallest boxes; likewise for the horizontal boxes. So one might think that this picture unambiguously yields a positive relation between atoms in these boxes. However, when we consider the picture as a whole, no *single* positive arrow overrides *all* of the negative arrows, so this picture is ambiguous. We include this kind of ambiguity because pictures of this type are difficult to interpret at a glance, and we wish to eliminate such pictures from our language.

V.2 Derivation

We will show that the intuitive definition of ambiguity given above follows logically from the other definitions. We begin by defining ambiguity as the condition when neither a positive nor a negative relation holds.

$$Z(p,f,t) \text{ is } \textit{ambig} \text{ iff } \neg(Z(p,f,t) \text{ is } \textit{pos}) \wedge \neg(Z(p,f,t) \text{ is } \textit{neg}) \tag{4}$$

We may now use the definitions in section IV.1, and apply one of DeMorgan's laws to the negative case, to expand formula (4) into the following:

$$\begin{aligned}
&Z(p,f,t) \text{ is } \textit{ambig} \text{ iff} \\
&\quad \neg \exists_{P,P'}\, p \in \textit{members}(P) \wedge f \in \textit{members}(P') \wedge \textit{POS}^t(P,\, P') \wedge \\
&\qquad \forall_{N,N'}\, (p \in \textit{members}(N) \wedge f \in \textit{members}(N') \wedge \textit{NEG}^t(N,\, N')) \\
&\qquad\qquad \Rightarrow \neg \left[\begin{array}{l} \boxed{1}\ (P \bowtie N \wedge P' \bowtie N') \vee \\ \boxed{2}\ N' \in \textit{inside}\ (P') \vee \\ \boxed{3}\ N \in \textit{inside}\ (P) \end{array} \right] \\
&\qquad\qquad\qquad \wedge \\
&\neg \exists_{N,N'}\, p \in \textit{members}(N) \wedge f \in \textit{members}(N') \wedge \textit{NEG}^t(N,\, N') \wedge \\
&\qquad \forall_{P,P'}\, (p \in \textit{members}(P) \wedge f \in \textit{members}(P') \wedge \textit{POS}^t(P,\, P')) \\
&\qquad\qquad \Rightarrow \neg \left[\begin{array}{l} \boxed{1}\ (P \bowtie N \wedge P' \bowtie N') \vee \\ \boxed{2}\ P' \in \textit{inside}\ (N') \vee \\ \boxed{3}\ P \in \textit{inside}\ (N) \end{array} \right] \\
&\qquad\qquad\qquad \wedge \\
&\qquad \neg \forall_{B,B'}\, B \in \textit{contains}\ (p) \wedge B' \in \textit{contains}\ (f) \Rightarrow \\
&\qquad\qquad \neg \textit{POS}^t(B,\, B') \wedge \neg \textit{NEG}^t(B,\, B')
\end{aligned} \tag{5}$$

Now we can use properties of first-order logic to push the negations through and express the condition for ambiguity as follows:

$$\begin{aligned}
&Z(p,f,t) \text{ is } \textit{ambig} \text{ iff} \\
&\qquad\qquad \boxed{\neg A} \\
&\forall_{P,P'}\, \neg(p \in \textit{members}(P) \wedge f \in \textit{members}(P') \wedge \textit{POS}^t(P,\, P')) \vee \\
&\quad \exists_{N,N'}\, p \in \textit{members}(N) \wedge f \in \textit{members}(N') \wedge \textit{NEG}^t(N,\, N') \wedge \\
&\qquad\qquad \left[\begin{array}{l} \boxed{1}\ (P \bowtie N \wedge P' \bowtie N') \vee \\ \boxed{2}\ N' \in \textit{inside}\ (P') \vee \\ \boxed{3}\ N \in \textit{inside}\ (P) \end{array} \right] \\
&\qquad\qquad \wedge \boxed{\neg B}
\end{aligned} \tag{6}$$

$$\forall_{N,N'} \neg (p \in members(N) \wedge f \in members(N') \wedge NEG^t(N, N')) \vee$$
$$\exists_{P,P'} p \in members(P) \wedge f \in members(P') \wedge POS^t(P, P') \wedge$$
$$\left[\begin{array}{l} \boxed{1} \ (P \bowtie N \wedge P' \bowtie N') \vee \\ \boxed{2} \ P' \in inside\ (N') \vee \\ \boxed{3} \ P \in inside\ (N) \end{array} \right]$$
$$\wedge \boxed{\neg C}$$
$$\exists_{B,B'} B \in contains\ (p) \wedge B' \in contains\ (f) \wedge$$
$$(POS^t(B, B') \vee NEG^t(B, B'))$$

