FORMAL SEMANTICS FOR VISUAL SPECIFICATION OF SECURITY

MARK W. MAIMONE, J. D. TYGAR, AND JEANNETTE M. WING

1. *Introduction*

Visual languages, like all languages, need a formal semantics. This chapter 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"* 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 described in Refs. 1, 3, 8, 9. Unlike the development of many other visual languages, our design

* 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.

MARK W. MAIMONE, J. D. TYGAR, and JEANNETTE M. WING • School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213. 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 U.S. Government.

proceeds in tandem with the development of its formal semantics. We therefore benefit from not only the concision of visual notation, but the precision of a rigorous semantics.

2. *Informal Description*

File system 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, the auxiliary information being 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 when access is permitted without inspecting the encoded information stored with each file or process. In UNIX,* 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 consider other UNIX features such as "set user ID" bits and "sticky" bits.

The constructs of Miró are simple: *boxes* and *arrows*, each optionally labeled.† In giving such constructs a semantics, it is essential to provide an interpretation for individual boxes and arrows and their compositions when they form *pictures*. Depending on the class of properties of interest, the interpretation of the boxes and arrows will change. In this chapter, we present a complete interpretation for the Miró constructs in the domain of file

* UNIX is a trademark of AT & T.

† Boxes are drawn as rectangles with rounded corners, inspired by Harel's Statechart notation (see Ref. 2).

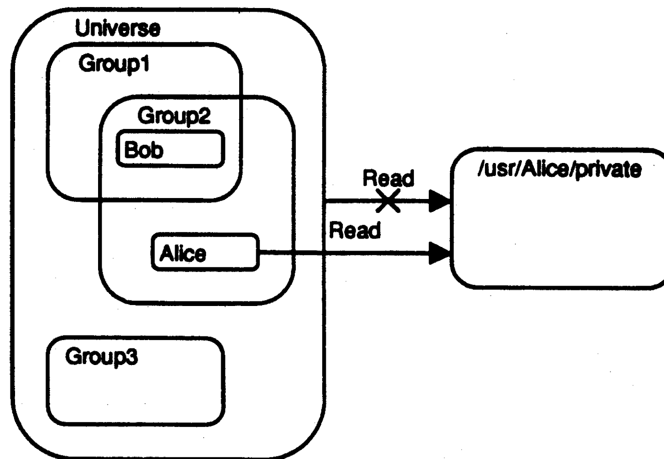

FIGURE 1. A sample Miró security specification.

system security properties. The underlying semantic model for security is simple: an *access-rights* matrix, where each *user* (represented by a process) has a (possibly empty) set of *rights* governing access (e.g., Read, Write, Execute) to each *file* (or program). Informally, boxes represent individual users and files or collections of users and files; arrows represent rights. Since we have *negative* arrows as well (expressed as arrows with slashes through them), we can express the absence of rights. The introduction of negative arrows causes some nontriviality in our semantics, since 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 left-hand 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.* The right-hand 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. Since the boxes for Alice, Bob, and Charlie are all in the Universe box, and there is a Read arrow from Universe to /etc./passwd, all of these users can read /etc./passwd. What of the rest of the users—do they or do they not have Write access to Alice's

* We do not address the property that a user must belong to at least one group here, though Miró does provide for this expressibility (see Ref. 3).

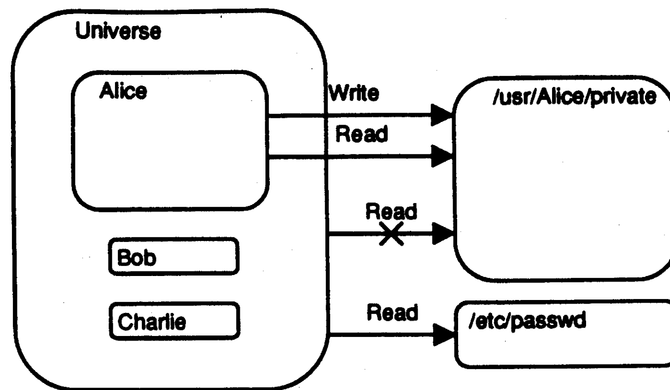

FIGURE 2. Another security specification.

private directory? The answer is no, they do not. We define the absence of an appropriate arrow to mean no access. Both Fig. 1 and Fig. 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 Fig. 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 chapter 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 nonatomic objects (e.g., Group2)? (What do "atomic" and "nonatomic" 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 formal semantics can answer precisely.

In what follows, we capture these visual notions in a logical setting: the set theoretic notions hinted at in Fig. 1 are made explicit. We formally present the syntactic domains in Section 3, and the semantic domains

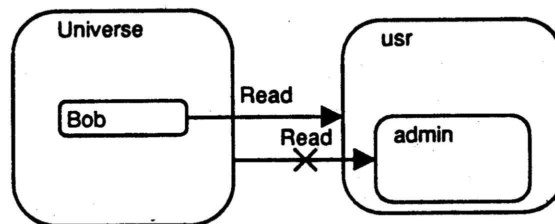

FIGURE 3. An ambiguous security specification.

(access-rights matrix) and interpretation in Section 4. We present a careful treatment of ambiguity in Section 5, and close with some final remarks about current and future work in Section 6.

3. 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.

Before beginning the presentation, we clarify some aspects of our notation. Throughout the chapter, 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.

3.1. Explanation of the Syntax Table

Consider the kinds of things that make up a picture. There are two types of boxes, User/Process and File boxes. Each 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 types of boxes, let us look carefully at just what these boxes stand for. We will ignore the fact that there are two types of boxes for now, and just consider generic boxes (i.e., we will leave the identifiers uninstantiated).

Just what does a box represent? The simplest kind of box, the atomic box (one that contains no other boxes), represents exactly one column in an access matrix; that column has the name of the box. The box's size and location do not affect the matrix, as long as it is not enclosed within another (nonatomic) box. 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.

TABLE 1
Miró Syntactic Entities; Examples are from Fig. 1 and 2

Entity	Symbol	Example
Set of file identifiers	F_{id}	{/usr/Alice/private, /etc/passwd}
Set of user/process identifiers	P_{id}	{Alice, Bob, Charlie}
Box constructor:	$B_I^0 = \{\langle \varnothing, id \rangle \mid id \in I\}$	B_I^0 is the set of atoms
Set of boxes with identifiers in I, where $I = F_{id}$ or P_{id}	$B_I^1 = B_I^0 \cup \{\langle x, id \rangle \mid x \in (2^{B_I^0} - \{\varnothing\}) \wedge id \in I\}$	B_I^1 adds the set of boxes containing atoms
	$B_I^i = \bigcup_{j=0}^{i-1} B_I^j \cup \{\langle x, id \rangle \mid x \in (2^{B_I^{i-1}} - \{\varnothing\}) \wedge id \in I\}$	$\langle \varnothing, \text{Alice} \rangle \simeq$ [Alice] $\langle \{\langle \varnothing, \text{Alice} \rangle, \langle \varnothing, \text{Bob} \rangle\}, \text{Group 2} \rangle$
Set of file boxes	$F \subseteq \bigcup_{i=0}^{\infty} B_{F_{id}}^i$	
Set of user/process boxes	$P \subseteq \bigcup_{i=0}^{\infty} B_{P_{id}}^i$	
Set of all boxes	$\text{BOXES} = F \cup P$	{ [Alice] , [Group 2 [Alice] [Bob]] }
Subbox operators	$\sigma_F : F \to 2^F$ $\sigma_P : P \to 2^P$ $\sigma_I(\langle X, id \rangle) = X$ $\sigma = \sigma_F \cup \sigma_P$ $\tau : 2^{\text{BOXES}} \to 2^{\text{BOXES}}$ $\tau(X) = \bigcup_{x \in X} \sigma(x)$ $\tau^*(X) = \bigcup_{j=0}^{\infty} \tau^j(X)$	
Atomic boxes	$\text{ATOMS} = \{x \mid x \in \text{BOXES} \wedge \sigma(x) = \varnothing\}$	
Set of relation types	TYPES	{Read, Write, Execute}
Arrows	$\text{ARROWS} \subseteq P \times F \times \text{TYPES} \times \{\text{pos}, \text{neg}\}$	

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 their intersection, 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 box layer $x(B_I^x)$ inductively, in terms of increasingly larger enclosing sets. B_I^0 is the set of atomic boxes, and B_I^1 is the set of atomic boxes *plus* all boxes that contain only atomic boxes. We continue in the same fashion, adding at each layer boxes that contain all of the boxes listed at the previous layer. Members of B_I^2 are boxes that contain either atoms or boxes containing atoms. Formally, $B_I^2 = B_I^0 \cup B_I^1 \cup \{\langle x, id \rangle \mid x \in (2^{B_I^1} - \{\varnothing\}) \wedge id \in I\}$. This definition ensures that our boxes are well-founded; no box is inside itself.