Lemma 3 *If the relation between* **p** *and* **f** *is ambiguous according to type* **t**, *then there must be at least two pairs of boxes surrounding both* **p** *and* **f**, *one pair connected by a positive arrow and the other by a negative arrow.*

Proof Suppose the relation between **p** and **f** is ambiguous according to type **t**, i.e., *Z (p, f, t)* is *ambig*. Then formula (6) is true, and in particular the conjuncts labeled $\boxed{\neg A}$ $\boxed{\neg B}$ and $\boxed{\neg C}$ are true. By $\boxed{\neg C}$ we may choose boxes B and B' which have either a positive or negative relation. Suppose the relation is positive. Then by instantiating P and P' to B and B' in $\boxed{\neg A}$ we can derive the existence of boxes with a negative relation. Suppose instead that B and B' have a negative relation. Then by instantiating N and N' to B and B' in $\boxed{\neg B}$ we can derive the existence of boxes with a positive relation. ♠

Figure 8 shows all of the pictures which satisfy the three clauses in $\boxed{\neg A}$, each one labeled with the disjuncts it satisfies and the value it would have according to these semantics. If we now inspect all pictures which satisfy some disjunct in $\boxed{\neg A}$ as well one in $\boxed{\neg B}$, we will find that they are exactly those pictures listed in Figure 6, which are all ambiguous.

VI Conclusions

This paper gave a precise syntax and semantics for the core of the Miró language. It also gave a precise definition of ambiguity. Determining whether a picture is ambiguous can be regarded as a *static semantic check*. Miró has two additional classes of static semantic checks dealing with *types* (of boxes and arrows) and *constraints* (on boxes and arrows). Type-checking a picture is similar to type-checking a program in a standard programming language. Constraint-checking a picture intuitively involves pattern-matching on pictures as well as checking boolean predicates. We have an informal definition of these checks and are currently formulating their precise semantics.

Miró solves several important problems in security. It provides, for the first time, a tool for configuring and visualizing complicated security constraints. This tool allows precise definitions of security environments in a convenient mathematical notation — a major advance over previous one-dimensional (i.e., textual) logic-based approaches (such as [3,4,5,6]). It is a practical tool designed to be used by people who are actively enforcing security constraints in real environments.

In the future, we intend to apply the Miró language to domains outside of security, e.g., concurrency, where we would reinterpret the meaning of boxes and arrows. Ideally, we would like to make that part of the semantics given here that is independent of security a separate library that can be reused by many entities, and reinterpret only that part that is dependent on the specific domain. Finally, we would like to explore the possibility of using a visual approach to giving semantics instead of the standard denotational approach as presented in this paper.