There is a quirk in the formalism. In order to make the mathematical expression for *contains all boxes at lower layers* more concise, we replicate each earlier layer in the later ones. Every earlier set of boxes is included in the later ones; more formally, $\forall_i \forall_{j \text{ where } 0 \leq j < i} B_I^j \subset B_I^i$. Thus we can conveniently use the power set notation $2^{B_x^{i-1}}$ (see Table 1). 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_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_I^∞ is the set of all possible boxes, but to avoid trying to prove that such a limit exists, we instead define the set of all boxes to be the union of all box layers. Since each earlier layer is contained in all later ones, the union is unnecessary for a particular (finite) picture, but most necessary for describing the infinite set of all pictures.

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 the file identifiers and the user/process identifiers in turn, giving us sets F and P of boxes. F and P are disjoint; File boxes contain only File boxes, and User/Process boxes contain only User/Process 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 that will be useful later on: subbox (σ) and all-subboxes (τ^*, defined below). Subbox 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 $\langle$input,

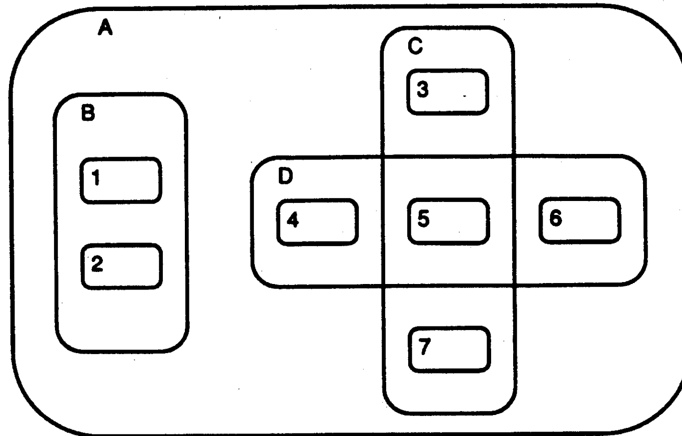

FIGURE 4. Illustration for the auxiliary definitions. Atomic boxes are labeled with numbers, nonatomic boxes with letters.

function value⟩, we can take (unsubscripted) σ to be the union of σ_F and σ_P; σ 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.

We can do something similar for sets of boxes. We define τ of a set of boxes X to be the union of all subboxes in X. τ^* returns a set containing all possible subboxes (those at every layer 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 layer 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/Process 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.

3.2. *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 Fig. 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

TABLE 2
Some Properties of the Picture in Fig. 4

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}

members(x) gives us the set of all atoms contained within box x. The other set constructors use members (x) in their definitions.

DEFINITION 1: *members*(x). The set of atoms contained in (or equal to) x.

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

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.

DEFINITION 2: *inside*(x). The set of boxes whose atoms form a proper subset of those in x. In particular, $x \notin \text{inside}(x)$. The reader may find it helpful to read $y \in \text{inside}(x)$ as "box y is strictly inside box x."

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

DEFINITION 3: *contains*(x). The set of boxes whose atoms are at least those of x. In particular, $x \in \text{contains}(x)$. Note that for a given x, inside(x) and contains(x) are disjoint sets. Read $y \in \text{contains}(x)$ as "box y is or contains box x."

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

We do not *require* strictly hierarchical pictures, however; pictures may contain overlapping boxes. We define the set *crisscrosses*(x) and operator $\bowtie$ to represent overlapping boxes. These will be useful in the Closure Lemma below.

DEFINITION 4: *crisscrosses*(x). The set of boxes that share some, but not all of their atoms with x. Note that this is symmetric, in that

$x \in \text{crisscrosses}(y) \Rightarrow y \in \text{crisscrosses}(x)$. Also note that no box can be both inside and crisscrossing another box (e.g., in Fig. 4, box B does not crisscross box A). Read $y \in \text{crisscrosses}(x)$ as "box y crisscrosses box x (and x crisscrosses y)."

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

DEFINITION 5: $x \underline{\bowtie} y$. x is equal to, or crisscrosses y. Note that this is not a transitive relation. For example, construct three boxes x, y, and z. Put y inside x, and put z crisscrossing both of these. Then $x \underline{\bowtie} z$ and $z \underline{\bowtie} y$, but $x \not\underline{\bowtie} y$. Read $x \underline{\bowtie} y$ as "box x is equal to or crisscrosses box y."

$$x \underline{\bowtie} y \triangleq (\text{members}(x) = \text{members}(y) \vee x \in \text{crisscrosses}(y))$$

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').

DEFINITION 6: $POS^t(P, P')$. A positive arrow of type t exists between P and P'.

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

DEFINITION 7: $NEG^t(N, N')$. A negative arrow of type t exists between N and N'.

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

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 $x \underline{\bowtie} y$.* If $x \in \text{inside}(y)$, y is said to have a *higher level* than box x, and x a *lower level* than box y. In Fig. 5, A and B have the same level, neither C nor D is related by level to any other box (since they have no members in common), 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 *at the same or*

* Just as $\underline{\bowtie}$ is not transitive, neither is *at the same level*. Note that the definition of box level should not be confused with that of box layer, introduced on page 103.

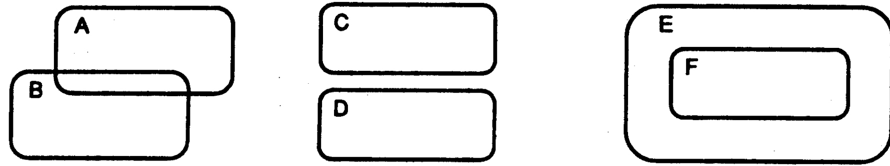

FIGURE 5. Illustration of *box level.*

lower level does *not* provide a partial ordering of boxes. 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 $B \bowtie B'$, $B \in \text{inside}(B')$, or $B' \in \text{inside}(B)$ is true.*

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

4. Semantics

4.1. Access Rights Matrix

The 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 $\mathcal{Z}$ is three dimensional, with axes being *Processes*, *Files*, and types of *Relations* (or *Access-Rights*). Entries in the matrix range over 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 user/process, and f an atomic box representing the file. The interpretation is that if $\mathcal{Z}(p, f, t)$ is *pos* then user/process p can access file f according to relationship type t. If $\mathcal{Z}(p, f, t)$ is *neg*, then p cannot

TABLE 3
Example of an Access Matrix; the Matrix for Fig. 2

	/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

access f according to t. If $\mathcal{Z}(p, f, t)$ is *ambig*, the access cannot be determined. We want to detect and eliminate all such ambiguity in the matrix.

Before going through the formal procedure for computing values in the matrix, consider an example. The access-rights matrix for the picture in Fig. 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 $\boxed{C}$ 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.

In what follows, P and P' will identify the boxes at the tail and head, respectively, of a positive arrow, and $\mathcal{N}$ and $\mathcal{N}'$ 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 $\mathcal{N}$ would label the same user/process box. Similarly, P' and $\mathcal{N}'$ might label the same file box. Boxed symbols (e.g., $\boxed{x}$) are used in the formulas below to name clauses for later reference, and have no semantic or logical interpretation.

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

$$\boxed{A}$$

$$\exists_{P,P'}\ p \in \text{members}(P) \wedge f \in \text{members}(P') \wedge \text{POS}^t(P, P')$$

$$\wedge\ \forall_{\mathcal{N},\mathcal{N}'}\ (p \in \text{members}(\mathcal{N}) \wedge f \in \text{members}(\mathcal{N}') \wedge \text{NEG}^t(\mathcal{N}, \mathcal{N}'))$$

$$\Rightarrow \neg \begin{bmatrix} \boxed{1}\ (P \bowtie \mathcal{N} \wedge P' \bowtie \mathcal{N}') \vee \\ \boxed{2}\ \mathcal{N}' \in \text{inside}(P') \vee \\ \boxed{3}\ \mathcal{N} \in \text{inside}(P) \end{bmatrix}$$

$\mathcal{Z}$ 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 $\boxed{1}$

above); one box at a lower level than P or P', and the other box at any level (cases [2] and [3] above).