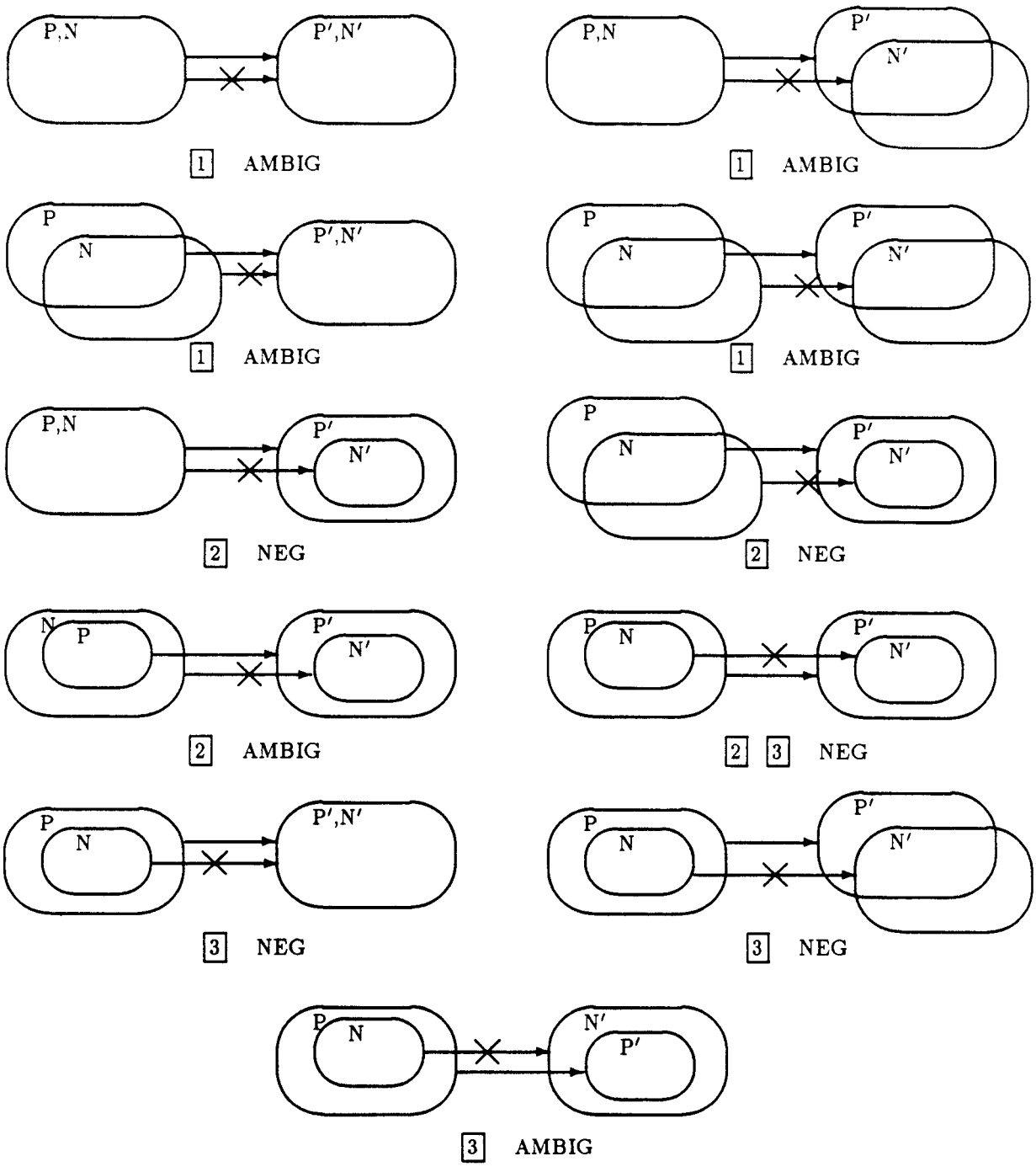

Figure 8: Pictures which contain positive arrows but do not have a positive interpretation.

VII Acknowledgments

We thank David Harel for his inspiration and enthusiastic support for our work. We also wish to thank Amy Moormann, Allan Heydon, and Kenneth McMillan for their comments on earlier versions of this paper.

This research was sponsored by IBM and the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976, under contract F33615-87-C-1499 monitored by the Avionics Laboratory, Air Force Wright Aeronautical Laboratories, Wright-Patterson AFB. Additional support for M. Maimone was provided by the Office of Naval Research under contract N00014-88-K-0641; for J.D. Tygar by a National Science Foundation Presidential Young Investigator grant; and for J.M. Wing by the National Science Foundation under grant CCR-8620027.

The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

VIII References

[1] J. D. Tygar and J. M. Wing, "Visual Specification of Security Constraints," in *Proceedings of the 1987 Workshop on Visual Languages*, (Linkoping, Sweden), Aug 1987.

[2] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, 1987.

[3] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Mathematical Foundations (3 volumes)," Tech. Rep. AD-770 768, AD-771 543, AD-780 528, Mitre Corporation, Nov 1973.

[4] D. Good, R. Cohen, C. Hoch, L. Hunter, and D. Hare, "Report on the Language Gypsy, Version 2.0," Tech. Rep. ICSCA-CMP-10, Certifiable Minicomputer Project, The University of Texas at Austin, September 1978.

[5] T. Benzel, "Analysis of a Kernel Verification," in *Proceedings of the 1984 Symposium on Security and Privacy*, (Oakland, California), pp. 125–131, May 1984.

[6] P. G. Neumann, R. S. Boyer, R. J. Feiertag, K. N. Levitt, and L. Robinson, "A Provably Secure Operating System: The System, Its Applications, and Proofs, Second Edition," Tech. Rep. CSL-116, SRI, May 1980.