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

$$\boxed{B}$$

$$\exists_{\mathcal{N},\mathcal{N}'}\ p \in \text{members}(\mathcal{N}) \wedge f \in \text{members}(\mathcal{N}') \wedge \text{NEG}^t(\mathcal{N}, \mathcal{N}')$$

$$\wedge\ \forall_{P,P'}\ (p \in \text{members}(P) \wedge f \in \text{members}(P') \wedge \text{POS}^t(P, P'))$$

$$\Rightarrow \neg \begin{bmatrix} \boxed{1}\ (P \bowtie \mathcal{N} \wedge P' \bowtie \mathcal{N}') \vee \\ \boxed{2}\ P' \in \text{inside}(\mathcal{N}') \vee \\ \boxed{3}\ P \in \text{inside}(\mathcal{N}) \end{bmatrix}$$

$$\vee\ \boxed{C}$$

$$\forall_{B,B'}\ B \in \text{contains}(p) \wedge B' \in \text{contains}(f) \Rightarrow$$

$$\neg \text{POS}^t(B, B') \wedge \neg \text{NEG}^t(B, B')$$

$\mathcal{Z}$ is negative when the smallest enclosing boxes have only negative arrows (call these boxes $\mathcal{N}$ and $\mathcal{N}'$), 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 $\mathcal{N}$ and $\mathcal{N}'$ (case [1] above); one box at a lower level than $\mathcal{N}$ or $\mathcal{N}'$, and the other box at any level (cases [2] and [3] above).

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

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

4.2. 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 $\mathcal{Z}(p, f, t)$ 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 $\boxed{C}$ [in formula (2)] is false because P and P' exist. Thus $\boxed{B}$ must be true since we assumed that formula (2) was true. Now from $\boxed{B}$ we may choose $\mathcal{N}$ and $\mathcal{N}'$ containing p and f with a negative arrow connecting them. Using this we can determine that clauses $\boxed{A1}$ through $\boxed{A3}$ must be false in order for $\boxed{A}$ to be true. Likewise, clauses $\boxed{B1}$ through $\boxed{B3}$ must be false because P and P' exist and have a positive arrow. Returning to the clauses in $\boxed{A}$, we have the following results: by $\neg\boxed{A1}$ we know that either $\mathcal{N} \not\bowtie P$ or $\mathcal{N}' \not\bowtie P'$; by $\neg\boxed{A2}$ we know that $\mathcal{N}' \notin \text{inside}(P')$; by $\neg\boxed{A3}$ we know that $\mathcal{N} \notin \text{inside}(P)$. Suppose $\mathcal{N} \not\bowtie P$. Then by Lemma 1 and $\neg\boxed{A3}$, $P \in \text{inside}(\mathcal{N})$. But $P \notin \text{inside}(\mathcal{N})$ by $\neg\boxed{B3}$, a contradiction. Suppose instead that $\mathcal{N}' \not\bowtie P'$. Then by Lemma 1 and $\neg\boxed{A2}$, $P' \in \text{inside}(\mathcal{N}')$. But $P' \notin \text{inside}(\mathcal{N}')$ by $\neg\boxed{B2}$, a contradiction. ■

5. *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.

5.1. *Intuition*

Before defining ambiguity, we introduce the notion of *smallest scope* for arrows. This is based on the *box level* concept defined in Section 3.2.

Let boxes p and f, and arrow type t be given. Construct the set $A \subseteq \text{ARROWS}$ of t-arrows such that for each $a \in A$, the tail box of a contains p, and the head box contains f. That is, A is the set of arrows (of type t) connecting all boxes around p and f. Then some arrow has the *smaller scope* than all arrows in A if and only if its tail (and head) box has lower level than all tail (and head) boxes mentioned in A. For example, consider the Read arrows in Fig. 2. The Read arrow connecting Alice and /usr/Alice/private has the smallest scope, since the only other Read arrow has Universe as its tail box (Universe has a higher level than Alice).

We now state an informal definition of ambiguity:

DEFINITION 8. A relation between p and f is ambiguous when no *single* arrow has smaller scope than all (similarly typed) arrows of opposite parity surrounding p and f. A picture is ambiguous when there exists some ambiguous relation between atoms within it. These cases are illustrated in Figs. 6 and 7.

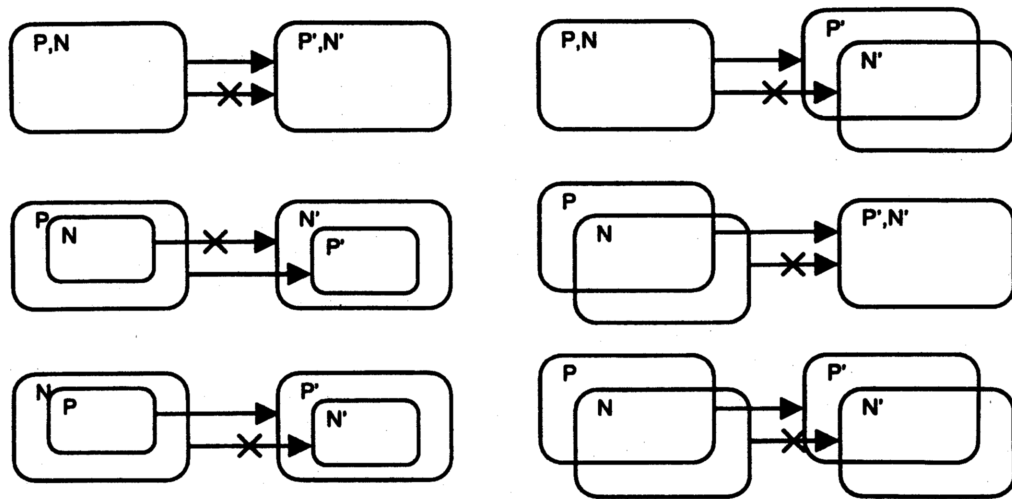

FIGURE 6. Ambiguous pictures.

Figure 6 shows the kinds of ambiguity most likely to arise in typical usage. The pictures on the left are inherently ambiguous, and those on the right are ambiguous when their boxes share some atoms. Suppose that P and $\mathcal{N}$, and P' and $\mathcal{N}'$, share at least one atom. Since crisscrossing boxes have the same level, each of these pictures has 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.

Regrettably, even the meaning of a picture that is *formally* unambiguous may not be immediately clear to a human reader. For instance, suppose that Fig. 7 were unambiguous; then what is the relationship between U and F? Conversely, a picture that is defined to be ambiguous may have an "obvious" interpretation (for instance, one might assume that positive arrows always override negative arrows in Fig. 6). We take a conservative approach to ambiguity: by increasing the number of ambiguous pictures, we reduce the number of potentially confusing pictures that can be expressed in our language.

Figure 7 shows that ambiguity cannot be determined locally. If we con-

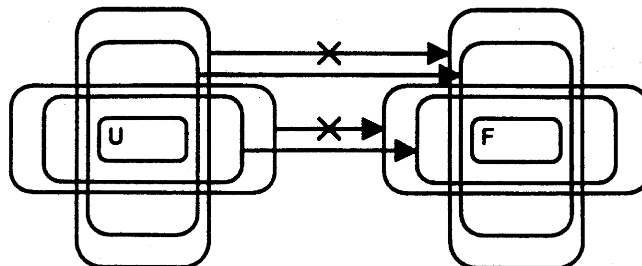

FIGURE 7. Another kind of ambiguous picture.

sider 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 defined to be 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.

5.2. 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:

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

We may now use the definitions in Section 4.1, and apply one of De Morgan's laws to the negative case, to expand formula (4) into the following:

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

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

$$\mathcal{Z}(p, f, t) \text{ is ambig iff} \tag{6}$$

$$\boxed{\neg A}$$

$$\forall_{P,P'} \neg(p \in \text{members}(P) \wedge f \in \text{members}(P') \wedge \text{POS}^t(P, P'))$$
$$\vee \exists_{\mathcal{N},\mathcal{N}'}\, p \in \text{members}(\mathcal{N}) \wedge f \in \text{members}(\mathcal{N}') \wedge \text{NEG}^t(\mathcal{N}, \mathcal{N}') \wedge$$
$$\begin{bmatrix} \boxed{1}\ (P \bowtie \mathcal{N} \wedge P' \bowtie \mathcal{N}') \vee \\ \boxed{2}\ \mathcal{N}' \in \text{inside}(P') \vee \\ \boxed{3}\ \mathcal{N} \in \text{inside}(P) \end{bmatrix}$$

$$\wedge\ \boxed{\neg B}$$

$$\forall_{\mathcal{N},\mathcal{N}'} \neg(p \in \text{members}(\mathcal{N}) \wedge f \in \text{members}(\mathcal{N}') \wedge \text{NEG}^t(\mathcal{N}, \mathcal{N}'))$$
$$\vee \exists_{P,P'}\, p \in \text{members}(P) \wedge f \in \text{members}(P') \wedge \text{POS}^t(P, P') \wedge$$
$$\begin{bmatrix} \boxed{1}\ (P \bowtie \mathcal{N} \wedge P' \bowtie \mathcal{N}') \vee \\ \boxed{2}\ P' \in \text{inside}(\mathcal{N}') \vee \\ \boxed{3}\ P \in \text{inside}(\mathcal{N}) \end{bmatrix}$$

$$\wedge\ \boxed{\neg C}$$

$$\exists_{B,B'}\, B \in \text{contains}(p) \wedge B' \in \text{contains}(f)$$
$$\wedge\ (\text{POS}^t(B, B') \vee \text{NEG}^t(B, B'))$$

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., $\mathcal{Z}(p, f, t)$ 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' that 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 $\mathcal{N}$ and $\mathcal{N}'$ 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 that satisfy the three clauses in $\boxed{\neg A}$, each one labeled with the disjuncts it satisfies and the value it would have

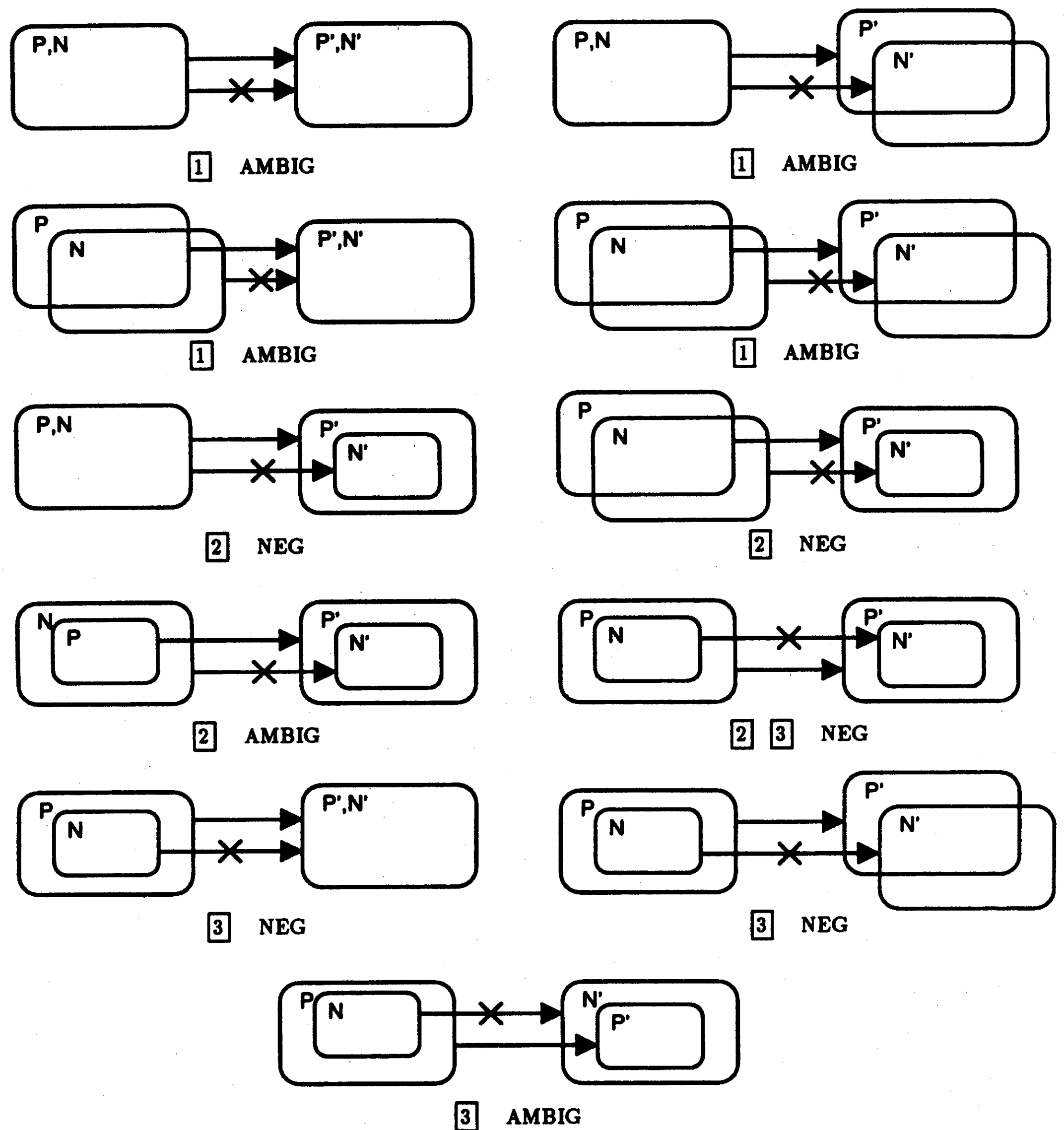

FIGURE 8. Pictures that contain positive arrows, but do not have a positive relationship. Boxed numbers refer to the disjuncts in part $\boxed{\neg A}$ of formula (6); the text gives the relation between each pair of atoms (where one atom is contained in P and N, and the other is in P' and N').

according to these semantics. If we now inspect all pictures that 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 Fig. 6, which are all ambiguous.

6. *Conclusions*

This chapter has given 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 in Ref. 3 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 Refs. 4–7). 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 the part of the semantics that is independent of security a separate library that can be reused by many entities, and reinterpret only the 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 chapter.

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. Support for J. Wing was provided in part by the National Science Foundation under grant No. CCR-8620027 and for J. D. Tygar under a National Science Foundation Presidential Young Investigator Award, contract No. CCR-8858087. M. Maimone is also supported by a fellowship from the Office of Naval Research, under contract No. N00014-88-K-0641.

References

1. J. D. Tygar and J. M. Wing, Visual specification of security constraints, in Proceedings of the 1987 IEEE Workshop on Visual Languages, Linkoping, Sweden, August 1987.
2. D. Harel, Statecharts: A visual formalism for complex systems, *Sci. Comput. Programm.* **8**, 231–274, 1987.

3. A. Heydon, M. Maimone, J. D. Tygar, J. Wing, and A. Moormann-Zaremski, Constraining pictures with pictures. In 11th IFIP World Computer Conference, August 1989.
4. D. E. Bell and L. J. LaPadula, Secure computer systems: Mathematical foundations (3 volumes), Technical Report AD-770 768, AD-771 543, AD-780 528, Mitre Corporation, November 1973.
5. D. Good, R. Cohen, C. Hoch, L. Hunter, and D. Hare, Report on the language Gypsy, version 2.0, Technical Report ICSCA-CMP-10, Certifiable Minicomputer Project, The University of Texas at Austin, September 1978.
6. T. Benzel, Analysis of a kernel verification, in Proceedings of the 1984 Symposium on Security and Privacy, Oakland, California, pp. 125–131, May 1984.
7. 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, Technical Report CSL-116, SRI, May 1980.
8. A. Heydon, M. Maimone, J. D. Tygar, J. Wing, and A. Moormann-Zaremski, Miró tools. In Proceedings of the 1989 IEEE Workshop on Visual Languages, October 1989.
9. A. Heydon, M. Maimone, J. D. Tygar, J. Wing, and A. Moormann-Zaremski, Miró: Visual Specification of Security, Technical Report CMU-CS-89-199, Carnegie Mellon, November 1989.